

Learning by Knowledge Exchange in Logical Agents

Stefania Costantini

Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
Email: stefcost@di.univaq.it

Arianna Tocchio

Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
Email: tocchio@di.univaq.it

Abstract—In this paper we introduce a form of cooperation among agents based on exchanging sets of rules. In principle, the approach extends to agent societies a feature which is proper of human societies, i.e., the cultural transmission of abilities. However, acquiring knowledge from untrustworthy agents should be avoided, and the new knowledge should be evaluated according to its usefulness. After discussing the general principles of our approach, we present a prototypical implementation.

I. INTRODUCTION

Adaptive autonomous agents are capable of adapting their behavior according to changes in the environment. Then, adaptive agents must take profit of past experiences using some learning approach. As it is widely acknowledged, the effects of learning should include at least one of the following:

- The range of behaviors is expanded: the agent can do more.
- The accuracy on tasks is improved: the agent can do things better.
- The speed is improved: the agent can do things faster.

According to [16] [2], three learning techniques are usable to develop adaptive autonomous agents: reinforcement learning, models learning and classifier systems.

In reinforcement learning, the mechanism consists in assigning rewards (weights) to actions that contribute to the resolution of a problem. This approach has been used for instance to improve coordination between autonomous agents [17]. In models learning, agents try to find causal relations between their actions and the events occurring in the environment. In general, they use either probabilistic models or logical models. The original ideas by [16] have then been widely developed in various approaches. Induction (and also inductive logic programming) is often considered as a particular form of models learning. Classification is the most common form of automatic learning. In the agent context, an agent can try to classify applicable rules by setting priorities and then updating these priorities according to the results achieved [12]. Memory-based reasoning (MBR) is based on the idea that if a given action took place in the past in a given situation s and gave good results, it will be useful in a new situation s' similar to s . Incremental learning techniques

have been recently introduced for improving MBR in dynamic applications where data arrive continuously [9].

In a multi-agent setting however, other forms of learning can be introduced that, though related to the classical ones, are specifically tailored to multi-agent systems (MAS) topics and issues. For instance in [10], in order to recall practical solutions to coordination problems, agents learn coordinated procedures from execution traces and store them into a case-base that is organized around expectations about other agents. Agents also learn better estimates for how likely individual actions are to succeed in order to improve the quality of decisions when planning, communicating, and adapting plans.

In this paper we discuss a learning approach useful to improve adaptive behavior in computational logic agents. We assume that whatever the formalism, these agents have a rule-based knowledge base. The approach is centered on the possibility of exchanging sets of rules between agents. These sets of rules can either define a procedure, or constitute a module for coping with some sort of situation, or be just a segment of a knowledge base. However, agents should then be able to evaluate how useful the new knowledge is. To this extent, we propose two techniques.

The first technique associates to the acquired knowledge a specific objective, meaning that the new rules should help the agent to reach that objective. After a while, the agent will evaluate whether (or to which extent) the objective has been reached. If the evaluation is unsatisfactory, the new knowledge can be discarded. There is a clear similarity with reinforcement learning, where here the action that is to be evaluated is the use of the new knowledge.

The second technique consists in acquiring the same knowledge from several other agents, and then comparing the results. The comparison is made based on a meta-specification, i.e., based for instance on efficiency, or on a measure of similarity of the results. The comparison will state which versions pass a given threshold, and which don't. The unsatisfactory ones will be discarded.

In a real application, a directory agent can be employed so as to inform agents of where to find the required knowledge. This directory agent may in principle be notified of the updates of the level of trust performed by agents that have

acquired a piece of knowledge from a certain agent, and thus compute and exhibit a value that represent the *reputation* of that agent. To this aim, the directory agent will employ suitable algorithms (for instance those presented in [19]) to assess the past behavior of agents, so as to allow avoidance of untrustworthy agents in future.

In Section 2 we discuss the proposed approach at some length. In Section 4 we present a prototypical implementation of the approach in the agent-oriented programming language DALI [3] [6], after shortly summarizing the main DALI features (Section 3). In DALI, all the above can take profit from the DALI communication architecture, that allows the agent to filter incoming and out-coming messages according to any kind of constraint, including trust [5]. Then for instance, new knowledge will be learned by trusted agents only; successful evaluation of the acquired knowledge can lead to an increase of the level of trust of the sending agent, while a decision to discard that knowledge can also result in a decrease of the level of trust. Moreover, the trial of different version of the same knowledge can be made in parallel, by exploiting the DALI children generation capability [7] that allows the agent to create sub-agents on specific tasks.

II. LEARNING BY RULE EXCHANGE

Learning may allow agents to survive and reach their goals in environments where a static knowledge is insufficient. The environmental context changes, cooperative or competitive agents can appear or disappear, ask for information, require resources, propose unknown goals and actions. Then, agents may try to improve their potentiality by interacting with other entities so as to perform unknown or difficult tasks.

One of the key features of MAS is the ability of “sub-contracting” computations to agents that may possess the ability to perform them. More generally, agents can try to achieve a goal by means of cooperative distributed problem-solving. However, on the one hand not all tasks can be delegated and on the other hand agents may need or may want to acquire new abilities to cope with unknown situations. In our view, an improvement in the effectiveness of MAS may consist in introducing a key feature of human societies, i.e., cultural transmission of abilities. Without this possibility, agents are limited under two important respects:

- they are unable to expand the set of perceptions they can recognize, elaborate and react to;
- they are unable to expand their range of expertise.

Indeed, the flexibility and thus the “intelligence” of agents will increase if they become able not only to refine but also to enlarge their own capabilities. The need of acquiring new knowledge can be recognized by an agent at least in relation to the following situations:

- 1) There is an objective that the agent has been unable to reach: it has been unable to relate a plan (in the KGP perspective [13]) or intention (in a BDI perspective [18]) to that objective (or desire) and it has to acquire

new knowledge (beliefs). As a particular case, there is a situation the agent is unable to cope with; for instance, there is an exogenous event that the agent does not recognize.

- 2) There is some kind of computation that the agent is unable to perform.

Assuming that the agent establishes that it cannot resort to cooperation to get its task performed, it can still resort to cooperation in order to try to acquire the necessary piece of knowledge from another agent. The problems involved in this issue are at least the following: how to ask for what the agent needs; how to evaluate the actual usefulness of the new knowledge; and, how this kind of acquisition can be semantically justified in a logical agent.

In this context, we make the simplifying assumption that agents speak the same language, and thus we overlook the problem of ontologies that in an actual implementation would of course arise. We also assume that, whatever the underlying formalism, agents have a rule-based knowledge base. Two feasible ways of asking other agents can be:

- Ask by keyword, assuming that other agents have a way of matching the keyword with a piece of knowledge. Some kind of pattern-matching will have to be used by an agent in order to establish whether it can answer a request.
- Ask by predicate name.

An agent that would accept to give the requested knowledge, should answer by providing, together with the piece of knowledge, some kind of “control” information that should include at least:

- A specification of the way of using that knowledge, that specifies whether the rules apply automatically, e.g., in the case of reactive rules, or if there is either a predicate or a procedure to be invoked.
- In the former case, specify the format of the external event that triggers the rules; in the latter, specifying the invocation pattern of the predicate/procedure.

The details of the above are left to the specific implementation, related to the language/formalism in which the agents are expressed. Notice that it is not required that the involved agents be based on the same inference mechanism. However, they should be somehow “compatible”, i.e., a prolog-based agent might acquire an Answer-Set program [21] and then use it, assuming that it is able to invoke an Answer-Set solver. Clearly, the exchanged piece of knowledge should include all the *relevant rules*, i.e., all the rules which are needed (directly or indirectly [8]) for actually exploiting that knowledge.

At this stage, the receiver agent has to face two problems:

- (a) Establish whether the new knowledge is consistent, or at least compatible, with its knowledge base. This is a topic which has long been studied in belief revision [1]. However, we assume that the new knowledge is not directly incorporated to the existing knowledge base. On the contrary, in the first stage the new knowledge

is distinct from the existing well-established knowledge base, as it must be evaluated before being accepted.

- (b) Establish whether the new knowledge is actually useful to the purposes for which it has been acquired. If so, it can possibly be asserted in the knowledge base. Otherwise, it can possibly be discarded.

Then, agents should be able to evaluate how useful the new knowledge is. Similarly to reinforcement learning, techniques must be identified so as to make this evaluation feasible with reasonable efficiency. Simple techniques to cope with this problem can be the following.

- 1) The new knowledge had been acquired in order to reach an objective: the agent can confirm/discharge the new knowledge according to its reaching/not reaching the objective. This evaluation can be related to additional parameters, like e.g. time, amount of resources needed, quality of results.
- 2) The new knowledge has been acquired for performing a computation: the agent can acquire the same knowledge by several sources, and compare the results. Results which are not “sufficiently good” (given some sort of evaluation) lead to the elimination of the related piece of knowledge. The others are used (compared/combined) to produce the accepted result.

A. Semantics of Learning by Rule Exchange

The semantics of Computational Logic agent languages may in principle be expressed as outlined in [3] for the DALI language. I.e., given program P_{Ag} , the semantics is based on the following.

- 1) An *initialization step* where P_{Ag} is transformed into a corresponding program P_0 by means of some sort of knowledge compilation (which can be understood as a rewriting of the program in an intermediate language).
- 2) A sequence of evolution steps, where reception of each event is understood as a transformation of P_i into P_{i+1} , where the transformation specifies how the event affects the agent program (e.g., it is recorded).

Then, one has a Program Evolution Sequence $PE = [P_0, \dots, P_n]$ and a corresponding Semantic Evolution Sequence $[M_0, \dots, M_n]$ where M_i is the semantic account of P_i (in [3] M_i is the model of P_i).

This semantic account can be adapted by transforming the initialization step into a more general knowledge compilation step, to be performed:

- (i) At the initialization stage, as before.
- (ii) Upon reception of new knowledge.
- (iii) In consequence to the decision to accept/reject the new knowledge.

III. DALI IN A NUTSHELL

DALI [3] [6] [20] is an Active Logic Programming language designed in the line of [14] for executable specification of logical agents. The Horn-clause language is a subset of

DALI, which however includes the following agent-oriented features. The reactive and proactive behavior of the DALI agent is triggered by several kinds of events: external events, internal, present and past events. All the events and actions are timestamped, so as to record when they occurred.

An external event is a particular stimulus perceived by the agent from the environment. In fact, we define the set of external events perceived by the agent from time t_1 to time t_n as a set $E = \{e_1 : t_1, \dots, e_n : t_n\}$ where $E \subseteq S$, and S is the set of the external stimuli that the agent can possibly perceive.

A single external event e_i is an atom indicated with a particular postfix in order to be distinguished from other DALI language events. More precisely:

Definition 1 (External Event): An external event is syntactically indicated by postfix E and it is defined as:

$ExtEvent ::= \langle \langle Atom_E \rangle \rangle | seq \langle \langle Atom_E \rangle \rangle$

where an *Atom* is a predicate symbol applied to a sequence of *terms* and a *term* is either a constant or a variable or a function symbol applied in turn to a sequence of terms.

External events allow an agent to react through a particular kind of rules, reactive rules, aimed at interacting with the external environment. When an event comes into the agent from its “external world”, the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event. The special token $:>$, used instead of $-$, indicates that reactive rules performs forward reasoning.

Definition 2 (Reactive rule): A reactive rule has the form:

$ExtEvent_E :> Body$ or
 $ExtEvent_{1E}, \dots, ExtEvent_{nE} :> Body$

The agent remembers to have reacted by converting the external event into a *past event* (time-stamped). Operationally, if an incoming external event is recognized, i.e., corresponds to the head of a reactive rule, it is added into a list called *EV* and consumed according to the arrival order, unless priorities are specified.

The internal events define a kind of “individuality” of a DALI agent, making it proactive independently of the environment, of the user and of the other agents, and allowing it to manipulate and revise its knowledge. More precisely:

Definition 3 (Internal Event): An internal event is syntactically indicated by postfix I :

$InternalEvent ::= \langle \langle Atom_I \rangle \rangle$

The internal event mechanism implies the definition of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen:

$IntEvent : -Conditions$
 $IntEvent_I :> Body$

The goal defined in the first rule is automatically attempted with a default frequency customizable by means of directives in the initialization file. Whenever it succeeds, the internal event “has happened”, and the reaction (second rule) is trig-

gered as if it were an external one. A DALI agent is able to build a plan in order to reach an objective, by using internal events of a particular kind, called *planning goals*.

Actions are the agent's way of affecting the environment, possibly in reaction to either an external or internal event. An action in DALI can be also a message sent by an agent to another one.

Definition 4 (Action): An action is syntactically indicated by postfix *A*:

Action ::=

$\ll Atom_A \gg | message_A \ll Atom, Atom \gg$

Actions take place in the body of rules.

If an action has preconditions, they are defined by action rules, emphasized by a new token:

Definition 5 (Action rule): An action rule has the form:

Action :< *Preconditions*.

Similarly to external and internal events, actions are recorded as past actions.

Past events represent the agent's "memory", that makes it capable to perform future activities while having experience of previous events, and of its own previous conclusions. Past events are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file. A past event is syntactically indicated by the postfix *P*.

Procedurally, DALI is based on an Extended Resolution Procedure that interleaves different activities, and can be tuned by the user via directives.

The operational semantics of DALI is based on Dialogue Games Theory [4] [20]: the DALI Interpreter is modeled as a set of cooperating players. By means of this approach one is able to prove formal properties of the language in the form of properties that the game will necessarily fulfil.

A. DALI Communication Architecture

The DALI communication architecture consists of four levels. The first and last levels implement the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not receive (*told* check level) or send a message (*tell* check level). The DALI communication filter is specified by means of meta-level rules defining the distinguished predicates *tell* and *told*. Whenever a message is received, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *told* rule. If such a rule is found, the interpreter attempts to prove *told(Sender,primitive(Content))*. If this goal succeeds, then the message is accepted, and *primitive(Content)* is added to the set of the external events incoming into the receiver agent. Otherwise, the message is discarded. Symmetrically, the messages that an agent sends are subjected to a check via *tell* rules. The second level includes a meta-reasoning layer, that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The third level consists of the DALI interpreter.

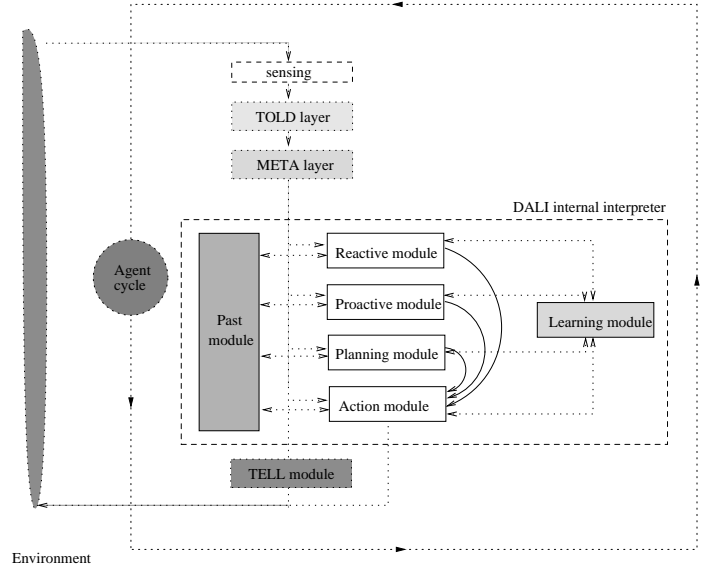


Fig. 1. DALI communication architecture

B. Children generation capability

We have introduced in the DALI framework the ability to generate children agents [7]. An important motivation for this improvement has been the need for our agents to face non-trivial planning problems by means of the invocation of a performant planner, such as for instance an Answer Set Solver. [11] [15] [21]. As a planning process can require a significant amount of time, the possibility for an agent to assign this time-expensive activity to its children can constitute a real advantage.

Another motivation for generating children is, more generally, that of splitting an agent goal into subgoals to be delegated to children. This possibly with the aim of obtaining different results by means of different strategies, and then comparing the various alternatives and choosing the best ones. The father provides the child with all the information useful to find the solution and, optionally, with an amount of time within which to resolve the assigned problem.

IV. BASIC LEARNING MECHANISMS

Agents that adopt forms of learning to improve their behavior can in perspective deal with more complex jobs, but expose themselves to some risks. The learned information could be either intentionally or accidentally wrong or simply not consistent with the agent specialization. Agents knowledge is generally divided into a set of facts and rules: the former represent the agent "beliefs" about itself and the world, while the latter determine the entity behavior. If learning one or more beliefs (plain facts) implies a certain degree of risk, adding rules coming from other agents to the knowledge base can

very dangerous. Thus, in our view it is necessary to elaborate different learning strategies for beliefs and rules, reserving to the latter case a more sophisticated acquisition process. In the following subsections we will first propose an approach to manage the exchange of facts and then we will discuss the more general problem of rules learning.

A. Beliefs learning

The belief base of DALI agents is composed of the facts which are present in the agent logic program, dynamically augmented by past events. Past events keep trace of what the agent has done/observed before: external and internal events, performed actions, reached internal conclusions and pursued planning activities.

Past events also represent external world knowledge. In fact, a DALI agent can ask for some facts by using the primitives *is_a_fact(Fact, Ag)* and *query_ref(Fact, Match_number, Ag)* where: *Fact* is the desired information, *Match_number* represents the number of matches that the agent intends to receive and *Ag* is the name of the agent asking for the fact. While the first primitive allows the entity to require a ground fact, the second one supports the requests of non-ground information. Then, if for instance agent *dave* wants to know who is the lover of *susy*, it can send the following message to, e.g., *susy*'s friend *kate*:

```
message_A(kate, query_ref(love(Y, susy), 1, dave)).
```

If the beliefs base of *kate* contains only the fact *love(susy, peter)*, no matching is directly found. This problem has been overcome in DALI via a meta-level support that, by using both ontologies and properties of relations, tries to “understand” message contents (namely, message contents are automatically subjected to a procedure *meta* which is predefined though user-customizable). In this case, if the ontology of *kate* contains information on the symmetry of the predicate ‘love’, *loves(Y, susy)* can be matched with *love(susy, peter)* and the agent *kate* will return the result:

```
send_message_to(dave,
  inform(query_ref(loves(Y, susy), 1),
    values([loves(susy, peter)]), kate),
  italian, [])
```

Once received the desired information, the agent *dave* will update its beliefs by adding, as a past event, the fact:

```
past_event(loves(susy, peter), 479379, kate).
```

where first value is the information about *susy* and *peter*, the second value is the acquisition time and the third value keeps track of the information source. The sender agent name is relevant: if trust in this agent reliability will be reduced under a certain threshold by negative cooperation experiences, all partial beliefs coming from it could be eliminated. At the same time, the *told filter* will get rid of at priori each communication act sent by the unreliable agent:

```
told(Sender, query_ref(Fact, Match_number)) : -
  not(unreliable_P(Sender_agent)).
```

Another direct acquisition beliefs method in DALI agent is based on the *confirm* primitive. This method allows an agent to send a fact to another one. Also in this case, the fact will be added to the agent beliefs only if the message will overcome the told filter. For example, if the agent *dave* intends to send to *peter* the information *bought(car, red)*, it will send the message:

```
message_A(peter, confirm(bought(car, red), dave)).
```

and the *peter* beliefs will contain the past event: *past_event(bought(car, red), 479379, dave)*.

A fact can be eliminated from the agent knowledge base by using the *disconfirm* primitive.

B. Rules learning

The rule-exchange approach to learning proposed in this paper is a first step into the complex world of learning rules. For instance, a DALI agent, when receiving a stimulus whose reaction is unknown, can ask other agents for acquiring rules capable of suggesting the right behavior to adopt. While retrieving and adding rules to the knowledge base is not difficult, the relevant problem of learning a correct information remains.

Intelligent agents can have different specializations for different contexts and a learning rules process cannot ignore this. Moreover, even agents having the same specializations can adopt behavioral rules which are mutually inconsistent. What could the solution be?

In the prototypical implementation that we present here, the learning rules process includes several steps starting from a verification of the source reliability. The solution is based on the introduction of a *mediator agent* that we call *yellow_rules_agent*, keeping track of the agents specialization and reliability. When an entity needs to learn something, it asks the *yellow_rules_agent* for the names agents having the same specialization and being more reliable.

Once obtained this information, the agent may acquire the desired knowledge by some of them. If the agent will finally decide to incorporate the learned rules in its program because they work correctly, it will also send to *yellow_rules_agent* a message indicating satisfaction. This will result in an increment of the reliability of the agent that has provided the rules. A negative experience will imply an unfavorable dispatch. In the present implementation agents return a numeric value indicating the “level of trust”. We mean to add also the objective that the receiver agent meant to reach via the new knowledge, so as to conditionally rate agents with respect to this point. This in order to avoid a low reliability esteem for agents which are actually reliable in their own area of expertise. In fact, it may happen that an agent has a very specialized (and accurate) rule set which is mistakenly matched against some requests. In updating the level of trust the *yellow_rules_agent* should

adopt a model that updates trust only when the information is sufficient, i.e., after a certain number of reports which are in accordance, sent by reliable agents.

The learned rules will be added to the agent knowledge base in the form of past events. A preliminary check will verify some properties such as, for example, the syntactic correctness or consistency. Rules that “survive” this check will be used by the agent in its activities, and their usefulness and efficiency will be recorded. After a certain time, according to results the acquired rules will be either definitely learned or eliminated. Below we describe in more detail all steps involved in our cooperative rules learning approach.

1) **New knowledge is needed.** A DALI agent behavior is described by: (i) a set of rules determining what reaction to apply in response to external stimuli; (ii) a set of rules useful to draw internal conclusions or to reach goals via planning strategies; (iii) a set of rules containing conditions for reacting external events or for performing actions; (iv) a set of horn-clauses. The need to acquire new knowledge arises whenever an agent receives a communication act whose content is unknown and the meta-level does not succeed in searching for a semantically equivalent content recognized by the entity. The communication act could be an external event, a proposed action, a request of information and so on. Having no internal means to cope with this situation, the agent activates the learning rules process. This process is risky enough, so the agent must try to search a suitable information source.

2) **Looking for information sources.** Our learning architecture allows a particular agent, *yellow_rules_agent*, to maintain the information useful to identify the desired rules source or sources. Each agent living in the environment is identified by the tuple:

$$source(A_i, S_i, KR_i, Q_i)$$

where the first parameter represents the agent identification and the second one is a string synthesizing the agent role in the environment. The third one is a list of rules keys that the agent A_i is willing to transfer to other agents. The fourth one is the reliability value, computed by *yellow_rules_agent* according to positive and negative feedbacks. In fact, agents that receive rules from $agent_i$, at the end of the verification phase send a message to *yellow_rules_agent* rating that knowledge. According to current and past values average, the *yellow_rules_agent* computes $agent_i$ reliability by means of some kind of evaluation. For example, if the agent *dave* is a barman and is available to give to others the rules useful to serve a drink, the tuple *source* might be for instance: $source(dave, barman, [serve_drink], 0.6)$.

Whenever an agent A_k having the specialization S_k needs some rules, it will send to *yellow_rules_agent* the message:

$$message_A(yellow_rules_agent, search_sources(A_k, S_k, Key_k))$$

where Key_k is meant to indicate the desired rules. More precisely, this parameter allows *yellow_rules_agent* to identify agents having the right information by finding a correspondence between Key_k and the elements of the rules keys list KR_i .

If one or more agents fulfill the correspondence, the agent A_k will receive as a response the list of reliable agents corresponding to the expected specialization S_k and the key Key_k :

$$L_s = [(A_1, Q_1), \dots, (A_n, Q_n)]$$

If no agents are available, the response will be the empty list.

Having chosen the names of one or more agents to which one can ask missing rules according to the *yellow_rules_agent* and personal reliability evaluation, the agent will then contact them.

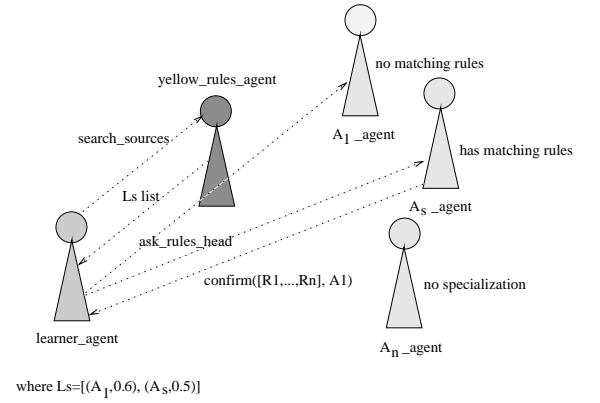


Fig. 2. A cooperative learning scenario

3) **Asking for missing rules.** In order to get the needed piece of knowledge, the agent can choose one of two techniques: the first one allows an agent to learn all required rules by specifying their heads. This implies that there be a strict correspondence between the heads of rules in two or more agents in order to be able to activate the learning process. But, agents often came from different platforms and technologies, so this correspondence could hardly be found. This limit can be overcome by adopting ontologies capable of matching rule heads which though looking syntactically different are semantically equivalent. If we consider the agent A_k having selected the couple (A_s, Q_s) and the head H_l , the following message will propose to the receiver agent the exchange of rules having the head H_l :

$$message_A(A_s, ask_rules_head(H_l, A_k))$$

The second technique allows an agent to ask for a specific key that can match with either the head or the

body of rules in the agent program. We may notice that in this manner the probability of finding corresponding rules will be higher, though the rules appropriateness and usefulness could be more in question. In this case, the message syntax will be:

$message_A(A_s, ask_rules_key(Key, A_k))$

Agents accepting the proposal to exchange rules that match with either the *Head* or the *Key* will pack all retrieved rules and will send them back to the A_k entity.

$message_A(A_k, sent_rules([R_1, \dots, R_n], Ch, A_s))$

The parameter *Ch* represents the goal that must be invoked in order to activate the rules. In particular, if the agent *bob* receives from the agent *dave* the rules:

$[danger_E :> call_policeA,$
 $call_police :< have_a_phone]$

the parameter *Ch* will correspond to $danger_E$.

As soon as these rules will be received by the learner agent, they will be unpacked and asserted as past events in its knowledge base with the suffix $learn_P(Rule, Time, Sender, Objective)$. Each rule will be re-asserted in a second version, where more information is associated to it, and in particular: the current time; the sender agent name; a parameter *Objective*, useful to remember what was the goal for which the request had been issued. For example, if the agent was in a dangerous situation when it requested the rule with the key *help*, it will memorize that the *Objective* of this learning rule process was to get safe. The objective introduction will allow the agent, after learning the rules, to check their effectiveness with respect to the associated goal.

- 4) **Add learned rules.** Rules added as past events are managed by a specific internal event, $gest_learning(Rule)$, that implements the first filter level. This internal event filters one rule at the time. In order not to slow down the agent, this operation is performed in suitable time slots, i.e., when the agent is not performing complex tasks and the events queues have few items to be processed. Each rule is taken into account in order to be added to the knowledge base, and must fulfill two conditions (expressed in the first rule of the internal event): $learn_if(Rule, Time, Sender)$ and $properties_true(Rule)$. The first condition, $learn_if(Rule, Time, Sender)$, is similar to a *told* one: the user can define in the same file of tell/told rules a set of constraints that the considered *Rules*, the *Time* and the *Sender* agent must respect:

$learn_if(Rule, Time, Sender) : -$
 $constraint_1, \dots, constraint_n.$

Constraints can avoid adding an incoming rule from an agent that was reliable for *yellow_rules_agent*, but is considered instead unreliable by the receiver agent under some different perspective. *Time* can be used to either

```
gest_learning(Rule) : -
    learn_P(Rule, Time, Sender),
    learn_if(Rule, Time, Sender),
    properties_true(Rule).
gest_learning_I(Rule) :>
    accept_at_present_A(Rule).
```

Fig. 3. First learning process filter

force or delay the assertion of the *Rules*. Other domain- or situation-dependent constraints can be expressed. The second condition, $properties_true(Rule)$, takes more specific properties of the *Rules* into account, e.g.:

- the syntactic correctness according to prolog and DALI language;
- the absence of procedure calls without a corresponding procedure;
- the overlap of rules originating from different agents;
- the rule consistence with respect to previously learned clauses.

If certified by the internal event, the rules are added to the agent program with a label indicating that they are to be submitted to second filter. A particular label will emphasize that the rules have been learned provisionally. Final learning will take place only if the rules will overcome the second filter level, based on usefulness and efficiency. Some rules discarded by the first filter can remain for some time in the agent knowledge base, waiting for a successive integration. In fact, the learning process can generate new contexts where some previously false properties become true.

In order to avoid a rule that cannot be learned to be kept for too long in the agent memory, we have introduced a particular internal event that eliminates all past events $learn_P(Rule, Time, Sender, Objective)$ that has been kept for an amount of time that exceeds a threshold.

C. Exploiting basic mechanisms

Rules added to the agent program in order to be evaluated wait for the moment in which the agent will be in need of them. The estimate of their usefulness depends strictly on the kind of learned rules. Some, expressing a set of actions that the agent needs to perform, can be evaluated by examining the correspondence between the entity objective and the exhibited behavior. Some, useful to execute complex operations, can be evaluated by examining for instance the time spent in the calculation and the result quality. Here we propose two sample methods to estimate partially learned rules.

- **On Objectives** Once introduced in the agent program, each piece of knowledge is used by the entity during its life, keeping always track of its performance with respect to the corresponding objective. This testing phase can be performed in two modalities. The first one is based on the correspondence between the expressed *Objective* in $learn_P(Rule, Time, Sender, Objective)$ and the effective rule application result. For example, if *Objective*

```

check_objective(Rules, Value) : -
    learnP(Rules, Time, Sender, Objective),
    expired_time(Time, T),
    evaluate(Rules, Objective, Value).
check_objectiveI(Rules, Value) : >
    evaluate_rule(Rules, Value).
evaluate_rule(Rules, Value) : -
    accept_definitelyA(Rules, Value).
accept_definitely_rule(Rules, Value) : <
    Value > Threshold.
evalreject_ruleA(Rules, Value).
reject_ruleA(Rules, Value) : <
    Value < Threshold.

```

Fig. 4. Second learning process filter

expresses the agent safety, we would expect that the agent be safe or at least having made some progress in this direction. The utility and efficiency test is implemented by an internal event that, whenever a learned rule is invoked, checks from time to time its effect. The evaluation is performed by the function *evaluate(Rules, Objective, Value)* that considers:

- the degree of correspondence between the *Objective* and past events generated by the *Rules* application;
- given the state snapshot of the program execution, the degree of correspondence between the saved state and the declared *Objective*.

For each usage of *Rules*, the returned *Value* increments/decrements the average calculated on the past evaluations. After some time, a negative result implies the rule elimination while a positive one determines a final learning. However, the agent maintains information on the *Rules* sources, so that also in future what is learned can be eliminated if, for example, the source becomes unreliable.

- **On comparison** If the learned rules are aimed at some kind of complex computation that returns a result, a suitable testing method can be adopted. The agent can generate children, and can assign each child a different set of rules acquired by different sources for the same calculation. The results, together with performance, time and resources spent, will be returned to the father that can decide which set of rules is better to adopt.

V. CONCLUSIONS

We have proposed a form of cooperation among agents that consists in improving each agent's skills by acquiring new knowledge from the others. The approach aims at extending to agent societies a feature which is proper of human societies, i.e., the cultural transmission of abilities. We have outlined the problems and advantages of this approach, and have discussed a prototype implementation in DALI. More experimental work is needed for proving the effectiveness of the approach, and for putting various methods of verification of the usefulness of learning at work. Indeed, the mechanisms for matching needs against rule sets of other agents (keywords or rule heads) is

quite preliminary and must be checked in real applications, as it might result in low "precision", i.e., too many matches are found and "recall", i.e., too many matches are discarded.

REFERENCES

- [1] G. Antoniou (with contributions by M.-A. Williams). *Nonmonotonic Reasoning*, The MIT Press, Cambridge, Massachusetts, 1997, ISBN 0-262-01157-3.
- [2] W. Brenner, R. Zarnekow and H. Wittig. *Intelligent Software Agents, Foundations and Applications*, Springer Verlag, Berlin, Germany, 1998.
- [3] S. Costantini and A. Tocchio. *A Logic Programming Language for Multi-agent Systems*, In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002.
- [4] S. Costantini, A. Tocchio and A. Verticchio. *A Game-Theoretic Operational Semantics for the DALI Communication Architecture*, In: Proc. of WOA04, Pitagora Editrice Bologna, ISBN: 88-371-1533-4, Also available on-line, at the URL: <http://woa04.unito.it/Pages/atti.html>
- [5] S. Costantini, A. Tocchio and A. Verticchio. *Communication and Trust in the DALI Logic Programming Agent-Oriented Language*, In: Proc. of the Italian Conference on Intelligent Systems AI*IA'04, 2004.
- [6] S. Costantini and A. Tocchio. *The DALI Logic Programming Agent-Oriented Language*, In: Proceedings of the 9th European Conference, Jelia 2004, Lisbon, September 2004. LNAI 3229, Springer-Verlag, Germany, 2004.
- [7] S. Costantini and A. Tocchio. *Enhancing Computational power: DALI child agents generation*, In: Electronic proceedings of CILC'05, Italian Conf. on Comp. Logic, Roma, 21-22 giugno 2005, URL <http://www.disp.uniroma2.it/CILC2005/Programma.html>.
- [8] J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties*, Fundamenta Informaticae 22(3), 1995.
- [9] F. Enembreck and J.-P. Barthès. *ELA - A new Approach for Learning Agents*, Journal of Autonomous Agents and Multi-Agent Systems, 3(10): 215-248, 2005.
- [10] A. Garland and R. Alterman. *Autonomous Agents that Learn to Better Coordinate*, J. Autonomous Agents and Multi-agent Systems, 2004.
- [11] M. Gelfond and V. Lifschitz. *The Stable Model Semantics for Logic Programming*, In: Proc. of the Fifth Joint International Conference and Symposium. The MIT Press, 1988, 1070-1080.
- [12] J. H. Holland. *Escaping brittleness: The possibility of general-purpose learning algorithms applied to parallel rule-based systems*, In: Machine Learning, an Artificial Intelligence Approach, Morgan-Kaufman, vol. 2, 1986.
- [13] A. Kakas, P. Mancarella, K. Stathis, F. Sadri and F. Toni. *The KGP Model of Agency*, In: ECAI 04, Proc. of the 16th European Conf. on Artificial Intelligence, 2004.
- [14] R. A. Kowalski. *How to be Artificially Intelligent - the Logical Way*, Draft, revised February 2004, Available on line, URL <http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html>.
- [15] V. Lifschitz. *Answer Set Programming and Plan Generation*. Artif. Intelligence 138 (1-2), Elsevier Science Publishers, 2002, 39-54.
- [16] P. Maes. *Modeling Adaptive Autonomuos Agents*, Artificial Life Journal, 1(1-2): 135-162, MIT Press, 1994.
- [17] Oliveira E. *Agent, advanced features for negotiation and coordination*, In M. Luck (ed.), ACAI 2001, LNAI 2086: 173-186, Springer-Verlag, 2001.
- [18] A. S. Rao and M. Georgeff. *Modeling rational agents within a BDI-architecture*, In: Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91). Morgan Kaufmann, 1991: 473-484.
- [19] J. Sabater and C. Sierra. *Reputation and social network analysis in multi-agent systems*, In: Proceedings of the First Int. Joint Conf. on Autonomous Agents and Multi-agent Systems: 475482. ACM Press, 2002.
- [20] A. Tocchio. *Multi-Agent Systems in computational logic*, Ph.D. Thesis, Dipartimento di Informatica, Università degli Studi di L'Aquila, 2005.
- [21] Material about Answer Set Programming (ASP) and web location of ASP solvers. <http://tinfp2.vub.ac.be/wasp/bin/view/Wasp/WebHome>.

Languages for Programming BDI-style Agents: an Overview

Viviana Mascardi, Daniela Demergasso, Davide Ancona
DISI, Università di Genova, Via Dodecaneso 35, 16146, Genova, Italy
mascardi@disi.unige.it, 1996s165@educ.disi.unige.it, davide@disi.unige.it

Abstract—The notion of an intelligent agent as an entity which appears to be the subject of mental attitudes like beliefs, desires and intentions (hence, the BDI acronym) is well known and accepted by many researchers. Besides the definition of various BDI logics, many languages and integrated environments for programming BDI-style agents have been proposed since the early nineties. In this reasoned bibliography, nine languages and implemented systems, namely PRS, dMARS, JACK, JAM, Jadex, AgentSpeak(L), 3APL, Dribble, and Coo-BDI, are discussed and compared. References to other systems and languages based on the BDI model are also provided, as well as pointers to surveys dealing with related topics.

I. INTRODUCTION

The notion of an intelligent agent as an entity which appears to be the subject of beliefs, desires, commitments, and other mental attitudes, is well known and accepted by many researchers [1]. The philosopher Dennett has coined the term *intentional system* to denote systems of this kind [2].

In order to formalise intentional systems, different logics have been developed, among which Cohen and Levesque's theory of intentions [3], and Rao and Georgeff's Belief-Desire-Intention logic [4], [5], [6]. However, intelligent software agents cannot just be formalised using ad-hoc logical languages: they must be programmed using executable languages, as any other piece of software. Hence, there is a pressing need of programming languages which can fill the gap between the logical theory and the practical issues concerned with software agents' development.

One of the computational models that gained more consensus as a candidate to fill this gap is the Belief-Desire-Intention (BDI) one [7], which, as the acronym itself suggests, is characterized by the following concepts:

- *Beliefs*: the agent's knowledge about the world.
- *Desires*: the objectives to be accomplished.
- *Intentions*: the courses of actions currently under execution to achieve the agent's desires.

Besides these components, the BDI model includes a *plan library*, namely a set of "recipes" representing the procedural knowledge of the agent, and an *event queue* where both events (either perceived from the environment or generated by the agent itself to notify an update of its belief base) and internal subgoals (generated by the agent itself while trying to achieve a desire) are stored.

The typical BDI execution cycle is characterised by the following steps:

1. observe the world and the agent's internal state, and update the *event queue* consequently;
2. generate possible new plan instances whose trigger event matches an event in the event queue (*relevant* plan instances) and whose precondition is satisfied (*applicable* plan instances);
3. select for execution one instance from the set of applicable plan instances;
4. push the selected instance onto an existing or new *intention stack*, according to whether or not the event is a (sub)goal;
5. select an intention stack, take the topmost plan instance and execute the next step of this current instance: if the step is an action, perform it, otherwise, if it is a subgoal, insert it on the event queue.

Usually, BDI-style agents do not adopt first principles planning at all, as all plans must be generated by the agent programmer at design time. The planning done by agents consists entirely of context-sensitive subgoal expansion, which is deferred until a point in time at which the subgoal is selected for execution.

This paper provides an overview of languages and implemented systems for programming BDI-style agents. In Section II, nine systems based on the BDI model are surveyed, and in Section III they are compared along seven dimensions. Our knowledge about these nine systems, comes both from our own readings and experience, and from talks with most of the authors of the systems themselves, in particular with R. Bordini (AgentSpeak), M. Dastani (3APL), M. J. Huber (JAM), A. Pokahr (Jadex), M. B. van Riemsdijk (Dribble), and M. Winikoff (AgentTalk). All of them have been asked to check the content of Section III before submission¹. Section III also contains pointers to other existing systems, together with references to related work.

II. BDI-STYLE LANGUAGES AND SYSTEMS

The choice of the nine languages and systems that we briefly introduce in this section is motivated either by their historical relevance (PRS, dMARS, AgentSpeak(L)) or by their current significance and with adoption (the remaining six ones). Clearly, there are many more BDI-based languages besides these ones, most of which are cited in Section III.

¹Despite to the precious advices given by these researchers, the paper might contain inaccuracies, whose responsibility is only ours!

Although we could not treat all the existing languages and systems in depth, our choice does not mean, in any way, that the languages surveyed in this section are “better” (according to whatever criterion) than those cited in Section III.

A. PRS

The SRI’s procedural reasoning system, PRS [8], [9], was developed for representing and using an expert’s procedural knowledge for accomplishing goals and tasks, based on the research of procedural reasoning carried out at the Artificial Intelligence Center, SRI International. It can be considered the ancestor of all the languages and architectures for practical reasoning discussed in this paper. Procedural knowledge amounts to descriptions of collections of structured actions for use in specific situations. PRS supports the definition of real-time, continuously-active, intelligent systems that make use of procedural knowledge, such as diagnostic programs and system controllers.

Main components of the language: PRS architecture consists of (1) a *database* containing current facts and beliefs, (2) a set of *goals* to be achieved, (3) a set of plans, called *Acts*, describing how sequences of conditional tests and actions may be performed to achieve certain goals or to react to certain situations, and (4) an *interpreter* that manipulates these components to select and execute appropriate plans for achieving the system’s goals.

Agent operation: The PRS interpreter runs the entire system. At any particular time, certain goals are established and certain events occur that alter the beliefs held in the system database. These changes in the system’s goals and beliefs trigger (invoke) various Acts. One or more of these applicable Acts will then be chosen and placed on the intention graph. Finally, PRS selects a task (intention) from the root of the intention graph and executes one step of that task. This will result either in the performance of a primitive action in the world, the establishment of a new subgoal or the conclusion of some new belief, or a modification to the intention graph itself.

Semantics: We were not able to find documents describing the formal semantics of the original PRS system; our understanding is that the work on giving a formal semantics to PRS started only with the research on dMARS (see Section II-B).

Implementation: A list of implemented PRS systems can be retrieved from M. Wooldridge’s page on BDI software, <http://www.csc.liv.ac.uk/~mjw/pubs/rara/resources.html>. The list includes for example PRS-CL (<http://www.ai.sri.com/~prs/>) and UMPRS (<http://ai.eecs.umich.edu/people/durfee/UMPRS.html>).

Industrial-strength applications: The PRS has been evaluated in a simulation of maintenance procedures for the space shuttle, as well as other domains [10].

B. dMARS

dMARS was implemented at the Australian AI Institute, under the direction of M. Georgeff. It was a kind of “second

generation PRS”, implemented in C++ and used for commercial agent development projects.

Main components of the language: An agent in dMars is characterised by a plan library, three main selection functions which select the intention to execute, the plan to adopt, and the event to manage respectively, and two auxiliary selection functions that are used during the agent’s functioning.

A plan is in turn constituted by an invocation condition, an optional context and a mandatory maintainance condition, a body – that is a tree representing the possible flows of actions; arcs are labelled with either an internal or an external action, or a subgoal, while nodes are labelled with states – two sequences of internal actions (updates to the belief base), to be executed if the plan succeeds or fails.

The state of an agent includes the current belief base, the set of current intentions (namely, plan instances which contain information about the current state of execution of the plans they originate from), and the event queue.

Agent operation: The dMARS operation cycle respects the basic cycle depicted in the introduction:

- If the event queue is not empty, an event is selected from it and relevant plans and, in turn, applicable plans are determined. An applicable plan is selected and used to generate a plan instance. With an external event, a new intention containing just the plan instance as a singleton sequence is created. With an internal event, the plan instance is pushed onto the intention stack which generated that (subgoal) event.
- If the event queue is empty, an intention is chosen and the action labelling the current branch in the body of its topmost plan is executed.

A plan fails if all its branches have been attempted, and all of them failed. In this case, the failure actions must be executed. Otherwise, a plan succeeds and the success actions must be executed.

Implementation: The first implementation of dMARS has been developed by the Australian Artificial Intelligence Institute (AII) - Melbourne, Australia - in 1995. AII implemented the dMARS platform for distributed reasoning agents consisting of graphical editors, a compiler and an interpreter for a logic goal-oriented programming language, a number of run-time libraries (including an in-memory knowledge database, a multi-threading package and a communication subsystem). dMARS was developed in C++ and ran on a variety of Unix platforms. At the end of 1997, dMARS was being ported to Windows/NT.

Semantics: In [11], an operational semantics of dMARS described using Z [12].

Industrial-strength applications: The dMARS system has been used for both research and production in factory automation, simulation, business and air traffic control systems. Among the customers of dMARS, there are NASA (space shuttle malfunction handling), AirServices, Thomson Airsys (air traffic control), Daimler Chrysler (supply chain management, resource & logistics management), Hazelwood Power (process control). A survey of the applications developed with dMARS can be found in [13].

C. JACK

JACK Intelligent Agents [14], [15] incorporates the BDI model and allows developers to create new reasoning models to suit their customers particular requirements. It is implemented in Java, and the JACK Agent Language extends Java with constructs for agent characteristics such as plans and events. JACK has been built by a team of experts who have worked on PRS and dMARS, and is a commercial product.

Main components of the language: The JACK Agent Language extends the regular Java syntax. It allows the programmers to develop the components that are necessary to define BDI agents and their behaviour, namely:

- Agents - which have methods and data members just like objects, but also contain capabilities that an agent has, namely belief bases (*beliefsets*), descriptions of events that they can handle, and plans that they can use to handle them.
- Capabilities - which serve to encapsulate and aggregate functional components of the JACK Agent Language for use by agents.
- Beliefsets - which are used to store beliefs and data that the agent has acquired.
- Views - which provide a way of modelling any data in a way easily manipulated by JACK.
- Events - which identify the circumstances and messages that it can respond to.
- Plans - which are executed in response to these events.

Agent operation: When an agent is instantiated in a system, it will wait until it is given a goal or it experiences an event to which it must respond. When it receives an event (or goal), the agent initiates activity to handle the event. If it does not believe that the goal or event has already been handled, it will look for the appropriate plan(s) to handle it. The agent then executes the plan or plans depending on the event type. The handling of the event may be synchronous or asynchronous relative to the posting. The plan execution may involve interaction with an agent's beliefset relations or other Java data structures. The plan being executed can in turn initiate other subtasks, which may in turn initiate further subtasks (and so on). Plans can succeed or fail. Under certain circumstances, if the plan fails, the agent may try another plan.

Implementation: JACK is implemented entirely in Java and should run on any Java-based platform. JACK stores all program files and data as normal text, allowing standard configuration management and versioning tools to be used.

Semantics: We did not find any formal semantics of JACK.

Industrial-strength applications: One of the most significant applications of JACK was an unmanned aerial vehicle (UAV) that was guided by an on-board JACK intelligent software agent that directed the aircraft's autopilot during the course of the mission.

D. JAM

JAM is an intelligent agent architecture that grew out of academic research and extended during the last five years of

use, development, and application. JAM combines ideas drawn from the BDI theories, the PRS system and its UMPRS and PRS-CL implementations, the SRI International's ACT plan interlingua [16], and the Structured Circuit Semantics (SCS) representation [17]. It also addresses mobility aspects from Agent Tcl [18], Agents for Remote Action (ARA) [19], Aglets [20] and others.

Main components of the language: Each JAM agent is composed of five primary components: a world model, a plan library, an interpreter, an intention structure, and an observer. The world model is a database that represents the beliefs of the agent. The plan library, interpreter, and intention structure has the same purpose of the corresponding components in dMARS. The observer is a user-specified lightweight declarative procedure that the agent interleaves between plan steps (in addition to the reasoning performed by the JAM interpreter) in order to perform functionality outside of the scope of JAM's normal goal/plan-based reasoning (e.g., to buffer incoming messages).

Note that, like in Coo-BDI (discussed in Section II-I), the set of plans available to the agent in the plan library can be augmented during execution through communication with other agents, generated from internal reasoning or by many other means.

Agent operation: The JAM interpreter is responsible for selecting and executing plans based upon the intentions, plans, goals, and beliefs about the current situation. The agent checks all the plans that can be applied to a goal to make sure they are relevant to the current situation. Those plans that are applicable are collected into what is called the Applicable Plan List (or APL). An utility value is determined for each instantiated plan in the APL and, if no meta-level plans are available to select between the APL elements, the JAM interpreter selects the highest utility instantiated plan (called an intention) and intends it to the goal. Note that neither the original PRS specification nor prior PRS-based implementations (such as PRC-CL) support utility-based reasoning.

Implementation: JAM is distributed freely for non-commercial use and can be downloaded from http://www.marcush.net/IRS/download_jam.html

Semantics: We are not aware of any formal semantics of JAM.

Industrial-strength applications: We could not find any information on industrial-strength applications developed using JAM.

E. Jadex

The Jadex research project is conducted by the Distributed Systems and Information Systems Group at the University of Hamburg. The developed software framework is currently in a beta-stage. A basic set of features already supports the development of rational agents on top of the the FIPA-compliant JADE platform [21]. The main purposes of Jadex are both to bring together BDI-style reasoning and FIPA-compliant communication [22], and to extend the traditional BDI-model (e.g. with explicit goals).

Main components of the language: Jadex agents have beliefs, which can be any kind of Java object and are stored in a belief base, goals, that are implicit or explicit descriptions of states to be achieved, and plans, that are procedural recipes coded in Java.

Agent operation: After initialisation, the Jadex runtime engine executes the agent by keeping track of its goals while continuously selecting and executing plan steps, based on internal events and messages from other agents. Jadex is supplied with some predefined functionalities and can integrate third party tools like the “beangenerator” plug-in for the ontology design tool Protégé [23].

Implementation: Jadex is implemented on top of JADE. To easily integrate the Jadex engine (implemented in Java) into JADE agents, a wrapper agent class is provided, which creates and initialises an instance of the Jadex engine with the beliefs, goals and plans from an agent definition file.

Semantics: For the basic operation of the Jadex interpreter, as well as for some specific aspects such as goal deliberation, an operational semantics has been sketched in [24].

Industrial-strength applications: From the Jadex home page (<http://vsis-www.informatik.uni-hamburg.de/projects/jadex/>), the pointers to three applications developed using Jadex, namely MedPAGE, Dynatech, and Blackjack, can be found.

F. AgentSpeak(L)

AgentSpeak(L) [25] takes as its starting point PRS and dMARS and formalizes its operational semantics. It can be viewed as a simplified, textual language of PRS or dMARS.

Main components of the language: AgentSpeak(L) is based on a restricted first-order language with events and actions. The beliefs, desires and intentions of the agent are not represented as modal formulas, but they are ascribed to agents, in an implicit way, at design time. The current state of the agent can be viewed as its current belief base; states that the agent wants to bring about can be viewed as desires; and the adoption of programs to satisfy such stimuli can be viewed as intentions.

Agent operation: Like in PRS and dMARS, at every interpretation cycle of an agent program, AgentSpeak(L) updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plan). In [26], R. Machado and R. H. Bordini have introduced a Belief Revision Function (BRF) in the architecture which is implicit in Rao’s interpreter, and in [27] R. H. Bordini, et al., enhance the interpreter with an efficient intention selection in BDI agents via decision-theoretic task scheduling.

Implementation: There are many implementations of the AgentSpeak(L) language, among which:

- SIM_Speak [26] (the first working AgentSpeak(L) interpreter), which runs on Sloman’s SIM_AGENT toolkit, a testbed for cognitively rich agent architectures [28], and

- Jason [29], which provides an interpreter for a version of AgentSpeak(L) extended with speech-acts [30]; Jason supports the distribution of the agents by means of SACI [31].

- AgentTalk [32], an interpreter for a simplified version of AgentSpeak(L) implemented by M. Winikoff.

Semantics: The agent operation described above is formalised in [25], [33], and [34].

Industrial-strength applications: AgentSpeak(L) has been used to program animated embodied agents in virtual environments.

G. 3APL

3APL [35] supports the design and construction of intelligent agents for the development of complex systems through a set of intuitive concepts like beliefs, goals and plans.

Although the 3APL architecture has many similarities with other cognitive architectures such as PRS, it departs from them in many ways. For example, the PRS architecture is designed to plan agents’ goals (desires) while the 3APL architecture is designed to control and revise agents’ to-do-goals. Moreover, there is no ingredient in the PRS architecture that corresponds to the practical reasoning rules, which are a powerful mechanism to revise mental attitudes. Finally, the deliberation cycle in 3APL is supposed to be a programmable component while the deliberation cycle in PRS is integrated.

Main components of the language: An agent in 3APL is characterised by two sets: the expertise of the agent, which is a set of actions, and the agent’s rule base, which is a set of rules.

A rule is formed by:

- an optional head and an optional body (both of which are goals, namely either basic actions, or queries, or achievement, or sequences of goals, or nondeterministic choices of goals, or goal variables);
- a guard, that is a belief;
- a type, that may be either *reactive*, or *failure*, or *plan*, or *optimisation*.

In 3APL goals represent both the target of the agent and the way to achieve this target, thus 3APL goals are similar both to dMARS achieve goals and to dMARS plans.

The state of a 3APL agent is constituted by its belief base and its goal base.

Agent operation: The architecture for 3APL is based on the think-act cycle, which is divided into two parts. The first part corresponds to a phase of practical reasoning by using practical reasoning rules, and the second corresponds to an execution phase in which the agent performs some action.

Think stage. The application of a rule to a goal results in the replacement of a subgoal which matches with the head of the rule by the body of the rule in case the head of the rule is non-empty. If the body of the rule is empty, the subgoal is simply dropped. In case the head of a rule is empty only the guard of the rule needs to be derivable from the beliefs of the agent, and a new goal (the body of the rule) is added to the goal base of the agent.

Act stage. The execution of a goal is specified through the computation steps an agent can perform on a goal. A computation step corresponds to a simple action of the agent, which is either a basic action or else a query on the beliefs of the agent.

Implementation: Both a Java version and an Haskell version of 3APL can be downloaded from <http://www.cs.uu.nl/3apl/download.html>.

Semantics: Originally, the operational semantics of 3APL was specified by means of Plotkin-style transition semantics [36], while in [37] 3APL has been re-specified in Z. In [38], the specification of a programming language for implementing the deliberation cycle of cognitive agents is shown, and 3APL has been used as the object language.

Industrial-strength applications: We are not aware of real applications developed using 3APL.

H. Dribble

Dribble [39] is a propositional language that constitutes a synthesis between the declarative features of the language GOAL [40], and the procedural features of 3APL.

Some attention should be devoted to the terminology used. In the original paper on 3APL [35], 3APL is defined to have beliefs and goals (no plans). These goals are however procedural (basically sequences of actions) and are actually the same as the plans of Dribble (modulo some details). The important feature of Dribble compared with the original version of 3APL, is the addition of declarative goals (based on GOAL). In the Dribble paper, the term “goal” has been used for declarative goals (in the sense of propositional formulas describing a situation that is to be achieved), and “plans” for the procedural part of the agent (which was termed “goals” in [35]). Further, the ideas of Dribble have been incorporated in the latest version of 3APL, as discussed in [41]. That paper presents a first order version of Dribble, with some minor extensions. It uses the Dribble terminology of “goals” for declarative goals and “plans” for the procedural part.

Main components of the language: The language Dribble incorporates beliefs, declarative goals, and plans (i.e., procedural goals, following 3APL terminology).

Agent operation: Dribble basically adopts a Think-Act cycle like 3APL.

Implementation: We were not able to find documents describing a working implementation of the Dribble language.

Semantics: In [39], an operational semantics of the possible mental state changes is defined using transition systems. A dynamic logic is also sketched in which one can reason about actions defined in that logic. These actions transform the mental state of the agent. In [42], a demonstration that mental state transitions defined by actions in the logic, correspond to the mental state transitions defined by the transition system is provided.

Industrial-strength applications: We are not aware of applications developed using Dribble.

I. Coo-BDI

Coo-BDI (Cooperative BDI) [43] is based on the dMARS specification and extends it by introducing *cooperations* among agents to retrieve external plans for achieving desires.

Main components of the language: The cooperation strategy of an agent *A* includes the set of agents with which is expected to cooperate (a set of agent names), the plan retrieval policy (*always*, *noLocal*) and the plan acquisition policy (*discard*, *add*, *replace*).

Coo-BDI plans are classified in *specific* and *default* ones; besides the standard components, they also have an *access specifier* which determines the set of agents the plan can be shared with (*private*, *public* and *only(TrustedAgents)*).

Coo-BDI intentions are characterized by “standard” components plus components introduced to manage the external plan retrieval mechanism.

Agent operation: The operation of a Coo-BDI agent is based on a three steps cycle:

- 1) process the event queue;
- 2) process suspended intentions;
- 3) process active intentions.

The mechanism for retrieving relevant plans involves cooperation with the trusted agents, in order to retrieve external plans, besides the local ones.

Implementation: An integration of the ideas underlying Coo-BDI into the Jason programming language has been designed [44], and its implementation is under way [45].

Semantics: No formal semantics of Coo-BDI has been defined. In [43] the Coo-BDI interpreter is fully described in Prolog, which gives an operational specification of its operation.

Industrial-strength applications: No applications have been developed using Coo-BDI.

III. COMPARISON AND RELATED WORK

In Tables 1 and 2, we summarise the analysis of the nine surveyed systems along seven dimensions. In particular, in Table 2 we take the ability of the agents to easily integrate ontologies (**Ont**) and to update the plan library at runtime (**Dyn**) into account. In Table 2, references are given if the analysed feature is not supported by the original system, but by some of its extensions. Instead, a “Yes” in the cell means that the original version of the system natively supports the corresponding feature.

Many resources on BDI-style languages are available to the research community, although, to the best of our knowledge, no exhaustive roadmap on this topic exists. An introduction to the BDI logics, architecture, and to some languages based on BDI concepts can be found both in [54] and in [1]. The paper [55] discusses and compares five MAS development toolkits, namely AgentBuilder [56], CaseLP [57], DESIRE [58], IMPACT [59], and ZEUS [60], that support the definition of agents in terms of mental attitudes.

Among the on-line resources, <http://www.csc.liv.ac.uk/~mjw/pubs/rara/resources.html> provides pointers to some

	Implementations	Formal semantics	Industrial-strength applic.
PRS	UMPRS [46], PRS-CL [47], others [48]	No	[10]
dMARS	In 1995, AAIL implemented a C++ platform running on Unix; in 1997 dMARS was ported to Windows/NT	Operational [11]	[13]
JACK	Java [49]	No	Unmanned vehicle
JAM	Java [50]	No	No
Jadex	Java [51]	Operational (sketched in [24])	[51]
AS(L)	SIM.Speak [26], AgentTalk [32], Jason [29]	Operational [33], [25], [34]	Virtual environments
3APL	Java and Prolog [52]	Operational [37], [36]; meta-level [38]	No
Dribble	No	Operational [39], dynamic logic-based [42]	No
Coo-BDI	Coo-AgentSpeak [44], [45]	Operational [43]	No

TABLE I
IMPLEMENTATIONS, SEMANTICS, APPLICATIONS OF THE SURVEYED SYSTEMS

	Basic components	Operation cycle	Ont	Dyn
PRS	Standard	Standard	No	No
dMARS	Standard	Standard	No	No
JACK	Standard + capabilities (that aggregate functional components) + views (to easily model data)	Standard	No	No
JAM	Standard + observer (user-specified declarative procedure that the agent interleaves between plan steps) + utility of plans	Utility-based	No	Yes
Jadex	Beliefs + goals + plans + capabilities (that aggregate functional components)	Standard	Yes	No
AS(L)	Standard	Standard; efficient [27]	Yes [53]	Yes [44]
3APL	Beliefs, plans, practical reasoning rules, basic action specifications	Think-act	No	Yes
Dribble	Beliefs, plans, declarative goals, practical reasoning rules, goal rules, basic action specifications	Think-act	No	Yes
Coo-BDI	Standard + cooperation strategy (trusted agents + plan retrieval and acquisition policies) + plans' access specifiers	Perceive-cooperate-act	No	Yes

TABLE II
OTHER FEATURES OF THE SURVEYED LANGUAGES AND SYSTEMS

implemented BDI systems, while <http://www.cs.rmit.edu.au/agents/SAC/survey.html> surveys systems based on the concepts of action, event, plan, belief, goal, decision and choice.

Besides the BDI-based languages and integrated environments that we have not discussed in Section II, we can cite: MYWORLD [61], in which agents are directly programmed in terms of beliefs and intentions; ViP [62], a visual programming language for plan execution systems with a formal semantics based upon an agent process algebra; CAN [63], a conceptual notation for agents with procedural and declarative goals; NUIN [64], a Java framework for building BDI agents, with strong emphasis on Semantic Web aspects; SPARK [65], that builds on PRS and supports the construction of large-scale, practical agent systems; and Jason [29], that supports many extensions to the AgentSpeak language and is a BDI system in the spirit of Jadex, JAM, JACK, but with much better formal basis.

When we move from BDI-style languages to the more general class of agent programming languages based on computational logics (which, however, includes the BDI-style languages), we can find two surveys that complement each other in many ways. The first one, [66], discusses different formalisms with the aim of putting in evidence the contribution of logic to knowledge representation formalisms and to basic mechanisms and languages for agents and MAS modeling.

The second one, [67], analyses a subset of logic-based executable languages whose main features are their suitability for specifying agents and MASs and their possible integration into an existing conceptual framework for agent-oriented software engineering based on computational logic.

ACKNOWLEDGEMENTS

The authors acknowledge Rafael Bordini, Mehdi Dastani, Marcus J. Huber, Alexander Pokahr, M. Birna van Riemsdijk, and Michael Winikoff for their precious advices.

This work was partially funded by the MIUR project "Sviluppo e verifica di sistemi multi-agente basati sulla logica", 2004-2005, coordinated by A. Martelli.

REFERENCES

- [1] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.
- [2] D. C. Dennett, *The Intentional Stance*. The MIT Press, 1987.
- [3] P. R. Cohen and H. J. Levesque, "Intention is choice with commitment," *Artificial Intelligence*, vol. 42, 1990.
- [4] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proc. of KR'91*, 1991, pp. 473–484.
- [5] —, "Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics," in *Proc. of IJCAI'91*, 1991, pp. 498–504.
- [6] —, "A model-theoretic approach to the verification of situated reasoning systems," in *Proc. of IJCAI'93*, 1993, pp. 318–324.
- [7] —, "BDI agents: from theory to practice," in *Proc. of ICMAS'95*, 1995, pp. 312–319.

- [8] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proc. of AAAI'87*, 1987, pp. 677–682.
- [9] K. L. Myers, "User guide for the procedural reasoning system," Artificial Intelligence Center, SRI International, Menlo Park, CA, Tech. Rep., 1997.
- [10] M. P. Georgeff and F. F. Ingrand, "Decision-making in an embedded reasoning system," in *Proc. of IJCAI'89*, 1989, pp. 972–978.
- [11] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge, "A formal specification of dMARS," in *Proc. of ATAL'97*, 1997, pp. 155–176.
- [12] M. Spivey, *The Z Notation: A Reference Manual*, 2nd edition. Prentice Hall International Series in Computer Science, 1992.
- [13] M. P. Georgeff and A. S. Rao, "A profile of the Australian AI institute," *IEEE Expert*, vol. 11, no. 6, pp. 89–92, 1996.
- [14] P. Busetta, R. Ronnquist, A. Hodgson, and A. Lucas, "JACK intelligent agents – components for intelligent agents in Java," *AgentLink News Letter*, vol. 2, 1999.
- [15] Agent Oriented Software Group, "What is JACK?" <http://www.agent-software.com/shared/products/index.html>.
- [16] K. L. Myers and D. E. Wilkins, "The Act Formalism, Version 2.2," SRI International AI Center Technical Report, SRI International, Menlo Park, CA, Tech. Rep., 1997.
- [17] J. Lee and E. H. Durfee, "Structured circuit semantics for reactive plan execution systems," in *Proc. of AAAI'94*, 1994, pp. 1232–1237.
- [18] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "Agent Tcl," in *Mobile Agents: Explanations and Examples*. Manning Publishing, 1997.
- [19] H. Peine, "ARA - Agents for Remote Action," in *Mobile Agents*. Manning Publishing, 1997.
- [20] D. Lange and O. Mitsuru, *Programming and Deploying Java Mobile Agents with Aglets*, 1998.
- [21] JADE Home Page, <http://jade.tilab.com/>.
- [22] FIPA Home Page, <http://www.fipa.org/>.
- [23] Protégé Home Page, <http://protege.stanford.edu/>.
- [24] A. Pokahr, L. Braubach, and W. Lamersdorf, "A flexible BDI architecture supporting extensibility," in *Proc. of IAT-2005*, 2005, pp. 379–385.
- [25] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Proc. of MAAMAW'96*, 1996, pp. 42–55.
- [26] R. Machado and R. H. Bordini, "Running AgentSpeak(L) agents on SIM-AGENT," in *Proc. of ATAL'01*, 2001, pp. 158–174.
- [27] R. H. Bordini, A. L. C. Bazzan, R. de O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser, "AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling," in *Proc. of AAMAS'02*, 2002, pp. 1294–1302.
- [28] A. Sloman and R. Poli, "SIM-AGENT: A toolkit for exploring agent design," in *Proc. of ATAL'95*. Springer-Verlag, 1995, pp. 392–407.
- [29] R. H. Bordini, J. F. Hübner, et al., *Jason: A Java-based AgentSpeak interpreter used with SACI for multi-agent distribution over the net*, Manual, release 0.5 ed., 2004.
- [30] A. F. Moreira, R. Vieira, and R. H. Bordini, "Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act Based Communication," in *Proc. of DALT'03*. Springer-Verlag, 2003, pp. 135–154.
- [31] J. F. Hübner and J. S. Sichman, *SACI — Simple Agent Communication Infrastructure*, 2003, sACI Home Page: <http://www.lti.pcs.usp.br/saci/>.
- [32] M. Winikoff, "The AgentTalk home page," <http://goanna.cs.rmit.edu.au/~winikoff/agenttalk/>.
- [33] M. d'Inverno and M. Luck, "Engineering AgentSpeak(L): A formal computational model," *Logic and Computation Journal*, vol. 8, no. 3, pp. 1–27, 1998.
- [34] R. H. Bordini and A. F. Moreira, "Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L)," *Annals of Math. and AI*, vol. 42, no. 1–3, pp. 197–226, 2004.
- [35] K. V. Hindriks, F. S. D. Boer, W. V. der Hoek, and J.-J. C. Meyer, "Agent programming in 3APL," *AAMAS Journal*, vol. 2, no. 4, pp. 357–401, 1999.
- [36] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer, "Formal semantics for an abstract agent programming language," in *Proc. of ATAL'97*, 1997, pp. 215–229.
- [37] M. d'Inverno, K. V. Hindriks, and M. Luck, "A formal architecture for the 3APL agent programming language," in *Proc. of ZB'00*, 2000, pp. 168–187.
- [38] M. Dastani, F. S. de Boer, F. Dignum, and J.-J. C. Meyer, "Programming agent deliberation – an approach illustrated using the 3APL language," in *Proc. of AAMAS'03*, 2003.
- [39] B. van Riemsdijk, W. van der Hoek, and J.-J. C. Meyer, "Agent programming in Dribble: from beliefs to goals using plans," in *Proc. of AAMAS'03*, 2003, pp. 393–400.
- [40] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer, "Agent programming with declarative goals," in *Proc. of ATAL'00*, 2000, pp. 228–243.
- [41] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. C. Meyer, "A programming language for cognitive agents: goal directed 3APL," in *Proc. of ProMAS'03*, ser. LNAI. Springer, 2004, vol. 3067, pp. 111–130.
- [42] M. B. van Riemsdijk, "Agent programming in Dribble: from beliefs to goals with plans," Master's thesis, Utrecht University, 2002.
- [43] D. Ancona and V. Mascardi, "Coo-BDI: Extending the BDI model with cooperativity," in *Post-proc. of DALT'03*, 2004, pp. 109–134.
- [44] D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini, "Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange," in *Proc. of AAMAS'04*, 2004, pp. 698–705.
- [45] D. Demergasso, "Coo-AgentSpeak: un linguaggio per agenti deliberativi e cooperativi," Master's thesis, DISI – Università di Genova, 2005, in Italian.
- [46] J. Lee, M. J. Huber, P. G. Kenny, and E. H. Durfee, "UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications," in *Proc. of CIRFSS'94*, 1994, pp. 842–849.
- [47] PRS-CL Home Page, <http://www.ai.sri.com/~prs/>.
- [48] M. Wooldridge's List of PRS Implementations, <http://www.csc.liv.ac.uk/~mjw/pubs/rara/resources.html>.
- [49] JACK Home Page, <http://www.agent-software.com/shared/products/index.html>.
- [50] JAM Home Page, http://www.marcush.net/IRS/irs_downloads.html.
- [51] Jadex Home Page, <http://vsiis-www.informatik.uni-hamburg.de/projects/jadex/>.
- [52] 3APL Home Page, <http://www.cs.uu.nl/3apl/>.
- [53] A. F. Moreira, R. Vieira, R. H. Bordini, and J. Hübner, "Agent-oriented programming with underlying ontological reasoning," in *Proc. of DALT'05*, 2005.
- [54] M. Wooldridge and N. R. Jennings, "Agent theories, architectures, and languages: A survey," in *Proc. of ECAI ATAL Workshop*, 1994, pp. 1–39.
- [55] T. Eiter and V. Mascardi, "Comparing Environments for Developing Software Agents," *AI Communications*, vol. 15, no. 4, pp. 169–197, 2002.
- [56] AgentBuilder Home Page, <http://www.agentbuilder.com/>.
- [57] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini, "Multi-agent systems development as a software engineering enterprise," in *Proc. of PADL'99*, 1999, pp. 46–60.
- [58] DESIRE Home Page, <http://www.cs.vu.nl/vakgroepen/ai/projects/desire/desire.html>.
- [59] T. Eiter, V. S. Subrahmanian, and G. Pick, "Heterogeneous active agents, I: Semantics," *Artificial Intelligence*, vol. 108, no. 1–2, pp. 179–255, 1999.
- [60] H. S. Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis, "ZEUS: A tool-kit for building distributed multi-agent systems," *Applied Artificial Intelligence Journal*, vol. 13, no. 1, pp. 129–185, 1999.
- [61] M. Wooldridge, "This is MYWORLD: The logic of an agent-oriented testbed for DAI," in *Proc. of ECAI ATAL Workshop*, 1994, pp. 160–178.
- [62] D. Kinny, "ViP: a visual programming language for plan execution systems," in *Proc. of AAMAS'02*, 2002, pp. 721–728.
- [63] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, "Declarative & procedural goals in intelligent agent systems," in *Proc. of KR'02*, 2002, pp. 470–481.
- [64] I. Dickinson and M. Wooldridge, "Towards practical reasoning agents for the semantic web," in *Proc. of AAMAS'03*, 2003, pp. 827–834.
- [65] D. Morley and K. Myers, "The SPARK agent framework," in *Proc. of AAMAS'04*, 2004, pp. 714–721.
- [66] F. Sadri and F. Toni, "Computational Logic and Multi-Agent Systems: a Roadmap," Department of Computing, Imperial College, London, Tech. Rep., 1999.
- [67] V. Mascardi, M. Martelli, and L. Sterling, "Logic-based specification languages for intelligent software agents," *TPLP Journal*, vol. 4, no. 4, pp. 429–494, 2004.

An Ontology-Based Similarity between Sets of Concepts

Valentina Cordì, Paolo Lombardi, Maurizio Martelli and Viviana Mascardi

Dipartimento di Informatica e Scienze dell'Informazione – DISI,

Università di Genova, Via Dodecaneso 35, 16146, Genova, Italy.

Email: cordi@disi.unige.it, 2001s003@educ.disi.unige.it, martelli@disi.unige.it, mascardi@disi.unige.it

Abstract—To help sharing knowledge in those contexts where documents and services are annotated with semantic information, such as the Semantic Web, defining and implementing the similarity between sets of concepts belonging to a common ontology may prove very useful. In fact, if both the required and the provided pieces of information (be they textual documents, services, images, or whatever) are annotated with sets of concepts taken from a reference ontology O , the evaluation of how good a piece of information P is, w.r.t. the required one R , may be based on the similarity between the two sets of concepts that describe P and R .

One of the first applications of the agent technology, aimed at “reducing work and information overload”, was that of retrieving and filtering information in an automatic way. Thus, the possibility to calculate the semantic distance between two sets of concepts finds a natural application in the agent field, in particular for improving those agents that act as “digital butlers” for their human owners, by exploring the Semantic Web and looking for useful documents and/or services.

Unfortunately, the metrics for calculating the semantic distance between two sets of concepts that can be found in the literature, are often very simple and do not meet some requirements that, up to us, make the metric closer to the common sense reasoning. For this reason, we have designed and implemented two new algorithms for computing the similarity between sets of concepts belonging to the same ontology.

I. INTRODUCTION

According to the Wikipedia encyclopedia (<http://en.wikipedia.org/wiki/>),

The Semantic Web is a project that intends to create a universal medium for information exchange by giving meaning (semantics), in a manner understandable by machines, to the content of documents on the Web.

The intent of the Semantic Web is thus to enhance the usability and usefulness of the Web by means of common metadata vocabularies (ontologies [7]) and standard languages suitable for defining them, such as XML (<http://www.w3.org/TR/2004/REC-xml-20040204/>), XML Schema (<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>), RDF [6], RDF Schema [2], and OWL [17].

In order to share knowledge within the Semantic Web, two problems must be addressed:

- 1) The similarity between sets of concepts belonging to the same ontology O must be defined, in order to allow a user (be it a human or a software agent) interested in

documents dealing with topics in the set S_1 , to retrieve also documents dealing with topics in the set S_2 , if S_1 and S_2 are “close enough” with respect to O .

- 2) Since it is not always possible to have a unique ontology O to use as a reference for comparing sets of concepts, documents are often tagged not only with a set of concepts, but also with the ontology from which the tagging concepts come from. In this case, maps between different ontologies must also be provided.

These two problems, although typical of the Semantic Web context, can be found in many other application scenarios, like Multiagent, Peer-to-Peer and Grid systems, where the comparison of set of concepts is required to provide more precise answers to the user’s requests. For example, [8] describes a multiagent system for semantic-driven information retrieval, based on the peer-to-peer model, where routing of requests is performed by computing the similarity between the set of concepts advertised by each agent (the concepts that are dealt with by the agent’s documents) and the set of concepts that characterise the documents looked for by the requesting peer. The similarity is evaluated by referring to a unique ontology that describes the system’s domain, and from which concepts appearing both inside advertisements and requests are taken.

More in general, all the application scenarios where an “intelligent” information retrieval and filtering is required, may take advantage of programs and instruments that allow to compare two sets of concepts w.r.t. an ontology. In fact, all the “agents that reduce work and information overload” [15] usually need to compare the meta-information extracted from the retrieved document, with the meta-information that describes the user’s interests. Very often, this meta-information consists either of a set of keywords whose similarity can be computed by using lexical ontologies such as WordNet, or of a set of concepts taken from a reference ontology.

In this paper, we address the problem of defining a similarity metric between sets of concepts belonging to the same ontology. In Section II we review some existing metrics defined in the literature to measure the similarity between two concepts (and, very rarely, between two sets of concepts) in either a taxonomy or an ontology. In Section III, we describe our algorithms and provide motivations for the choice we made in their design, and in Section IV we discuss our algorithms and we conclude the paper with the future directions of our work.

II. AN OVERVIEW OF EXISTING METRICS

There are several techniques used to measure the similarity of two concepts belonging to the same taxonomy, and these techniques can be also applied to ontologies.

In particular the three main techniques are either 1) based on the distance between concepts, or 2) based on information content, or 3) based on a glossary. In this paper we take into account only the first one.

A. Bouquet, Kuper, Scoz and Zanobini's metric

In [1], Bouquet, Kuper, Scoz and Zanobini introduce two kinds of distances, one between simple concepts, and one between sets of simple concepts.

– *Ontological Distance between simple concepts.* The Ontological Distance between c and c' , written $D_s(c, c')$, is the length of the minimal path between the nodes corresponding to c and c' in the ontology O , if such a path exists, and is 0 otherwise. The ontology is defined in the usual way, as a graph where nodes are labelled with concepts and arcs are labelled with relations between couples of concepts.

– *Ontological Distance between sets of simple concepts.* Let A and B be two sets of simple concepts. The ontological distance between the sets A and B , $D_c(A, B)$, is the sum of $D(c, c')$ for each c in A and c' in B .

Since this definition involves some redundancy, the notion of normalized set of simple concepts is introduced:

– *Normalized set of simple concepts.* Let K be the set of simple concepts occurring in a complex concept, namely a concept that is built from simple concepts defined in some ontology O and organized in a classification structure. A normalized set of simple concepts K' contained in K is defined as the set of all c belonging to K such that there is no path from c' to c in O for some c' belonging to K .

The ontological distance between complex concepts CA and CB is then defined as $D_c(A', B')$, where A' and B' are the normalized sets of simple concepts for A and B respectively, and A and B are the sets of simple concepts occurring in CA and CB , respectively.

B. Haase, Siebes, and van Harmelen's metric

In [9], the similarity between two concepts belonging to the same ontology, where a *SubTopic* relation is defined, is evaluated as

$$S(t_1, t_2) = \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } t_1 \neq t_2 \\ 1 & \text{otherwise} \end{cases}$$

where l is the length of the shortest path between topic t_1 and t_2 according to the *SubTopic* relation, h is the level in the tree of the direct common subsumer from t_1 and t_2 , and $\alpha \geq 0$ and $\beta \geq 0$ are parameters scaling the contribution of shortest path length l and depth h , respectively. The intuition behind using the depth of the direct common subsumer in the definition of the similarity is that topics at upper layers of hierarchical semantic nets are more general and are semantically less similar than topics at lower levels.

Given the function for calculating the similarity between two individual topics, it is possible to define the distance between two sets of concepts as

$$SF(s, e) = \frac{1}{|s|} \cdot \sum_{t_i \in s} \max_{t_j \in e} S(t_i, t_j)$$

C. Castano, Ferrara, Montanelli, and Racca's metric

In [5], a term affinity function $A(t, t')$ is defined to evaluate the affinity between two terms t and t' with respect to a thesaurus Th of terms and terminological relationships among them.

$A(t, t')$ is equal to the value of the highest-strength path of terminological relationships between t and t' in Th if at least one path exists, and is 0 otherwise. A path strength is computed by multiplying the weights associated with each terminological relationship involved in the path, that is:

$$A(t, t') = \begin{cases} \max_{i=1..k} \{W_{t \rightarrow_i t'}\} & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases}$$

where: k is the number of paths between t and t' in Th ; $t \rightarrow_i^n t'$ denotes the i th path of length $n \geq 1$; $W_{t \rightarrow_i^n t'} = W_{1tr} \cdot W_{2tr} \cdot \dots \cdot W_{ntr}$ is the weight associated with the i th path, where W_{jtr} such that $j = 1, 2, \dots, n$ denotes the weight associated with the j th terminological relationship in the path.

In [3], Bulskov, Knappe, and Andreasen use basically the same measure, namely the maximal multiplicative weighted path length, on ontologies expressed in ONTOLOG [16].

D. Rada, Mili, Bicknell, and Blettner's metric

In [18] the conceptual distance between any two concepts is defined as the shortest path through a semantic network. The semantic network taken into account is MeSH, a hierarchical network of biomedical concepts that (at the time the paper was published, namely in 1989) consisted of about 15,000 terms organized into a nine-level hierarchy, with concepts related by a *broader-than* relationships, which includes both *is-a* and *part-of* relationships.

E. Leacock and Chodorow's metric

The measure presented in [13] is similar to that defined by Rada, Mili, Bicknell, and Blettner, since it is based on the length of the shortest paths between noun concepts in a *is-a* hierarchy. Leacock and Chodorow's measure of similarity is thus defined as follows:

$$sim_{LeaCho}(c1, c2) = \max \left[-\log \left(\frac{length(c1, c2)}{(2D)} \right) \right]$$

where $length(c1, c2)$ is the shortest path length between the two concepts and D is the maximum depth of the taxonomy.

As we can see, the value of the shortest path length is scaled by the depth D of the hierarchy, where depth is defined as the length of the longest path from a leaf node to the root node of the hierarchy.

F. Wu and Palmer's metric

Wu and Palmer [20] define a measure of similarity that is also based on path lengths, however, they focus on the distance between a concept to the root node.

Resnik [19] reformulates their measure slightly. This measure finds the distance to the root of the most specific node that intersects the path of the two concepts in the *is-a* hierarchy. This intersecting concept is the most specific concept that the two concepts have in common, and is known as the “lowest common subsumer” (*lcs*). The distance of the *lcs* is then scaled by the sum of the distances of the individual concepts to the node.

The measure is formulated as follows:

$$sim_{WuPal}(c1, c2) = \frac{2 \cdot depth(lcs(c1, c2))}{depth(c1) + depth(c2)}$$

where *depth* is the distance from the concept node to the root of the hierarchy.

G. Hirst and St.Onge's metric

Hirst and St.Onge [10] introduce a measure of relatedness that considers many other relations beyond the *is-a* one, and that is used for lexical ontologies. This measure classifies relations as: horizontal, upward, or downward.

- *Upward relations* connect more specific concepts to more general ones (i.e., *is-a*)
- *Downward relations* connect more general concepts to more specific ones (i.e., *is-a-kind-of*)
- *Horizontal relations* maintain the same level of specificity.

The measure defined by Hirst and St.Onge has three levels of relatedness: extra strong, strong and medium strong. An extra strong relation is based on the syntactic form of the words, while two words representing the same concept (i.e., synonyms) have a strong relation between them. The medium-strong relation is determined by a set of allowable paths between concepts. If a path that is neither too long nor too winding exists, then there is a medium-strong relation between the concepts. The score given to a medium-strong relation considers the path length between the concepts and the number of changes in direction of the path:

path weight =

$$C - path\ length - (k \times \#changes\ in\ direction)$$

III. COMPUTING THE SIMILARITY OF TWO SETS OF CONCEPTS: A NEW PROPOSAL

The two algorithms we have designed and implemented to compute the similarity of two sets of concepts, work on ontologies represented in OWL, RDF and DAML+OIL [11]. Both algorithms are based on the definition of the “sim.c” function for calculating the similarity of a concept w.r.t. a set of concepts. Thus, we first introduce “sim.c” in Section III-A, and then we introduce the two algorithms in Section III-B.

A. Computing the similarity between a concept and a set of concepts

Our algorithm for computing the similarity between a concept and a set of concepts is an extension of Dijkstra's algorithm (shown in Algorithm 1), where there may be more than one destination node.

Algorithm 1: *Dijkstra*(G, w, s)

```

foreach vertex  $v \in V[G]$  do
   $d[v] := \infty$ ;
   $previous[v] := nil$ ;
end
 $d[s] := 0$ ;
 $S := \emptyset$ ;
 $Q := V$ ;
while  $Q \neq \emptyset$  do
   $u := extract\_min(Q)$ ;
   $S := S \cup \{u\}$ ;
  foreach edge  $(u, v)$  outgoing from  $u$  do
    if  $d[v] > d[u] + w(u, v)$  /* Relax( $u, v$ ) */ then
       $d[v] := d[u] + w(u, v)$ ;
       $previous[v] := u$ ;
       $Q := update(Q)$ ;
    end
  end
end

```

Initialisation: The difference in the initialisation phase w.r.t. Dijkstra's algorithm is that in ours, the value of a path is evaluated as the product of the weights of the path's edges (kept in a $w[.,.]$ matrix), and we prefer paths with a higher value¹. Thus, we must initialise the similarity (kept in a $d[.]$ array) of the source concept s from itself as if it were the best possible similarity (1), and the other similarities as if they were the worst possible ones (0).

In our algorithm for calculating the “sim.c” function we use the $pi[.]$ array that, for each concept in the ontology, keeps track of its predecessor (if any) in the current best estimated path towards the destination node(s). The pi data structure allows us to store a spanning tree of the ontology, characterised by the nodes $pi[j]$ for $1 \leq j \leq |O|$ (where $|O|$ is the number of concepts belonging to the ontology) and by the edges $(pi[j], j)$, for $1 \leq j \leq |O|$.

¹The weights between couples of concepts kept in the $w[.,.]$ matrix, representing the similarity of pairs of adjacent concepts, are decided at design time by the ontology developer who is supposed to be an expert of the ontology domain.

Algorithm 2: *initialise_single_source*(*Ontology* *o*, *Concept* *s*)

```

foreach concept c in Concepts(o) do
  | d[c] := 0 ;
  | pi[c] := nil ;
end
d[s] := 1;

```

Relaxation: The relaxation procedure checks whether the current best estimate of the similarity between *s* and *v* (*d*[*v*]) can be improved by going through *u* (i.e. by making *u* the predecessor of *v*). With respect to Dijkstra's algorithm, here we changed a > with a < in the condition of the **if** statement, and a + with a * in the evaluation of the path's total weight.

Algorithm 3: *relax*(*Concept* *u*, *Concept* *v*, *double* *w*[][])

```

if d[v] < d[u] * w[u, v] then
  | d[v] := d[u] * w[u, v] ;
  | pi[v] := u ;
end

```

Implementation of the sim_c function: The *sim_c* algorithm, whose pseudo-code is shown in Algorithm 4, evaluates the similarity between a concept *s* and a set of concepts *target*.

The algorithm returns a value in [0,1]. If *s* belongs to *target*, 1 is returned and the algorithm stops, otherwise the algorithm starts by exploring the paths from *s* to the nodes in *target*. The *lower_bound* parameter is the value under which results are considered no longer relevant, and may range in [0, 1].

Post processing phase: All the paths with a better value than *lower_bound* have been explored; the *current_highest_similarity* array contains 0 for those nodes in the local ontology whose similarity with *s* is not relevant; they contain a value different from 0 (and surely > *lower_bound*) for the other ones. Now, we only need to combine these values in order to obtain a final value in [0, 1]. The mathematical function used to combine these values is the following:

$$\text{combine_final_value}(x_1 \dots x_n) = f(x_1)$$

where the sequence $(x_1 \dots x_n)$ is ordered and

$$f(x_i) = \begin{cases} x_i + (1 - x_i) * f(x_1) & \text{if } i < n \\ x_i & \text{if } i = n \end{cases}$$

This function, when applied to the ordered sequence of values is in the [0, 1] range representing the similarity of the source concept with all the other concepts of the matrix (when this similarity is different from 0), allows us to obtain a value which is in the [0, 1] range, and that gives more weight to the higher values, but still allowing the lower values to contribute to the final outcome.

Algorithm 4: *double sim_c*(*Ontology* *o*, *Concept* *s*, *set*(*Concept*) *target*, [0..1] *lower_bound*)

```

if s ∈ target then
  | return 1
else
  initialise_single_source(o, s);
  foreach concept c ∈ target do
    current_highest_similarity[c] := 0;
    go_on := true;
    /* Make S equal to the source concept */
    S := s;
    /* the current concept is the source concept */
    u := s;
    C := Concepts(o);
    /* while there are still concepts to explore, and the paths through these concepts may prove to be better than the current path to one of the concepts in target */
    while C ≠ S and go_on do
      foreach concept c ∈ Adjacent(u) do
        /* the path's weights going through "v" are evaluated and updated */
        relax(u, c, w);
      end
      /* the most promising concept to reach one of the concepts in Loc namely the one such that d[u] is higher is found */
      u := extract_most_similar(C \ S);
      if d[u] > lower_bound then
        S := S ∪ {u};
        /* if the path leads to the destination, we wonder if this path is better than the previous one found, leading to the destination (if any), and we eventually update the value of the current best path */
        if u ∈ target then
          if d[u] > current_highest_similarity[u] then
            | current_highest_similarity[u] := d[u];
          end
        end
      end
      /* if all the paths that still remain to be considered, are worst than the lower_bound, the algorithm stops */
      go_on := false;
    end
    odered_val := order(current_highest_similarity);
    return combine_final_value(odered_val);
  end

```

B. Computing the similarity between two sets of concepts

In the following we present two different algorithms for computing the similarity between two sets of concepts. The first is based on a mathematical formula while the second is based on a recursive approach. There are some conditions that, in our opinion, an algorithm for computing the similarity between two sets of concepts should meet :

- 1) It is not sufficient that a concept in the source set matches a concept in the target set to obtain 1 as result.
- 2) To obtain 1 as result, all the concepts in the source set should have a related concept in the target set.
- 3) If there are many concepts with high similarity in the two sets, we would like to “prize” them by having a similarity function that respects the inequality:

$$\frac{\text{similarity}(o, \text{target}, \text{source}, \text{lower_bound})}{\sum_{s \in \text{source}} \frac{\text{sim}_c(o, \text{target}, \text{source}, \text{lower_bound})}{|\text{source}|}} >$$

On the other hand, if there are many concepts with low similarity in the two sets of concepts, we would like to “punish” them by having a similarity function that respects the inequality:

$$\frac{\text{similarity}(o, \text{target}, \text{source}, \text{lower_bound})}{\sum_{s \in \text{source}} \frac{\text{sim}_c(o, \text{target}, \text{source}, \text{lower_bound})}{|\text{source}|}} <$$

1) *Algorithm similarity_by_m.th.root.*: The first algorithm is based on the following formula:

$$\sqrt[n]{\sum_{k=0}^n (a_k)^m} \quad \text{where } a_i \in \mathbb{Z}$$

The algorithm that uses this formula is described below in pseudo-code; it takes as input five parameters: the ontology, the target set, the source set, a coefficient representing the value of m in the previous formula and the *lower_bound* under which results are considered no longer relevant.

Algorithm 5: *double similarity_by_m.th.root(Ontology o, set(Concept) target, set(Concept) source, int m, [0..1] lower_bound)*

```

result := 0;
foreach concept c ∈ source do
    result :=
        result + sim_c(o, c, target, lower_bound)^m;
end
return  $\frac{\sqrt[n]{\text{result}}}{|\text{source}|}$ ;

```

2) *Algorithm similarity_by_recursive_eval.*: The second algorithm is also based on the *sim_c* algorithm for computing the similarity between a concept and a set of concepts illustrated previously, but the different idea is to evaluate the similarity between the target set and the elements of the source set, and to use a recursive function to calculate the similarity. The entire pseudo-code is illustrated below.

Algorithm 6: *double similarity_by_recursive_eval(Ontology o, set(Concept) target, set(Concept) source, [0..1] lower_bound)*

```

similarity := nil;
foreach concept c ∈ source do
    similarity.add(sim_c(o, c, target, lower_bound));
end
return calculate(similarity);

```

Algorithm 7: *double calculate(set(Double) similarity)*

```

if similarity.isEmpty() then
    return 0
else
    value = max(similarity);
    similarity.remove(value);
    return (value + (1 - value) * calculate(similarity));
end

```

IV. DISCUSSION AND FUTURE WORK

The *similarity_by_m.th.root* algorithm meets the requirements specified in Section III-B. It defines a means where high values have an higher impact in computing the final result than low values. The m parameter allows us to specify how much the higher values should be “prized” with respect to lower ones.

The *similarity_by_recursive_eval*, instead, does not satisfy the third requirement: in fact, if one (and only one) concept belongs both to the *source* set and to the *target* set, the result of the algorithm is 1, and thus the first condition specified in Section III-B is not met. However our experiments, discussed in [14], demonstrated that *similarity_by_recursive_eval* has a better performance than *similarity_by_m.th.root*.

Although many techniques for computing the similarity between concepts belonging to the same ontology or taxonomy exist, the use of that based on the path length is very common in the scientific community. Our definition of the similarity between concepts is also very common, and thus it is not a novelty w.r.t. the existing literature. However, we have also defined the similarity between a single concept and a set of concepts, and between two sets of concepts, establishing some criteria that our definitions should meet. Up to our knowledge, the only metrics for defining the similarity between two sets of concepts are those defined in [1] and [9], but none of them meets our requirements, that, instead, are met by our *similarity_by_m.th.root* definition.

Our algorithms have been implemented using the Jena framework (<http://jena.sourceforge.net>), and can be downloaded from <http://www.disi.unige.it/person/MascardiV/Software/SoftwarePaoloLombardi.html>. The main direction of our work consists of integrating our implemented metrics into the P2P system described in [8], that is being developed using JXTA (<http://www.jxta.org>). Since both Jena and JXTA are based on Java, the integration should be easy to implement.

Although this system was born as a pure P2P system, we can easily see it as a MAS. In fact, peers are autonomous (they actively push their expertise to the other peers in the system, as discussed below), reactive (they react to incoming requests), proactive (they have a long term goal of retrieving as many relevant documents as possible), and social (communication is asynchronous and uses a structured, XML-based communication language). Finally, peers are aware of the features, the topology and the inhabitants of the P2P system where they live, so they are in some sense situated in their software environment. Since peers respect the well-known definition of agent given in [12], in the following we will use the term agent instead of the term peer.

In the system under consideration, agents may register to one or more thematic groups. Relevant information retrieval is achieved through the use of a thematic global ontology (*TGO*) for each theme dealt with by the system; the *TGO* associates a semantics with the resources to be shared within the thematic group. All the agents that register to a thematic group share the *TGO* of the group. Each arc between two concepts belonging to the *TGO* is weighted with a value in $[0, 1]$ that represents the similarity between the two concepts. Each agent A is characterised by a set of concepts of interest CoI_A such that $CoI_A \subseteq V$, where V are the concepts in the *TGO*. Agents actively and autonomously push their expertises by sending *advertisements*, containing the concepts of the *TGO* that better describe the resources they share, so each agent A is also characterised by the sets of advertised concepts that it pushes towards the system, Adv_{A_i} , for $i \in 1, \dots, n$ such that $Adv_{A_i} \subseteq V$.

In order to allow an agent A to understand if an agent B sending an advertisement Adv_{B_j} , shares resources that may be of interest for A , the similarity between Adv_{B_j} and CoI_A w.r.t. the *TGO* should be evaluated. By integrating our *similarity_by_nth_root* algorithm in the system, we would allow agent A to evaluate *similarity_by_nth_root*(*TGO*, CoI_A , Adv_{B_j} , *lower_bound*) thus obtaining the required similarity.

REFERENCES

- [1] P. Bouquet, G. Kuper, M. Scoz and S. Zanobini. Asking and answering semantic queries. In *Proc. of Meaning Coordination and Negotiation Workshop (MCNW-04) in conjunction with International Semantic Web Conference (ISWC-04)*, 2004.
- [2] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen and I. Horrocks. Enabling knowledge representation on the Web by extending RDF schema. In *Proc. of the 10th international conference on World Wide Web*, pp. 467–478, 2001.
- [3] H. Bulskov, R. Knappe and T. Andreassen. On Measuring Similarity for Conceptual Querying. In *Proc. of the 5th International Conference on Flexible Query Answering Systems*, Springer-Verlag publisher, pp. 100–111, 2002.
- [4] S. Castano, A. Ferrara, S. Montanelli, E. Pagani, and G. P. Rossi. Ontology-addressable contents in P2P networks. In *Proc. of the 1st SemGRID Workshop*, 2003.
- [5] S. Castano, A. Ferrara, S. Montanelli, G. Racca. Semantic Information Interoperability in Open Networked Systems. In *Proc. of the Int. Conference on Semantics of a Networked World (ICSNW)*, in cooperation with ACM SIGMOD 2004, pp. 215–230, 2004.
- [6] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann and I. Horrocks. The Semantic Web: The Roles of XML and RDF, *IEEE Internet Computing*, 4(5), pp. 63–74, 2000.
- [7] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.
- [8] G. Guerrini, V. Mascardi, M. Mesiti. A Semantic Information Retrieval Advertisement and Policy Based System for a P2P Network. In *Proc. of the DBISP2P Conference*, 2005.
- [9] P. Haase, R. Siebes, F. van Harmelen. Peer Selection in Peer-to-Peer Networks with Semantic Topologies. In *Proc. of International Conference on Semantics of a Networked World: Semantics for Grid Databases*, 2004.
- [10] G. Hirst, D. St. Onge. Lexical chains as representations of context for the detection and correction of malapropisms, In Fellbaum MIT Press, pp. 305–332, 1998.
- [11] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+oil: an ontology language for the semantic web. In *Proc. of the 18th National Conference on Artificial Intelligence*, pp. 427–428, 2002.
- [12] N. R. Jennings and K. Sycara and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.
- [13] C. Leacock, M. Chodorow. Combining local context and WordNet similarity for word sense identification, In Fellbaum MIT Press, pp. 265–283, 1998.
- [14] P. Lombardi. Progettazione ed implementazione di una nuova metrica sulle ontologie. Master Thesis, DISI, Università degli Studi di Genova, 2005. In Italian. <http://www.disi.unige.it/person/MascardiV/Download/Lombardi.zip>.
- [15] P. Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7), 1994.
- [16] J. F. Nilsson. A Logico-algebraic Framework for Ontologies ONTOLOG. In *Proc. of the First International OntoQuery Workshop Ontology-based interpretation of NP s*, 2001.
- [17] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Web Ontology Language (OWL) Abstract Syntax and Semantics. Technical report, W3C.
- [18] R. Rada, H. Mili, E. Bicknell, M. Blettner. Development and application of a metric on semantic nets. *IEEE Transaction on Systems, Man and Cybernetics* 19(1), pp. 17–30, 1989.
- [19] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research* 11, pp. 95–130, 1998.
- [20] Z. Wu, M. Palmer. Verb semantics and lexical selection, In 32nd Annual Meeting of the Association for Computational Linguistics, Las Cruces, New Mexico, pp. 133–138, 1994.

Asking and answering queries semantically

P. Bouquet, G. Kuper, S. Zanobini

Department of Information and Communication Technology

University of Trento

Via Sommarive, 14

38050 Trento (Italy)

{bouquet,kuper,zanobini}@dit.unitn.it

Abstract—In this paper we propose a new method, called SEMQUERY, for querying information sources whose data are organized according to different schemata. The method is based on the idea of *semantic elicitation*, namely a process which takes in input the structural part of a query (e.g. the XPath part of an XQuery) and returns an expression in a logical language which represent the meaning of the query in a form which ideally is independent from its original syntactic formulation. Since the same process of elicitation can be performed on any path of schemata used to organize data, the decision on whether there is any logical relation between a query and a path in the schema is made via logical reasoning.

I. INTRODUCTION

The distribution of knowledge across a large number of different and autonomous providers raises the problem of retrieving information from semantically heterogeneous sources. A crucial issue is how to allow users to query heterogeneous information sources without assuming that they know their conceptual structure. The problem, of course, is not new. It has been studied for a long time in the database community, in the form of querying distributed and heterogeneous databases (e.g. [3]). However, the proposed solutions either cannot be straightforwardly extended to other domains (e.g. querying document repositories based on a classification schema, or according to a hierarchy of web directories), or are based on assumptions which limit their applicability (e.g. assuming that mappings across schemata are available from the start). In this paper, we propose a new approach, which builds on our experience in the Semantic Web, but can be generalized to any information source which is structured according to some explicit schema, such as databases, product and service catalogs, document directories (e.g. web directories in search engines like Google or Yahoo), file systems (e.g. in peer-to-peer file sharing applications). As a concrete example, and without any loss of generality, we will discuss the problem, and present our results, using XMLSchema [8], [9] as a syntax for schemata, and XQuery [10] as a query language (more precisely, the XPath [7] fragment of an XQuery).

The main contribution of the paper is a method, called SEMQUERY, which, given a query containing an XPath expression q and a collection of XMLSchema specifications Σ , computes the set of *semantically equivalent* rewritings of q with respect to each $\sigma \in \Sigma$. SEMQUERY is based on the concept of *semantic elicitation*, which is a process that takes

an XPath expression and encodes its meaning in a logical language L^1 . This encoding, which in SEMQUERY is performed fully automatically, makes explicit the meaning of an XPath expression in a form which is (relatively) independent from its original syntactic form, and ideally is logically equivalent to any other semantically equivalent XPath expression. To achieve this result, we assume that the names of elements and attributes in a schema are meaningful noun phrases of some natural language (e.g. English). As we argued in [1], this assumption is crucial, as it allows us to exploit lexical and domain knowledge to construct a deep interpretation of XPath expressions.

Semantic elicitation can be applied both to XPath expressions occurring in a query and to XPath expressions which describe a path in a XMLSchema. Therefore, the way a query q is processed against a schema is the following: the meaning of the XPath part of the query is elicited, the meaning of each path in the schema is elicited as well, and then the decision on whether there is any logical relation between the query q and a path in the schema is made via logical reasoning (in the paper, we check for concept equivalence or subsumption, but of course this is only a special case). As we shall show, SEMQUERY can be implemented quite efficiently, as the semantic elicitation of a schema's paths can be performed at design time and stored with the schema (this enriched version of a schema is what we call a *context*). At execution time, we only need to elicit the meaning of the query and match it against the concepts (already) available in a schema.

The structure of the paper is as follows: Section II defines the problem, and Section III describes our method, SEMQUERY, for solving the problem. Finally, Section IV provides a detailed description of the semantic elicitation phase.

II. THE PROBLEM

Imagine that we have two schemata σ_1 and σ_2 such as those depicted in Figure 1, and suppose they are used to structure two multimedia document repositories. Consider the paths which lead to the node LANDSCAPES in the schema on the left hand side and to the node JPEG in schema on the right hand side. Despite their syntactical difference, they seem

¹In this paper we use Description Logic, as we deal mostly with concepts and attributes. However, in previous work on semantic coordination, a much simpler encoding in propositional logic was used [1].

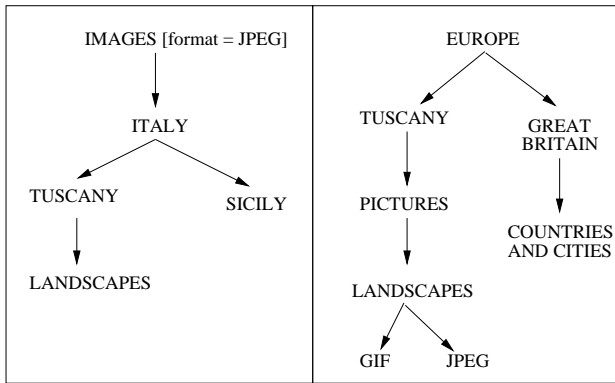


Fig. 1. Two simple schemata

to have the same meaning, something like *images of Tuscan landscapes in JPEG format*. For classification schemata, the intuition behind the notion of “having the same meaning” is that a human user would classify the same documents under the two nodes. However, XPath expressions refer directly to the syntactical features of schemata; as a result, there is no single XPath expression that can be used in a query to refer to the two semantically equivalent nodes LANDSCAPES and JPEG, and therefore to retrieve the associated documents.

We therefore need a way of recognizing that the two XPath expressions

`/IMAGES[format = 'JPEG']/ITALY/TUSCANY/LANDSCAPES` (1)

and

`//IMAGES[about = 'Tuscany']/LANDSCAPES/JPEG` (2)

are semantically equivalent, regardless of their concrete syntactic form, and therefore that an XQuery expression containing the first path should allow us to retrieve not only documents from the corresponding path in the first schema, but also document from the path in the second schema.

In addition, one might want to recognize that, for example, a query containing the XPath expression

`/IMAGES[format = 'JPEG']/ITALY/FLORENCE/LANDSCAPES`

can be also a valuable answer for a query containing the expression

`/IMAGES[format = 'JPEG']/ITALY/TUSCANY/LANDSCAPES`

even though in this case the relation would not be semantic equivalence, but rather subsumption (after all, JPEG pictures of landscapes of Florence are a special case of JPEG pictures of landscapes of Tuscany).

To sum up, the examples show that syntactically different XPath expressions may be used to refer to *semantically equivalent* concepts. The problem we address is to define an automated method for *asking and answering queries semantically*, namely to ask queries which are (relatively) independent from their syntactic form, and to answer a semantic query by looking for semantic relations between concepts.

III. SEMQUERY: A SHORT DESCRIPTION

SEMQUERY is a method for asking semantic queries based on what we call the *meaningfulness hypothesis*, namely that

the schemata which are used to organize information sources have meaningful labels². As we discussed in [1], there are two reasons for making this hypothesis in a framework in which semantic relations between schema elements are to be discovered and exploited in a principled way:

- 1) Firstly, if we didn't assume that labels were meaningful, there would be no reason to say, for example, that there is a relation between IMAGES and PICTURES. Indeed, as mere strings, there is no similarity, and synonymy is definitely a semantic relation between meaningful words. Conversely, we don't want to conclude that DIG and DOG are more similar than DIG and EXCAVATION, though the first two are syntactically much more similar than the others. We stress this issue, as many approaches to semantic interoperability, e.g. those based on graph matching, use thesauri or other type of lexical information in a way which is sound only if one makes the assumption of meaningfulness;
- 2) The second, more important observation, is that if we ignore the meaning of labels, we will not be able to discover relations between paths in schemata that depend only on the meanings of these labels. For example, consider the two pairs of isomorphic schemata in Figure 2. Intuitively, the relation between the two pairs of

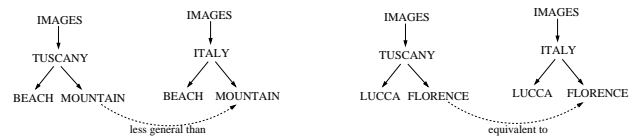


Fig. 2. Relations across schemata with meaningful labels

nodes is different, subsumption on the left hand side and equivalence on the right hand side, even though the two schemata are isomorphic. The explanation is that we use what we know about the concepts corresponding to the labels, in order to decide what a node really means.

Closely related to the meaningfulness hypothesis is the idea that SEMQUERY should use knowledge about labels to improve the quality of its results. Indeed, only domain knowledge can allow us to realize that the concept of ‘images of Florence’ is less general than the concept of ‘images of Tuscany’, no matter how the two concepts are expressed syntactically. This again is crucial to discover semantic relations across paths, which do not depend only on what is explicitly said, but also on what we know about the corresponding concepts.

We now turn to a general description of our method. Let Λ be the set of noun phrases that can be built in English from a set λ of English words, and Λ^* the set of all finite XPath expressions using only elements of Λ as tags, the child and descendant axes, and the wild card $*$. \mathcal{C} is the set of terms that

²More precisely, we assume that they would be interpreted as meaningful by humans via some simple manipulation; for example, labels like `ProgrammingLanguages`, `Programming_Languages` or even `ProgLang` would be easily recognized as meaningful – and basically equivalent – by humans on the web site of, say, a Computer Science Department. In the rest of the paper, we will pretend that labels are English noun phrases, but in many real applications it may be necessary to go through a normalization phase in which labels are transformed into correct English words and noun phrases.

can be built in a Description Logic language like \mathcal{ALC}^3 from a set T of primitive terms and a set R of primitive roles, and \mathcal{O} is a (possibly empty) set of axioms defined over \mathcal{C} . The process of semantic elicitation can be viewed as a function $\Upsilon : \Lambda^* \rightarrow \mathcal{C}$ which takes as input an element of Λ^* and returns a (complex) term in \mathcal{C} which expresses its meaning.

How to compute this function is a crucial issue of our work, and this will be discussed in Section IV. For now, suppose that Υ is defined. We can then divide the set Λ^* of all the possible XPath expressions into sets of semantically equivalent expressions, namely expressions with an equivalent meaning. Formally:

Definition 1 (Equivalence class): Let p and p' be two XPath expressions from Λ^* , Υ the semantic elicitation function, and let $L = \langle \mathcal{C}, \mathcal{O} \rangle$ be a T-Box containing terminological axioms. We say that p and p' belong to the same equivalence class $[\Lambda^*]_{(\Upsilon, L)}$ of Λ^* with respect to Υ and L iff:

$$\mathcal{O} \models (\Upsilon(p) \equiv \Upsilon(p'))$$

We write $\mathcal{J}(p)$ to denote the equivalence class containing p .

We can now define the set of *semantically equivalent rewritings* of a query over a collection of schemata. Intuitively, given an XPath expression p and a set of schemata Γ , the problem of answering queries semantically can be defined as the problem of determining the set P of XPath expressions occurring in Γ which belong to the same equivalence class of p . Formally:

Definition 2 (Semantically equivalent answer): Let p be an XPath expression occurring in a query, Γ a set of schemata, and $P \subseteq \Lambda^*$ the set of all the XPath expressions which denote a path occurring in at least one schema in Γ . Then P' is the set of *semantically equivalent answers* for p if it is the maximal subset of P such that

$$\text{for all } q \in P', \mathcal{J}(q) = \mathcal{J}(p)$$

A weaker, but still useful, notion of semantic answer can be defined as follows. Suppose that a query containing the XPath expression `/IMAGES/JPEG/ITALY/LANDSCAPES` is performed over the structure on the left in Figure 1. Intuitively, the associated concept, ‘JPEG images of Italian landscape’, is subsumed by the concept ‘JPEG images of Tuscany’s landscape’ corresponding to the path `/IMAGES[format = 'JPEG']/ITALY/TUSCANY/LANDSCAPE`. Therefore, the corresponding XPath expression is not semantically equivalent to the query, but can be considered as a *semantically less general answer*.

Formally, let \leq be a partial order over the set $[\Lambda^*]_{(\Upsilon, \mathcal{O})}$ w.r.t. Υ and \mathcal{O} , let X and Y be two equivalence classes in $[\Lambda^*]_{(\Upsilon, \mathcal{O})}$,

and let x and y be the witnesses of X and Y respectively. Then

$$x \leq y \quad \text{iff} \quad \mathcal{O} \models \Upsilon(x) \sqsubseteq \Upsilon(y)$$

We can then define the set of all the semantically related answers as follows:

Definition 3 (Semantically related answer): Let p an XPath expression occurring in a query, Γ a set of schemata, and $P \subseteq \Lambda^*$ the set of all XPath expressions which denote a path occurring in at least one schema in Γ . Then P' is the set of *semantically related answers* for p if it is the maximal subset of P such that, for all $q \in P'$, one of the following conditions hold:

- 1) $\mathcal{J}(p) \leq \mathcal{J}(q)$ (semantically less general answer)
- 2) $\mathcal{J}(q) \leq \mathcal{J}(p)$ (semantically more general answer)

In most real applications, only less general answers are likely to be used; however, we cannot exclude that in some situations one might be interested in broadening the scope of a search and look for concepts that are more general than the initial one.

IV. SEMANTIC ELICITATION

A crucial issue for our approach is the definition of a reasonable implementation of the semantic elicitation function Υ . In this section we provide an algorithm which approximates Υ under the assumptions of meaningfulness. The current version is adapted from [6].

Semantic elicitation is not just a (whatever complex) syntactic rephrasing an XPath expression into an expression of some formal language. To explain what we mean, consider the two following XPath expressions:

$$\text{IMAGES/JPEG/TUSCANY} \quad (3)$$

$$\text{IMAGES/ITALY/TUSCANY} \quad (4)$$

Intuitively, the two XPath expressions could be translated into the two DL terms respectively:

$$\text{Image} \sqcap \exists \text{format.JPEG} \sqcap \exists \text{about.Tuscany} \quad (5)$$

$$\text{Image} \sqcap \exists \text{about.}(\text{Tuscany} \sqcap \exists \text{partOf.Italy}) \quad (6)$$

where **Image** is the concept of “a visual representation of an object or scene or person or abstraction produced on a surface” (from WordNet2.0, sense 1), **JPEG** is a format for electronic images, **Tuscany** is the Italian region, and so on and so forth.

Despite their isomorphic syntactical structure, the XPath expressions (3) and (4) do not have an isomorphic semantic structure. Indeed, in (3), the second and the third elements **JPEG** and **TUSCANY** are modifiers of the element **IMAGES**, while in (4) the first element, **IMAGES**, is modified by the third element, **TUSCANY**, which is in turn modified by the second element, **ITALY**. Thus, the process of semantic elicitation should be a process of deep interpretation of an XPath expression, as a human being would do. [2] argues that such a deep interpretation must take into account two general kinds of knowledge:

³The choice of the logical language depends on what kind of structures one is querying. Indeed, it’s all very well to say that we deal with XPath expressions, but one thing is to query a hierarchical classification, and one thing is to query a service description. Indeed, it is well-known that the sub-element relation in XML does not have any pre-defined meaning, and can be used to organize concepts in a taxonomy, objects in a partonomy, or even to decompose actions in a service description. A method for semantic elicitation must take into account this pragmatic aspect, and choose the most appropriate language for each case. In the situation we describe below, we are interested in querying classifications, where each node corresponds to a (complex) concept (e.g. ‘photos of my holidays in Italy’), and therefore we will adopt the language \mathcal{ALC} ; however, no DL logic language would not do for a service description.

- **Lexical knowledge:** it allows us to determine the (set of) concept(s) possibly denoted by a lemma⁴; for example, the fact that the lemma ‘image’ can mean ‘a visual representation’ and ‘a standard or typical example’. Conversely, it can be used to recognize that two different lemmas may refer to the same concept; for example, the words ‘image’ and ‘picture’ can both denote the concept of a visual representation, and therefore – under this interpretation – they are to be taken as synonyms. Formally, let m be the set of lemmas that can be denoted by words occurring in Λ . A lexicon $\mathcal{L} : m \rightarrow 2^{T \cup R}$ is a function that associates each lemma to a set of primitive concepts or roles belonging to the signature of the T-Box L . In the current version we shall use $\text{Image}\#n$ for the n -th concept that can be denoted by the lemma ‘Image’.
- **Ontological/World knowledge:** this type of knowledge concerns relations between primitive concepts. For example, the fact that there is a **PartOf** relation between the concept $\text{Italy}\#1$ (‘a republic in southern Europe’) and the concept $\text{Tuscany}\#1$ (‘a region in central Italy’). We formally define the ontological knowledge \mathcal{O} to be a set of axioms of the T-Box L . In this paper we will assume that we have a “black box” function $\mathcal{R} : T \times T \rightarrow R$ which takes as input two concepts and returns a role which holds between them. For further details, see [6].

For the sake of simplicity, we shall assume the set Λ consists of single words in English⁵. Furthermore, we shall use WORDNET⁶ as our source \mathcal{L} of lexical knowledge; finally, the terms T and the roles R of the signature S will be interpreted as WORDNET synsets; R contains two predefined roles, **IsA** and **PartOf**; the set of concepts \mathcal{C} is the set of all the allowed expression built using the signature S ; finally, the ontological knowledge \mathcal{O} contains the **IsA** and **PartOf** relations defined over WORDNET, and possibly other relations from some domain ontology. To make the presentation clearer, we show how the process works with a running example over the X-Path (2). The process of semantic elicitation is split into four main steps:

1) *Local Interpretation:* In the first phase, we try to build the space of all possible interpretations for each element of the query. Each element consists of a label and (possibly) a set of attributes. We interpret an attribute as an object which qualifies the meaning expressed by the label. Formally, we interpret a node by the expression

$$\text{label} \sqcap \exists \text{attName}_1.\text{filler}_1 \sqcap \dots \sqcap \exists \text{attName}_n.\text{filler}_n$$

In particular, the attribute name is interpreted as a role and the attribute filler as a range. We obtain the space of all possible interpretations of a node by replacing the words that occur in the pattern elements by all concept that could possibly

be denoted by the words, with respect to the lexicon \mathcal{L} . For example, our lexicon provides 7 concepts for the lemma ‘image’, 7 for ‘about’, 1 for ‘Tuscany’, 4 for ‘landscape’ and 1 for ‘JPEG’. As a result, the space of the possible interpretations for the elements of example (2) is:

$li(\text{IMAGES})$	$li(\text{LANDSCAPE})$	$li(\text{JPEG})$
$\text{Images}\#1 \sqcap \exists \text{about}\#1.\text{Tuscany}\#1$	$\text{Landscape}\#1$	$\text{JPEG}\#1$
$\text{Images}\#1 \sqcap \exists \text{about}\#2.\text{Tuscany}\#1$	\vdots	
\vdots	$\text{Landscape}\#4$	
$\text{Images}\#7 \sqcap \exists \text{about}\#7.\text{Tuscany}\#1$		

2) *Semantic Enrichment:* We now look for semantic relations that hold between the concepts defined in the previous step. This is done by accessing the ontological knowledge \mathcal{O} using the \mathcal{R} function. In particular, we search for the relations that hold between two different kinds of elements:

- **Attribute Roles:** Consider $\text{IMAGES}[\text{about} = \text{'Tuscany'}]$ in our example. In the previous step, we built the set $li(\text{IMAGES})$ of all the possible interpretations for this node. We now use the ontology \mathcal{O} to determine if it *explicitly supports* one or more of these possible interpretations. For example, we discover that $\mathcal{R}(\text{Image}\#2, \text{Tuscany}\#1) = \text{about}\#1$, i.e., that the first interpretation is supported by the ontology.
- **Structural roles:** Here, we search for semantic relations between different elements in the same XPath expression. In our example the relation $\mathcal{R}(\text{Image}\#2, \text{JPEG}\#1) = \text{format}\#1$ holds.

Table I shows the semantic relations that hold between the terms in our example.

1	$\langle \text{Image}\#2, \text{Tuscany}\#1, \text{about}\#1 \rangle$
2	$\langle \text{Landscape}\#1, \text{Image}\#2, \text{IsA} \rangle$
3	$\langle \text{Images}\#2, \text{JPEG}\#1, \text{format}\#1 \rangle$
4	$\langle \text{Image}\#2, \text{Landscape}\#3, \text{about}\#1 \rangle$

TABLE I
SET OF RELATIONS

3) *Semantic Filtering:* This step filters out the concepts and relations which do not seem to be the right ones for the XPath expression under analysis. Such a filtering applies the following rules to every concept extracted in the previous phase:

- **Weak rule:** A concept c associated to a word w occurring in an XPath tag n can be removed if c is not involved in any relation, and there is some other concept c' that is also associated to w in n , which is involved in some relations.
- **Strong rule:** A concept c associated to a word w occurring in an XPath tag n can be removed if c is not involved in any **IsA** or **PartOf** relation and there is some other concept c' associated to w in n which is involved in some **IsA** or **PartOf** relation.

An example of the use first rule is as follows. In Table I we see that $\text{Image}\#2$ occurs in relations 1–4, while $\text{Image}\#1$ and $\text{Image}\#3 \dots \text{Image}\#7$ do not occur in any relation. It is therefore likely that the “right” concept expressed by the

⁴We assume that we are able to determine the lemma of each word occurring in a label through some standard lemmatizer.

⁵For the case when Λ contains complex noun phrases, and for further discussion, see [6].

⁶WORDNET [4], a well-known Lexical/Ontological repository which contains the set of concepts possibly denoted by a word (called synsets, i.e. set of synonyms), and a set of relations (essentially the **IsA** and **PartOf**) that holding between senses

lemma ‘Image’ in this context is the second one, and the other concepts can be discarded. The second rule is stronger, as here a concept can be discarded even if it is involved in some relation. The idea is that we consider **IsA** and **PartOf** relations be stronger than the other ones, and give priority to these over others. For example, consider relations 2 and 4. Because there is a **IsA** relationship between **Landscape#1** and **Images#2** (relation 2), we can discard the concept **Landscape#3** even though this concept occur in an **about#1** relation (relation 4).

The goal of this step is to reduce the space of possible interpretations of a node, by discarding some concepts which are unlikely to be relevant. In our example, we would obtain the following terms.

$li(\text{IMAGES})$	$li(\text{LANDSCAPE})$	$li(\text{JPEG})$
$\text{Images\#2} \sqcap \exists \text{about\#1.Tuscany\#1}$	Landscape\#1	JPEG\#1

Note that if more than one concept satisfies our conditions, all of them are retained (ambiguity partially solved). Furthermore, if the concepts associated to a word are not involved in any relation, no filtering is done (ambiguity is not solved). The same filtering process is then applied to the set of relations, i.e., we discard all relations involving discarded concepts, as these ones refer to concepts that no longer exist. Table II shows the current set of relations.

1	$\langle \text{Image\#1, Tuscany\#1, about\#1} \rangle$
2	$\langle \text{Landscape\#1, Image\#1, IsA} \rangle$
3	$\langle \text{Images\#1, JPEG\#1, format\#1} \rangle$

TABLE II
SET OF FILTERED RELATIONS

4) *Constructing the representation of the semantics:* The final step is to construct the logical representation of the semantics of the query. This is done in two steps.

First, we construct the *local meaning* of an element of the query, namely its meaning considering only the label and the attributes. We define the $LM(n)$, the local meaning of an element n , as the disjunction (\sqcup) of all terms occurring in $li(n)$, the space of all the possible interpretations.

We then combine the local meanings to obtain the *global meaning* of a node. First of introducing the method, we want to make the following observation. Consider Table II: it essentially says that there is a relation **IsA** between the (concepts belonging to the interpretations of the) node **LANDSCAPE** and the (concepts belonging to the interpretations of the) node **IMAGES**, and that there is a relation **format#1** between the (concepts belonging to the interpretations of the) node **IMAGES** and the (concepts belonging to the interpretations of the) node **JPEG**. In short, we have the following set of relations between nodes: $\text{LANDSCAPE} \xrightarrow{\text{IsA}} \text{IMAGES} \xrightarrow{\text{format\#1}} \text{JPEG}$. Essentially, axioms can be interpreted as edges relating nodes. Such chain of relations can be rephrased with the following pattern⁷:

$$\text{LANDSCAPE} \sqcap \text{IMAGES} \sqcap \exists \text{format\#1.JPEG}$$

⁷Ambiguity can arise in the axioms. As an example, two elements can be modifiers of each. See [6] for a set of heuristics for solving some ambiguity problems.

At this time is quite simple to build the *global meaning* of the X-Path: indeed we need just to substitute the node labels with the local meanings provided by function $LM()$. Going on with our example, the global meaning for the X-Path (2) is the following Description Logic term:

$$\text{Landscape\#1} \sqcap \text{Images\#1} \sqcap \exists \text{about\#1.Tuscany\#1} \sqcap \exists \text{format\#1.JPEG\#1} \quad (7)$$

5) *Dealing with special symbols:* The XPath symbols $*$ and $//$ do not have an explicit semantic counterpart in some concept in C . The first is a wild card, which can be replace by any tag; the second allows us to find elements at any depth in an XML document. Here we propose a simple treatment of these two special symbols in SEMQUERY.

From Section IV-4 results that we essentially combine each element as a conjunction (\sqcap). Following this idea, we can argue that each element of an XPath is a specification of the meaning of the others elements. Consider the element $\text{IMAGES}[\text{about} = \text{'Italy'}]$. Its intuitive meaning is ‘Images about Italy’. The further element **LANDSCAPE** reduces its meaning to the ‘Images about Italy that are Landscapes’, so as the last element **JPEG** reduces the meaning to the set of ‘Images about Italy with JPEG format that are Landscapes’. Following this intuition, we can interpret a sign as $*$ as one potential element that reduces the meaning of the XPath, and the sign $//$ as a possibly empty set of potential elements that reduce the meaning of the XPath.

So, instead of introducing some redundant place-holder for that symbols, we prefer to play on the class of equivalence which an XPath where such symbols occur belong to.

Let q be an XPath where the sign $*$ ($//$) occurs. Let $Y \subseteq \Lambda^*$ be the set of all the XPaths allowed by Λ such that the symbol $*$ ($//$) in q is substituted with some element (a finite, possibly empty, sequence of elements) of Λ . Let $q^* (q//)$ be the XPath resulting by removing (substituting with $/$) element $*$ ($//$) from q . For each $y \in Y$, if

$$\mathcal{O} \models \Upsilon(y) \sqsubseteq \Upsilon(q^*) \quad (\mathcal{O} \models \Upsilon(y) \sqsubseteq \Upsilon(q//))$$

then $q^c = y^c$, namely q belongs to all the equivalence classes of equivalence to whom belongs the XPaths in Λ^* such that (i) they are generated by substituting $*$ ($//$) in q with an element (a set of elements) of Λ and (ii) their meanings are a specification of the meaning of q . Multiple occurrence of $*$ ($//$) can be defined recursively in the same way.

6) *Concluding example:* Following this approach, we can state that the class of equivalence where the XPath of example 3, namely $//\text{IMAGES}[\text{about} = \text{'Tuscany'}]/\text{LANDSCAPE}/\text{JPEG}$, belongs to is the same of the class of equivalence where the XPath $/\text{IMAGES}[\text{about} = \text{'Tuscany'}]/\text{LANDSCAPE}/\text{JPEG}$ belongs to.

Now consider the XPath expression

$$\text{IMAGES}[\text{format} = \text{'JPEG'}]/\text{ITALY}/\text{TUSCANY}/\text{LANDSCAPES}$$

from left hand schema of Figure 1. Running the semantic elicitation process, we obtain the following DL term:

$$\begin{aligned} \text{Images\#1} \sqcap \exists \text{about\#1.} (\text{Tuscany\#1} \sqcap \exists \text{partOf\#1. Italy\#1}) \\ \sqcap \text{Landscape\#1} \sqcap \exists \text{format\#1. JPEG\#1} \end{aligned} \quad (8)$$

Imagine then to have an ontology \mathcal{O} which contains the axiom $\text{Tuscany\#1} \sqsubseteq \exists \text{partOf. Italy\#1}$ (such an axiom can be found, as an example, in WORDNET), then we can say that

$$\mathcal{O} \models (7) \equiv (8)$$

so they belong to the same class of equivalence. From Definition 2 we can conclude that the XPath $\text{IMAGES}[\text{format} = \text{'JPEG'}]/\text{ITALY}/\text{TUSCANY}/\text{LANDSCAPES}$ of left schema of Figure 1 is a semantically equivalent answer for the query $//\text{IMAGES}[\text{about} = \text{'Tuscany'}]/\text{LANDSCAPE}/\text{JPEG}$.

V. CONCLUSIONS

The main idea of the paper is that querying heterogeneous information sources requires to abstract from the syntactic form of local schemata and lift the representation to a level in which only semantic differences are preserved. This is what we called semantic elicitation. Here we proposed a general method for processing these type of semantic queries called SEMQUERY, and described a technique for semantic elicitation derived from our experience on the problem of semantic interoperability in the Semantic Web.

We are perfectly aware that this is only a starting point. Future work will explore the following directions. First, we will extend the approach to other kinds of data sources (e.g. relational databases) and other query languages (e.g. SQL).

Second, we want to test the approach on real cases, and see how it performs from a user's point of view; however, we must say that similar tests have been done in our work on the Semantic Web (see e.g. [5]) in the domain of web directories and e-commerce catalogs, and the results were quite promising.

REFERENCES

- [1] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: a new approach and an application. In K. Sycara, editor, *Second International Semantic Web Conference (ISWC-03)*, Lecture Notes in Computer Science (LNCS), Sanibel Island (Florida, USA), October 2003.
- [2] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: a new approach and an application. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *The Semantic Web – 2nd international semantic web conference (ISWC 2003)*, volume 2870 of LNCS, Sanibel Island, Fla., USA, 20-23 October 2003.
- [3] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, 1999.
- [4] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, US, 1998.
- [5] B. M. Magnini, L. Serafini, A. Doná, L. Gatti, C. Girardi, , and M. Speranza. Large-scale evaluation of context matching. Technical Report 0301-07, ITC-IRST, Trento, Italy, 2003.
- [6] S. Sceffer, L. Serafini, and S. Zanobini. Semantic coordination of hierarchical classifications with attributes. Technical Report 706, Department of Informatics and Telecommunications, University of Trento, December 2004. <http://eprints.biblio.unitn.it/archive/00000706/>.
- [7] W3C. XML Path Language (XPath). <http://www.w3.org/TR/xpath>, November 1999.
- [8] W3C. XML Schema Part 1: Structures Second Edition. <http://www.w3.org/TR/xmlschema-1/>, October 2004.
- [9] W3C. XML Schema Part 2: Datatypes Second Edition. <http://www.w3.org/TR/xmlschema-2/>, October 2004.
- [10] W3C. XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery/>, October 2004.

Building Semantic Agents in eXAT

Antonella Di Stefano, Corrado Santoro

University of Catania - Engineering Faculty

Department of Computer and Telecommunication Engineering

Viale A. Doria, 6 - 95125 - Catania, Italy

EMail: {adistefa, csanto}@diit.unict.it

Abstract—This paper describes the FIPA-ACL semantics support provided by eXAT, an Erlang-based FIPA-compliant agent platform, developed by the authors, which uses the Erlang language to offer a complete environment for the realization of the behavioral, intelligent and social parts of an agent. eXAT agents can thus exploit a FSM-based abstraction for the behavioral part and an Erlang-based rule processing engine (with its own knowledge base) for the implementation of agent's reasoning process. In this architecture, a *SL Semantics Layer* is introduced to support FIPA-ACL semantics; such a module is activated during messaging and is able to automatically check and perform the *feasibility precondition* and *rational effect* relevant to the communicative act sent or received. This is performed by manipulating the knowledge base of the inference engines of the sender/receiver agent, by checking for the presence of suitable facts and/or asserting other facts, according to FIPA-ACL semantics specification. ACL semantics handling is also enriched with a reasoning module, charged with the task of providing an “higher-level” messaging, based on agent actions—rather than messages—that, after a semantics-aware reasoning process, are transformed into communicative acts.

Keywords—Interaction Semantics, FIPA-ACL, Ontologies, Agent Programming Platform, eXAT, Erlang

I. INTRODUCTION

To date, FIPA specification [26] is the most widely used and referenced standard for the development of software agents in both academic/research institutions and industries. Even if there are few projects [30] that use different ad-hoc agent architectures and models, FIPA is recognized as the leading reference architecture for open and interoperable multi-agent systems.

On this basis, some Java-based FIPA-compliant platforms have been developed [9], [1], [7], and, among them, JADE [9] can be now considered “the FIPA platform”, as it is the most widely used in agent-based projects. All of these platforms take care of only some aspects of FIPA specification, as they provide a support for agent management, directory service, ACL interaction, ontology specification and encoding, agent behavior programming. But all of them fail to take into account “agent intelligence”: As a consequence, one of the main contribution of FIPA, which is the FIPA-ACL semantics [24], cannot be supported¹. As we argued many times [17], [18], [19],

this gap between agent nature, FIPA specification and FIPA-compliant platforms is due to the fact that the Java language is not able to offer native statements to express logic constructs, like those needed by FIPA-SL language [25]. Neither logic predicates nor production rules can be described in Java and, to this aim, additional tools must be introduced [2], [28], which, however, use a different language and, in any case, are not able to support FIPA-ACL semantics.

The first attempt to fill such a gap is represented by the eXAT² platform [14], [16], [15], [18], [19], which has been developed by the authors (since 2003) with the aim of offering an environment that takes care of the three main aspects of agent-oriented programming: *behavior*, *intelligence* and (*semantic and syntactic*) *interoperability*. The key feature of eXAT that enables such an integration is the use of the *Erlang* language [5], [8] for agent programming: It is a functional language that, thanks to two main features—pattern matching in function clause declaration and handling of symbols in data—is very well suited for the implementation of both (reactive) behaviors and (intelligent) production rules [18], [13]. eXAT, designed to be FIPA-compliant, includes an inference engine that can be tightly coupled with ACL message exchanging, in order to support the reasoning process deriving from the meaning of messages sent and received, according to the semantics of FIPA-ACL. This mechanism exploits the Erlang-native data types—atom (symbols), lists, tuples and records—in order to represent *SL sentences*, handled as *facts* or *predicated* for the knowledge base of running agents.

This paper describes the functionalities of the eXAT platform from the point view of the support provided to build *semantics-aware* agents. The paper focuses on the tools and abstractions provided to write *ontologies* and use them not only in agent messaging but also in rule-based inference engines. Then the paper describes the support for FIPA-ACL semantics dealing with the primitives and mechanisms for semantic reasoning. A key aspect of the architecture is the use of *pluggable semantics*, i.e. user-defined software modules that can be plugged in the platform in order to implement an ad-hoc semantic reasoning process. A comparison of the semantics support of eXAT with other solution is also provided, showing that, even if eXAT is still a work-in-progress³, its features seem

¹We consider FIPA-ACL semantics specification very important because we argue that, without taking into account “mental attitudes” (i.e. goals, plans, beliefs, etc.), FIPA specification can be considered a standard for the interoperability of distributed reactive entities that could not necessarily be “software agents”.

²erlang eXperimental Agent Tool.

³eXAT is an “experimental” platform and it has been mainly designed for investigation purposes.

```

1 -module (reactive_agent ).
2
3 agent_loop () ->
4   E = wait_for_next_event (),
5   act (E),
6   agent_loop ().
7
8 act ({switch , on}) -> % act when switch is turned on
9 act ({switch , off}) -> % act when switch is turned off
10 act ({temperature , X}) when X > 30 ->
11   % act when the temperature is greater than 30
12 act ({temperature , X}) when X < 20 ->
13   % act when the temperature is less than 20
14 act (_) -> % unknown event, no action

```

Fig. 1. A simple pure-reactive agent in Erlang

able to provide a “semantic environment” more flexible and complete than that of other similar proposals.

The paper is organized as follows. Section II illustrates the motivations behind our choice of employing the Erlang language in writing agent systems (and thus the reasons why we developed eXAT). Section III gives a brief overview of the eXAT platform. Section IV describes the support for SL and ontology handling in eXAT. Section V deals with the semantic framework. Section VI compares our approach with other solutions. Section VII concludes the paper.

II. WHY ERLANG?

Some of the main reasons that led us to choose Erlang as a possible language for the development of agent systems, and that in turn guided us in realizing the eXAT platform, are discussed in [17], [19]. In those papers, the authors first derive an abstract model of intelligent agent, based on the concepts of finite-state machine and rule-production system, and then introduce some properties that should be met by an agent programming language. Here, the basic properties of agents listed in [32]—*reactivity*, *pro-activeness* and *social ability*—are instead taken into account and, starting from them, the reasons for the use of Erlang are subsequently derived.

A. Reactivity

An agent has the basic capability of reacting to incoming events. This includes e.g. a change of the state of the reference environment, the arrival of a messages from the user or other agents, the occurrence of exceptional conditions, etc. An event can be considered featured by a *type* and *additional data* bound to the event itself (e.g. for an incoming message, the additional data could be the payload) and, on this basis, suitable predicates on bound data can discriminate various reaction cases to events of the same kind.

From the programming point of view, reacting to events implies to provide (i) an abstraction for modeling events and (ii) some constructs or library calls to specify the computation to be triggered when a particular event occurs, also given that the bound data could be subject to certain conditions. Erlang seems particularly suitable to face such requirements for the following reasons:

```

rule (Engine , {'child -of ' , X, Y}, {female , Y}) ->
  eresye :assert (Engine , {'mother -of ' , Y, X});

rule (Engine , {'child -of ' , X, Y}, {male , Y}) ->
  eresye :assert (Engine , {'father -of ' , Y, X}).

```

Fig. 2. Some Erlang function clauses expressing inference rules

- 1) Erlang is a symbolic language (like Prolog or LISP), and it is known that the use of *literal symbols* (*atoms*) facilitates the representation of constants in data⁴. Structured information can be represented as *tuples*⁵ and, since they are untyped, are well-suited for heterogeneous data [31] and thus particularly appropriate for event types that could be very different one another. For example, the state of a switch can be represented as {switch, on} or {switch, off}, a sensed temperature with {temperature, 25}, an incoming message as {message, 'QUERY-IF', {sender, 'UserAgent'}}, etc.
- 2) Erlang is a functional language and functions can have multiple clauses, each one expressing a match on one or more parameters; clauses can also have guards to specify more complex matching expressions. Matching on function definition can be exploited to specify the computation to execute following an incoming event formed as desired: Function (clause) declaration will specify the matching criteria relevant to a triggering event, while function body will implement the associated action.

The example in Figure 1 shows a practical usage of the concepts indicated above. The listing in the Figure reports a possible implementation of a (very simple) pure reactive agent programmed in Erlang. Agent's main loop (function `agent_loop`, lines 3–6) waits for an incoming event and then executes the associated action; computations tied to events are specified by using multiple clauses of the function `act`, each one specifying a different matching value for the parameter: When the function is invoked using the event acquired (line 5), only the matching clause is activated (if one exists, otherwise the default clause—line 14—is chosen). As the reader can appreciate, using symbols, structured data and function with several clauses improve not only engineering and implementing reactive agents, but also the readability of the source code.

B. Pro-Activeness

Pro-activeness means the capability of an agent to develop and execute *plans*, in order to achieve a specific goal. Unless specific BDI tools are employed [6], [28], such an ability is generally supported by means of a rule production system [3],

⁴A symbol (atom), in Erlang, is a string constant beginning with a lowercase letter or any string literal enclosed in single quotes, e.g. 'My_atom'.

⁵A tuple, in Erlang, is a comma-separated set of identifiers enclosed in graph braces.

[2], [4], [13], featuring a *knowledge base* and a set of *inference rules*. In this context, Erlang's features are particularly interesting for the following reasons:

- 1) Symbols and primitive types (i.e. atoms and tuples) are well suited to represent *facts* of a knowledge base; moreover the use of the same types for facts and events (i.e. tuples) facilitates agent programming, allowing the direct use of event data in the knowledge base.
- 2) Function clauses, which indeed represent *predicates* on parameters that if matched activate the clause, fit well in the representation of the *precondition* part of a rule; at the same time, the function body can represent the *action* part.
- 3) The Erlang-native pattern matching mechanism facilitates the implementation of rule-handling algorithms, also improving processing performances.

Note that despite Erlang's capability to represent rules, the language and run-time system do not include an engine for rule processing, which has to be provided by an external tool⁶. For this reason, the **ERESYE** system has been designed by the authors [13] and it has been included in the **eXAT** platform. **ERESYE** is an Erlang-based rule production system featuring the same characteristics (from both the syntactic and semantic point of view) of other well-known similar tools, such as OPS5 [20], [21], CLIPS, Jess, etc.

The example in Figure 2 gives a sketch of Erlang function clauses used as rules of an **ERESYE** inference system. In the example, the rules shown permit to enrich the knowledge by deriving the concepts of 'father-of' and 'mother-of', on the basis of the knowledge of the 'child-of' and "gender" concepts.

In the **eXAT** platform, **ERESYE** is used not only to support a (user-defined) agent's reasoning process but also the inference process required by ACL message semantics, as it will be explained in Section V.

C. Social Ability

Agent-oriented engineering is based on subdividing a whole application into a set of goals to be achieved by several cooperating agents; thus the possibility of supporting interaction among agents is a mandatory functionality of any agent programming language or platform. The Erlang language and its run-time system have been explicitly designed to support communication, thus providing the programmer with a set of smart and flexible language constructs to perform message exchanging among (local or remote) processes. Messages that can be exchanged are basic Erlang data types and include atoms, tuples, strings, lists, etc., no further manipulation (e.g. enveloping, etc.) is needed. Moreover, language constructs for interaction do not change should a receiving process be local or remote. As reported in [8], [18], the programming model of applications in Erlang is based on subdividing a problem into a set of tasks to be assigned to the same number of *concurrent processes* that *share nothing* and interact each other

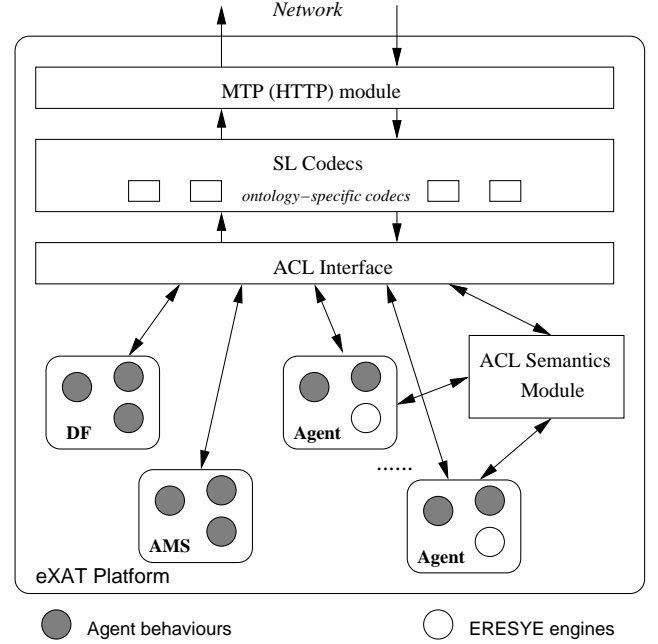


Fig. 3. Architecture of the eXAT Platform

only by means of *message passing*. The reader can appreciate the similarity between this model and the basics of multi-agent systems: Erlang concurrency model and interaction constructs seem thus perfect "as-is" to support interactions among (Erlang-programmed) agents. The only concern is with the exchanging protocol and data representation, which is Erlang-proprietary and thus non-standard (even if it is documented). An agent platform is thus needed when standard messaging, as in FIPA, is required to favor the interoperability with different platforms and agents written with other programming languages.

III. OVERVIEW OF EXAT

Even if the **eXAT** platform has been already described in [14], [15], [16], [17], [19], [18], [13], it is worthwhile to give an overview of it, in order to help the reader in better understanding the remaining part of the paper.

The **eXAT** platform has been designed with the objective of providing an "all-in-one" environment to execute agents and to program them in their *behavioral (reactive)*, *intelligent (pro-active)* and *cooperative (social)* parts, all with the same language (Erlang).

Agent behaviors can be programmed by means of finite-state machines (FSMs), enriched with the possibility of using *composition*, i.e. serial and parallel execution of sub-FSMs, and *extension*, i.e. refining some parts of an existing FSM (according to the concept of virtual inheritance proper of the object-oriented technology) in order to support new requirements.

Agent intelligence is instead programmed by means of rule-based code, supported and executed by the **ERESYE** tool (as briefly illustrated in the Section II). An **ERESYE**

⁶Erlang is functional, not logic.

(a) wine.onto

```

class (wine_grape ) ->
{ name = [string , mandatory , nodefault ] };

class (wine ) ->
{ name = [string , mandatory , nodefault ],
  color = [string , mandatory , nodefault ],
  flavor = [string , mandatory , nodefault ],
  grape = [set_of (wine_grape ), mandatory , nodefault ],
  sugar = [string , mandatory , nodefault ]};

class ('red -wine ') -> is_a (wine ),
{ color = [string , mandatory , default (red )] };

class ('white -wine ') -> is_a (wine ),
{ color = [string , mandatory , default (white )] };

class ('Chianti ') -> is_a ('red -wine '),
{ sugar = [string , mandatory , default (dry )] }.

```

(c) wine_agent.erl

```

-module (wine_agent ).
-include ("wine .hrl "). %include the ontology records

on_starting (Self ) ->
  ontology_service :register_codec ("wine ",
                                   wine_ontology_sl_codec ).

send_inform_action (Self , _ , _ , _ ) ->
  acl :inform (
    #aclmessage { sender = Self ,
                  receiver = Dest ,
                  ontology = wine ,
                  content = #Chianti ' {
                                name = 'Barone Ricasoli ',
                                grape = ...,
                                flavor = ... }
                  } ).

```

(b) wine.hrl

```

-record (' wine_grape ', {
  'name ' }).

-record (' wine ', {
  'name ' ,
  'color ' ,
  'flavor ' ,
  'grape ' ,
  'sugar ' }).

-record (' red -wine ', {
  'name ' ,
  'color ' = 'red ' ,
  'flavor ' ,
  'grape ' ,
  'sugar ' }).

-record (' white -wine ', {
  'name ' ,
  'color ' = 'white ' ,
  'flavor ' ,
  'grape ' ,
  'sugar ' }).

-record (' Chianti ', {
  'name ' ,
  'color ' = 'red ' ,
  'flavor ' ,
  'grape ' ,
  'sugar ' = 'dry ' }).

```

Fig. 4. The “wine” ontology and an excerpt of the generated include file

engine, together with its programmed rules, can be bound to an agent of the platform in order to support agent’s inference: The knowledge base of the engine can thus represent agent’s mental state, while production rules support agent’s reasoning process. ERESYE engine’s events can be bound to behaviors, thus allowing reasoning processes to also trigger user-defined agent actions.

Agent interaction is performed by means of the exchange of FIPA-ACL messages; this is supported by the eXAT’s ACL modules that include library functions to send and receive communicative acts and codecs for user-defined ontologies. Message exchanging is mainly connected to behavior execution in order to make possible the occurrence of a proper event when a new message is delivered to the agent. But message exchanging is also able to influence agent’s mental state thanks to the support of FIPA-ACL semantics: An incoming message is processed by the ACL semantics module and, according to the performative name and the message content, suitable actions are performed on the knowledge base of the ERESYE engine bound to the receiving agent. The details of such a process are reported in Section V.

Figure 3 reports a sketch of the architecture of the platform.

According to FIPA abstract architecture [22], the platform (at runtime) includes also the MTP module⁷, as well as AMS and Directory Facilitator agents, which provide the agent directory and the service directory, respectively.

IV. WRITING AND USING ONTOLOGIES IN EXAT

One of the key features that allows interoperability in multi-agent systems is to make interacting agents sharing the same concepts in their “universe of discourse”: In other words, they should share the same *ontology*. Ontology writing and manipulation is thus a mandatory characteristic that any FIPA-compliant agent platform has to feature, as well as modules to translate messages, written in the SL language [25], [23], into constructs and data types proper of the programming language employed (and vice-versa).

In order to comply with these requirements, eXAT provides a support for ontologies—i.e. concepts organized in *classes* with hierarchies—and for their use in agent behaviors, agent messaging and ERESYE engines. Ontologies can be written, in a specification file, using a (more or less) standard notation;

⁷Current eXAT version supports only the HTTP message transport protocol.

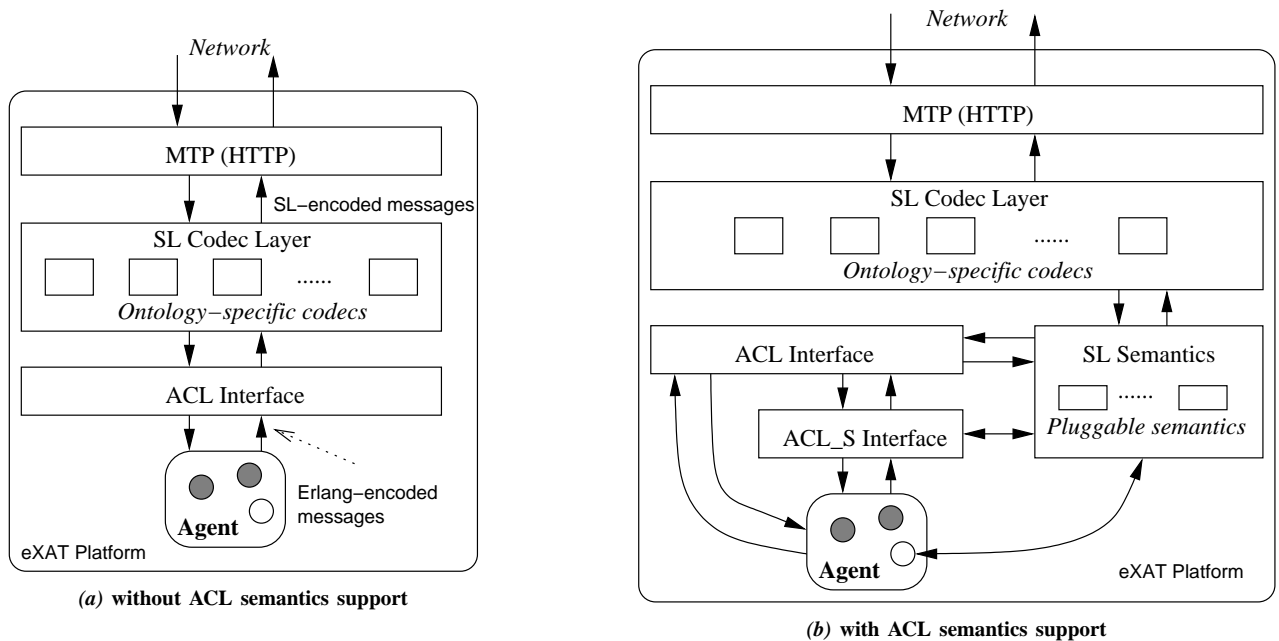


Fig. 5. Architecture of modules for message exchanging and handling in eXAT

in the current version of eXAT, ontologies can be written using an ad-hoc Erlang-like syntax, as Figure 4a illustrates, while the ability to translate files written in standard notations, such as OWL, or by means of visual tools, such as Protégé, will be available in the next releases of the platform. Then a suitable *Ontology Compiler*, provided with eXAT⁸, is able to parse such ontology specification files and generate the relevant Erlang type definitions to be used in agent source code. Since Erlang is not object-oriented, a task of the Ontology Compiler is also to transform the object-based ontology specification into an Erlang-readable (non-object-based) form, while maintaining semantics. This is performed by generating some functions that reflect the class hierarchy.

In detail, the Ontology Compiler generates, from the ontology specification file, the following Erlang sources:

- i) An Erlang (*.hrl*) include file, which reports the definition of an Erlang record for each class⁹, provided that the hierarchy is “flattened” by incorporating each attribute of a class/record into all the relevant child class/records; therefore, creating a fact referring to an object of class ‘T’ implies to create an Erlang record of type ‘T’. Figure 4b reports an excerpt of the include file generated from the “wine” ontology in Figure 4a.
- ii) An Erlang source (*.erl*) file (**class-hierarchy file**), containing information on class hierarchy, which is lost in the include file, and encoded by means of suitable `is_a` and `childof` functions. This source file also contains some functions to perform class typecasting (up- and down-casting).

⁸Also the Ontology Compiler is written in Erlang.

⁹An record in Erlang is like a “struct” in C, it has a name and a set of named fields; however, according to Erlang syntax and unlike C, fields are untyped.

- iii) An Erlang source (*.erl*) file (**parser file**), containing the parser (*codec*) for the translation of the concepts defined with Erlang records from/to FIPA-SL language.

Once generated, the *.hrl* file can be included in the agent source code in order to allow a programmer to directly use the generated Erlang records in the specification of and access to a message content. The other files, once compiled, are instead used as libraries. Functions provided by the class-hierarchy file can be used by ERESYE engines and/or agent’s code to perform check or manipulation of ontology records. Functions provided by the parser file are instead internally used by the eXAT platform to perform automatic encoding/decoding of message contents. To this aim, eXAT provides a function call that agents can use to *register* an ontology by giving its name and the name of the parser module (*codec*) generated by the Ontology Compiler. This means that, when a message is received through the network by the platform’s MTP module (see Figure 5a), its SL-encoded payload is passed to the SL codec layer: If the ontology specified in the message is registered, the relevant ontology-specific codec is called and the message content is automatically translated into the relevant Erlang record(s)¹⁰. A similar process is performed when a message has to be sent: Erlang record(s) can be directly used in the source code and it’s up to the ontology-specific codec to perform automatic Erlang-to-SL translation.

As an example, Figure 4c shows a piece of code of an agent that, after startup, registers the codec for the “wine” ontology (function `on_starting`) and, when `send_inform_action` is called, sends an “inform” speech act containing information on a Chianti wine.

¹⁰If the ontology is not registered the content is passed as is, i.e. encoded in a string.

(a) SL

```

(B
  (agent -identifier
    :name alice@JADE
    :address
      (set (http :// csanto .diit .unict .it:7778/ acc )))
  (temperature 50 C)
)

-----

(I
  (agent -identifier
    :name alice@JADE
    :address
      (set (http :// csanto .diit .unict .it:7778/ acc )))
  (done
    (action
      (agent -identifier .... )
      (purchase computer 500)
    )
  )
)

-----

(iota
  ?x
  (temperature ?x C))

```

(b) Erlang

```

#'B ' {
  identifier =
    #'agent -identifier ' {
      name = "alice@JADE ",
      addresses = ["http :// csanto .... it:7778/ acc "]
    },
  formula =
    #temperature {value = "50 ", um = "C" }
}

-----

#'I ' {
  identifier =
    #'agent -identifier ' {
      name = "alice@JADE ",
      addresses = ["http :// csanto .... it:7778/ acc "]
    },
  formula = #done {
    action = #action {
      identifier = #'agent -identifier ' { .... },
      action = #purchase { item = "computer ",
                           price = "500 " }}
  }
}

-----

#iota { term = #var {name = "x"},
        formula = #temperature {
          value = #var {name = "x"},
          um = "C" }}

```

Fig. 6. Correspondence between some SL constructs and the relevant constructs translated in Erlang by eXAT

V. THE SEMANTIC FRAMEWORK OF eXAT

A. eXAT and the FIPA Semantic Language

Supporting FIPA-ACL semantics in an agent platform means to tie the acts of sending and receiving a message to agent's mental state and reasoning process. In fact, the basic principles regulating FIPA-ACL semantics are in the so-called *feasibility precondition (FP)* and *rational effect (RE)*: For each communicative act type, *FP* is a predicate, on sender's mental state, that has to be true for the message to be sent, while *RE* represents a condition, on sender's and receiver's mental state, to be met when the message has been delivered [24]. These conditions are expressed using modal logic constructs that have their concrete representation and implementation in the SL language. Moreover, the semantics of many communicative acts is based on the fact that the content field of a message is also expressed in SL or, if this is not the case, in a language that is able to represent the SL's modal logic semantic constructs. SL can be thus considered not only a simple content language but also a mandatory building block for a concrete support of FIPA-ACL semantics.

Following the statement above, and given that eXAT allows agents to handle message contents using Erlang types, not SL constructs, a suitable way to represent SL logic expressions is also needed in the platform. In this sense, eXAT handles SL constructs using a model similar to that of ontologies: SL sentences and operators are translated into suitable *Erlang records*, where the record name is equivalent to the name of the SL operator, while the other fields represent the arguments. As an example, Figure 6 reports the correspondence between

some SL constructs and the relevant constructs translated in Erlang records; in particular the Figure shows the “B” (*believes*) and “I” (*intends*) modal operators, and the “i” (*iota*) referential operator. This means that, in encoding/decoding a message content (see Figure 5a), SL-specific constructs and operators are first taken into account by the SL Codec Layer; then all other non-SL-specific constructs that appear in the message are passed to the ontology-specific codec, thus building the final message in the proper representation. Note that the use of Erlang records to represent SL constructs is not a case, since such types can be directly used in ERESYE engines; this means that not only message contents but also SL constructs can take concrete part to the agent's reasoning process.

B. Architecture and Functionality

FIPA-ACL semantics is supported, in eXAT, by means of several modules that connect the incoming and outgoing messages to the “agent's mind”, i.e. the ERESYE engine representing agent's mental state. With reference to Figure 5b, which reports the architecture of eXAT with ACL semantics support, such modules are *ACL Interface*, *SL Semantic Layer* and *ACL_S Interface*. The first two are the main modules responsible for handling the basic FIPA-ACL semantics, while the third module, *ACL_S Interface*, is charged with the task of providing an “higher-level” messaging, based on agent actions—rather than messages—that, after a semantics-aware reasoning process, are then transformed into communicative acts (this functionality is detailed in Section V-C).

In order to use the semantic support, an agent has to activate it; this is performed by means of a suitable function, to be called in the agent's body, that also associates an ERESYE engine to the agent, to be used as "agent's mind". After that, as Figure 5b depicts, each incoming (resp. outgoing) message is processed by the SL Semantic Layer before being delivered to the agent (resp. sent through the network). On the basis of the message's direction (incoming or outgoing), the SL Semantic Layer performs the following tasks:

- a. **Outgoing messages.** Before sending a message, the SL Semantic Layer checks for its *feasibility precondition*, according to the communicative act being issued. Since *FP* is based on SL modal logic predicates, this operation is performed by checking that the relevant Erlang-translated SL expressions are *asserted* (i.e. present)—or *not asserted*—in the knowledge base of the ERESYE engine representing the sender agent's mental state. For example, for a "confirm" communicative act whose content is X , the *FP* is $B_i X \wedge B_i U_j X$ ¹¹, thus the task of the SL Semantic Layer is to verify that facts " X " and " $\#U' \{ \text{identifier} = j, \text{formula} = X \}$ " are present in i 's mind. When the message has been successfully sent, the SL Semantic Layer performs the *rational effect* for the sender agent, that is, it asserts the facts that reflect, in sender agent's mental state, the communicative act semantics following message forwarding. For "confirm", for example, the SL Semantic Layer will assert the fact " $\#B' \{ \text{identifier} = j, \text{formula} = X \}$ " in i 's mind. Appropriate internal rules are also implemented to avoid consistency problems in the presence of contradictory facts; as instance, the assertion of both $B_j X$ and $U_j X$ results in a contradiction, so an internal rule is used to remove (in this case) the latter fact, leaving the former asserted.
- b. **Incoming messages.** When a message is received in a platform, before forwarding it to the destination agent, the SL Semantic Layer is charged with the task of asserting the *FP* and performing the *RE* (for the receiver agent¹²), that is (once again) to assert the proper facts, in the agent's mind, according to the communicative act and message context. For a "confirm" communicative act with content X , for example, the *RE* will be the assertion of X in receiver agent's mind.

The internal architecture of the SL Semantic Layer is organized in a way as to provide a great flexibility in semantics handling, allowing a programmer to define and implement its own semantic support. Such a functionality is achieved by means of *pluggable semantics module*, i.e. Erlang modules¹³ that can be plugged-in at run-time in order to support user-defined semantics. In fact, it should be noted that, even if

FIPA-ACL semantics is a FIPA-approved standard, it has been often criticized¹⁴ and alternative proposals have been provided [12]; therefore the possibility of employing user-defined semantics is, in the authors' opinion, a very important characteristics that any semantics-aware agent platform should feature.

In eXAT, plugging-in operation is performed at the agent level, using the same function call that enables ACL semantics for an agent; this function, called `agent:set_rational`, takes two arguments: (i) the name of the ERESYE engine representing agent's mind and (ii) the name of the Erlang module implementing the code for semantic support (in particular, for FIPA-ACL standard, the module is "`fipa_std_semantics`").

In order to be plugged-in, semantics modules must export two functions: `is_feasible` and `rational_effect`. The former is called by the SL Semantic Layer before sending the message, by passing, together with the message to be sent, the (identifier of the) sender agent and the (identifier of the) ERESYE engine representing sender agent's mind. Multiple clauses of this function can be used to discriminate the action to be taken on the basis of the different communicative act carried by the message. The latter function—`rational_effect`—is instead called when the message is sent (from sender's side) and received (from receiver's side). The function takes the same data of the former function plus an additional parameter, which can assume the value of one of the atoms "sender" or "receiver" and indicates the peer at which the function is called. Also in this case, multiple clauses can discriminate the various cases, i.e. sender or receiver side, as well as the communicative act, thus allowing the implementation of the rational effect appropriate for the message.

An additional feature of the pluggable semantics support is the possibility of refining some parts of another semantics module, according to the principles of code inheritance, proper of the object-oriented technology. Programmers can "inherit" all the functionalities of a yet existing semantics module and modify only some parts of it, e.g. the *FP* of one or more communicative act, the *RE* of only one communicative act at sender side, etc. In this case, the programmer has to specify the name of the module to extend and then to write only the functions implementing the new functionalities. Such an object-based behavior (which is not Erlang-standard but provided by eXAT as an additional feature) introduces great flexibility and improves semantics engineering a lot. As an example, Figure 7 shows a user-defined semantics module that inherits all functionalities from "`fipa_std_semantics`" (see functions `extends`) and overrides the actions to be taken for the "inform" communicative act. In particular, no checks

¹¹The formula means that " i believes X and it believes that j is uncertain about X ", where i is the sender and j is the receiver.

¹²These operations are performed only if the receiver agent has enabled the support for ACL semantics.

¹³A "module" in Erlang is a set of functions belonging to the same source file.

¹⁴One of the main critique is that FIPA agents are "benevolent", e.g. issuing an "inform" implies, as a precondition, that the sender has to believe what it is saying: So a FIPA agent cannot lie. But this could not be a practical case, as in auctions, for example, a competitive behavior could also consider lying in order to try to convince other agents to give up bidding and thus win the auction.

```

-module (fipa_semantics_simple ).
-export ([extends/0, is_feasible/4, rational_effect /5]).
-include ("acl.hrl").
-include ("sl.hrl").

extends () -> fipa_std_semantics .

is_feasible (Self, Agent, Engine,
            AclMessage =
              #aclmessage { speechact = 'INFORM ' }) ->
  true .

rational_effect (Self, Agent, Engine,
                AclMessage =
                  #aclmessage { speechact = 'INFORM ' },
                sender) ->
  % rational effect on sender side
  Fact = #'B' { identifier = AclMessage #aclmessage .receiver,
                formula = AclMessage #aclmessage .content },
  eresye :assert (Engine, Fact),
  true ;

rational_effect (Self, Agent, Engine,
                AclMessage =
                  #aclmessage { speechact = 'INFORM ' },
                receiver) ->
  % rational effect on receiver side
  eresye :assert (Engine, AclMessage #aclmessage .content ),
  true .

```

Fig. 7. A pluggable semantics module

are performed for the *FP*, while some facts are asserted for the *RE*.

C. Semantics-aware Messaging in eXAT

The interaction model of eXAT, as that of other agent platforms, is based on the assumption that agents explicitly perform the actions of sending and receiving communicative acts as part of their behavior. To this aim, appropriate “send” and “receive” primitives (or equivalent mechanisms) are available to agent programmer.

But the situation could change when ACL semantics is considered, since the actions of sending and receiving a message are intrinsically and tightly connected to the agent’s mental state, which is dynamic in nature. A clear example is the “confirm”/“inform” question; given that both of these communicative act are used for the same purpose (communicating that a given proposition is true), the use of one of these depends on the feasibility precondition: If the sender believes that the receiver is uncertain about the proposition then a “confirm” should be used, otherwise an “inform” is made necessary.

Another example, is the fact that some communicative acts carrying certain proposition are considered equivalent to other communicative acts. As instance, saying that agent *j* agrees to perform a requested action *a* (“agree” comm. act) is equivalent to inform that *j* intends to do *a*, i.e. $I_j Done(a)$; this means that issuing an “inform” communicative act with $I_j Done(a)$ as content, and given that *j* has received a prior request to do *a*, should result in an “agree”, instead of “inform”.

For these reasons, in ACL semantics-aware agents, a better approach seems to avoid the direct use of communicative acts and replace them “higher level” actions [10], [11].

In eXAT, such a support is provided by the *ACL_S Interface* (see Figure 5b), which offers to the agent a set of primitives for high-level actions derived from grouping the various communicative acts into some *categories*. Such a categorization has been performed following not only the principles of the speech act theory [29], but also the semantic equivalence of some communicative acts as reported in [24]. However, note that the functionalities of the ACL_S Interface, as well as the communicative act categorization, is still at a preliminary experimental level.

The categories considered are:

- **Assertive**—This category holds all communicative acts that express the truth of a proposition. The acts are “accept-proposal”, “agree”, “cancel”, “confirm”, “disconfirm”, “inform”, “refuse” and “reject-proposal”.
- **Directive**—This category holds all communicative acts that express the desire of the sender agent that an action has to be performed, i.e. “cfp”, “request”, “request-when”, “request-whenever”, “propagate” and “proxy”.
- **Interrogative**—This category holds all communicative acts that express a query to a given agent. Communicative acts of this category are “query-if” and “query-ref”.
- **Exceptional**—This category holds all communicative acts that express an (exceptional) error condition, i.e. “not-understood” and “failure”.

The ACL_S Interface provides a different primitive—i.e. *assert*, *perform*, *query*, *report*—for each of the four categories; each of these primitives, on the basis of the message content passed as parameter and the sender agent’s mental state, builds the proper communicative act and then sends it. For example, invoking the *query* primitive with a referential operator as message content (i.e. #iota, #all or #any) will automatically results in a “query-ref” communicative act, while, if the content is a simple proposition, a “query-if” is issued.

The ACL_S Interface is also able to react to incoming messages by automatically sending a reply on the basis of the knowledge of the receiver agent, present in the associated ERESYE engine. For example, if a “request” is received and the knowledge base of the receiver agents contains a fact expressing that the agent (already) has the intention of doing the action, then an “agree” is automatically replied. Similarly, if a “query-ref” is received, the knowledge base of the ERESYE engine of the receiver agent is queried for facts meeting the referential expression, and a subsequent “inform” is generated and issued.

Such an automatic reaction process can be however controlled by the receiver agent, in order to allow agents maintaining their autonomy, as this is a mandatory characteristic of agent technology.

VI. RELATED WORK

Currently, the sole proposal¹⁵ dealing with FIPA-ACL semantics is the JADE Semantic Agent [27] (JSA), presented, for

¹⁵At the time this paper has been written.

the first time, at AAMAS 2005. JSA is a JADE add-on that implements a reasoning engine that, on the basis of agent's knowledge and some built-in production rules, is able to automatically generate and send the messages needed for an agent to achieve its goal. Similarly, incoming communicative acts are processed in order to affect receiver agent's knowledge on the basis of the rules of FIPA-ACL semantics, thus automatically generating a proper reply, if needed.

A great advantage of JSA is that it can be integrated in JADE, even if this integration is not so tight. In fact, JSA requires to handle concepts directly in SL forms, using Java strings in agent's code and without any connection to JADE ontology framework. This impedes (or burdensome) the operation of porting agents that yet use JADE ontologies to a "semantics-aware" form, but such agents will have to be almost entirely rewritten. Moreover, even if the reasoning framework provides a programming capability (e.g. user-defined "listeners" can be defined when a fact has been asserted), the rules implementing the reasoning process are built-in and cannot be modified to implement an ad-hoc semantic support.

From this point of view, the semantic support of eXAT seems more flexible, as it is basically possible to use the same Erlang structures (and the same data) to represent message contents, SL expressions and facts, thus tightly avoid any form of data conversion to use a message content or SL expression in a rule-based reasoning (and vice-versa). Moreover, not only a user-defined reasoning process can be implemented through the provided ERESYE tool, but also the FIPA-ACL semantics support is programmable, thus giving the programmer a full control over the reasoning mechanisms behind semantics handling and providing a flexible and complete agent programming environment. The only issue could be that eXAT is based on Erlang, not Java, but, as argued in [14], [15], [16], [17], [19], [18], Java does not seem the best choice for agent implementation.

VII. CONCLUSIONS

In this paper, the semantic framework of eXAT has been presented. Such a framework is able to support FIPA-ACL semantics, thus allowing the implementation of "really rational" agents. This objective has been achieved by means of a platform architecture that integrates and connects one another the modules for *messaging*, *ontology handling* and *rule-based reasoning*. Moreover, the full programmability of such modules provides a very flexible environment for the development of semantics-aware multi-agent systems.

REFERENCES

- [1] "http://fi pa-os.sourceforge.net/. FIPA-OS Web Site." 2003.
- [2] "http://herzberg.ca.sandia.gov/jess/. JESS Web Site." 2003.
- [3] "http://www.ghg.net/clips/CLIPS.html. CLIPS Web Site." 2003.
- [4] "http://www.drools.org. Drools Home Page." 2004.
- [5] "http://www.erlang.org. Erlang Language Home Page." 2004.
- [6] "http://www.agent-software.com." 2004.
- [7] "http://sourceforge.net/projects/zeusagent/. ZEUS Agent Toolkit Web Site." 2005.
- [8] J. L. Armstrong, M. C. Williams, C. Wikstrom, and S. C. Virding, *Concurrent Programming in Erlang, 2nd Edition*. Prentice-Hall, 1995.
- [9] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework," *Software: Practice and Experience*, vol. 31, no. 2, pp. 103–128, 2001.
- [10] F. Bergenti and A. Poggi, "A development toolkit to realize autonomous and interoperable agents," in *5th International Conference on Autonomous Agents (Agents 2001)*, Montreal, Quebec, Canada, 2001.
- [11] —, "Formalizing the Reusability of Software Agents," in *4th International Workshops on Engineering Societies in the Agents World (ESAW 2003)*, London, UK, 2003.
- [12] M. Colombetti, N. Fornara, and M. Verdicchio, "A Social Approach to Communication in Multiagent Systems," in *First International Workshop on Declarative Agent Languages and Technologies (DALT 2003)*, vol. LNCS 2990. Melbourne, Australia: Springer, 2003.
- [13] A. Di Stefano, F. Gangemi, and C. Santoro, "ERESYE: Artificial Intelligence in Erlang Programs," in *Erlang Workshop at 2005 Intl. ACM Conference on Functional Programming (ICFP 2005)*, Tallinn, Estonia, 25 Sept. 2005.
- [14] A. Di Stefano and C. Santoro, "eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2003)*, Villasimius, CA, Italy, 10–11 Sept. 2003.
- [15] —, "eXAT: A Platform to Develop Erlang Agents," in *Agent Exhibition Workshop at Net.ObjectDays 2004*, Erfurt, Germany, 27–30 Sept. 2004.
- [16] —, "Designing Collaborative Agents with eXAT," in *ACEC 2004 Workshop at WETICE 2004*, Modena, Italy, 14–16 June 2004.
- [17] —, "On the use of Erlang as a Promising Language to Develop Agent Systems," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2004)*, Torino, Italy, 29–30 Nov. 2004.
- [18] —, "Supporting Agent Development in Erlang through the eXAT Platform," in *Software Agent-Based Applications, Platforms and Development Kits*. Whitestein Technologies, 2005.
- [19] —, "Using the Erlang Language for Multi-Agent Systems Implementation," in *2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, Compiègne, France, 19–22 Sept. 2005.
- [20] C. Forgy, "OPS5 Users Manual," Dept. of Computer Science, Carnegie-Mellon Univ., Tech. Rep. CMU-CS-81-135, 1981.
- [21] —, "The OPS Languages: An Historical Overview," *PC AI*, Sept. 1995.
- [22] Foundation for Intelligent Physical Agents, "FIPA Abstract Architecture Specification—No. SC00001L," 2002.
- [23] —, "FIPA ACL Message Representation in String Specification—No. SC00070I," 2002.
- [24] —, "FIPA Communicative Act Library Specification—No. SC00037J," 2002.
- [25] —, "FIPA SL Content Language Specification—No. SC00008I," 2002.
- [26] —, "http://www.fi pa.org," 2002.
- [27] T. Martinez and L. Vincent, "JADE Semantic Framework," in *JADE Workshop at 4th AAMAS 2005*, Utrecht, The Netherlands, 2004.
- [28] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *Telecom Italia Journal: EXP - In Search of Innovation (Special Issue on JADE)*, vol. 3, no. 3, Sept. 2003.
- [29] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
- [30] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa, "The RETSINA MAS Infrastructure," *Special joint issue of Autonomous Agents and Multi-Agent Systems Journal*, vol. 7, no. 1 and 2, July 2003.
- [31] C. van Reeuwijk and H. J. Sips, "Adding tuples to Java: a study in lightweight data structures," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 5–6, pp. 423–438, 2005.
- [32] M. J. Wooldridge, *Multiagent Systems*. G. Weiss, editor. The MIT Press, April 1999.

Integrating Ontologies in Mobile Agents

F. Corradini, R. Culmone, M.R. Di Berardini and E. Merelli

Dipartimento di Matematica e Informatica

Università di Camerino

via Madonna delle Carceri

62032 Camerino, Italy

Email: {flavio.corradini, rosario.culmone, mariarita.diberardini, emanuela.merelli}@unicam.it

Abstract—The process of information extraction and data integration in a global information system demands automatic techniques for quickly determining semantic similarity among concepts across different ontologies. This paper presents a graph based approach for computing, on-the-fly, semantic similarities among ontologies of a specific domain. The approach consists of integrating mobile agents and ontologies to support a variety of applications in distributed environments. The resulting technique is illustrated on Hermes, agent-based middleware for mobile computing, by an example in molecular biology domain.

I. INTRODUCTION

In recent years, ontologies [13], [14], [8], have played an important role in many research areas such as information retrieval and data integration; ontologies are useful for semantic interoperability among heterogeneous information systems [20]. In the information and computer science, an ontology is a type of knowledge-base that describes the concepts, through definitions, that are sufficiently detailed to capture the semantics of a specific domain [13], [15]. An ontology captures a certain view of the world, it provides a vocabulary of terms and relations to model the domain [8]; it supports intentional queries regarding the content of one or more data repositories, and it reflects the relevance of data by providing a description of semantic information independent of the data representation.

In a global environment, the interoperability of information systems, is based on the possibility to offer a query environment in which users may enter a request without knowing how and where the requested information are stored. Thus, due to the heterogeneity of distributed information sources, the use of ontologies become essential to support the semantic interoperability; as well as, the availability of automatic techniques for quickly determining semantic similarity among concepts to describe queries and information sources to be queried.

In addition, the new Web applications, as described by T.B. Lee et al. in [2], aim at guaranteeing the almost completely automatic execution of complex distributed processes, where autonomy, adaptability and cooperation are essential requirements. Agents technology [19] is an appealing approach to build automatic applications. Agents being an autonomous entity, able to react and adapt in a proactive way, in a dynamic execution context, can encapsulate the execution of several independent activities. The agent ability to cooperate with other agents allows to have a useful interaction within an

heterogeneous environment [31]. The integration of agents and ontologies as discussed in Hendler [17], provides a powerful approach to automate distributed computation, to support semantic interoperability and to allow meaningful agents interaction. Furthermore, an agent can move from one environment to another. In some specific domains as computational biology and bioinformatics, the quantity of data to be processed is often prohibitively large to be retrieved in an acceptable time, thus the possibility to move the computation is a promising approach. In an environment with multiple information systems, such those visited by mobile agents, different domain ontologies can coexist [16]. Although the use of single shared ontology would ensure the complete integration across information systems, it is quite impractical because it forces information systems to commit to this single ontology by making difficult the input of new concepts. Thus, a mobile agent has to face two problems: the ability to measure, on-the-fly, the *similarity* among concepts of different ontologies, (its own, e.g. its knowledge base and those used in the visited sites, e.g. conceptual schema of local data repository), and the ability to enrich his own knowledge with new concepts.

In this work, we propose integrating mobile agents with suitable tools for managing ontologies during their migration across distributed heterogeneous information systems. To that purpose, we have defined an abstract data model, the *ontological graph*, derived from the graph-based conceptual model proposed in Mitra et al. [24], and we have defined a minimum set of operators essential to manage *ontological graphs* to determine the similarity. We define three algebraic operators, to isolate a concept in an ontology (*projection*), to measure similarity between two concepts (*similarity*) and to enrich an ontology with a new concept (*enriching*).

The proposed approach shows some advantages: any domain ontology, being represented by RDF or OWL or DAML+OIL, can be mapped into the *ontological graph*; every information system can use a local conceptualization of the domain without to commit to a single global one; every agent can enrich its own knowledge by generating a collection of synonyms; and it can choose the most suitable similarity function [22], [27], [10], [11], to relate domain-specific ontologies. Last but not least, the algebraic operators can be considered the ground for designing a declarative language to specify the agent behaviour.

The three operators has been implemented in Hermes, mid-

middleware for mobile computing, to support the description of the mobile agents behaviour in a distributed environment. In particular, the similarity operator has been implemented over three algorithms: the semantic similarity algorithm proposed by Maedche et al in [22], that proposed by Rodriguez and Egenhofer in [27] and over a new algorithm based on structural similarity, proposed in this work as an extension of our previous work [7].

The paper is structured as follows: Section II describes a motivating scenario with an example in Bioinformatics. Section III defines the *ontological graph* and the algebraic operators. Section IV proposes a new similarity function. Section V, discusses the behaviour of a mobile agent by an example in Bioinformatics. Finally, Section VI remarks conclusions and future work.

II. MOTIVATING SCENARIO

Nowadays, the widespread interconnection of distributed systems, with the global distribution of information sources and computational tools, offers a scenario where to build distributed applications in every domain of social and life science (Medicine, BioMedicine, Computational and Systems Biology, Health Care ...). In a wide view, we can think to design a scenario where a user (human or application) describes his goal (e.g complex queries, workflows of activities) by using a vocabulary of terms and relations close as much as possible to his application domain. He will not worry about where information are stored, what data formats have been used, how tools can be integrated and services coordinated. The achievement of the user goal is delegated to one or more software entities or agents that are responsible for a correct achievement of the user goal.

In particular, in Bioinformatics, a biologist would be able to specify his experiment like a workflow of activities, ranging from researching and integrating information, to coordinating computational tools executed over specific data. In this domain, where the amount of interrelated information exponentially increases, it is very difficult for a human to exploit all available data, to identify, select, clean and use all relevant data, also because of different data formats with different semantics.

If on the one hand the integration of heterogeneous data can be achieved in different ways, on the other to manage all suitable data in an acceptable time asks for remote computation. In fact, data can be extracted and integrated either in a unique datawarehouse to which users can submit a query using a global schema, or instantaneously in native data sources. In the first approach data are centralized, there is no instant schema translation, but it is difficult to add new data repositories, to maintain data updated and to modify any schema. In the second approach, data remain in native repositories where they are constantly updated and free to be represented in any format, but an instant (on-the-fly) schema mapping must be done by the data collector; in fact different data sources may use different names and formats to refer the same object, or the same name to refer objects with different meaning.

The latter is a typical scenario where a mobile agent works on; therefore, integrating mobile agents with suitable tools for managing ontologies would enrich its capabilities. It is worth pointing out that an act of communication between two agents is feasible only if a common ontology is shared. Even if this restriction guarantees an agreement on the semantics of exchanged data, not all data are a priori shared, thus agents must be able to reach an on-the-fly agreement by measuring semantic similarity of different ontologies. Whenever an agent acquires new information, it can enrich its personal knowledge base.

A. Running Example

Suppose a biologist has prepared his experiment within an interactive virtual laboratory for Bioinformatics. The experiment consists of the set of concurrent and coordinated activities each of which is described by using the specific terminology taken from an ontology of the biological domain. The execution of the experiment is delegated to the runtime support of the virtual laboratory. Suppose to have a computational environment based on middleware for mobile computation, where every experiment is compiled in a pool of mobile agents activated to support the execution of the whole experiment. Also, suppose that one of the agents is involved in the execution of a query, which implies its migration across several places to query different data repositories. Let

“Find all Complementary DNA transcribed from Messenger RNA whose DNA is”

be a meaningful query for a biologist. We can observe that *“Complementary DNA”*, *“Messenger RNA”* and *“DNA”* are terms that identifies domain specific concepts while *“Transcribed from”* is a relation between two concepts.

As the mobile agent reaches a destination, it will interact with a local stationary service agent, passing on to it the query and the reference of the domain ontology. If the service agent shared the domain ontology it will just translate the query in the local format. If not, it will offer its local ontology to the mobile agent, which in turn will decide whether to come to an agreement or to move to next place. The final decision could be taken over the result given by the similarity function measured between the two ontologies. Once the mobile agent has decided which are the most similar concepts to those describing its query, it will rewrite the query in terms of new concepts, and submit the query to the local service agent. Afterward, the service agent will be able to convert the incoming query to a corresponding local one. In some cases, the mobile agent, could decide to enrich his knowledge with new learned concepts, which could be also used by the biologist to interpret the extracted data. Then, the mobile agent moves to the next place.

What are suitable tools to support the interaction of mobile agents with local service agent? To provide an answer, we define the *ontological graph* an abstract data model more flexible and light than an ontology; suitable to map every ontology a mobile agent will manage and analyze during its migration. In the next section, before the definition of

ontological graph the general concept of ontology will be introduced.

III. ONTOLOGICAL GRAPH MODEL

Gruber in [13] defines an ontology as an explicit specification of a conceptualization. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. We can note that every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. Usually, agents share a common specification (common ontology) which supports the communication with each other and they commit “on-the-fly” specifications (local ontology) to operate on a specific domain. Guarino ([14]) underlines that interoperating systems need two types of ontologies: a top-level and a domain-level ontology. The top-level ontology describes very general common concepts (e.g primitives of a communication protocol [9] or concepts as time, space, event, etc.) which are independent of a particular problem or domain, by which to verify the consensus on sharing a generic domain ontology. The domain ontology describes the vocabulary related to a specific domain (like Biomedicine, Molecular Biology, etc.) or a specific task or activity (like Protein, Enzyme, etc.) by specializing the terms introduced in a top-level ontology. The evolution of ontology research in computer science recently shifted from theoretical to practical issues. Noy and Klein report in [26] that issues like what a formal ontology is - what requirements an ontology must satisfy - what representation language is suitable to define and exchange ontologies, shifted to issues associated with the use of ontologies in real-world, for large-scale applications, like how to use multiple distributed heterogeneous ontologies - how to maintain updated an ontology - how to integrate similar ontologies, etc.

An ontology, denoted by O , is a formal specification of a conceptualization, that is the knowledge structure that describes, using a lexicon, the semantics of a given domain. A lexicon is defined in [28], as a “knowledge-base about some subset of words in the vocabulary of a natural language denoting concepts of the domain and relations among concepts”.

Notations

For ease of notation and retention of all definitions, in the sequel of this paper, we will use the following notation.

Let:

\mathcal{L} be the set of lexicons, ranged over by $L_1, L_2 \dots L, L', L''$;

\mathcal{O} be a set of ontologies, ranged over by $O_1, O_2, \dots O, O', O''$;

C be a set of ontological concepts, ranged over by $c_1, c_2, \dots c, c', c''$;

R be a set of ontological relations, ranged over by $r_1, r_2, \dots r, r', r''$;

$N, N_1, N_2 \dots H, K$ be a set of nodes, ranged over by $n_1, n_2, \dots n, m$;

$A, A_1, A_2 \dots$ be a set of arcs, ranged over by a, a_1, a_2, \dots

Since, a lexicon contains terms to identify both concepts and relations which semantically describe the domain, in the sequel of this paper, we will separately use concepts and relations as terms of a given lexicon. Therefore, a lexicon L can be represented as a finite, not empty set C of concepts and a not empty set R of relations among concepts. L can be represented as $L = \langle C, R \rangle$. We assume that R contains a special relation *similar* which will be used to denote a similarity relation between concepts.

Formally, an ontology O is a, node and arc labelled, graph, where the labelling functions are expressed over a lexicon $L \in \mathcal{L}$; the set of graph nodes represents concepts and the set of arcs represents relations between concepts. The association between a node and a concept, so as between an arc and a relation, is unique. Any concept can be described by its lexical name and the set of relations it has with other concepts. A concept is represented by a rooted subgraph.

An *ontological graph* is formalized by the following definition:

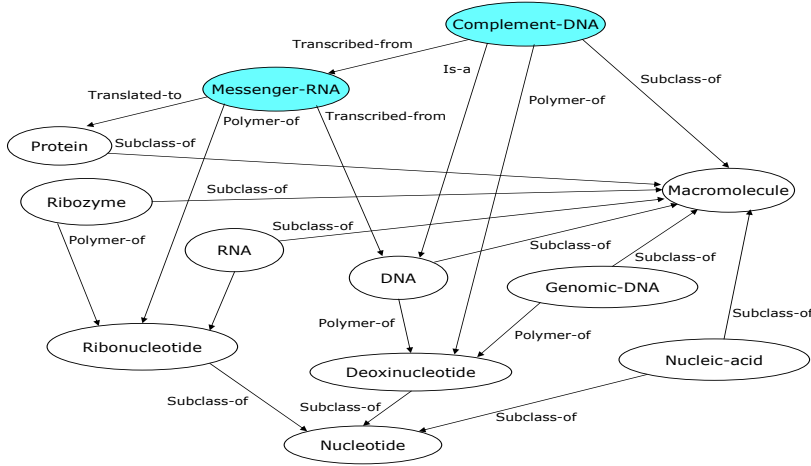
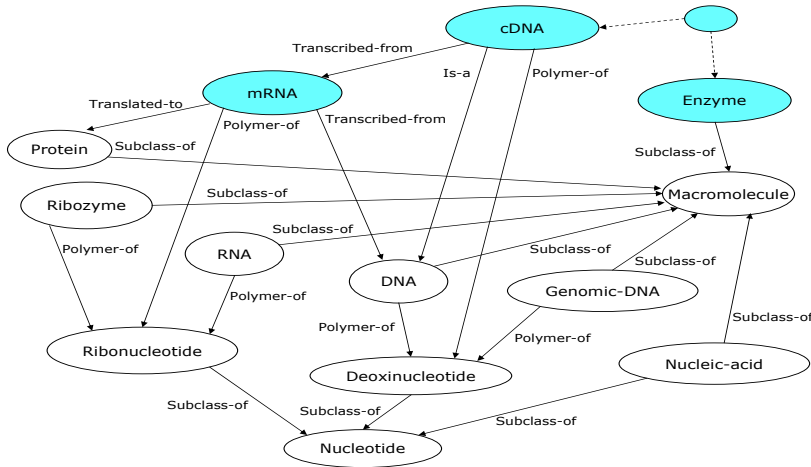
Definition 3.1 (Ontological Graph): An *ontological graph* $O = (N, A, n)$ is a directed, rooted, node and arc labelled over a lexicon $L = \langle C, R \rangle$, graph. Where N is the finite set of ontology concepts, A is the finite set of relations among concepts and n is the root. The node labelling function, $\lambda : N \rightarrow C$ uniquely associates a node to a concept in the lexicon. The arc labelling function $\delta : A \rightarrow R$ uniquely associates an arc to a relation in the lexicon.

The functions, λ and δ , are neither injective nor surjective mapping function; this property allows the existence of concepts and relations in the ontology, that are not expressed in the *ontological graph*.

Each ontology $O \in \mathcal{O}$ is associated to the corresponding *ontological graph* O . Each concept $c \in C$ is associated to the corresponding subgraph. Each subgraph is, in turn, an *ontological graph*. Each node n of an *ontological graph* is associated to the name of the concept described by the *ontological graph* rooted on the node n .

In the sequel of this paper, both ontology and *ontological graph*, so as concept name and node, concept and *ontological graph*, will be used interchangeable.

Figure 1 shows the *ontological graph* corresponding to a small set of concepts in Bioinformatics, whose lexicon is $L_1 = \{DNA, RNA, Ribozyme, Nucleotide, Ribonucleotide, Deoxynucleotide, Nucleic-acid, Protein, Macromolecule, Complementary-DNA, Messenger-RNA, Polymer-of, Subclass-of, Transcribe-from, Translate-to\}$. In the figure, the concept of *Ribozyme* is described in the subgraph rooted at the node labelled by *Ribozyme*. Thus, the *Ribozyme* is *Subclass-of* a *Macromolecule* and a *Polymer-of* the *Ribonucleodite*; this latter, in turn is a *Subclass-of* *Nucleotide*. A *Nucleotide*, in this conceptualization, is a primitive concept of the domain, a leaf node of the graph. The Molecular Biology ontology used to derived the *ontological graph* has been taken from TAMBIS

Fig. 1. An ontological graph for the lexicon L_1 Fig. 2. ontological graph for the lexicon L_2

project [12] in OIL [18], [29] description.

In Figure 2, we consider an *ontological graph* corresponding to a different lexicon $L_2 = \{DNA, RNA, Ribozyme, Nucleotide, Ribonucleotide, Deoxynucleotide, Nucleic-acid, Protein, Macromolecule, cDNA, mRNA, Enzyme; Polymer-of, Is-a, Subclass-of, Transcribe-from, Translate-to\}$. Figure 2 shows that the *Ribozyme* concept is described by the same subgraph of Figure 1. Whereas, the concept of *Complementary-DNA* from the lexicon L_1 and *cDNA* from the lexicon L_2 are described by two different subgraphs whose degree of similarity will be later discussed.

A. Algebraic operators

To allow the manipulation of ontologies by *ontological graph*, we concentrate on a minimum set of operators necessary to measure on-the-fly the similarity among concepts of different ontologies. The three main operators are: *projection*, *similarity* and *enriching* (see Table I).

$$\text{projection} \quad \pi : \mathcal{O} \times N \rightarrow \mathcal{O}$$

$$\text{similarity} \quad \sigma : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$$

$$\text{enriching} \quad \epsilon : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{O}$$

TABLE I
ALGEBRAIC OPERATORS

The *projection* allows to reduce the *ontological graph* into a subgraph whose root node corresponds to a given concept. The *similarity* operator is a function which measures the similarity of two concepts and returns a coefficient that ranges over by $[0, 1]$; the coefficient is 1, if the two concepts are equal; it is

0 if they completely mismatch. The *enriching* operator allows to enrich the *ontological graph* with new concepts.

The small set of operators could be easily extended with other operators, for example those proposed by Mitra et al. in [25] for ontology composition: *Select*, *Intersection*, *Union* and *Difference*;

In the following, the description of behaviour of the proposed operators is given. We have omitted the formal definitions which can be found in Appendix I.

1) *Projection* π : The projection of an *ontological graph* over a given concept, reduces the *ontological graph* by isolating the subgraph consisting of those concepts and relations – except for the relation *similar* – that describe the given concept. The root node of the projected graph is represented by the concept itself.

The *projection* is a binary operator defined over an *ontological graph* and a concept name.

$$\pi : \mathcal{O} \times N \rightarrow \mathcal{O}$$

Given an *ontological graph* $O=(N,A,n)$ and a node $m \in N$, the *projection* of O over m returns the subgraph O' , rooted in m corresponding to the concept associated to m . Suppose we wish to isolate the concept *Ribozyme* in the *ontological graph* O given in Figure 1, the *projection* operator can be used as follows

$$\pi(O, \text{Ribozyme}) = O'$$

Figure 3 shows the graphical behaviour of the *projection* operator over the example.

2) *Similarity* σ : The measure of the similarity between two concepts determines how much the two corresponding *ontological graphs* are similar. There are several ways to measure the similarity among two concepts. Giunchiglia in [11] proposes to classify the process of discovering the graphs mapping in syntactic and semantic matching. The syntactic similarity (matching) [21] is based on searching the semantic correspondence among node labels, the resulting coefficient, that ranges over $[0,1]$, measures the similarity between the labels of the given nodes by performing linguistic analysis. The semantic similarity is based on analyzing the position that a node has in a graph, that can be done either analysis the position of a given node in terms of neighbours nodes [11], or by following a path in the graph [3] or by analyzing both the semantic and the syntactic concepts matching, as Maedche et al. propose in [22]. In the above cases, the similarity algorithm returns a coefficient that ranges over $[0,1]$, except for the Giunchiglia algorithm that returns a set of values that range over $\{=, \subseteq, \supseteq, \perp\}$ (equality, more specific, more general and mismatch respectively). In Section IV, we propose a new approach which determines semantic similarity by clustering concepts satisfying common relations. The algorithm allows to measure on-the fly semantic similarity without sharing a domain ontology.

The *similarity* operator allows to measure the similarity between two concepts of different *ontological graphs*. The

similarity returns a coefficient that ranges over $[0,1]$. How the similarity is measured, it depends on the algorithm chosen to implement the operator. In any case, two concepts are equal if the *similarity* returns 1, two concepts mismatch (no affinity) if the *similarity* returns 0.

The *similarity* is a function defined over two *ontological graphs*.

$$\sigma : \mathcal{O} \times \mathcal{O} \rightarrow [0,1]$$

Given two *ontological graphs* $O=(N_1, A_1, n)$ and $O'=(N_2, A_2, m)$ respectively, the *similarity* of O and O' over the two root nodes n and m returns a real number $\alpha \in [0,1]$ that quantitatively estimates the similarity degree of the concepts described by the *ontological graphs*.

Suppose we wish to measure the similarity between the concept *Ribozyme* in the *ontological graph* $O=(N_1, A_1, \text{Ribozyme})$ given in Figure 1, and *Ribozyme* in the *ontological graph* $O'=(N_2, A_2, \text{Ribozyme})$ given in Figure 2, the *similarity* operator most likely will return the value 1.

$$\sigma(O, O') = 1$$

If we measure the similarity between *Complementary-DNA* and *cDNA* described in the two graphs respectively, the *similarity* operator will return the value α depending on the algorithm that implements the operator.

$$\sigma(\text{Complementary-DNA}, \text{cDNA}) = \alpha$$

3) *Enrichment* ϵ : When an agent discovers a new concept, he can decide to enrich his knowledge by storing the new knowledge. This can be done in several ways, by creating a new data structure or by adding the projection of the new concept to its *ontological graph*. The *enriching* is an operator defined over two concepts, i.e. two *ontological graphs*:

$$\epsilon : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{O}$$

Given two *ontological graphs* $O=(N_1, A_1, n)$ and $O'=(N_2, A_2, m)$, representing two similar concepts, the enrichment of O with O' is obtained by adding a new arc from the root n of O to the root m of O' labelled by “*similar*”; *similar* is a special relation meaningful only for the agent purpose. Suppose we wish to store the knowledge that *cDNA* is similar to *Complementary-DNA*, we can use the *enriching* operator between $O=(N_1, A_1, \text{Complementary-DNA})$, $O'=(N_2, A_2, \text{cDNA})$ as follows

$$\epsilon(O, O') = \tilde{O}$$

The resulting graph \tilde{O} is depicted in Figure 4.

IV. A STRUCTURAL SEMANTIC SIMILARITY FUNCTION

In this section, a new function to assess the semantic similarity between concepts is proposed. We only compare the structural (topological) similarity among sets of concepts without considering syntactic matching between node labels.

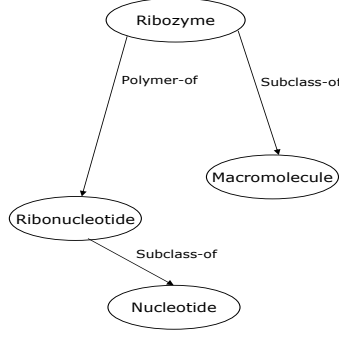


Fig. 3. The projected *ontological graph* in Figure 1 over the node “Ribozyme”

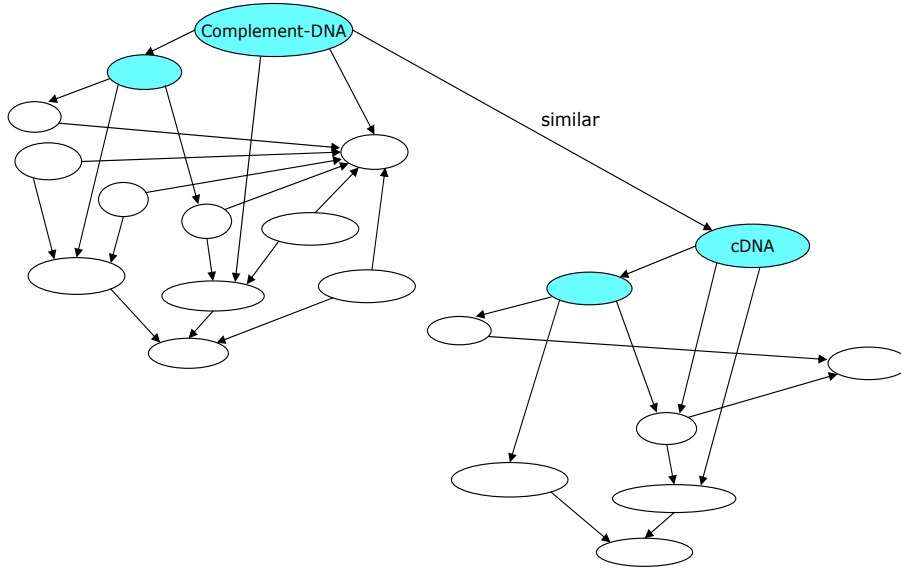


Fig. 4. Enriching operator over the running example

Given two *ontological graph* $O_1=(N_1, A_1, n)$, $O_2=(N_2, A_2, m)$ and two sets of nodes $H \subseteq N_1$ and $K \subseteq N_2$, with $H \neq \emptyset \neq K$, the similarity between H and K is measured by the function in Table II, where, for any given set of nodes N , $\tau(N)$ is the number of outgoing arcs from (nodes in) N , R_N is the set of relations – different from *similar* – associated to the arcs outgoing from N , and $\text{sons}_r(N)$ is the set of nodes reachable from any node in N through the relation $r \in R$.

Intuitively, two set of concepts H and K are equal (mismatch) if both of them (one but not the other) contain only primitive concepts, that is, have no outgoing arcs. Otherwise, if both H and K contain no primitive concepts (the number of outgoing arcs from H and K is greater than zero), we consider the set $R_H \cap R_K$ of the relations they have in common. For

each $r \in R_H \cap R_K$, we recursively apply f to the sets of nodes reachable from H and K through the relation r .

It is worth noting, that the proposed similarity function is relevant when the relation are meaningful for the application domain.

V. ON-THE-FLY CONCEPTS COMPARISON ACROSS ONTOLOGIES

As we mention in the introduction, in the context of multiple information systems, the semantic interoperability must allow users to enter a request without knowing where and how data are stored.

We have implemented the biological example described in Section II, in the framework of Hermes, middleware for mobile computation proposed by Corradini et al. in [5], [6].

$$f(H, K) = \begin{cases} 1 & \text{iff } \tau(H) = \tau(K) = 0 \\ 0 & \text{iff } \tau(H) = 0 \text{ xor } \tau(K) = 0 \\ \frac{\sum_{r \in R_H \cap R_K} f(\text{sons}_r(H), \text{sons}_r(K))}{\text{card}(R_H \cup R_K)} & \text{otherwise} \end{cases}$$

TABLE II
THE SIMILARITY FUNCTION

As Figure 5 shows, Hermes is structured as a component-based system with 3-layer software architecture: *user layer*, *system layer* and *run-time layer*. At the user layer, it allows users to specify their application as a workflow of activities using the graphical notation provided by DroFlo (OpenWFE, 2005) and JaWE editor (Enhydra, 2003). At the system layer, it provides a context-aware compiler to generate a pool of user mobile agents from the workflow specification. At the run-time layer, it supports the activation of a set of specialized service agents, and it provides all necessary components to support agent mobility and communication. Hermes can be configured for specific application domains by adding domain-specific components.

Biological workflow	User Layer
Workflow Management	
Bio-agents	System Layer
Context-aware compiler	
Bio-Service Agent	Run-Time Layer
Core	

Fig. 5. Hermes Software Architecture

Suppose to have Hermes as middleware that allows the interoperability among different information systems and suppose to have a service agent, *ontology service agent* which interfaces local repositories and interacts with mobile agents to allow semantic querying. Let us consider the running example described in Section II, and suppose that the definition of a workflow in the biological domain consists of a single task: retrieval of information about *Complement-DNA*. Also assume that the ontological graphs in Figures 1 and 2 (denoted in the following by O_1 and O_2) are used to describe the query and the remote data schema repositories, respectively.

In the Hermes context, a mobile agent (called bio-agent) is created, by using L_1 related to O_1 on platform 1, to support the execution of the workflow activity. The bio-agent has the goal to move to platform 2 and to search the concept *Complement-DNA*. The bio-agent starts to extract the subgraph of *Complement-DNA* from the derived *ontological*

graph O_1 (Figure 1) and moves to platform 2. There, it will interact with a local stationary service agent, passing on to it the query and the reference of the domain ontology. If the service agent shared the domain ontology it will just translate the query in the local format. If not, it will offer its local ontology to the bio-agent. The bio-agent will search for the most similar *Complement-DNA* concept on O_2 . Note that nodes and arcs of the subgraphs are indicated by corresponding labels in the lexicon. The *Complement-DNA* concept is compared to all the subgraphs, thus the agent will compute $f(\text{Complement-DNA}, \text{mRNA}) = 0.3333^1$ and $f(\text{Complement-DNA}, \text{Enzyme}) = 0.333$. To evaluate $f(\text{Complement-DNA}, \text{cDNA})$, the agent needs to evaluate the following values: $f(\text{Messenger-RNA}, \text{mRNA}) = \frac{1+1+1}{4} = \frac{3}{4} = 0.75$ and $f(\text{Deoxynucleotide}, \text{Deoxynucleotide}) = 1$.

The agent can now complete the comparison between *Complement-DNA* and *cDNA*, computing $f(\text{Complement-DNA}, \text{cDNA}) = \frac{0.75+1}{4} = \frac{1.75}{4} = 0.4375$, which yields *cDNA* as the concept most similar to *Complement-DNA*, with a similarity degree of 0.4375.

Having obtained the degree of similarity between *Complement-DNA* on platform 1 and *cDNA* on platform 2, the mobile agent can ask service agent to extract information (projection) about *cDNA* from platform 2, having learned that *cDNA* is “sufficient” similar to *Complement-DNA*, it will enrich its knowledge. Then, it will rewrite the query and proceed in the task execution.

A. Related Work

A general approach to data integration has been to map the local terms of distinct ontologies onto a single shared ontology, as described in [27]. In this work the semantic similarity is typically determined as a function of the path distance between terms in the hierarchical structure underlining this ontology [4]. Another strategies for ontology integration are based on the mapping of a local ontology onto a more generic ontology [1], [30]. The ONIONS methodology [1] integrates local ontologies by inheriting from shared generic ontologies, but does not automatically compare concepts (as proposed herein). The OBSERVER [23] ontology-based system combines intensional and extensional analysis to calculate lower and upper bounds for the precision and recall of queries that are translated across ontologies.

¹Given two nodes n and m , we write $f(n, m)$ to denote $f(\{n\}, \{m\})$

Weinstein et al. [30] propose differentiated ontologies as support to communications in distributed systems subject to semantic heterogeneity. Ontologies are described by using Description Logic. Concepts are formally defined in relation to other concepts, so that concepts in local ontologies inherit definitional structure from concepts in shared ontologies.

Recently, Rodriguez in [27], has suggested that in the area of information retrieval and data integration, the use of ontologies and semantic similarity functions have recently been emphasized as a mechanism for comparing objects that can be retrieved or integrated across heterogeneous repositories [16]. The authors proposed a model for semantic similarity among Entity Classes from different ontologies. Ontologies are described as objects by using BNF. The similarity model provides a systematic way to detect similar entity classes across ontologies based on the matching process of each of the specification components in the entity class representation (i.e., synonym sets, distinguishing features, and semantic neighborhoods).

The approach presented in this paper measures the structural similarity by only considering the relations among concepts.

VI. CONCLUSIONS AND FUTURE WORK

The present work aims at integrating ontologies in mobile agents. This approach allows the information retrieval and data integration in a scenario where a pool of mobile agents can migrate across different data repositories where updated information can be instantaneously retrieved.

The integration ontologies and mobile agents allows to discover new knowledge by combining information extracted from different data repositories and to move computational tools over data, by delegating a software entity. This approach supports the decentralization of the execution of local activities, to avoid the warehousing of highly dynamic data, to reduce network traffic and to free the users from network faults and from the need to be continuously connected to a laptop.

Future work will be geared towards reducing complexity through hypergraphs in place of graphs, generalizing the similarity function to cyclic graphs, exploiting the similar relation in the definition of similarity function and validating the proposed approach on a real application.

Acknowledgements

This research was partially supported by the MIUR strategic project Oncology over Internet and FIRB project LITBIO.

REFERENCES

- [1] A. Gangemi, M. Pisanelli, and G. Steve. Some requirements and experiences in integrating terminological ontologies in medicine. In *Eleventh Workshop on Knowledge Acquisition, Modelling and Management*, 1998.
- [2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.
- [3] P. Bouquet, G. Kuper, M. Scoz, and S. Zonobini. Asking and answering semantic queries. In *Proceedings of Meaning Coordination and Negotiation Workshop (MCNW-04) in conjunction with International Semantic Web Conference (ISWC-04)*, 2004.
- [4] M. Bright, A. Hurson, and S. Pakzad. Automated resolution of semantic heterogeneity. *ACM Transaction on Distributed Systems*, 19(2):212–253, 1997.
- [5] F. Corradini, L. Mariani, and E. Merelli. An agent-based approach to tool integration. *Journal of Software Tools Technology Transfer*, 6:231–244, 2004.
- [6] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Tutorial Book of SFM-05*. Springer-Verlag, 2005. LNCS 3465.
- [7] R. Culmone, Gloria Rossi, and Emanuela Merelli. An ontology similarity algorithm for bioagent. In *NETTAB Workshop on Agents in Bioinformatics*, July 12–14 2002.
- [8] D. Fensel. *Ontologies: a silver bullet for Knowledge Management and Electronic Commerce*. Springer, 2001.
- [9] The Foundation for Intelligent Physical Agents. <http://www.fipa.org/>.
- [10] Gustavo A. Gimnez-Lugo, Analía Amandi, Jaime Simo-Sichman, and Daniela Godoy. Enriching information agents' knowledge by ontology comparison: A case study. In M. Toro F.J. Garijo, J.C. Riquelme, editor, *Advances in Artificial Intelligence - IBERAMIA 2002: 8th Ibero-American Conference on AI, Seville, Spain, November 12–15, 2002. Proceedings*, volume 2527 of *Lecture Notes in Artificial Intelligence*, pages 546–555. Springer-Verlag, 2003.
- [11] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.
- [12] C. A. Goble, R. Stevens, G. Ng, S. Bechhofer, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent access to multiple bioinformatics information sources. *IBM Systems Journal*, 40(2):532–551, 2001.
- [13] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report 93-04, Knowledge Systems Laboratory, Stanford University, 1993.
- [14] N. Guarino. Formal ontology and information systems. In *Proceedings of FOIS - Formal Ontology in Information Systems*, 1998.
- [15] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases*. IOS Press, 1995.
- [16] N. Guarino, C. Masolo, and G. Verete. Ontoseek: Content-based access to the web. *IEEE Tran. on Information Systems*, 14(3):70–80, 1999.
- [17] J. Hendler. Agents and the semantic Web. *IEEE Intelligent Systems*, 16(1):30–37, January/ February 2001.
- [18] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. The ontology inference layer OIL. Technical report, Free University of Amsterdam, 2000.
- [19] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
- [20] Yannis Kalfoglou, Marco Schorlemmer, Michael Uschold, Amit Sheth, and Steffen Staab. 04391 – semantic interoperability and integration: Executive summary. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005. <<http://drops.dagstuhl.de/opus/volltexte/2005/50>> [date of citation: 2005-01-01].
- [21] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [22] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW2002)*, pages 251–263, Madrid, Spain, October 1–4 2002.
- [23] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. Observer: an approach for query processing in global information system based on interoperation across preexisting ontologies. *Distributed and parallel Databases*, 8(2):223–271, 2000.
- [24] P. Mitra, G. Wiederhold, and M. L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Extending Database Technology*, pages 86–100, 2000.
- [25] Prasenjit Mitra and Gio Wiederhold. An algebra for semantic interoperability of information sources. In IEEE, editor, *2nd IEEE International Symposium on Bioinformatics and Bioengineering (BIBE'01)*, 2001.
- [26] F. N. Noy and M. Klein. Ontology evolution: not the same as schema evolution. *Knowledge and Information Systems*, (6):428–440, march 2004.

- [27] A. Rodriguez and M. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Tran. on Knowledge and Data Eng.*, 16(2):442–456, 2003.
- [28] J.F Sowa. *Knowledge representation, logical, philosophical and Computational Foundations*. Brooks/Cole, 2000.
- [29] R. Stevens. An ontology of molecular biology and the questions that can be performed on the resources containing data about molecular biology. <http://img.cs.man.ac.uk/stevens/tambis-oil.html>.
- [30] P. C. Weinstein and W. P. Birmingham. Comparing concepts in differentiated ontologies. In *Workshop on Knowledge Acquisition, Modeling and Management (KAW)*, 1999.
- [31] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures and languages: a survey. In Michael Wooldridge and Nicholas R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Computer Science, pages 1–22. Springer-Verlag, Berlin, 1995.

APPENDIX I

FORMAL DEFINITION OF ALGEBRAIC OPERATORS

In this appendix we provide the formal definition of the three operators performing the minimal set of operators that a mobile agent can use across platforms for determining ontology mapping. The three operators introduced in Section III-A are: Projection, Similarity and Enriching.

1) *Projection* π : The *projection* is a binary operator defined over an *ontological graph* and a concept.

$$\pi : \mathcal{O} \times N \rightarrow \mathcal{O}$$

Definition 1.1 (Projection): Given an *ontological graph* $O=(N, A, n)$ and a concept name $m \in N$ the projection of O on m is defined as:

$$\pi(O, m) = \bar{O}$$

where $\bar{O} = (N_1, A_1, m)$ is the subgraph of O such that:

- 1) N_1 is the set of nodes $n_j \in N$ such that either $n_j = m$ or $\exists n_0, n_1, \dots, n_j \in N$, with $j \geq 1$, such that $n_0 = m$ and $(n_0, n_1), \dots, (n_{j-1}, n_j) \in A$
- 2) $A_1 = \{(n_1, n_2) \in A \mid n_1, n_2 \in N_1\}$

Properties of the projection

- $\pi(O, null) = (\emptyset, \emptyset, null)$
- $\pi(O, n) = (\emptyset, \emptyset, null)$ se $n \notin N$
- $\pi(\emptyset, n) = (\emptyset, \emptyset, null)$

2) *Similarity* σ : The *similarity* is a function defined over two *ontological graphs*. Given two *ontological graphs* O, O' and two node n, m , the *similarity* of O and O' over n and m returns a real number $\alpha \in [0, 1]$; α , quantitatively estimates the similarity degree of the two concepts.

Definition 1.2 (Similarity): Let $O = (N, A, n)$ and $O' = (N', A', n')$ be two *ontological graphs*. The *similarity* of O and O' is defined as:

$$\sigma(O, O') = \alpha$$

where $\alpha \in [0, 1]$.

The similarity operator can be implemented with one of the algorithms available in literature [7], [22], [27] or the one introduced in Section IV, in that case the similarity will be as follows

$$\sigma(O, O') = f(n, n')$$

Properties of the similarity

- $\sigma(O, O) = 1$
- $\sigma(O, \emptyset) = \sigma(\emptyset, O) = \sigma(\emptyset, \emptyset) = 0$
- $\sigma(O, O') = \sigma(O', O)$

Several similarity functions can be defined over two ontologies, as long as, they return a real positive number that ranges over by $[0, 1]$; 0 means no affinity and 1 overlapping (equivalence) of the two ontological concepts.

As an example, the similarity function has been discussed in details in Section IV.

3) *Enrichment* ϵ : The *enriching* is a binary operator defined over *ontological graphs*. Given two *ontological graphs* O, O' the *enriching* of O with O' returns O enriched by adding a new arc from the root of O to the root of O' labelled by *similar*.

Definition 1.3 (Enrichment): Let $O=(N, A, n)$, $O'=(N', A', n')$ be two *ontological graphs*. The enrichment of O over O' is defined as:

$$\epsilon(O, O') = \bar{O}$$

where $\bar{O}=(\{N \cup N'\}, \{A \cup A' \cup \{(n, n')\}, n\})$ and $\delta(n, n') = \text{similar}$.

Un'Infrastruttura per la Mobilità in AgentService

A. Grosso, A. Boccalatte, C. Vecchiola

Abstract—L'articolo presenta la soluzione alle problematiche legate alla mobilità degli agenti adottata nella piattaforma AgentService. L'infrastruttura per il trasferimento degli agenti sfrutta il modello di agente della piattaforma che prevede la separazione tra lo stato e le attività dell'agente. L'implementazione dell'infrastruttura per la mobilità si avvantaggia della struttura modulare della piattaforma e si integra in modo del tutto trasparente per gli agenti e gli altri componenti. AgentService offre un servizio di mobilità debole, nonostante ciò garantisce il mantenimento dello stato degli agenti tramite il trasferimento delle strutture dati, della coda dei messaggi, delle conversazioni e dello stato dei comportamenti. Il servizio di mobilità degli agenti è inoltre sfruttato per l'applicazione di politiche di bilanciamento di carico tra piattaforme federate

Index Terms—Agent Mobility, Load Balancing Policy, Agent Framework

I. INTRODUZIONE

LE metodologie, le architetture e le tecnologie che vengono utilizzate per lo sviluppo di applicazioni distribuite manifestano i propri limiti quando sono applicate a sistemi distribuiti di notevoli dimensioni, potenzialmente illimitati, come si verifica per applicazioni Internet. In particolare ciò è ancor più evidente quando si ha a che fare con sistemi che debbano offrire un elevato grado di configurabilità, alta scalabilità e facilità di personalizzazione. Un metodo efficiente per la soluzione di problemi di questo tipo ci viene offerto dalle tecnologie basate sulla mobilità del codice, in altre parole la possibilità di spostare codice attraverso i nodi di una rete [1].

La mobilità del codice non è un concetto nuovo, gli *Applet Java* ne sono un lampante esempio, ma il lavoro di ricerca in questo campo è sempre attivo, sia per quanto riguarda le disquisizioni concettuali sia per ciò che riguarda gli aspetti puramente tecnologici, si veda ad esempio [2], [3].

Un ambito a cui i concetti di mobilità ben si adattano è quello degli agenti software. Gli agenti ci forniscono, in effetti, un'astrazione tale da rendere semplice l'applicazione del concetto di mobilità. Le caratteristiche intrinseche degli

agenti quali l'autonomia, l'inserimento in un ambiente, la proattività e la cooperazione, consentono di introdurre con naturalezza il concetto di agente mobile.

L'idea di agente mobile, vale a dire la possibilità di trasferirne il codice e lo stato tra nodi di una rete, porta a diverse considerazioni: la prima è che deve esistere una struttura che consenta all'agente di spostarsi; la seconda è che l'agente dovrebbe avere una certa intelligenza, tale da renderlo particolarmente autonomo nel decidere sugli spostamenti. Una volta realizzata un'infrastruttura adatta e una volta che all'agente saranno forniti tutti gli strumenti perché se ne avvalga, si potranno ottenere svariati vantaggi. Grazie agli agenti mobili, gli amministratori delle piattaforme possono disporre di uno strumento più efficiente per il bilanciamento delle risorse. Dall'altro lato la mobilità sembra rafforzare l'autonomia dell'agente stesso, che ad esempio potrebbe decidere su quale computer migrare in base alla disponibilità delle risorse.

In questo articolo viene presentata l'infrastruttura di supporto alla mobilità per la piattaforma AgentService. La soluzione proposta è risultata particolarmente efficace grazie al modello di agente adottato dalla piattaforma; tale modello consente di gestire con facilità lo stato degli agenti e quindi di mantenerlo persistente e/o trasferirlo altrove. Inoltre lo sviluppo degli strumenti per la mobilità risulta facilitato dall'architettura modulare di AgentService, la quale permette di arricchire i servizi offerti agli agenti attraverso la realizzazione di moduli aggiuntivi; il tutto può avvenire in maniera trasparente e senza la necessità di sostanziali modifiche all'infrastruttura software. Infine la tecnologia offerta dalla Common Language Infrastructure (CLI) [4, 5], su cui AgentService si basa, fornisce gli strumenti necessari ad un'implementazione efficiente del servizio grazie ad alcune particolarità quali la serializzazione, la presenza di primitive di comunicazione remota, l'utilizzo di domini applicativi e la gestione delle unità di distribuzione del codice (Assembly).

Nelle sezioni successive descriveremo le caratteristiche principali della piattaforma AgentService e dei componenti che influiscono maggiormente sullo sviluppo di un'infrastruttura per la mobilità, quali il modello di agente ed alcuni dei moduli che implementano le funzionalità di base (Sezione II). Dopo una breve introduzione sui problemi relativi alle problematiche associate agli agenti mobili (Sezione III), descriveremo in dettaglio le caratteristiche architetturali dell'infrastruttura per la mobilità e la sua

Manuscript received October 4, 2005.

A. Grosso, A. Boccalatte, and C. Vecchiola are with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy (phone: +39-010-353-2812; e-mail: {agrosso, nino, christian}@dist.unige.it).

interazione con gli altri componenti della piattaforma AgentService (Sezione IV). Valuteremo perciò gli aspetti riguardanti le possibili applicazioni dell'infrastruttura quali l'attuazione di politiche di bilanciamento di carico tra più piattaforme (Sezione V). Alcune osservazioni finali sulle caratteristiche dell'architettura, i suoi punti di forza ed i suoi punti deboli, seguiranno nelle conclusioni.

II. ARCHITETTURA DI AGENTSERVICE

A. Caratteristiche Principali

AgentService [6] è un framework per lo sviluppo di applicazioni orientate agli agenti basato sulla Common Language Infrastructure, di cui un'implementazione è il .NET framework. AgentService offre un particolare modello di agente ed un ambiente di run-time per l'esecuzione degli agenti conforme alle specifiche FIPA [7]. In letteratura sono presenti numerosi lavori riguardanti piattaforme ad agenti, tra i più diffusi si veda Zeus [8], FIPA-OS [9] e JADE [10]. L'architettura della piattaforma è estremamente modulare: sia le funzionalità di base sia quelle aggiuntive sono implementate utilizzando l'astrazione del modulo. Questa soluzione architeturale rende la piattaforma ad agenti un ambiente di run-time molto flessibile e facilmente personalizzabile a particolari esigenze, nonché facilmente estendibile nelle sue funzionalità. I moduli si dividono in due classi: i moduli fondamentali e quelli addizionali. I moduli fondamentali implementano quei servizi necessari alla piattaforma per attivarsi; questi comprendono la gestione degli assembly in cui sono contenuti i tipi degli agenti (*Storage*), la gestione della messaggistica, la gestione della persistenza e le funzionalità di logging. Con i moduli addizionali vengono implementate tutte quelle funzionalità che arricchiscono di servizi la piattaforma, ma la cui assenza non è vincolante per l'attivazione della piattaforma stessa.

Allo scopo di rendere più fruibile la trattazione dell'implementazione della mobilità in AgentService descriveremo brevemente alcuni elementi fondamentali dell'architettura di AgentService: il modello di agente adottato ed i moduli per la gestione dello storage e del servizio di messaggistica.

B. Il Modello di Agente

AgentService modella un agente come un'entità software la cui base di conoscenza è definita da un insieme di dati chiamati *Knowledge* e le cui attività sono descritte da task concorrenti che prendono il nome di *Behavior*. L'insieme delle *knowledge* utilizzate da un agente ne definisce lo stato, mentre il suo comportamento è descritto dall'insieme dei *behavior* che sono in esecuzione. Una *knowledge* è di fatto simile ad un *record* del linguaggio Pascal od ad una *struct* del linguaggio C, sebbene possa essere caratterizzata anche da metodi è principalmente pensata per esporre delle proprietà (queste possono fare riferimento ai tipi base della CLI od a classi anche definibili dal programmatore). I *behavior* sono a

tutti gli effetti delle classi generiche definibili dall'utente e presentano un metodo particolare che costituisce l'*entry point* della loro esecuzione. I *behavior* di un agente possono condividere le *knowledge* che ne definiscono lo stato e l'ambiente di run-time garantisce l'accesso esclusivo alle *knowledge*, in modo relativamente trasparente.

Dal punto di vista implementativo ad ogni istanza di agente viene associato un differente *Application Domain* che garantisce l'esecuzione in maniera autonoma del codice in esso contenuto. L'*Application Domain* è una struttura nuova introdotta con la CLI ed è assimilabile ad un "processo leggero" in quanto ha un proprio spazio di memoria, è possibile creare in esso più thread di esecuzione ed associarvi differenti permessi di esecuzione, ma il suo *setup* è più leggero di quello di un processo. L'esecuzione di un agente all'interno di un *Application Domain* ne garantisce l'autonomia e l'isolamento dagli altri agenti: l'unico modo per mantenere dei riferimenti ad oggetti in *Application Domain* differenti è l'impiego di tecniche di comunicazione esplicite quali il *Remoting*.

C. Gestione dello Storage

Lo *Storage* della piattaforma costituisce un repository virtuale di tutte le classi necessarie al ciclo di vita degli agenti che vengono creati ed ospitati all'interno della piattaforma. In particolare nello *Storage* sono memorizzate le classi che definiscono i tipi di agente, le *knowledge* ed i *behavior* usati da tali agenti nonché i tipi logicamente dipendenti se non già compresi nella class library della CLI. I tipi eseguibili ed istanziabili dalla CLI sono fisicamente memorizzati in un file od in un insieme di file logicamente collegati che prende il nome di assembly. In ultima analisi lo *Storage* si occupa della gestione degli assembly che contengono i tipi di agenti e tutto ciò che serve per crearne delle istanze. La CLI per identificare in modo univoco gli assembly utilizza una tecnica di crittografia asimmetrica basato su chiave pubblica e privata normalmente opzionale ma richiesto da AgentService.

Il modulo dello *Storage* viene interrogato ogni volta che la piattaforma deve creare l'istanza di un particolare tipo di agente e restituisce tutti gli assembly necessari a creare l'istanza. Per poter caricare nello *Storage* un assembly occorre firmarlo, l'apposizione della firma ad un assembly permette di identificarlo in maniera univoca indipendentemente dal nome dei file fisici che lo costituiscono. Questo è un elemento che diventerà di fondamentale importanza quando verrà illustrata l'infrastruttura che permette la mobilità.

D. Gestione del Servizio di Messaggistica

Il servizio di messaggistica è gestito da un opportuno modulo che rende di fatto possibile la comunicazione tra agenti attraverso scambio di messaggi (*Messaging*). Il modulo di messaggistica mantiene una coda di messaggi per ogni agente che è ospitato nella piattaforma in cui è installato il modulo. All'atto della creazione di un agente il modulo fornisce un opportuno client per il servizio di messaggistica. Le specifiche di AgentService per l'implementazione del

modulo di messaggistica prevedono che sia garantito il semplice servizio di scambio messaggi, ma è prevista come funzionalità addizionale la possibilità da parte degli agenti di usufruire delle conversazioni che forniscono un servizio di comunicazione connesso tra due agenti. Il modulo fornito con l'installazione di default della piattaforma offre nativamente questo servizio. I messaggi che possono essere scambiati tra due agenti oltre ad aderire alle specifiche fornite da FIPA in tale ambito devono contenere oggetti serializzabili, requisito fondamentale dal momento che due agenti risiedono in Application Domain diversi.

III. NOZIONI DI MOBILITÀ

La mobilità è sicuramente una proprietà importante per gli agenti così come per gli oggetti; ciò è dovuto al fatto che la mobilità del software è in grado di portare al sistema maggiore robustezza, prestazioni, scalabilità ed espressività [11]. Gli agenti mobili sono quindi divenuti un paradigma per strutturare applicazioni distribuite.

In letteratura è possibile trovare una definizione che caratterizza in maniera sufficientemente esaustiva l'essenza di un agente mobile [12, 13]: un'agente mobile è un'entità software che esiste in un dato ambiente e possiede parte delle caratteristiche dell'agente. Un agente mobile deve contenere tutti i seguenti modelli: un modello di agente, un ciclo di vita, un modello computazionale, un modello di sicurezza, un modello di comunicazione ed infine un modello di navigazione.

In particolare per quanto riguarda il modello di navigazione, FIPA ha definito un adeguato ciclo di vita dell'agente. Tale specifica estende il consueto ciclo di vita prevedendo in più lo stato *transit* e due azioni aggiuntive che consentono rispettivamente di entrare e di lasciare tale stato (*move* ed *execute*). Questo consente di rappresentare ogni stato assumibile dall'agente nel contesto dell'AMS (*Agent Management System*) [7]. L'agente stesso è in grado di richiedere l'azione *move*, mentre è la piattaforma, attraverso l'AMS, ad essere responsabile di completare la migrazione eseguendo l'operazione di *execute*.

Gli agenti mobili richiedono naturalmente un opportuno ambiente di run-time in grado di fornire un servizio di trasferimento per essere spostati da un nodo ad un altro: l'ambiente è costruito sopra ad un sistema host. Il compito primario è fornire un ambiente in cui gli agenti mobili possano operare. Essi devono poter comunicare tra loro sia localmente sia in remoto in modo trasparente.

In letteratura c'è una distinzione tra due differenti tipi di mobilità basata sul fatto che lo stato dell'esecuzione sia o no trasferito assieme all'unità di computazione [14]. I sistemi in grado di fornire il trasferimento dello stato dell'esecuzione si dice supportino la mobilità forte (*strong mobility*), al contrario i sistemi che perdono lo stato dell'esecuzione durante il trasferimento si dice forniscano una mobilità debole (*weak mobility*). Nei sistemi in cui la mobilità è forte, la migrazione

risulta completamente trasparente al programma trasferito, mentre con la *weak mobility* è richiesta un ulteriore sforzo di programmazione per salvare manualmente parte dello stato dell'esecuzione.

Ad esempio un semplice agente scritto come un *Applet Java* fornisce mobilità del codice attraverso lo spostamento dei file delle classi da un server web ad un browser, ma naturalmente non vi sono informazioni associate allo stato. Al contrario in Aglets [15], piattaforma basata su Java e sviluppata da IBM, vengono trasferiti anche i valori delle variabili istanziate, senza però tenere conto dello stack e del program counter. Infine un esempio di mobilità forte ci viene fornito da Sumatra [16], sviluppato presso l'Università del Maryland, che consente il trasferimento del contesto di run-time dei thread di Java assieme al codice durante la migrazione.

IV. MOBILITÀ IN AGENTSERVICE

A. Introduzione

Il modello di agente su cui AgentService si basa, sembra possedere tutte le caratteristiche richieste dalla definizione di agente mobile, a partire dalla gestione del ciclo di vita che è quella definita da FIPA.

AgentService implementa un modello di mobilità debole, ma fornisce alcuni servizi aggiuntivi che possono consentire la sua attuazione in maniera trasparente ai programmatori. L'idea base è quella di sfruttare il modello di agente adottato da AgentService trasferendo dell'agente solamente lo stato, vale a dire le strutture dati contenenti le *knowledge* e lo stato dei *behavior*. All'interno della piattaforma di destinazione le attività dell'agente possono essere riavviate in conformità a ciò che è indicato nello stato persistito dell'agente. Inoltre, il framework fornisce agli sviluppatori un punto d'accesso per controllare lo stato e le attività dell'agente prima che riprenda la sua esecuzione. I dettagli di questo processo sono analizzati nella sezione seguente.

B. Implementazione

Grazie al sottostante modello di agente, la mobilità in AgentService può essere ottenuta con relativa facilità: la separazione tra lo stato dell'agente e le sue attività consente una semplice implementazione del processo di migrazione. Per poter schedare le attività di un agente, l'ambiente di run-time di AgentService necessita dei dati che definiscono lo stato dell'agente e degli assembly contenenti la definizione del tipo di agente. Quindi, spostare un agente tra due installazioni di AgentService implica, innanzi tutto, la presenza sulla piattaforma di destinazione del dato tipo di agente (*AgentTemplate*) all'interno dello *Storage* e richiede il trasferimento dello stato dell'agente.

Una volta che l'agente è stato spostato è possibile riavviare le sue attività istanziando un nuovo agente del tipo specifico e ripristinandone lo stato in maniera simile a ciò che avviene per un qualsiasi agente di AgentService dopo un crash o un riavvio del sistema. Il ripristino dello stato implica quindi il

caricamento degli oggetti *knowledge* trasferiti, la ricostruzione delle conversazioni in corso e dei messaggi presenti nella coda e l'attivazione di tutti i *behavior* in esecuzione quando l'agente è stato fermato. Le informazioni sugli oggetti *knowledge* e lo stato di ciascun *behavior* (ready, active, suspended) sono tutto ciò che veramente occorre per trasferire un agente. Discorso a parte merita la gestione delle comunicazioni e della reperibilità dell'agente mobile che è trattata nel paragrafo successivo.

Il servizio di mobilità è implementato all'interno di un modulo addizionale della piattaforma che si occupa di seguire la migrazione dello stato dell'agente e, quando necessario, del trasferimento dei relativi assembly. Quando un agente richiede un'azione *move*, gli agenti AMS delle piattaforme coinvolte contrattano la possibilità di uno spostamento e successivamente delegano al modulo di mobilità il trasferimento.

La fase di contrattazione può essere controllata dall'amministratore della piattaforma in due differenti modi, entrambi portano ad influenzare il comportamento dell'AMS. Il primo, quello più semplice, consiste nella modifica, al momento dell'installazione di AgentService, del file di configurazione della piattaforma; attraverso questo è possibile indicare se consentire o no l'hosting di agenti provenienti da altre piattaforme e in maniera duale se permettere l'invio di agenti verso altre piattaforme. Di default sono entrambi negati. Come si vede questo meccanismo di controllo è molto semplice, ma estremamente limitativo, è quindi necessario un approccio che fornisca maggiore flessibilità e potere decisionale.

Il secondo modo attraverso il quale è possibile controllare, da parte dell'amministratore, la mobilità degli agenti consiste nell'implementazione di due metodi specifici. Tali metodi vengono invocati dall'AMS nel momento in cui si verifica la necessità di prendere decisioni sul trasferimento di un agente. L'implementazione di default delle procedure si basa appunto sulle informazioni contenute nel file di configurazione della piattaforma, è, infatti, attraverso l'implementazione di default che viene applicato il controllo descritto nel paragrafo precedente. Ridefinendo invece tali metodi, è possibile modificare da codice il comportamento dell'AMS riguardo alla gestione della mobilità degli agenti nel contesto della data piattaforma di appartenenza. Tale meccanismo consente di personalizzare al meglio le decisioni, ma richiede, rispetto al primo, un maggior sforzo per l'amministratore.

Discorso differente merita l'applicazione di politiche di *load balancing* che sono analizzate nella sezione successiva.

Il processo di trasferimento di un agente può quindi essere attivato direttamente dall'agente stesso, attraverso l'invio di una richiesta all'AMS, o in alternativa utilizzando l'interfaccia di programmazione della piattaforma (*IPlatformController*): gli amministratori della piattaforma possono decidere di spostare agenti tra differenti installazioni di AgentService. Attraverso l'*IPlatformController* anche altre applicazioni software sono in grado di controllare la mobilità degli agenti ed applicare ad esempio algoritmi di distribuzione del carico.

C. Comunicazione e Reperibilità degli Agenti Mobili

Nel processo di trasferimento di un agente è importante considerare come le comunicazioni, intrattenute dall'agente mobile, possano proseguire in maniera trasparente al programmatore anche una volta avvenuto il trasferimento. I problemi di comunicazione e reperibilità sono legati al fatto che gli agenti che hanno comunicazioni in corso con l'agente mobile probabilmente saranno in possesso, relativamente ad esso, di un *agent identifier* (AID) non aggiornato.

In AgentService la comunicazione tra agenti può avvenire in due differenti modi: attraverso l'invio o la ricezione di messaggi semplici (one-shot) oppure tramite lo scambio di messaggi nel contesto di una conversazione. Per garantire la trasparenza, dal punto di vista dei programmatori, nelle comunicazioni con agenti mobili si sono definite delle specifiche aggiuntive per il modulo di messaggistica della piattaforma. L'implementazione del modulo di AgentService che decide di seguire tali specifiche deve garantire la corretta gestione delle conversazioni anche in presenza di agenti mobili. Tale problematica è risolta attraverso l'aggiornamento delle strutture dati che sono alla base delle conversazioni ogni qual volta che un agente coinvolto in una conversazione viene trasferito. Così facendo entrambi gli agenti sono in grado di continuare a scambiarsi messaggi nel contesto di una conversazione in modo indipendente rispetto alla mobilità.

Per quello che concerne invece la comunicazione basata sullo scambio di messaggi semplici, la reperibilità è garantita attraverso l'utilizzo dei cosiddetti *resolvers* presenti negli AID degli agenti. All'interno dell'AID di ciascun agente mobile dovranno essere indicati anche gli AID degli AMS delle piattaforme federate, vedi Sezione V, in modo che possano essere contattati per avere il nuovo indirizzo dell'agente trasferito.

D. Le Fasi del Processo di Mobilità

Le fasi del processo di trasferimento possono essere riassunte come segue:

- contrattazione con la piattaforma di destinazione per la mobilità dell'agente prendendo informazioni sullo *Storage* di destinazione (la contrattazione avviene tra gli agenti AMS delle rispettive piattaforme);
- blocco delle attività dell'agente, persistenza del suo stato e passaggio allo stato *transit* attraverso l'azione *move* (di queste operazioni si fa carico l'AMS della piattaforma di partenza);
- se necessario, trasferimento degli assembly richiesti dal dato *Agent Template*, tramite servizio ftp implementato nel modulo di mobilità;
- trasferimento dello stato persistito (lo stato dell'agente contenente tutti gli elementi della *knowledge*, i messaggi, le conversazioni, l'AID e lo stato di ciascun *behavior*), tramite servizio ftp implementato nel modulo di mobilità;
- creazione di un'istanza dell'agente sulla piattaforma di destinazione, ripristino della *knowledge*, creazione degli oggetti *behavior* posti nello stato in cui erano prima della migrazione (di questa operazione si fa carico l'AMS della

piattaforma di destinazione);

- invocazione del metodo *Resume(...)* per consentire al programmatore di personalizzare la riattivazione dell'agente (*Resume* è un metodo dell'Agent Template implementato dal programmatore dell'agente che in questa fase viene invocato dall'AMS);

- rilascio dell'agente e passaggio allo stato *active* attraverso l'azione *execute* (l'AMS attiva l'agente e lo *scheduler* della piattaforma manda in esecuzione i suoi *behaviour*).

E. Aspetti di Sicurezza

Nelle precedenti sezioni abbiamo discusso dei requisiti strutturali necessari ad implementare un'architettura per la mobilità in AgentService. La sicurezza è un ulteriore requisito che si aggiunge a tale architettura al fine di garantire che gli agenti mobili portino avanti le loro attività senza costituire un pericolo per l'ambiente che li ospita. In particolare ciò che si vuole evitare è l'esecuzione arbitraria di codice trasferito attraverso l'infrastruttura che permette la mobilità.

Un primo livello di sicurezza è fornito dallo *Storage* che permette l'esecuzione di solo codice verificato: gli assembly caricati nello *Storage* e quindi anche quelli trasferiti per attuare la mobilità devono essere firmati. La firma di un assembly ci fornisce una prima garanzia sul codice in esso contenuto, in quanto ci permette di riconoscere qualcosa identificato in precedenza. Tale livello di sicurezza può anche non bastare e per tal motivo l'architettura di AgentService permette di personalizzare attraverso la gestione utenti l'insieme dei permessi che sono associati ad un agente, fornendo ad essi solo quei permessi che l'amministratore della piattaforma ritiene necessario conferirgli. La gestione dei profili utente all'interno di AgentService è implementata in un opportuno modulo la cui installazione non è obbligatoria. Le funzionalità avanzate di gestione della sicurezza sono perciò possibili solo in presenza di questo modulo. Questa è una scelta che garantisce la massima flessibilità anche in tale ambito: in contesti in cui non è richiesta una particolare attenzione agli aspetti di sicurezza, l'amministratore della piattaforma può decidere se accettare la mobilità oppure no. Nel caso in cui si voglia abilitare la mobilità, gli agenti ospitati e trasferiti nella piattaforma verranno eseguiti con l'utente predefinito, che disporrà di un set di permessi associati al corrispondente utente del sistema operativo su cui è installata la piattaforma. In presenza del modulo di gestione utenti, l'amministratore potrà decidere se conferire agli agenti trasferiti un predefinito profilo di sicurezza oppure se richiedere che questi siano associati ad un particolare profilo utente presente nel proprio insieme di utenti.

Osserviamo che la gestione della sicurezza attraverso i profili utente è qualcosa che si sovrappone in modo del tutto trasparente al normale ciclo di vita della piattaforma ad agenti: il modello di agente adottato basato sugli Application Domain permette, infatti, di conferire in maniera molto semplice un particolare profilo utente con cui eseguire il codice in esso contenuto. In mancanza di un'esplicita specifica del profilo

utente questo viene ereditato dall'Application Domain che lo ha creato.

V. POLITICHE DI BILANCIAMENTO DI CARICO

A. Introduzione

La mobilità degli agenti può essere vantaggiosamente sfruttata per risolvere il problema della distribuzione del carico in un a rete di entità computazionali: i sistemi multi-agente sono in grado di decentralizzare la distribuzione del carico computazionale. Infatti, un'applicazione complessa può essere suddivisa in parti autonome, ognuna delle quali delegata ad un agente mobile. Ogni agente mobile ha il compito di cercare il nodo/piattaforma nella rete a lui più conveniente. Durante l'esecuzione, gli agenti possono spostarsi verso altri nodi dove vi sono risorse computazionali disponibili per poter quindi meglio distribuire il carico.

In alternativa la gestione della distribuzione del carico può essere centralizzata per consentire l'applicazione di politiche/algoritmi che siano in grado di forzare, quando possibile, lo spostamento degli agenti. La scelta di centralizzare le politiche di bilanciamento è volta ad ottimizzare la distribuzione di carico dell'intero sistema piuttosto che ad avvantaggiare il singolo agente.

B. Load Balancing Policy Module

Il bilanciamento del carico in AgentService è gestito dal relativo modulo Load Balancing Policy Module (LBPM). Il modulo fornisce un servizio di federazione di piattaforme per creare un ambiente unico in cui gli agenti sono in grado di muoversi. Per il trasferimento degli agenti, il LBPM sfrutta naturalmente il servizio di mobilità offerto dal relativo modulo. Per implementare correttamente il bilanciamento di carico il modulo ha la necessità di accedere alle informazioni che definiscono il profilo della piattaforma. In particolare deve essere in grado di monitorare il numero degli agenti in esecuzione ed ottenere informazioni pertinenti alle risorse fisiche dell'host. Inoltre, potranno rivelarsi fondamentali al modulo LBPM, ed in particolare alle politiche da esso applicate, le informazioni relative al comportamento a run-time del sistema multi-agente, come ad esempio il numero di messaggi scambiati. La piattaforma è in grado di garantire ad ogni modulo un contesto all'interno del quale è possibile sia reperire informazioni riguardanti il suo profilo, sia registrarsi agli eventi che scandiscono il ciclo di vita del sistema, ad esempio l'attivazione di un agente.

Viste le considerazioni di cui sopra, il modulo LBPM è quindi posto nella condizione di operare sulla mobilità sfruttando tutte le informazioni messe a disposizione dalla piattaforma. Inoltre, collaborando con i moduli LBPM presenti su altre installazioni di AgentService, è in grado di creare un quadro completo della situazione a run-time delle piattaforme coinvolte nel processo di bilanciamento. Queste piattaforme costituiscono, di fatto, una federazione che determina i confini all'interno dei quali sono applicate le politiche di bilanciamento.

Di default, il LBPM fornisce due semplici politiche di bilanciamento orientate alla distribuzione del carico: una politica bilancia il numero di agenti tra le piattaforme, mentre l'altra ha l'obiettivo di spostare nella stessa installazione di AgentService gli agenti che interagiscono più frequentemente. In aggiunta il modulo è progettato per poter applicare nuove politiche di bilanciamento definite dall'utente, caricabile all'interno del modulo attraverso un'architettura a *plug-in*.

C. Processo di federazione delle piattaforme e applicazione delle politiche di bilanciamento

L'applicazione di politiche di balancing richiede, come detto, la formazione di federazioni di piattaforme AgentService. Occorre in pratica che le piattaforme coinvolte siano in grado di conoscere i profili delle piattaforme federate e vi sia un meccanismo centralizzato per lo spostamento degli agenti in funzione della data politica che si vuole applicare. Il sistema di federazione utilizza un modello client/server.

In fase di installazione l'amministratore di sistema deve indicare, attraverso uno specifico file di configurazione del modulo LBPM, la piattaforma AgentService che svolgerà le funzioni di server. A questo punto le altre eventuali installazioni di AgentService possono essere configurate come nodi secondari e fare riferimento alla piattaforma server come nodo primario, comunicando ad essa l'adesione alla federazione, tali comunicazioni avvengono attraverso il servizio di messaggistica normalmente utilizzato dagli agenti. Una volta creata una federazione è possibile per il modulo LBPM del nodo primario (server) richiedere agli altri moduli le informazioni sullo stato delle rispettive piattaforme. Avendo a disposizione i profili delle piattaforme, il LBPM (server) è in grado di applicare correttamente l'algoritmo di balancing prescelto. Tale politica applicata, sulla base dei profili e d'altre eventuali informazioni, determina una differente distribuzione degli agenti tra le piattaforme.

La realizzazione della distribuzione degli agenti è controllata dal modulo ed avviene ciclicamente attraverso le seguenti fasi:

- LBPM server interroga la politica richiedendo il successivo agente da spostare (invoca il metodo `GetNextAgentToMove(...)` sull'interfaccia ILBP);
- la politica, in base all'algoritmo di bilanciamento, fornisce l'AID dell'agente da trasferire e l'identificativo (`PlatformDescription`) della piattaforma di destinazione;
- LBPM server comunica al LBPM della piattaforma che ospita l'agente di operare il trasferimento richiesto (la comunicazione, come già detto, avviene attraverso il modulo di messaggistica standard di AgentService);
- LBPM client chiede quindi al locale modulo responsabile del servizio di mobilità l'esecuzione fisica del trasferimento (la comunicazione tra moduli è gestita dalla coda di comandi della piattaforma);
- a trasferimento avvenuto il modulo client notifica al LBPM server il successo dell'operazione;
- LBPM server aggiorna il *profile* delle piattaforme coinvolte ed esegue un nuovo ciclo.

Si noti che il modulo LBPM prevarica il controllo dell'AMS

sul ciclo di vita degli agenti. Il modulo, infatti, opera sulla mobilità degli agenti senza effettuare alcuna richiesta all'AMS, elude la fase di contrattazione tra gli AMS. Sebbene questo violi in parte i principi della teoria degli agenti e le direttive FIPA a riguardo, sembra accettabile che per esigenze non strettamente legate alla comunità di agenti ma, ad esempio, alle disponibilità delle risorse hardware, si possa operare sulle infrastrutture degli agenti stessi in maniera quasi trasparente all'AMS. In alternativa il coinvolgimento dell'AMS di ciascuna piattaforma per contrattare il trasferimento degli agenti appesantirebbe eccessivamente il protocollo di distribuzione degli stessi e rallenterebbe le normali attività degli AMS coinvolti.

Per un corretto funzionamento della politica di bilanciamento è opportuno che la configurazione delle piattaforme federate sia tale da non permettere ai singoli agenti di spostarsi autonomamente. Tale vincolo evita eventuali conflitti ed è facilmente imponibile attraverso il file di configurazione dell'installazione di AgentService.

D. Definizione di nuove politiche di bilanciamento del carico

AgentService permette al programmatore di definire nuove politiche di balancing per la distribuzione degli agenti sulle piattaforme. La definizione di una nuova politica risulta un'operazione relativamente semplice: allo sviluppatore non viene richiesto di modificare il modulo, ma solamente di implementare una specifica interfaccia (ILBP) che deve caratterizzare ogni politica di bilanciamento. Tale interfaccia espone i metodi che le consentono di fornire al modulo i trasferimenti da effettuare, di ricevere dallo stesso le informazioni sui profili delle piattaforme federate e di essere notificata degli eventi originatisi nella piattaforma host. Di seguito viene presentata la struttura dell'interfaccia:

```
interface ILBP
{
    PolicyDescription GetDescription();
    void ConsumeEvent(PlatformEvent evt);
    void GetNextAgentToMove(
        out AID aid,
        out PlatformDescription dest);
    void AddProfile(PlatformProfile p);
    void UpdateProfile(PlatformProfile p);
    void RemoveProfile(PlatformProfile p);
}
```

Di particolare interesse è il metodo `GetNextAgentToMove(...)` nel quale viene implementato l'algoritmo di bilanciamento. Eventuali informazioni sui comportamenti a run-time del sistema, l'invio di un messaggio o la creazione di un agente, sono fornite dal modulo di bilanciamento (LBPM) attraverso il metodo `ConsumeEvent(...)`.

La classe che implementa tale interfaccia dovrà essere inserita in un assembly della CLI e quindi aggiunta alle politiche della piattaforma attraverso il relativo file di

configurazione in fase d'installazione della stessa.

Una volta resa disponibile la nuova politica, il relativo algoritmo di balancing potrà essere applicato dall'amministratore della piattaforma della federazione attraverso le procedure già descritte: file d'installazione o interfaccia di programmazione della piattaforma del nodo primario.

E. Test Case

L'infrastruttura per la gestione della mobilità è stata testata con buoni risultati su una federazione di 10 piattaforme AgentService.

Sono stati creati in maniera casuale sulle piattaforme della federazione 100 agenti, la cui attività principale è data dal semplice scambio di messaggi con i *peer* della comunità. Al nodo primario è stata applicata la politica basata sulla limitazione del numero di messaggi interpiattaforma scambiati. Una volta definita la nuova distribuzione ed effettuati i trasferimenti applicati dalla politica di bilanciamento, il numero di messaggi interpiattaforma è diminuito sensibilmente, ed è andato stabilizzandosi anche in funzione di quegli agenti che modificavano il destinatario dei loro messaggi, vedi Tabella I.

L'esito positivo del test relativamente alla politica applicata è naturalmente dipeso dal tipo di attività svolta dagli agenti coinvolti, ma quello che si è voluto attestare è la bontà dell'infrastruttura di mobilità e dei meccanismi di applicazione delle politiche di balancing, non tanto l'efficacia degli algoritmi di balancing stessi. Questi ultimi dovranno essere, in effetti, valutati ed eventualmente ridefiniti in funzione del contesto applicativo della comunità di agenti su cui andranno ad operare.

VI. CONCLUSIONI

L'architettura modulare di AgentService consente la progettazione e l'implementazione di molte funzioni aggiuntive ai normali servizi della piattaforma e l'integrazione con essi. È possibile per tal motivo arricchire la piattaforma con un'infrastruttura che garantisce la mobilità degli agenti implementando tale funzionalità in un modulo. Il modulo della mobilità implementa un servizio di mobilità debole con persistenza dello stato che avviene in maniera del tutto trasparente al programmatore dell'agente. La realizzazione di un'infrastruttura ad agenti mobili più robusta e sicura richiede invece la collaborazione di tale modulo con altri componenti della piattaforma che devono soddisfare alcuni requisiti; in particolare è stata sottolineata l'interazione con il modulo di messaggistica che deve disporre di alcune funzionalità aggiuntive rispetto alle specifiche richieste per tale modulo dalla piattaforma AgentService. Allo scopo di fornire un servizio più sofisticato, il modulo di messaggistica dovrebbe essere in grado di tenere traccia degli agenti che si sono trasferiti e notificare agli agenti residenti nella piattaforma lo spostamento di tale agente. Un modulo di messaggistica che soddisfa i requisiti richiesti dalla mobilità permette inoltre il

TABELLA I
ANDAMENTO DEI MESSAGGI INTERPIATTAFORMA SCAMBIATI DURANTE IL

Tempo (secondi)	TEST # Agenti Trasferiti	# Messaggi Interpiattaforma
0-60	0	2412
60-240	32	2157
240-420	9	1123

mantenimento delle conversazioni tra due agenti, anche se uno di essi si trasferisce in un'altra piattaforma; tale operazione viene effettuata in maniera del tutto trasparente agli agenti stessi. Osserviamo che i servizi aggiuntivi richiesti dal modulo che implementa la mobilità sono, di fatto, una violazione della struttura modulare della piattaforma in quanto comportano un debole accoppiamento con il servizio di messaggistica. Occorre però tenere conto del fatto che tali servizi sono richiesti per implementare un'infrastruttura per la mobilità più sofisticata e che le funzionalità di base della mobilità sono garantite indipendentemente dalla presenza o meno di tali servizi. Poiché la scelta di dotare una piattaforma dell'infrastruttura per la mobilità avviene molto spesso in fase di installazione della piattaforma, l'accoppiamento debole con il modulo di messaggistica non costituisce di fatto un problema reale.

La presenza di un'infrastruttura per la mobilità permette di arricchire la piattaforma con servizi sofisticati come ad esempio la gestione del bilanciamento di carico tra piattaforme federate. Tale funzionalità è nuovamente implementata sfruttando i vantaggi dell'architettura modulare di AgentService: un opportuno modulo (LBPM) appoggiandosi al servizio di mobilità, applica algoritmi di bilanciamento del carico per la gestione delle risorse hardware. Le politiche di bilanciamento fornite di default si basano sul numero dei messaggi scambiati e sul numero degli agenti presenti in una piattaforma ma è possibile estendere tali politiche definendo regole personalizzate.

Il processo di trasferimento degli agenti è stato testato con esiti positivi assieme alla funzionalità di bilanciamento del carico. Si è osservato che l'aspetto più oneroso per l'architettura di AgentService è dato dallo spostamento dagli assembly contenenti la definizione dei tipi di agenti (AgentTemplate). Tale trasferimento non avviene sempre, esso dipende dal tipo applicazione, ma in ogni caso dovrebbe essere abbastanza limitato, in quanto normalmente si tende ad avere applicazioni con più agenti dello stesso tipo o perlomeno a riusare i loro elementi base, *knowledge* e *behavior*, vista la modularità del modello di agente adottato da AgentService.

Una limitazione di cui soffre l'infrastruttura per la mobilità qui descritta è la mancanza di interoperabilità con altre piattaforme che non siano installazioni di AgentService, Jade [10] in particolare. La realizzazione di un'infrastruttura per la mobilità tra piattaforme di diversa natura è, di fatto, un aspetto molto difficile da concretizzare in quanto occorre superare i problemi dovuti alla differenza delle tecnologie utilizzate, delle architetture implementate e del modello di agente adottato nelle diverse piattaforme.

RIFERIMENTI

- [1] A. Fuggetta, G. P. Picco, G. Vigna. *Understanding Code Mobility*, IEEE Trans. on Software Engineering, Maggio 1998.
- [2] Muhammad Kamran Naseem, Sohail Iqbal, Khalid Rashid, *Implementing Strong Code Mobility*, Information Technology Journal 3(2): pp. 188-191, 2004.
- [3] R. R Brooks, N. Orr. *A Model for Mobile Code Using Interacting Automata*, IEEE Trans. Mobile Computing 1(4): pp. 313-326, 2002.
- [4] Standard ECMA-335: *Common Language Infrastructure (CLI)*, 2nd Edition, Dicembre 2002, ECMA, disponibile presso: <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [5] Standard ISO/IEC 23271:2003: *Common Language Infrastructure*, 28 Marzo 2003, ISO.
- [6] A. Boccalatte, A. Gozzi, A. Grosso, C. Vecchiola. *AgentService*, The Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 Giugno 2004.
- [7] FIPA Abstract Architecture Specification, <http://www.fipa.org/specs/fipa00001/>
- [8] H.S. Nwana, D.T. Ndumu, L.C. Lee, *ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems*, in Proceedings of PAAM98, pp. 377-391, London, U.K., 1998.
- [9] Stefan Poslad, Phil Buckle, Rob Hadingham, *The FIPA-OS agent platform: Open Source for Open Standards*, Published at PAAM2000, Manchester, UK, April 2000.
- [10] F. Bellifemine, G. Rimassa, A. Poggi, *JADE - A FIPA-compliant Agent Framework*, in Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.
- [11] N. M. Karnik, A. R. Tripathi. *Design Issues in Mobile-Agent Programming Systems*, IEEE Concurrency 6(3): pp. 52-61, Luglio-Settembre 1998.
- [12] D. Chess, C. Harrison, A. Kershenbaum. *Mobile Agents: Are they a good idea?*, Technical Report, IBM T.J. Watson Research Center, NY, Marzo 1995.
- [13] H. Nwana. *Software agents: An Overview*, Knowledge and Engineering Review, 11(3), Novembre 1996.
- [14] G. Cabri, L. Leonardi, F. Zambonelli. *Weak and Strong Mobility in Mobile Agent Applications*, Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK), Aprile 2000.
- [15] D. Lange, M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.
- [16] A. Acharya, M. Ranganathan, J. Salz. *Sumatra: A Language for Resourceaware Mobile Programs*, Mobile Object Systems: Towards the Programmable Internet, J. Vitek and C. Tschudin (Eds.), Springer-Verlag, Lecture Notes in Computer Science No. 1222: pp. 111-130, Aprile 1997.

PACMAS: A Personalized, Adaptive, and Cooperative MultiAgent System Architecture

Giuliano Armano, Giancarlo Cherchi, Andrea Manconi, and Eloisa Vargiu

University of Cagliari

Piazza d'Armi

I-09123, Cagliari, Italy

Email: {armano,cherchi,manconi,vargiu}@diee.unica.it

Abstract—In this paper, a generic architecture, designed to support the implementation of applications aimed at managing information among different and heterogeneous sources, is presented. Information is filtered and organized according to personal interests explicitly stated by the user. User profiles are improved and refined throughout time by suitable adaptation techniques. The overall architecture has been called PACMAS, being a support for implementing Personalized, Adaptive, and Cooperative MultiAgent Systems. PACMAS agents are autonomous and flexible, and can be made personal, adaptive and cooperative, depending on the given application. The peculiarities of the architecture are highlighted by illustrating three relevant case studies focused on giving a support to undergraduate and graduate students, on predicting protein secondary structure, and on classifying newspaper articles, respectively.

I. INTRODUCTION

Accessing the widespread amount of distributed information resources, such as the World Wide Web (WWW), entails relevant problems (e.g., “information overload” [19]). Moreover, different users are typically interested in different parts of the available information, so that personalized and effective information-filtering procedures are needed. Software agents have been widely proposed for dealing with this kind of information retrieval and filtering problems [13] [8] [15] [25].

From our perspective, assuming that information sources are a primary operational context for software agents, the following categories can be identified focusing on their specific role: (i) *information agents*, able to access to information sources and to collect and manipulate such information [19], (ii) *filter agents*, able to transform information according to user preferences [18], (iii) *task agents*, able to help users to perform tasks by solving problems and exchanging information with other agents [10], (iv) *interface agents*, in charge of interacting with the user such that she/he interacts with other agents throughout them [17], and (v) *middle agents*, devised to establish communication among requesters and providers [7]. Although this taxonomy is focused on a quite general perspective, alternative taxonomies could be defined focusing on different features. In particular, one may focus on capabilities rather than roles, a software agent being able to embed any subset of the following capabilities: (i) *autonomy*, to operate without the intervention of users; (ii) *reactivity*, to react to a stimulus of the underlying environment according to a stimulus/response behaviour; (iii) *proactiveness*, to exhibit

goal-directed behavior in order to satisfy a design objective; (iv) *social ability*, to interact with other agents according to the syntax and semantics of some selected communication language; (v) *flexibility*, to exhibit reactivity, proactiveness, and social ability simultaneously [24]; (vi) *personalization*, to personalize the behavior to fulfill user’s interests and preferences; (vii) *adaptation*, to adapt to the underlying environment by learning how to react and/or interact with it; (viii) *cooperation*, to interact with other agents in order to achieve a common goal; (ix) *deliberative capability*, to reason about the world model and to engage planning and negotiation, possibly in coordination with other agents; (x) *mobility*, to migrate from node to node in a local- or wide-area network.

In this paper, we present a generic multiagent architecture designed to support the implementation of applications aimed at: (i) retrieving heterogeneous data spread among different sources (i.e., generic html pages, news, blogs, forums, and databases), (ii) filtering and organizing them according to personal interests explicitly stated by each user, and (iii) providing adaptation techniques to improve and refine throughout time the profile of each selected user.

Each agent is autonomous and flexible, and may implement (one or more of) the following capabilities: personalization, adaptation, and cooperation. The overall architecture has been called PACMAS, being designed to support the implementation of Personalized, Adaptive, and Cooperative MultiAgent Systems. The PACMAS architecture can easily give rise to specific systems by (1) identifying the characteristics of the dataflow that occurs from information sources to users (and vice versa), and (2) customizing each involved agent according to its actual role and capabilities.

The remainder of this paper is organized as follows: In Section 2 the Personalized, Adaptive, and Cooperative architecture, called PACMAS, is depicted. In Section 3, three case studies are presented, each one customized for a specific application. Section 4 draws conclusions and future work.

II. THE PACMAS ARCHITECTURE

PACMAS is a generic multiagent architecture aimed at retrieving, filtering and reorganizing information according to users’ interests. PACMAS agents can be personalized, adaptive, and cooperative, depending on their specific role.

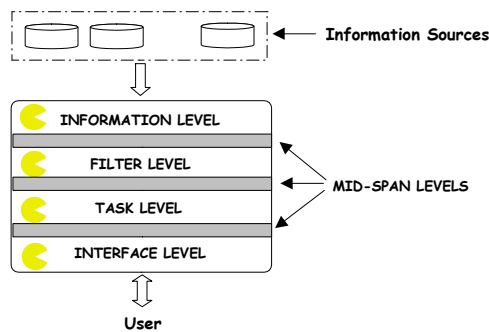


Fig. 1. The PACMAS Architecture.

PACMAS Macro-Architecture

The overall architecture (depicted in Figure 1) encompasses four main levels (i.e., information, filter, task, and interface), each being associated to a specific role. The communication between adjacent levels is achieved through suitable middle agents, which form a corresponding mid-span level.

Each level is populated by a society of agents, so that communication may occur both horizontally and vertically. The former kind of communication supports cooperation among agents belonging to a specific level, whereas the latter supports the flow of information and/or control between adjacent levels through suitable middle-agents.

At the information level, agents are entrusted with extracting data from the information sources. Each information agent is associated to one information source, playing the role of wrapper. Upon extraction, the information is then made available to the underlying filter level.

At the filter level, agents are aimed at selecting information deemed relevant to the users, and cooperate to prevent information from being overloaded and redundant. Two filtering strategies can be adopted: generic and personal. The former applies the same rules to all users; whereas the latter is customised for a specific user. Each strategy can be implemented through a pipeline of filters, since data undergo an incremental refinement process. The information filtered so far is then made available to the task level.

At the task level, agents arrange data according to users' personal needs and preferences. In a sense, they can be considered as the core of the architecture. In fact, they are devoted to achieve users' goals by cooperating together and adapting themselves to the changes of the underlying environment. In general, they can be combined together according to different connection modes, depending on the specific application.

At the interface level, a suitable interface agent is associated to each different user interface. In fact, a user can generally interact with an application through several interfaces and devices (e.g., pc, pda, mobile phones, etc.). Interface agents usually act individually without cooperation. On the other hand, they can be personalized to display only the information deemed relevant to a specific user. Moreover, in complex applications, they can adapt themselves to progressively improve their ability in supplying information to the user.

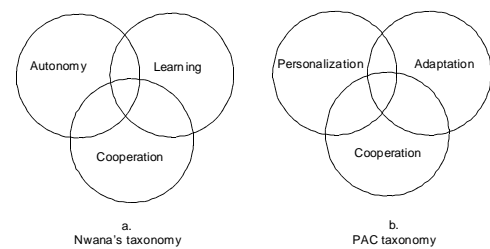


Fig. 2. Agents taxonomies.

At the mid-span level, agents are aimed at establishing communication among requesters and providers. In the literature, several solutions have been proposed: e.g., blackboard agents, matchmaker or yellow page agents, and broker agents (see [7] for further details). In the PACMAS architecture, agents at the mid-span level can be implemented as matchmakers or brokers, depending on the specific application.

PACMAS Micro-Architecture

Keeping in mind that agents may be classified along several ideal and primary capabilities that they should embed, let us first recall the agent taxonomy proposed in [20]. In such taxonomy, three primary capabilities have been identified: autonomy, learning, and cooperation (see Figure 2-a). In our view, agents are always autonomous and flexible, hence we deem that autonomy should not be explicitly listed in a diagram. On the contrary, we claim that personalization should be taken into account as a primary feature while depicting the characteristics of software agents, the resulting taxonomy is depicted in Figure 2-b.

As for personalization, an initial user profile is provided in form of a list of keywords, representing users' interests. The information about the user profile is stored by agents belonging to the interface level. It is worth noting that, to exhibit personalization, filter and task agents may need information about the user profile. This flows up from the interface level to the other levels through the middle-span levels. In particular, agents belonging to mid-span levels (i.e., middle agents) take care of handling synchronization and avoiding potential inconsistencies. Moreover, the user behavior is tracked during the execution of the application to support explicit feedback, in order to improve her/his profile.

As for adaptation, a model centered on the concept of "mixtures of experts" has been employed. Each expert is implemented by an agent able to select relevant information according to an embedded string of feature-value pairs, features being selectable from an overall set of relevant features defined for the given application. The decision of adopting a subset of the available features has been taken for efficiency reasons, being conceptually equivalent to the one usually adopted in a typical GA-based environment [11], which handles also don't-care symbols. The system starts with an initial population of experts, during the evolution of the system further experts are created according to a covering, crossover, or mutation mechanism.

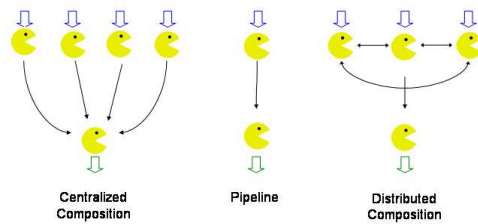


Fig. 3. Agents Connections.

As for cooperation, agents at the same level exchange messages and/or data to achieve common goals, according to the requests made by the user. Cooperation is implemented in accordance with the following modes: centralized composition, pipeline, and distributed composition (see Figure 3). In particular: (i) centralized compositions can be used for integrating different capabilities, so that the resulting behavior actually depends on the combination activity; (ii) pipelines can be used to distribute information at different levels of abstraction, so that data can be increasingly refined and adapted to the user's needs; and (iii) distributed compositions can be used to model a cooperation among the involved components aimed at processing interlaced information. The most important form of cooperation concerns the "horizontal" control flow that occurs between peer agents. For instance, filter agents can interact in order to reduce the information overload and redundancy, whereas task agents can work together to solve problems that require social interactions to be solved.

III. CASE STUDIES

In order to highlight the peculiarities of the architecture, three relevant case studies are presented. The first one is focused on giving a support to undergraduate and graduate students; the second one is concerned with the problem of predicting protein secondary structure; and the third one is devoted to classify newspaper articles.

All the proposed case studies have been implemented using Jade [4] as the underlying framework.

PACMAS for Supporting Students in University Activities

This case study is focused on giving a support to undergraduate and graduate students¹.

Motivation: Let us consider a typical University Department. It generally makes available the information about courses, seminars, exams, professors, and students on different areas: web sites, forums, and news (NNTP) servers. All the relevant information is spread on the department portal, on the web site of each course, and on the personal page of each professor. Furthermore, each professor might activate her/his news and forum service. Some of the information potentially interests all students, such as lesson timetables, exam dates,

taxes, and student tutoring. On the other hand, students belonging to different courses are interested in different lessons and exams. For example, a student attending the MSc in Computer Science may be interested in the *Object Oriented Programming Languages I* course rather than in the *Processors and Embedded Systems Architectures* one. Similarly, a student attending the MSc in Digital Microelectronics may be interested in the *Processors and Embedded Systems Architectures* course rather than in *Object Oriented Programming Languages I* one. Typically, a student in search of relevant information about her/his University activities browses web sites, and reads announcements from forum and news services. This is a repetitive and boring task that can be automated. From our perspective, personalization and adaptation represent the added value of such an automated system.

Implementation: Using PACMAS, we developed a system devoted to support undergraduate and graduate students in their University activity at the Department of Electrical and Electronic Engineering (DIEE) of the University of Cagliari. Let us note that supporting students involves several activities: information extraction, information retrieval and filtering, information processing, and results presentation. Each activity corresponds to a suitable level of the PACMAS architecture.

Information Extraction. It is carried out at the information level by information agents that play the role of wrappers, devised to process information sources. Each wrapper is specialized for dealing with a specific information source: e.g., web pages, forums or news services. In the current implementation, information agents are not personalized, not adaptive, and not cooperative (\overline{PAC}). Personalization is not supported, since information agents are aimed at retrieving information potentially relevant to all students, regardless of their personal interests and preferences. Adaptation is also not supported, being the system mainly concerned with changes in users needs rather than in the underlying environment². Cooperation is also not supported, cause each information agent is devoted to wrap a different information source.

Information Retrieval and Filtering. It is carried out at the filter level. In particular, this level contains a set of "redundancy filters" (one for each information source), an anti-spam filter agent and a population of personal filter agents (one for each user of the system). Redundancy filters cooperate together to remove the redundancy of data provided by the information sources (throughout the information agents). Redundancy filters are not personalized, not adaptive, and cooperative (\overline{PAC}). Similarly to information agents, personalization and adaptation are not required. On the other hand, cooperation is required to prevent the information from being redundant. The anti-spam filter is not personalized, not adaptive, and not cooperative (\overline{PAC})³. Being not dependent from a specific student, it filters the same information by removing undesirable contents according to a rule-based mechanism. Personal filters

²In this particular case the variability of the information sources

³In the current release of the system anti-spam agents are not permitted to implement adaptation, although in principle this property may be supplied in a future release.

¹This work has been partially funded by the Italian Ministry of University and Research under the program PRIN 2003 *Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale*.

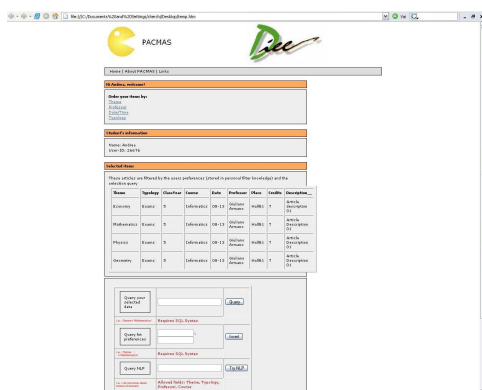


Fig. 4. JSP graphical interface.

are personalized, adaptive and not cooperative (PAC). As for personalization, they are sensible to any explicit change imposed by the corresponding student or to a change that occurs in the curriculum of the student. As for adaptation, they are able to progressively adapt their filtering capabilities according to the choices performed by the corresponding student during the lifetime of the agent. Cooperation is not supported; in fact, in the current release of the system, only a specific support for implementing voting policies according to the guidelines of GA-based systems is supplied.

Information Processing. It is carried out at the task level, where agents are devoted to perform different tasks according to the requirements imposed by the corresponding user. In particular, each task agent is customized for a specific task (e.g., lessons timetable, seminars, and exams scheduling). Agents belonging to the task level exploit a model centered on the concept of “mixtures of experts”, each expert being implemented by an agent. The system supports each user with a specific population of experts, handled in accordance with the basic guidelines of online systems, especially the ones that characterize evolutionary environments. Task agents are personalized, adaptive, and cooperative (PAC). Personalization is required since different behaviors are associated to different students. Adaptation is required since they adapt themselves to the needs of the corresponding student through a GA-based feedback mechanism. Cooperation is required since they usually need other task agents to successfully achieve their own goals.

Results Presentation. It is carried out at the interface level, through agents aimed at interacting with the users. Agents and users interact through a suitable graphical interface that can be run on several devices, including mobile phones. A different interface agent has been associated to each device. In the current implementation, the system embodies a graphical interface that runs on several devices, including MIDP 1.0 compliant devices, and JSP web pages (as the one shown in Figure 4)⁴.

Interface agents are also devoted to handle user profile and propagate it by the intervention of middle agents. Furthermore,

any feedback provided by the user can be exploited by the adaptive mechanism to improve the user profile. Interface agents are personal, adaptive, and not cooperative (PAC). Personalization is required to allow each student the customization of her/his interface. Adaptation is supported, since an interface agent must adapt to the changes that occur in the preferences and interests of the corresponding student. Cooperation is not supported by agents that belong to this architectural level.

PACMAS for Predicting Protein Secondary Structures

In this section we briefly describe an application concerned with the problem of predicting protein secondary structure using PACMAS (for further details see [2]).

Motivation: Difficulties in predicting protein structure are mainly due to the complex interactions between different parts of the same protein, on the one hand, and between the protein and the surrounding environment, on the other hand. Actually, some conformational structures are mainly determined by local interactions between near residues, whereas others are due to distant interactions in the same protein. Moreover, notwithstanding the fact that primary sequences are believed to contain all information necessary to determine the corresponding structure [1], recent studies demonstrate that many proteins fold into their proper three-dimensional structure with the help of molecular chaperones that act as catalysts [9], [12]. The problem of identifying protein structures can be simplified by considering only their secondary structure; i.e. a linear labeling representing the conformation to which each residue belongs to. Thus, secondary structure is an abstract view of amino acid chains, in which each residue is mapped into a secondary alphabet usually composed by three symbols: alpha-helix (α), beta-sheet (β), and random-coil (c).

Implementation: Keeping in mind that the PACMAS architecture encompasses several levels, each one hosting a set of agents, in the following, we illustrate how each level supports the implementation of the proposed application.

At the information level, agents play the role of wrappers, which –in our view– can be considered a particular kind of filters, devised to process information sources. Each wrapper is associated to one information source: (i) the selected training set (the TRAIN database), (ii) the test set (the R126 database), and (iii) a database containing information about the domain knowledge (the AAindex database). Datasets are briefly summarized in Table I. In the current implementation, information agents are not personalized, not adaptive, and not cooperative (shortly PAC). Personalization is not supported at this level, since information agents are only devoted to wrap the datasets containing proteins. Adaptation is also not supported, since information sources are invariant for the system and are not user-dependent. Cooperation is also not supported by the information agents, since each agent retrieves information from different sources, and each information source has a specific role in the chosen application.

At the filter level, agents embody encoding methods. Let us briefly recall that encoding methods play an important role

⁴Available at: <http://iascw.diee.unica.it/PacmasWWW>

TABLE I
INFORMATION SOURCES FOR PREDICTING PROTEIN SECONDARY
STRUCTURES

Dataset	Description
TRAIN	It has been derived from a PDB selection obtained by removing short proteins (less than 30 aminoacids), and with a resolution of at least 2.5 Å. This dataset underwent a homology reduction, aimed at excluding sequences with more than 50% of similarity. The resulting training set consists of 1180 sequences, corresponding to 282,303 amino acids.
R126	It has been derived from the historical Rost and Sander's protein dataset (RS126) [22], and corresponds to a total of 23,363 amino acids (the overall number has slightly varied over the years, due to changes and corrections in the PDB.)
AAindex	It contains information about hydrophobicity, dimension, charge and other features required for evaluating the given metrics. In the current application eight domain-specific metrics have been devised and implemented. A sample metrics is: Check whether hydrophobic amino acids occur in a window of predefined length according to a clear periodicity, whose underlying rationale is that sometimes hydrophobic amino acids are regularly distributed along alpha-helices.

in the prediction of protein secondary structures. In fact, they describe the chemical-physics properties of aminoacid deemed more interesting for the prediction. Several populations of filter agents have been implemented, each of them performing a different encoding techniques: one-shot, substitution matrices, multiple alignment algorithms, and a technique that combines the specificity of the multiple alignment technique with the generality of the substitution matrices. Personalization is not supported by filter agents, since they always embody the same encoding methods for all users. Adaptation is also not supported either, since encoding methods do not change during the application. Cooperation is supported by filter agents, as some implemented encoding methods brings together several algorithms (e.g., the encoding method that combines multiple alignment with substitution matrices).

At the task level, a population of task agents, which are the core of this case study, perform the protein secondary structure prediction. The “internals” of each task agent is based on the micro-architecture proposed for the NXCS-Experts [3]. In its basic form, each NXCS expert E can be represented by a triple $\langle g, h, w \rangle$, where: (i) g is a “guard” devised to check whether an input x can be processed or not, (ii) h is an embedded predictor whose activation depends on $g(x)$, and (iii) w is a weighting function used to perform output combination. Hence, the output of E coincides with $h(x)$ for any input x “acknowledged” (i.e., matched) by g , otherwise it is not defined. Typically, the guard g of a generic NXCS classifier is implemented by an XCS-like classifier, able to match inputs according to a set of selected features deemed relevant for the given application, whereas the embedded predictor h consists of a feed forward ANN, trained and activated on the inputs acknowledged by the corresponding guard. In the case E contributes to the final prediction (together with other experts), its output is modulated by the value $w(x)$, which represents the

expert strength in the voting mechanism. It may depend on several features, including $g(x)$, the overall fitness of the corresponding expert, and the reliability of the prediction made by the embedded predictor. It is worth noting that matching can be “flexible”, meaning that the matching activity returns a value in $[0,1]$ rather than “true” or “false”. In this case, only inputs such that $g(x) \geq \sigma$ will be processed by the corresponding embedded predictor (σ being a system parameter). Task agents are not personalized, adaptive, and cooperative (shortly \overline{PAC}). Personalization is not required, since task agents exhibit the same behaviors for all the users. Adaptation is required, since each expert is suitably trained through a typical evolutionary behavior. Cooperation is required, since they usually need other task agents to successfully achieve their own goals.

At the interface level, agents are aimed at interacting with the user. In the current implementation, this kind of agents has not been developed. Nevertheless, we are investigating how to implement a flexible behavior at the user side. In particular, a suitable web interface is under study. We envision an interface personalized for each user, in which the user can input a protein to be predicted also being given the possibility of selecting the encoding technique to be applied. The resulting information agents will be personalized, adaptive, and not cooperative (shortly PAC). Personalization will be required in order to allow each user to customize the user interface. Adaptation will be required, since agents could adapt themselves to the changes that occur in the user preferences. Cooperation will not be required by the agents belonging to this architectural level.

As for the mid-span levels, the corresponding middle agents exhibit a different behavior depending on the mid-span level that they belong to. In particular, let us recall that, in the PACMAS architecture, there are three mid-span levels, one between information and filter levels (in the following, IF level), one between filter and task levels (in the following, FT level), and one between task and interface levels (in the following, TI level). In this specific application personalization and adaptation are not supported by middle agents, since they are only devoted to connect together agents belonging to adjacent levels. Cooperation is supported by agents belonging to the IF and the FT levels, since in the training phase they are used to verify the prediction.

PACMAS for Newspaper Articles Classification

In this section we briefly describe the case study concerned with the problem of classifying newspaper articles using PACMAS (see [6] for details).

Motivation: All the information sources belonging to the WWW make it hard for users to choose the most suitable according to their interests. Finding useful information of personal interest has become difficult for Internet users. Ideally, users should be able to take advantage of the wide range of available information while being able to find the one she/he is interested in. In particular, manually selecting newspaper articles is quite difficult or not feasible within the time constraints common for most users also considering that

the results could not perfectly fit with the user interests. Some systems try to perform that task automatically, performing content-based filtering. In particular, software agents have been widely proposed for retrieving information from the web ([23], [16], and [5]).

Implementation: At the information level, agents play the role of wrappers, each one being associated to a different information source. In particular, in the current implementation a set of agents wraps databases containing italian newspaper articles⁵. Furthermore, an agent wraps the proposed taxonomy that is a subset of the one proposed by the International Press Telecommunications Council⁶. Information agents are not personalized, not adaptive, and not cooperative (shortly *PAC*). Personalization is not supported at this level, since information agents are only devoted to wrap information sources. Adaptation is also not supported, since we assume that information sources are invariant for the system and are not user-dependent. Cooperation is also not supported by the information agents, since each agent retrieves information from different sources.

At the filter level, a population of agents manipulates the information belonging to the information level through suitable filter strategies. First, a set of agents removes all non-informative words such as prepositions, conjunctions, pronouns and very common verbs by using a standard stop-word list. After stop-words removal, a set of agents performs a stemming algorithm [21] to remove the most common morphological and inflexional endings from words. Then, for each class, a set of agents selects the features relevant to the classification task according to the information gain method⁷. Filter agents are not personalized, not adaptive, and cooperative (shortly *PAC*). Personalization is not supported at this level, since the adopted filter strategies are user-independent. Adaptation is also not supported, since the adopted strategies do not change during agents activities. Cooperation is supported by the filter agents, since agents cooperate continuously in order to perform the filtering activity.

At the task level, a population of agents have been developed, each one embedding a k -NN classifier⁸. Each agent has been trained in order to recognize a specific class, and it is also devoted to measure the classification accuracy according to the confusion matrix [14]. Task agents are not personalized, adaptive, and cooperative (shortly *PAC*). Personalization is not supported at this level, since, in the current implementation, the adopted classification strategies are user-independent. Adaptation is supported by the task agents since they learn the classification rules during their life. Cooperation is supported by the task agents, since agents sometimes have to interact

⁵In general they may wrap any web sites containing newspaper articles (e.g., online newspapers).

⁶<http://www.iptc.org/>

⁷It measures the number of bits of information obtained for category prediction by knowing the presence or absence of a term in a document.

⁸The k -nearest neighbor is a classification method based upon observable features. The algorithm selects a set which contains the k nearest neighbours and assigns the class label to the new data point based upon the most numerous class with the set.

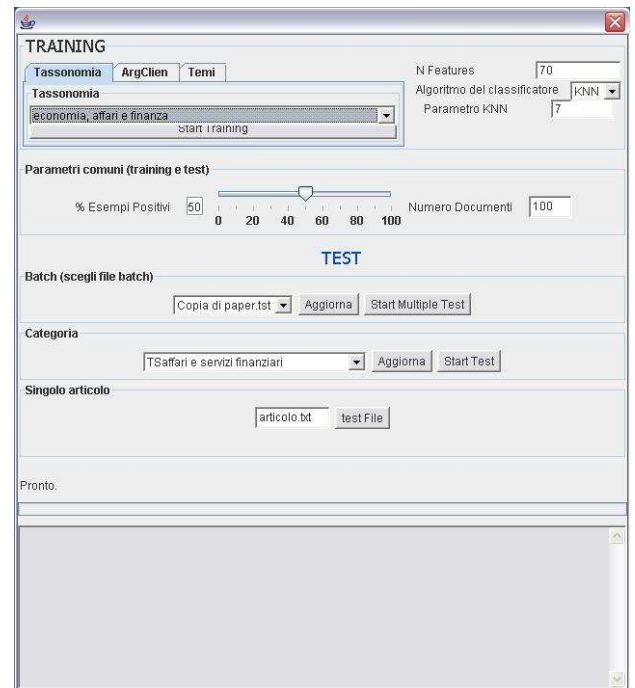


Fig. 5. Interface for the newspaper articles classifying system.

each other in order to achieve their goals.

At the interface level, agents are aimed at interacting with the user. In the current implementation, agents and users interact through a suitable graphical interface that runs on a pc (see Figure 5). Interface agents are also devoted to handle user profile and propagate it by the intervention of middle agents. Interface agents are personal, not adaptive, and not cooperative (shortly *PAC*). Personalization is required to allow each user the customization of her/his interface. In the current implementation adaptation is not supported, but in general an interface agent might adapt to the changes that occur in the preferences and interests of the corresponding user. Cooperation is not supported by agents that belong to this architectural level.

Discussion

The peculiarities of the architecture have been highlighted by depicting three relevant case studies. Table II shows agents and their capabilities for the proposed case studies. In particular, the added value of the proposed approach is that PACMAS agents are polymorphic in the sense that they can exhibit a different behavior depending on the specific application in which they operate.

IV. CONCLUSIONS AND FUTURE WORK

In this paper a generic architecture designed to support the implementation of applications aimed at managing information among different and heterogeneous sources has been presented. Information is filtered and organized according to personal interests explicitly stated by the user. User profiles are improved and refined throughout time by suitable adaptation

TABLE II
AGENTS CAPABILITIES

Agents	Case study 1	Case study 2	Case study 3
Information	PAC	PAC	PAC
Filter	Redundancy: \overline{PAC} Anti-spam: \overline{PAC} Personal: \overline{PAC}	\overline{PAC}	\overline{PAC}
Task	PAC	\overline{PAC}	\overline{PAC}
Interface	PAC	PAC	PAC
Middle	\overline{PAC}	IF: \overline{PAC} FT: \overline{PAC} TI: \overline{PAC}	\overline{PAC}

techniques. The overall architecture has been called PACMAS, being a support for implementing Personalized, Adaptive, and Cooperative MultiAgent Systems. PACMAS agents are autonomous and flexible, and can be personalized, adaptive and cooperative depending on the implemented application.

As for the future work, we are investigating how to improve the intelligent capabilities of agents with more complex forms of personalization, adaptation, and cooperation. Moreover, the possibility to implement further intelligent applications using PACMAS is currently under study.

V. ACKNOWLEDGMENTS

We would like to thank Andrea Addis for participating in the implementation of the prototype.

REFERENCES

- [1] C. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.
- [2] G. Armano, G. Mancosu, A. Orro, M. Saba, and E. Vargiu. Biopacmas: A personalized, adaptive, and cooperative multiagent system for predicting protein secondary structure. In *AI*IA 2005: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence (AI*IA 2005)*. LNAI 3673, Springer, September 2005.
- [3] G. Armano, A. Murru, and F. Roli. Stock market prediction by a mixture of genetic-neural experts. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 15(16):501–526, 2002.
- [4] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with jade. In *Eventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, 2000.
- [5] R. Carreira, J. M. Crato, D. Goncalves, and J. A. Jorge. Evaluating adaptive user profiles for news classification. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 206–212, New York, NY, USA, 2004. ACM Press.
- [6] G. Cherchi, A. Manconi, E. Vargiu, and D. Deledda. Text Categorization Using a Personalized, Adaptive, and Cooperative MultiAgent System. In *Workshop dagli Oggetti agli Agenti, Simulazione e Analisi Formale di Sistemi Complessi (WOA 2005)*, November 2005.
- [7] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 578–583, 1997.
- [8] O. Etzioni and D. Weld. Intelligent agents on the internet: fact, fiction and forecast. *IEEE Expert*, 10(4):44–49, 1995.
- [9] S. J. Gething, M.J. Protein folding in the cell. *Nature*, 355:33–45, 1992.
- [10] J. Giampapa, K. Sycara, A. Fath, A. Steinfeld, and D. Siewiorek. A multi-agent system for automatically resolving network interoperability problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1462–1463, 2004.
- [11] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [12] F. Hartl. Secrets of a double-doughnut. *Nature*, 371:557–559, 1994.
- [13] C. A. Knoblock, Y. Arens, and C.-N. Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada, 1994. University of Toronto Press.
- [14] R. Kohavi and F. Provost. Glossary of terms. *Special issue on applications of machine learning and the knowledge discovery process, Machine Learning*, 30(2/3):271–274, 1998.
- [15] J. Kramer. Agent based personalized information retrieval, 1997.
- [16] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.
- [17] H. Lieberman. Autonomous interface agents. In *Proceedings of the ACM Conference on Computers and Human Interface (CHI-97)*, pages 67–74, 1997.
- [18] E. Lutz, H. Kleist-Retzow, and K. Hoernig. Mafiaan active mail-filter-agent for an intelligent document processing support. *ACM SIGOIS Bulletin*, 11(4):16–32, 1990.
- [19] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.
- [20] H. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.
- [21] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- [22] B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal Molecular Biology*, 232:584–599, 1993.
- [23] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In I. Press, editor, *9th Conference on Artificial Intelligence for Applications (CAIA-93)*, pages 345–352, 2003.
- [24] M. Wooldridge and N. Jennings. *Intelligent Agents*, chapter Agent Theories, Architectures, and Languages: a Survey, pages 1–22. Berlin: Springer-Verlag, 1995.
- [25] J. Yang, V. Honavar, L. Miller, and J. Wong. Intelligent mobile agents for information retrieval and knowledge discovery from distributed data and knowledge sources. In *IEEE Information Technology Conference*. Syracuse, NY, 1998.

Text Categorization Using a Personalized, Adaptive, and Cooperative MultiAgent System

Giancarlo Cherchi, Andrea Manconi, Eloisa Vargiu
University of Cagliari
Piazza d'Armi, I-09123, Cagliari, Italy
Email: {cherchi,manconi,vargiu}@diee.unica.it

Dario Deledda
Arcadia Design
Loc. Is Coras, I-09028 Sestu, Cagliari, Italy
Email: dario.deledda@arcadiadesign.it

Abstract—In this paper, a multiagent system for supporting users in retrieving information from heterogeneous data sources, and classifying them according to users' personal preferences, is presented. The system is built upon PACMAS, a generic architecture that supports the implementation of Personalized, Adaptive, and Cooperative MultiAgent Systems. Preliminary tests have been conducted to evaluate the effectiveness of the system in retrieving and classifying newspaper articles. Results show an average accuracy of about 80%.

I. INTRODUCTION

The information available on the WWW is continuously growing from different points of view: information sources are increasing, topics discussed are becoming more and more heterogeneous, and stored data has reached a considerable size. It has become a difficult task for Internet users to select contents according to their personal interests, especially if contents are continuously updated (e.g., news, newspaper articles, Reuters, RSS feeds, blogs, etc.). Unfortunately, traditional filtering techniques based on keyword search are often inadequate to express what the user is really searching for. Furthermore, users often need to refine by hand the achieved results.

Supporting users in handling with the enormous and widespread amount of web information is becoming a primary issue. To this end, an automated system able to retrieve information from the Internet, and to select the contents really deemed relevant for the user, through a text categorization process, would be very helpful.

In the literature, software agents have been widely proposed for retrieving information from the web (see for example [9] [7] [11]). Furthermore, several machine learning techniques have been applied to text categorization (see [18] for a detailed comparison).

In this paper, we focus on the problem of retrieving articles from Italian online newspapers, and classifying them using suitable machine learning techniques. In particular, we exploit the PACMAS architecture [2] to build a personalized, adaptive, and cooperative multiagent system.

The outline of the paper is organized as following: in Section II some related work on agent-based information retrieving is briefly recalled; Section III briefly illustrates the text categorization problem; Section IV sketches the PACMAS architecture; In Section V, all customizations devised for ex-

PLICITLY dealing with text categorization are presented, together with some experimental results; Section VI draws conclusions and points to future work.

II. AGENT-BASED SYSTEMS FOR INFORMATION RETRIEVING

Several multiagent systems have been proposed to support the user in the task of retrieving information from the web. Among them let us recall NewT [16], Letizia [13], Web-Watcher [3], and SoftBot [7].

NewT [16] is designed as a collection of information filtering interface agents. Interface agents are intelligent and autonomous computer programs, which learn users' preferences and act on their behalf. This system uses a keyword-based filtering algorithm. The learning mechanisms used are relevance feedback and genetic algorithms.

Letizia [13] is a user interface agent that assists a user browsing the World Wide Web. The model adopted by this system is that the search for information is a cooperative venture between the human user and an intelligent software agent. Letizia and the user both browse the same search space of linked web documents, looking for "interesting" ones.

WebWatcher [3] is an information search agent that follows web hyperlinks according to users' interests, returning a list of interesting links to the user.

In contrast to systems for assisted browsing or information retrieval, the SoftBot [7] accepts high level user goals and dynamically synthesizes the appropriate sequence of Internet commands using a suitable ad-hoc language to satisfy those goals.

Finally, let us point out that current web search engines basically rely only on purely syntactical textual information retrieval. There are only a few approaches that try to integrate a set of different and specialized sources, but unfortunately it is very difficult to maintain and to develop this kind of systems [9].

III. TEXT CATEGORIZATION

The main goal of text categorization is to classify documents into a set of predefined categories. Each document can be in multiple or exactly one category. Using machine learning, the objective is to learn classifiers from examples, which

perform the category assignments automatically, according to a supervised learning approach.

A major characteristic, or difficulty, of text categorization problems is the high dimensionality of the feature space. The native feature space consists of the unique terms (words or phrases) that occur in documents, which can be tens or hundreds of thousands of terms, even for a moderate-sized text collection. This is prohibitively complex for many learning algorithms. Thus, the first step in text categorization is to transform documents into a representation suitable for the underlying learning algorithm and the classification task.

After counting the number of occurrences of a word w in a document –giving rise to an unordered *bag of words* [1]– suitable stemming algorithms [15] are applied to avoid unnecessarily large feature vectors. Each distinct word stem w_i corresponds to a feature, with the number of occurrences (in the entire document) of the word w_i as value. Words are considered as features only if they occur in the training data at least a predefined number of times except when they are considered as *stop-words* (like *and*, *or*, *is*, etc.).

To further reduce the number of considered terms, suitable feature selection methods can be applied. Automatic feature selection methods include the removal of non-informative terms according to corpus strategies, and the construction of new features which combine lower-level features (i.e., terms) into higher-level orthogonal dimension. Among different feature selection methods, let us recall document frequency, information gain, mutual information, a χ^2 statistic, and term strength (see [21] for a detailed comparison among them).

After selecting the terms, for each document a feature vector is generated, whose elements are the feature values of each term. A commonly used feature value is the TF (Term Frequency) \times IDF (Inverse Document Frequency) measure.

Among machine learning techniques applied to text categorization, let us cite multivariate regression models [19], k -Nearest Neighbor classification [20], Bayes probabilistic approaches [17], decision trees [12], neural networks [6], symbolic rule learning [14] and inductive learning algorithms [4].

IV. THE PACMAS ARCHITECTURE

PACMAS, which stands for Personalized Adaptive and Cooperative MultiAgent System, is a generic multiagent architecture, aimed at retrieving, filtering and reorganizing information according to the users' interests. PACMAS agents can be personalized, adaptive, and cooperative, depending on their specific role (see [2] for details).

PACMAS Macro-Architecture

The overall architecture (depicted in Figure 1) encompasses four main levels (i.e., information, filter, task, and interface), each being associated to a specific role. The communication between adjacent levels is achieved through suitable middle agents, which form a corresponding mid-span level.

Each level is populated by a society of agents, so that communication may occur both horizontally and vertically. The

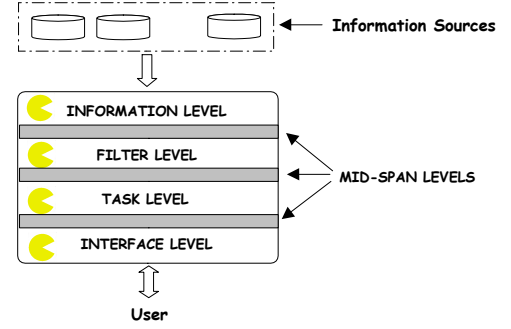


Fig. 1. The PACMAS Architecture.

former kind of communication supports cooperation among agents belonging to a specific level, whereas the latter supports the flow of information and/or control between adjacent levels through suitable middle-agents.

Information Level: At the information level, agents are entrusted with extracting data from the information sources. Each information agent is associated to one information source, playing the role of wrapper.

Filter Level: At the filter level, agents are aimed at selecting information deemed relevant to the users, and cooperate to prevent information from being overloaded and redundant. Two filtering strategies can be adopted: generic and personal. The former applies the same rules to all users; whereas the latter is customised for a specific user.

Task Level: At the task level, agents arrange data according to users' personal needs and preferences. In a sense, they can be considered as the core of the architecture. In fact, they are devoted to achieve users' goals by cooperating together and adapting themselves to the changes of the underlying environment.

Interface Level: At the interface level, a suitable interface agent is associated with each different user interface. In fact, a user can generally interact with an application through several interfaces and devices (e.g., pc, pda, mobile phones, etc.).

Mid-span Level: At the mid-span level, agents are aimed at establishing communication among requesters and providers. In the literature, several solutions have been proposed: e.g., blackboard agents, matchmaker or yellow page agents, and broker agents (see [5] for further details). In the PACMAS architecture, agents at the mid-span level can be implemented as matchmakers or brokers, depending on the specific application.

PACMAS Micro-Architecture

Keeping in mind that agents may be classified along several ideal and primary capabilities that they should embed, in our view agents are always autonomous and flexible. Moreover, we claim that personalization, adaptation and cooperation should be taken into account as a primary feature while depicting the characteristics of software agents.

Personalization: As for personalization, an initial user profile is provided in form of a list of keywords, representing users' interests. The information about the user profile is stored

by the agents belonging to the interface level. It is worth noting that, to exhibit personalization, filter and task agents may need information about the user profile. This flows up from the interface level to the other levels through the middle-span levels. In particular, agents belonging to mid-span levels (i.e., middle agents) take care of handling synchronization and avoiding potential inconsistencies. Moreover, the user behavior is tracked during the execution of the application to support explicit feedback, in order to improve her/his profile.

Adaptation: As for adaptation, a model centered on the concept of “mixtures of experts” has been employed. Each expert is implemented by an agent able to select relevant information according to an embedded string of feature-value pairs, features being selectable from an overall set of relevant features defined for the given application. The decision of adopting a subset of the available features has been taken for efficiency reasons, being conceptually equivalent to the one usually adopted in a typical GA-based environment [8], which handles also don't-care symbols. The system starts with an initial population of experts, during the evolution of the system further experts are created according to a covering, crossover, or mutation mechanism.

Cooperation: As for cooperation, agents at the same level exchange messages and/or data to achieve common goals, according to the requests made by the user. The most important form of cooperation concerns the “horizontal” control flow that occurs between peer agents. For instance, filter agents can interact in order to reduce the information overload and redundancy, whereas task agents can work together to solve problems that require social interactions to be solved.

V. PACMAS FOR TEXT CATEGORIZATION

In this section, we describe how the generic architecture has been customized to implement a system to perform text categorization.

The PACMAS Levels

In the following, we illustrate how each level of the architecture supports the implementation of the proposed application.

Information Level: At the information level, agents play the role of wrappers, each one being associated to a different information source. In particular, in the current implementation a set of agents wraps databases containing italian news articles¹. Furthermore, an agent wraps the adopted taxonomy that is a subset of the one proposed by the International Press Telecommunications Council² (a fragment is depicted in Figure 2).

Information agents are not personalized, not adaptive, and not cooperative (shortly \overline{PAC}). Personalization is not supported at this level, since information agents are only devoted to wrap information sources. Adaptation is also not supported, since we assume that information sources are invariant for the system and are not user-dependent. Cooperation is also not



Fig. 2. A fragment of the adopted (italian) taxonomy and its english translation.

supported by the information agents, since each agent retrieves information from different sources, and each information source has a specific role in the chosen application.

Filter Level: At the filter level, a population of agents manipulates the information belonging to the information level through suitable filtering strategies. First, a set of filter agents removes all non-informative words such as prepositions, conjunctions, pronouns and very common verbs by using a standard stop-word list. After removing the stop words, a set of filter agents, performs a stemming algorithm to remove the most common morphological and inflexional suffixes from all the words. Then, for each class, a set of filter agents selects the features relevant to the classification task according to the information gain method. Let us recall that information gain measures the number of bits of information obtained for category prediction by knowing the presence or absence of a term in a document.

Filter agents are not personalized, not adaptive, and cooperative (shortly \overline{PAC}). Personalization is not supported at this level, since all the adopted filter strategies are user-independent. Adaptation is also not supported, since all the adopted strategies do not change during the agents activities. Cooperation is supported by the filter agents, since agents cooperate continuously in order to perform the filtering activity.

Task Level: At the task level, a population of agents has been developed, each of them embedding a k NN classifier. Let us briefly recall that the k -nearest neighbor is a classification method based upon observable features. The algorithm selects a set which contains the k nearest neighbours and assigns the class label to the new data point based upon the most numerous class with the set. All the agents have been trained in order to recognize a specific class. Given a document in the test set, each agent, through its embedded k NN classifier, ranks its nearest neighbors among the training documents to a distance measure, and uses the most frequent category of the k top-ranking neighbors to predict the categories of the input document. Task agents are also devoted to measure the classification accuracy according to the confusion matrix [10].

Task agents are not personalized, adaptive, and cooperative (shortly \overline{PAC}). Personalization is not supported at this level,

¹More generally, they may wrap any web site containing news (e.g., online journals).

²<http://www.iptc.org/>

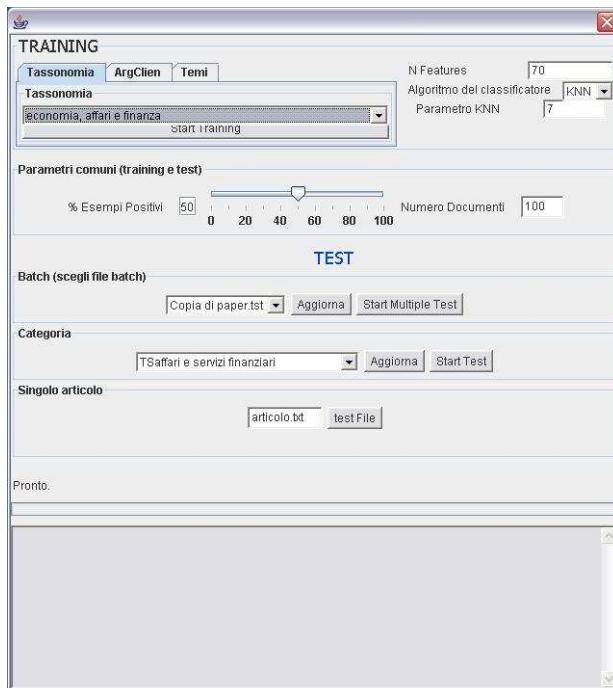


Fig. 3. Interface for the news classifying system.

since, in the current implementation, the adopted classification strategies are user-independent. Adaptation is supported by the task agents since they continuously adapt themselves to the underlying environment. Cooperation is supported by the task agents, since agents sometimes have to interact each other in order to achieve their own goals.

Interface Level: At the interface level, agents are aimed at interacting with the user. In the current implementation, agents and users interact through a suitable graphical interface that run on a pc. Interface agents are also devoted to handle user profile and propagate it by the intervention of middle agents.

Interface agents are personal, not adaptive, and not cooperative (shortly *PAC*). Personalization is required to allow each user the customization of her/his interface. In the current implementation, adaptation is not supported, but -at least in principle- an interface agent might adapt to the changes that occur in the preferences and interests of the corresponding user. Cooperation is not supported by agents that belong to this architectural level.

Table I summarizes the involved agents and their capabilities.

Training Task Agents

As for the training activity, task agents have been trained by a set of newspaper articles classified by human experts. Through a suitable graphical interface (see Figure 3), the user interacts with the interface agents setting her/him preferences. In particular, she/he can adjust the following parameters:

- the classification algorithm³;

³in the current implementation only *k*NN is supported

TABLE I
AGENTS ROLES AND CAPABILITIES

Agents	The ability of ...	Capabilities
information	wrapping databases containing news articles, and wrapping the taxonomy	<i>PAC</i>
filter	preprocessing the documents	<i>PAC</i>
task	classifying news articles	<i>PAC</i>
interface	interacting with the user	<i>PAC</i>
middle	allowing interactions among agents belonging to different levels	<i>PAC</i>

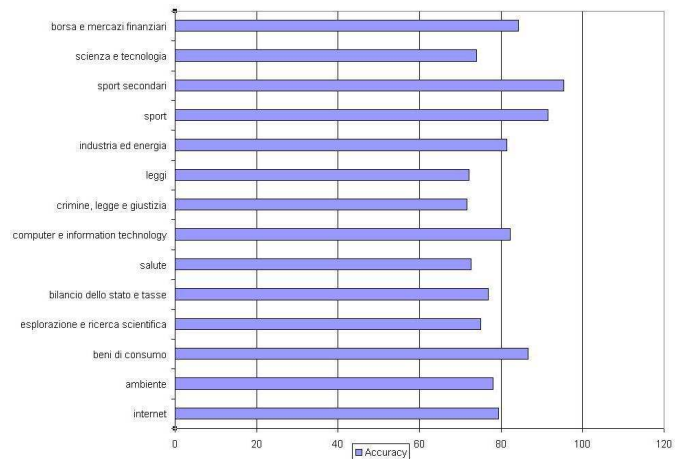


Fig. 4. Accuracy of the system.

- the number of documents forming the dataset;
- the training category;
- the percentage of positive examples;
- the number of features to be considered.

User choices are sent from the interface agent to the task level through the cooperation of the middle agent that belongs to the *task-interface* middle level (TI agent). The TI agent generates a task agent that embodies the corresponding classifier algorithm and asks it to perform the classification with the user preferences. The dataset needed for the classification is provided by information agents and subsequently pruned by the filter agents. After the classification activity, the task agent saves its own state in a suitable xml-like format in order to make it available for the test phase.

Experiments and Results

To evaluate the effectiveness of the system, several tests have been conducted using articles belonging to online newspapers. For each item of the taxonomy, a set of 200 documents has been selected to train the corresponding classifier, being *k*NN the adopted algorithm (with $k = 7$). To validate the training procedure, the system has been fed by the same dataset used in the training phase, showing an accuracy between 96% and 100%.

Then, random datasets for each category have been generated to test the performance of the system. The accuracy for fourteen categories is summarized in Figure 4. On the average, the accuracy of the system is 80.05%. Particular

care has been taken in limiting the phenomenon of “false negatives” (FN), which –nevertheless– had a limited impact on the percent of “false positives” (FP). In particular, the ratio $FN/(FN + FP)$ has been kept under 25% by weighting positive prototypes with an additional factor of 1.05 with respect to negative ones.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a system devoted to retrieve articles from Italian online newspapers, and classify them using suitable machine learning techniques. The system has been built upon PACMAS, a generic architecture designed to support the implementation of applications explicitly tailored for information retrieval tasks. PACMAS stands for Personalized, Adaptive, and Cooperative MultiAgent Systems, since PACMAS agents are autonomous and flexible, and can be personalized, adaptive, and cooperative depending on the implemented application. The categorization capability has been evaluated using several newspaper articles, showing an average accuracy of about 80%.

As for the future work, we are extending the system to handle with an automatic composition of the categories taken from the taxonomy in order to better fit the user profile.

VII. ACKNOWLEDGMENTS

We would like to thank Ivan Manca and Andrea Addis for participating in the development of the application.

REFERENCES

- [1] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *Information Systems*, 12(3):233–251, 1994.
- [2] G. Armano, G. Cherchi, A. Manconi, and E. Vargiu. Pacmas: A personalized, adaptive, and cooperative multiagent system architecture. In *Workshop dagli Oggetti agli Agenti, Simulazione e Analisi Formale di Sistemi Complessi (WOA 2005)*, November 2005.
- [3] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, pages 6–12, 1995.
- [4] W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In H.-P. Frei, D. Harman, P. Schauble, and R. Wilkinson, editors, *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 307–315. ACM Press, New York, US, 1996.
- [5] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 578–583, 1997.
- [6] A. S. W. Erik Wiener, Jan O. Pedersen. A neural network approach to topic spotting. In *Proceedings of 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 317–332, Las Vegas, US, 1995.
- [7] O. Etzioni and D. Weld. Intelligent agents on the internet: fact, fiction and forecast. *IEEE Expert*, 10(4):44–49, 1995.
- [8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [9] C. A. Knoblock, Y. Arens, and C.-N. Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada, 1994. University of Toronto Press.
- [10] R. Kohavi and F. Provost. Glossary of terms. *Special issue on applications of machine learning and the knowledge discovery process, Machine Learning*, 30(2/3):271–274, 1998.
- [11] J. Kramer. Agent based personalized information retrieval, 1997.
- [12] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, Las Vegas, US, 1994.
- [13] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.
- [14] I. Moulinier, G. Raskinis, and J.-G. Ganascia. Text categorization: a symbolic approach. In *Proceedings of 5th Annual Symposium on Document Analysis and Information Retrieval*, pages 87–99, Las Vegas, US, 1996.
- [15] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- [16] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In I. Press, editor, *9th Conference on Artificial Intelligence for Applications (CAIA-93)*, pages 345–352, 2003.
- [17] K. Tzeras and S. Hartmann. Automatic indexing based on Bayesian inference networks. In R. Korfhage, E. Rasmussen, and P. Willett, editors, *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval*, pages 22–34, Pittsburgh, US, 1993. ACM Press, New York, US.
- [18] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.
- [19] Y. Yang and C. Chute. An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems*, 12(3):252–277, 1994.
- [20] Y. Yang and X. Liu. A re-examination of text categorization methods. In M. A. Hearst, F. Gey, and R. Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US.
- [21] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning*, pages 412–420, 1997.

PRACTIONIST: implementing PRACTical reasONing sySTems

Vito Morreale*, Susanna Bonura*, Fabio Centineo*,
Alessandro Rossi*, Massimo Cossentino[†] and Salvatore Gaglio^{†‡}

*R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A.

[†]ICAR-Italian National Research Council

[‡]DINFO-University of Palermo

Abstract—One of the best known approaches to the development of rational agents is the BDI (Belief-Desire-Intention) architecture. In this paper we propose a new framework, PRACTIONIST (PRACTical reasONing sySTem), to support the development of BDI agents in Java (using JADE) with a Prolog belief base.

In PRACTIONIST we adopt a goal-oriented approach with a clear separation between the deliberation and the means-ends reasoning, and then between the states of affairs to pursue and the way to do it. Besides, PRACTIONIST allows developers to implement agents that are able to reason about their beliefs and the other agents' beliefs, expressed by modal logic formulas.

Our approach also includes a specific tool that provides the developer with the possibility to effectively monitor the components involved in the execution cycle of an agent.

I. INTRODUCTION

The Belief-Desire-Intention (BDI) architecture [1] derives from the philosophical tradition of practical reasoning first developed by Bratman [2], which states that agents decide, moment by moment, which actions to perform in order to pursue their goals. Practical reasoning involves two processes: (1) *deliberation*, to decide what states of affairs to achieve; and (2) *means-ends reasoning*, to decide how to achieve these states of affairs. Besides, in such a theory intentions are important, as they influence the selection of the actions to perform.

In the context of rational agents, the BDI model appears very attractive, because the abstractions of belief, desire and intention are quite intuitive. Moreover the model provides a clear functional decomposition that indicates what sort of subsystems might be required to build an agent. Nevertheless, the development of this abstract architecture involves several issues in efficiently implementing the deliberation process and the means-ends reasoning [3].

Moreover, since the BDI agent model suggests a declarative approach to represent the internal states, the debugging of BDI agents, the effective observation of their mental attitudes and execution flow are critical and difficult activities. Thus, having some development and debugging tools is crucial when implementing BDI agents, especially in real case scenarios or complex application domains.

Actually, several concrete implementations of the most known BDI agent architecture, the Procedural Reasoning System (PRS) developed by Georgeff and Lansky [4], have been

proposed in the literature. Among them, it is worth mentioning dMARS [5] developed at the Australian AI Institute, the UM-PRS implemented in C++ at the University of Michigan [6], and JAM [7], a Java version of PRS.

In order to enable the testing of BDI agents, the 3APL platform [8], an experimental multiagent platform, provides a graphical interface by which designers can develop, execute, and monitor the agents. JADEX, an add-on to the JADE platform [9] that supports the development of BDI agents, provides two tools as a support of the JADE introspector agent: the *debugger*, which allows the visualization and re-configuration of the internal BDI concepts, and the *logger agent*, which allows developers to detect the agent's sequence of outputs [10]. Finally, the JACK software [11], a commercial suite of tools with a programming language that extends the Java language with BDI features, provides an agent debugging environment, which allows inspection of messages and the internal execution states.

In several PRS-related BDI implementations, mental states, deliberation, and means-ends reasoning, when actually implemented, somewhat differ from their original meaning. As an example, often executing plans are considered as intentions. But intentions should be related to ends, while plans should be related to means to achieve such ends.

PRACTIONIST (PRACTical reasONing sySTem) is a new framework we have been developing, which adopts a goal-oriented approach and stresses the separation between the deliberation process and the means-ends reasoning. Indeed, the abstraction of goal is used to formally define both desires and intentions during the deliberation phase. Unlike some of existing BDI implementations, in our approach we actually adopt the plans as recipes to achieve the intentions. Besides, PRACTIONIST allows developers to implement agents that are able to reason about their beliefs and the other agents' (including humans') beliefs, since beliefs are not simple grounded literals or data structures but modal logic formulas [12].

Finally, our framework provides the developer with the PRACTIONIST Agent Introspection Tool, to entirely and easily monitor the components involved in the execution cycle of an agent. Throughout this paper we show how PRACTIONIST agents actually work by means of snapshots of our monitoring tool and through the well-known *blocks world* example. In this simple case study, we developed a *blocks world agent*, which,

		block7
block8	block3	block6
block5	block1	block10
block4	block2	block9
table1	table2	table3

Fig. 1. An initial situation for the blocks world problem.

starting from an initial situation (see figure 1), is requested (through an ACL message) to order some numbered blocks.

This paper is organized as follows: in section II we give a brief overview of the PRACTIONIST framework and the agent model; then in sections III to V we provide a brief description of the main agent components; finally in section VI we describe the execution flow of PRACTIONIST agents referring to their previously described components.

II. THE PRACTIONIST FRAMEWORK: AN OVERVIEW

The PRACTIONIST framework aims at supporting the programmer in developing agents (i) endowed with a symbolic representation about their internal states and environment, (ii) able to plan their activities in order to pursue some objectives, and (iii) provided with both proactive and reactive behaviours.

PRACTIONIST has been designed on top of JADE, a widespread platform that implements the FIPA specifications [13] and provides some core services, such as a communication support, interaction protocols, life-cycle management, and so forth. Therefore, the PRACTIONIST agents are executed within JADE containers and the main cycle is implemented by means of a JADE cyclic behaviour.

In PRACTIONIST, an agent is a software component with the following elements: (i) a set of *perceptors* able to listen to some relevant perceptions; (ii) a set of *beliefs*, which represents the information the agent has got about both its internal state and the external world; (iii) a set of *goals*, which are some objectives related to some states of affairs to bring about or actions to perform; (iv) a set of *plans* that are the means to achieve its intentions; (v) a set of *actions* the agent can perform to act over its environment; (vi) and a set of *effectors* that actually support the agent in performing its actions. The main components of PRACTIONIST agents are described in more details in the following sections.

The framework also provides developers with the PRACTIONIST Agent Introspection Tool (PAIT), a visual integrated monitoring and debugging tool, which supports the analysis of the agent's state during its execution. In particular, the PAIT can be suitable to display, test and debug the agents' relevant entities and execution flow. Each of these components can be observed at run-time through a set of specific tabs (see figure 2); the content of each tab can be also displayed in an independent window.

All the information showed at run-time could be saved in a file, providing the programmer with the possibility to perform an off-line analysis. Moreover, the PAIT provides a dedicated area for log messages inserted in the agent source code, according to the Log4j approach [14]. The usage of this console and the advantages it provides are described in more details in the following sections along with the agent's components.

III. BELIEFS

In general, the BDI model refers to beliefs instead of knowledge, as agents' information about the world is usually incomplete or incorrect, due to uncertainty and problems with perceptions and communication in their dynamic and possibly unpredictable environment [1]. Indeed, *beliefs* are not necessarily true, while *knowledge* usually refers to something that is definitely true [12]. According to this, an agent may believe true something that is false from another agent's and/or the designer's point of view, but in the BDI model the idea is just to provide the agents with a subjective window onto the world.

The PRACTIONIST framework adopts the common approach of modeling agents' beliefs by the *doxastic* modal logic, which is based on the axioms K, D, 4, and 5 (see [12] for more details). Thus, in our framework beliefs are expressed through the modal operator $Bel(\alpha, \varphi)$, whose arguments are the agent α (the believer) and what it believes (φ , the fact). Each fact φ may be believed true or false by a PRACTIONIST agent α , i.e.:

- $Bel(\alpha, \varphi)$: the agent α believes that φ is true;
- $Bel(\alpha, \neg\varphi)$: the agent α believes that φ is false.

Moreover, in order to assert that the agent α does not have any belief about φ , we defined the following operator:

$$Ubif(\alpha, \varphi) \Leftrightarrow \neg Bel(\alpha, \varphi) \wedge \neg Bel(\alpha, \neg\varphi)$$

Each fact can either be a closed formula of the classical modal logic or a belief of any agent. In other words, an agent may believe something (e.g. 'it is possible that it is raining in Rome'), or have *nested beliefs*, that is it may believe that some agent (even itself) believes something (e.g. 'the agent Jim believes that it is raining in Rome').

Finally, in PRACTIONIST agents it is possible to link an agent's beliefs to others' beliefs or other elements, obtaining new entailed beliefs, through the *belief formulas* (BFs). An example of belief formula follows:

$$Bel(tom, Bel(john, raining)) \wedge Bel(tom, trust(who : john)) \Rightarrow Bel(tom, raining)$$

Therefore BFs define relationships among two or more beliefs, allowing to infer new beliefs not explicitly asserted.

In regards to the architectural aspects, each PRACTIONIST agent is endowed with a Prolog belief base, where beliefs are asserted, removed, or entailed through the inference on the basis of KD45 rules and user-defined belief formulas. Therefore, in any moment the agent's belief set (BS) is composed of the beliefs that have been both directly asserted

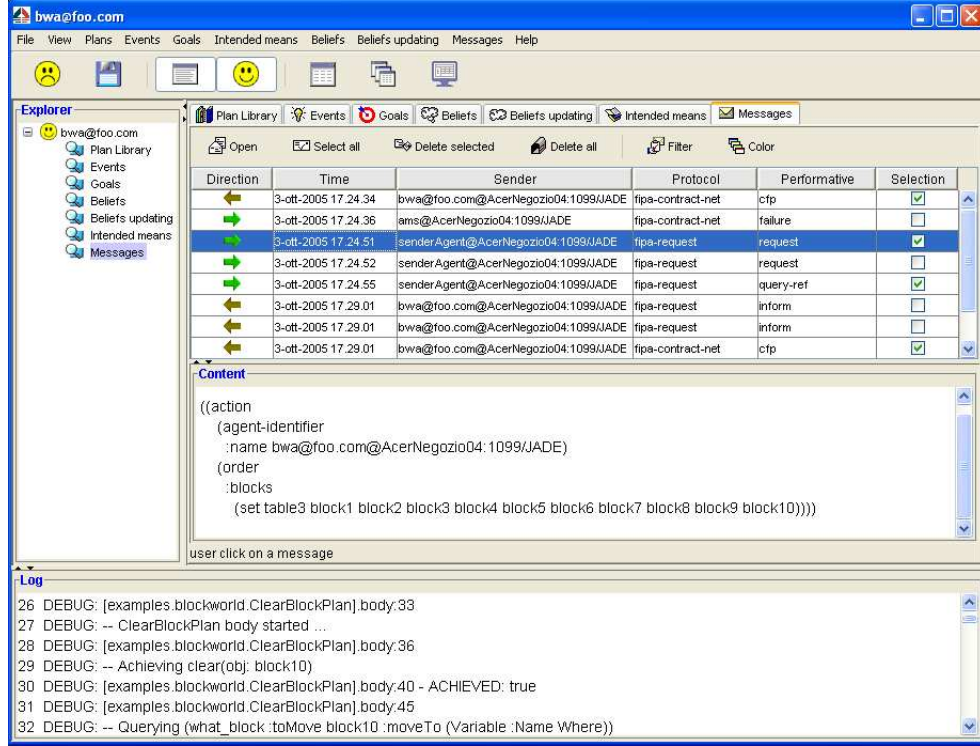


Fig. 2. The PRACTIONIST Agent Introspection Tool (PAIT).

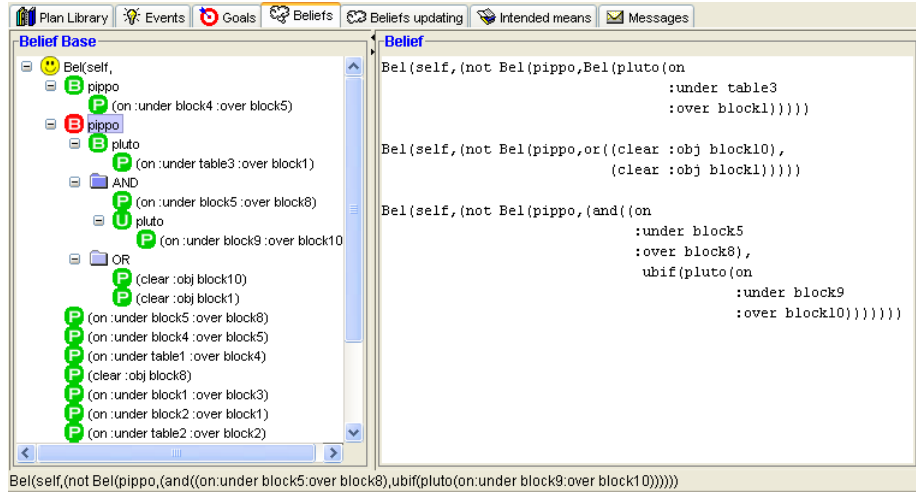


Fig. 3. The belief base view of PAIT.

and inferred by means of the BFs and the other built-in theorems.

Referring to the blocks world example, the initial set up of the blocks shown in figure 1 is graphically represented in PAIT in terms of *Bels* and *Ubifs* (figure 3). As it can be noticed, the structure of the belief base is represented as a tree, whose root refers to the current agent (i.e. *self*) and nodes refer to the believed facts. This structure lets the developer easily explore the belief base and select groups of beliefs ranging from the whole belief set (by selecting the root) to a specific belief (by selecting the corresponding leaf). Between them the

developer can choose an intermediate node in order to select the corresponding set of beliefs that share the same prefix.

Moreover, the icon of nodes represents the type of believed facts, such as *predicate* or *modal logic formula* (P), *Bel* (B), *Ubif* (U), while the color represents their truth value, that is *true* (green) and *false* (red).

As an example, selecting the node related to the agent *pippo*, the beliefs with

$$Bel(self, not(bel(pippo,$$

as prefix will be detailed on the right frame, as shown in figure

3. On the other hand, if the developer selects the leaf related to the agent *pluto* within the above-mentioned node, PAIT will show only the belief

*Bel(self, not(Bel(pippo, Bel(pluto, on(under :
table1, over : block3))))))*

It should be also noticed that this feature of the console is very useful when developing and testing agents, as it provides the user with a real-time snapshot of the agent's information attitudes.

IV. PLANS

In the PRACTIONIST framework plans represent an important container by which developers define the actual behaviours of agents. Therefore, each agent may own a declared set of plans (the *plan library*), which specify the activities the agent should undertake in order to pursue its intentions, or to handle incoming perceptions, or to react to changes of its beliefs.

Though the structure of plans can be defined by Java classes, the preferred approach relies on a declarative *plan description*, which specifies the complete set of information (the *plan slots*) used when actually executing the plans, as described in section VI. The complete list of such slots is reported in table I.

Thus, a plan represents a possible recipe to manage the trigger event; it may be related to a goal, an external event, or an event which notifies a change of the belief set. How to actually handle a certain event is reported within the body, which is an *activity*, that is a set of *acts*, such as *desiring* to bring about some states of affairs or to perform some action, *adding* or *removing* beliefs, *sending* ACL messages, *doing* an action and so forth.

It should be noticed that acts and actions are different, as one of the possible acts concerns with doing an action that lets the agent influence the environment, while other acts may produce internal effects only.

Regarding to the blocks world running example, one of the developed plans is the *TopLevelPlan*, which provides the agent with the capability of receiving and handling the request for ordering the blocks. In section VI we illustrate how the plans and their components are used during the execution flow and how the execution of plans affects the overall behaviour of PRACTIONIST agents.

V. ACTIONS

In PRACTIONIST, actions are described by tuples of four elements: (1) *arguments*, which are the objects each action acts over; (2) *results*, which are some kind of direct responses received from the environment; (3) *preconditions*, which should be satisfied before executing the action; and (4) *effects* (for both successfully and failing action execution), which are the state of affairs that will be true or false after executing the action (as long as preconditions are satisfied). It should be noticed that arguments and results are objects, while preconditions and effects are modal logic closed formulas.

TABLE I
THE STRUCTURE OF PRACTIONIST PLANS.

<i>Identifier</i>	Unambiguous (within each agent) identifier of plans
<i>Trigger event</i>	If this event matches the selected event, this plan can be activated. In this case the plan is defined as <i>practical</i>
<i>Context</i>	A modal logic formula that, when believed true by the agent, makes <i>applicable</i> a practical plan, so that the agent can select it to pursue its objectives
<i>Success condition</i>	When the agent believes that this condition holds, the plan ends with success, regardless its execution state
<i>Cancel condition</i>	When the agent believes that this condition holds, the plan ends with failure, regardless its execution state
<i>Body</i>	Set of acts that are performed during the execution of the plan. The body defines the actual behaviour of the plan
<i>Invariant</i>	Condition that must remain true during the execution of the plan. As soon as it becomes false (at least according to the agent's point of view), it will try to restore it
<i>Belief updates in case of success</i>	Effects of this plan, in terms of belief updates in case the plan ends with success
<i>Belief updates in case of failure</i>	Effects of this plan, in terms of belief updates in case the plan ends with failure

Actions are implemented in PRACTIONIST through Prolog structures or Java classes that include the above-mentioned elements. An example of action description from the blocks world agent follows:

```
action(move(block: Block, to: To),
  inputs: [Block, To],
  outputs: [],
  preconditions:
    [ on(over: Block, under: From),
      clear(obj: To), clear(obj: Block) ],
  success:
    [ -clear(obj: To),
      -on(over: Block, under: From),
      +clear(obj: From),
      +on(over: Block, under: To) ],
  failure: [])
```

It states that the action *move* takes two arguments as inputs (respectively the block to move and the block where it is moved over). Moreover, preconditions *on(over : Block, under : From)*, *clear(obj : To)*, and *clear(obj : Block)* must be satisfied before performing the action in order to have a proper execution. Finally, once the action has been executed, in case of success the agent will believe that both *clear(obj : To)* and *on(over : Block, under : From)* are false, while it will believe that both *clear(obj : From)* and

Fig. 4. An example of plan description: the *TopLevelPlan*.

on(over : Block, under : To) are true. Otherwise, in case of failure in executing the action, no update of the agent's beliefs has to be done.

Planning attitudes and the decision making of PRACTIONIST agents will rely on action description information, especially preconditions and effects.

VI. EXECUTION FLOW

In this section, we present the execution flow of PRACTIONIST agents in terms of relationships among the abstractions described above. Referring to figure 5, an agent cyclically performs the following steps:

- 1) it searches for stimuli from the environment through the perceptors that transform perceptions into *events* (we call them *external events*), which in turn are put into the *Event Queue* (in figure 6 a snapshot of the events tab of PAIT is shown), along with *internal events* (those generated by belief updates or goal commitments). In the PRACTIONIST console, the user could examine the current, the suspended and the handled events. Each event is tagged by some information about its arrival time and when it has been actually handled. Referring to the blocks world example, the agent will receive an ACL message, by which another agent requests it for putting the blocks in a specified order (see the selected message in figure 2). This component of the PAIT deals with the interactions established with other agents: all incoming and outgoing messages are registered in a single table, even though a complete description of messages can be shown in a different dialog box whose structure reflects the well-known FIPA ACL message. Besides, the messages can also be ordered (e.g. by their arrival time, direction, etc.) or filtered according to the relative performative;
- 2) it selects and extracts an event from the queue (*Event Selection*). In the blocks world example, let us suppose that the event corresponding to the above-mentioned

received message is extracted from the queue and then handled as follows;

- 3) it selects *practical* plans from the *Plan Library*, which are those plans whose trigger event matches the selected event (*Options* in figure 5); if the selected event is related to a goal and no plan has been triggered, an automatic planning is performed on the basis of available action descriptions in order to build a new dynamically-generated plan, which is able to pursue that goal (*Planning* in figure 5). If the planner is not able to figure out any plan, the calling plan will fail, then the selected event will have not been successfully managed. As stated, in the blocks world example we have defined a plan (i.e. *TopLevelPlan*) that will be activated by the above-mentioned message, as it has the following trigger event:

```
msg(request
  (action
    (agent-identifier :name bwa@foo.com)
    (order(blocks: BlockList))))
```

Once the plan is triggered, the following substitution is made:

```
BlockList = [ table3, block1, block2,
              block3, ..., block8,
              block9, block10]
```

- 4) among practical plans, the agent detects the *applicable* ones, which are those plans whose context is believed true by the agent. Then the agent builds the intended means (*Build Intended Means*), which represents the means the agent has just chosen and committed to in order to satisfy a goal, or to react to a perception or a change in its beliefs. Therefore, the intended means will contain the main plan and the other alternative practical plans (see figure 7).

If the event selected at the step (2) concerns with a goal, this means that some executing intended means has generated it after the deliberation phase (see below). Thus, the selected plan is put on top of such an intended means (figure 7a). On the other hand, in case of external events or belief update events, a new intended means stack is created (figure 7b).

The set of intended means stacks is monitored by means of a corresponding tab in PAIT (figure 8), which shows nested intended means through a tree data structure, used as an explorer to select the ones to trace. In particular, once an intended means is selected, its log messages (managed inside the plan's source code by the developer) will be shown in the frame on the right, along with the logs of nested intended means. The PRACTIONIST console makes the process of building intended means stacks easy to follow, providing the users with the opportunity to examine the entire means-ends reasoning;

- 5) all intended means stacks are concurrently executed in separate threads. In other words, the main plan at the top of each stack is extracted and then executed (see *Execute intended means* in figure 5). In section VI-A we provide more details on the execution of plans in PRACTIONIST agents, in terms of the several kind of

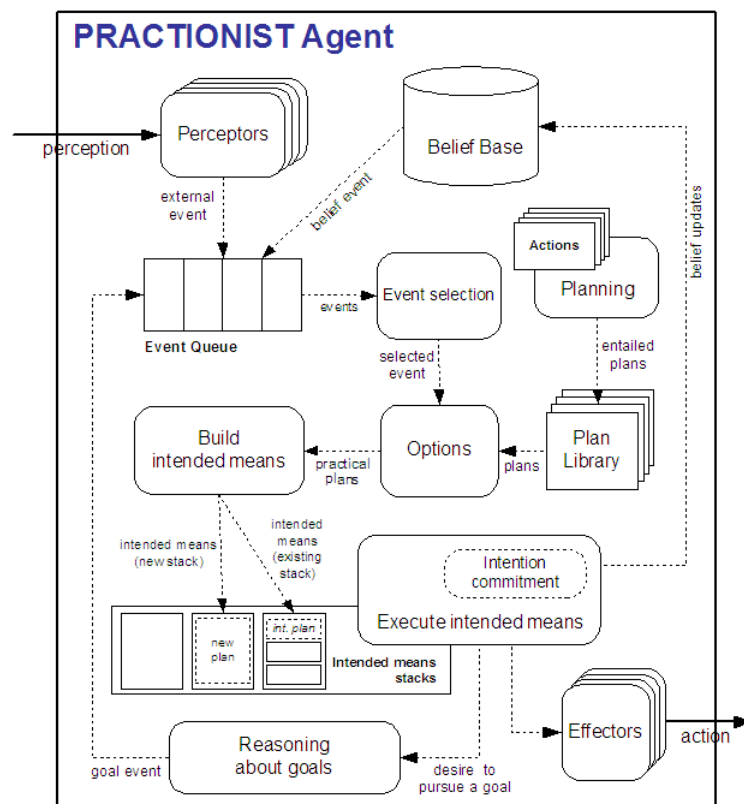


Fig. 5. PRACTIONIST agent architecture.

Plan Library
Events
Goals
Beliefs
Beliefs updating
Intended means
Messages

Discharge
Discharge automatically
Color

Type	Object	Arrive time	Handle time	Handled
GoalEvent	Achieve[(fix :under table3 :over bloc...	3-ott-2005 17.24.52	3-ott-2005 17.24.53	✓
MsgDelEvent	Msg[QUERY-REF :sender (agent-I...	3-ott-2005 17.24.52	3-ott-2005 17.24.55	✓
BeliefBaseUpdatedEvent	(fixing:obj block1)	3-ott-2005 17.24.54	3-ott-2005 17.25.06	✓
GoalEvent	Achieve[(clear :obj table3)]	3-ott-2005 17.24.55	3-ott-2005 17.24.57	✓
GoalEvent	Achieve[(clear :obj block9)]	3-ott-2005 17.24.57	3-ott-2005 17.24.59	✓
AchievedGoalEvent	Achieve[(clear :obj block10)]	3-ott-2005 17.24.59	3-ott-2005 17.25.11	
AchievedGoalEvent	Achieve[(clear :obj block9)]	3-ott-2005 17.25.02	3-ott-2005 17.25.13	

Fig. 6. The event queue view of PAIT.

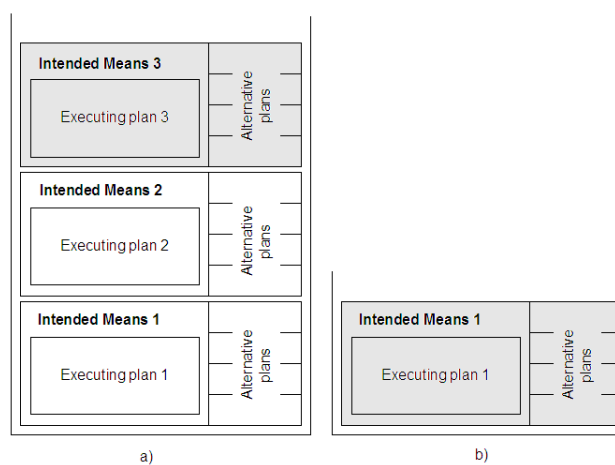


Fig. 7. Building of intended means stacks: a) the new intended means is put on top of executing stack; b) the new intended means is put in a new stack.

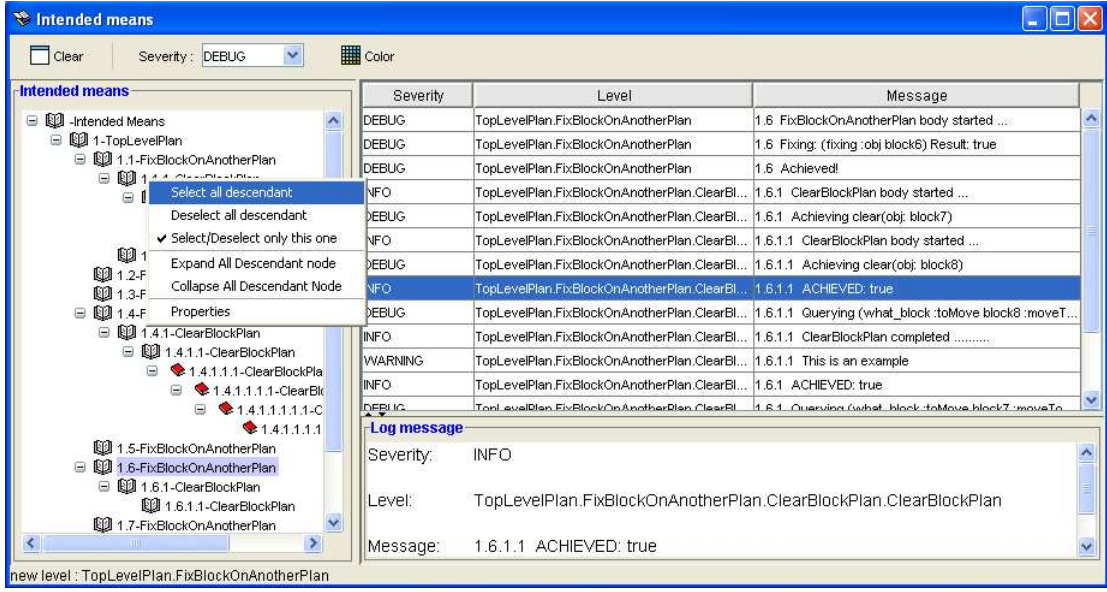


Fig. 8. Intended Means View.

acts the framework provides.

During the execution of the main plan, if the agent believes that its success condition (see table I) holds, that plan ends with success regardless its execution point and state. On the other hand, if the agent believes that the cancel condition is true, the plan ends with failure. Thus, referring to the *blocks world agent* and the *TopLevelPlan* described in figure 4, if during the execution of such a plan the required final order of blocks is achieved (for example due to some external reasons, e.g. other agents do the work of ordering the blocks), the agent will stop executing the plan, being successful in achieving its goal. Likewise, if the agent suddenly believes that the predicate *ordering* is false, it will stop executing the plan, but in this case failing in pursuing its objectives. Moreover, while executing the plan the agent checks the condition to be maintained (e.g. *ableToOrder(who : self)* in figure 4): if it is believed false, the agent will try to bring it about, by desiring and possibly intending to achieve it. If the agent succeeds in doing that, the plan will continue executing, otherwise it will fail. When the plan has completed its execution, the agent will update some beliefs, according to whether the plan has succeeded or not.

It should be noted that, when the executing plan fails, since a PRACTIONIST agent has a strong commitment to handling the selected event, it selects another plan from the above-mentioned alternative practical plans and checks whether it is applicable or not. Then, if some other applicable plan exists, the agent replaces the failed plan with it (which becomes the new main plan) in the executing intended means; otherwise, the whole intended means fails and in turn the plan below fails too.

A. Executing Plans

While executing its plans, the agent will have different behaviours according to the type of acts. The plans in turn will fail as soon as one of such acts fails. Some of the possible acts a PRACTIONIST agent can manage are:

- *do an action*: if its preconditions are satisfied (at least according to the agent's beliefs), the agent executes the action with the proper inputs and gets both the outputs and the general overcome of the action (that can be *true* or *false*). If the preconditions are not satisfied or the outcome is *false*, the action fails. Finally, the effects of such an action are applied to the belief base after the execution. It should be noted that actions are actually executed by some effectors. Thus, the agent will search for a proper effector that is able to execute an action and then delegate the actual execution to it;
- *add or remove beliefs*: as soon as it is executed, the corresponding *belief updated event* is generated. This event will be then handled in the following cycles;
- *query* over the belief base according to several criterion;
- *send an ACL message* to other agents;
- *desire* to bring about some state of affairs or perform an action, both expressed as goals. The agent processes such a desire in order to figure out whether it can be promoted to an intention or not, according to the processing described in section VI-B. If it can, a corresponding internal event is created, put in the event queue, and then considered at the step (2) in one of the following cycles. Then, the corresponding intended means is built and nested inside the one that contains the above-mentioned desire invocation.

After executing the nested intended means, if the agent does not believe that the goal has succeeded, the desire

act will fail; otherwise, the goal will be satisfied and the act will succeed.

- *wait for external messages*, by specifying an abstract structure (a template) of them. Thus, as soon as a message is received by the agent, if the message matches the template, this intended means can continue its execution;
- *wait for successful goals*, which lets the agent synchronize its activities with other intended means, but at the goal level. In other words, there is not an explicit synchronization among plans or intended means. Instead, some intended means and their executing plans can be directly synchronized with some ends (the intentions).

Several other acts are provided in PRACTIONIST in order to support the developers in actually implementing effective plans.

B. The Deliberation Process

In the PRACTIONIST framework, a goal is an objective to pursue and we refer to it as an abstraction to make the distinction between the state of affairs to be achieved and the way to achieve it. Besides, we use goals as a mean to transform desires into intentions through the satisfaction of some properties during the deliberation phase. Thus, in PRACTIONIST two families of goals were defined, as follows:

- *state goals*, which refer to some states of affairs the agent desires/intends to bring about, or cease, or preserve, or avoid. We provided PRACTIONIST agents with the capability of managing and reasoning about the following state goals: *achieve*, which represents what kind of world state to bring about; *cease* (as opposed to achieve), which represents a world state an agent wants to stop; *maintain*, which has the purpose to observe some world state and continuously re-establish this state when it does not hold; *avoid* (as opposed to maintain), which has the purpose to observe some world state and continuously prevent it;
- *perform goals*, which are not related to some world states but to some actions the agent desires/intends to perform.

In our framework states of affairs are represented by closed formulas of the FOL. Moreover, for each type of state goals, PRACTIONIST defines a success condition formula which is, in turn, defined by a modal logic closed formula that depends on its state of affairs.

We also defined the properties of *inconsistency* and *entailment* for goals. More precisely, we define a goal G_1 as *inconsistent* with a goal G_2 if and only if when G_1 succeeds, then G_2 fails. On the other hand, a goal G_1 *entails* a goal G_2 (or equivalently G_2 is *entailed by* G_1) if and only if when G_1 succeeds, then also G_2 succeeds.

These properties of goals are used by PRACTIONIST agents when reasoning about goals during the deliberation phase.

Thus, as described in section VI-A, an agent α may have the *desire* to pursue a goal G . Before committing to it, the agent will check if the goal G is inconsistent with some of current *active goals*, which are those goals the agent is already committed to. Then, if G is inconsistent with at least one of

		block10
		block9
		block8
		block7
		block6
		block5
		block4
		block3
		block2
		block1
table1	table2	table3

Fig. 9. The intended final situation in the blocks world problem.

those goals, G will remain just a desire and the agent will not "intend" it. On the other hand, if the goal G is not inconsistent with active goals, then it can be promoted to an *intention*.

But before moving to means-ends reasoning, the agent α will check if it believes that the goal G has already succeeded or if the goal G is entailed by some of current active goals. If so, there is no reason to really pursue the goal, that is the agent does not need to make any means-ends reasoning to figure out how to achieve such a goal, else the agent will try to figure out a set of plans to achieve this intention, and build the intended means, as explained in section VI.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we presented PRACTIONIST, a framework we have been developing for the implementation of agents according to the BDI model of agency.

PRACTIONIST implements the practical reasoning proposed by Bratman, trying to provide developers with usable abstractions and processing capabilities. In this direction, we believe that with respect to some of existing BDI agent implementations, our approach provides a more clear separation between the ends the agent wishes/wants to bring about and the means to do it. As a matter of fact, referring to the blocks world example, the agent is mainly programmed in terms of states to achieve instead of actions to perform, according to the philosophy of our framework. The figure 9 shows the final situation of blocks (the ends) *intended* and pursued by the agent through its plans (the means).

Moreover, our framework provide a very expressive way to represent and reasoning about beliefs through modal logic formulas.

We also presented the PRACTIONIST Agent Introspection Tool (PAIT), which supports the developer in the debugging of agents according to our model. We argue that such a tool is important especially when defining BDI agents in real case scenarios and in complex environments, due to the intrinsic declarative nature of mental attitudes when compared to the adopted imperative programming languages.

However, some further work should be done with respect to the several issues that a BDI model involves. Among them, our intention is to improve the execution flow by adding some functionalities like timing, new acts, and so on, that could help in the successful application of our framework in real problems.

ACKNOWLEDGMENT

This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

REFERENCES

- [1] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484. [Online]. Available: <http://citeseer.nj.nec.com/rao91modeling.html>
- [2] M. E. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.
- [3] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999. [Online]. Available: <http://jmvidal.cse.sc.edu/library/WeissBook/>
- [4] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proc. of AAAI-87*, Seattle, WA, 1987, pp. 677–682.
- [5] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge, "A formal specification of dMARS," ser. LNAI, M. P. Singh, A. Rao, and M. J. Wooldridge, Eds., vol. 1365. Springer, 1997, pp. 155–176.
- [6] J. Lee, M. J. Huber, P. G. Kenny, and E. H. Durfee, "UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications," in *Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS)*, Houston, Texas, 1994, pp. 842–849. [Online]. Available: citeseer.ist.psu.edu/lee94umprs.html
- [7] M. J. Huber, "Jam: A bdi-theoretic mobile agent architecture," in *Agents*, 1999, pp. 236–243.
- [8] M. Dastani, F. de Boer, F. Dignum, W. van der Hoek, M. Kroese, and J.-J. Meyer, "Implementing cognitive agents in 3APL," pp. 515–516.
- [9] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - a FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents*, 1999. [Online]. Available: <http://jmvidal.cse.sc.edu/library/jade.pdf>
- [10] L. Braubach, A. Pokahr, and W. Lamersdorf, "Jadex: A short overview," in *Main Conference Net.ObjectDays 2004*, 9 2004, pp. 195–207.
- [11] P. Busetta, R. Rnnquist, A. Hodgson, and A. Lucas, "Jack intelligent agents - components for intelligent agents in java," 1999.
- [12] B. F. Chellas, *Modal Logic: An Introduction*. Cambridge: Cambridge University Press, 1980.
- [13] "FIPA Abstract Architecture Specification," <http://www.fipa.org/specs/fipa00001/>, August 2001.
- [14] "Jakarta Log4J Homepage," <http://jakarta.apache.org/log4j/>. [Online]. Available: <http://jakarta.apache.org/log4j/>

A Discrete-Event Simulation Framework for the Validation of Agent-based and Multi-Agent Systems

Giancarlo Fortino, Alfredo Garro, Wilma Russo
DEIS – Università della Calabria, I-87036 Rende (CS), Italy
{g.fortino, garro, w.russo}@unical.it

Abstract

Simulation of agent-based systems is an inherent requirement of the development process which provides developers with a powerful means to validate both agents' dynamic behavior and the agent system as a whole and investigate the implications of alternative architectures and coordination strategies. In this paper, we present a discrete-event simulation framework which supports the validation activity of agent-based and multi-agent systems which are modeled and programmed as a set of event-driven agents by means of the Distilled StateCharts formalism and related programming tools. The simulation framework is equipped with a discrete-event simulation engine which provides support for the execution of agents by interleaving their events processing, the exchange of events among agents, the migration of agents, and the clustering of agents into agent servers interconnected by a logical network. Using this framework, an agent-based complex system can be easily validated and evaluated by defining a simulator program along with suitable test cases and performance measurements.

1. Introduction

Agent-based and multi-agent systems (MAS), like other complex software systems, must be tested and evaluated before being deployed [10]. Simulation of agent-based systems is an inherent requirement in all phases of the development process. Modeling and simulation help developers learn more about agents' interactive behavior and investigate the implications of alternative architectures and coordination strategies. In particular, discrete-event simulators are highly required for evaluating how complex agent-based systems work on scales much larger than those achievable in real testbeds.

Currently few development processes for agent-based systems which explicitly incorporate a simulation phase have been proposed. In [13] an integrated development environment for the engineering of MAS as Electronic Institutions is presented. An Electronic Institution is a

performative structure of multi agent protocols (or scenes) along with a collection of normative rules that can be triggered off by agents' actions. The development environment is composed of a set of tools supporting the design, validation through simulation, development, deployment and the execution of MAS as Electronic Institutions. Such a development environment is aimed at facilitating the iterated and progressive refinement of the development cycle of MAS. In particular, SIMDEI, a simulation tool, allows for the animation and analysis of the specification of the rules and protocols in an Electronic Institution. In [12, 15] a modeling and simulation framework (DynDEVS) for supporting the development process of MAS from specification to implementation is proposed. The authors advocate the use of controlled experimentation in order to allow for the incremental refinement of agents while providing rigorous observation facilities. The benefits of using modeling and simulation for the evaluation of cooperative agents is illustrated through a simple example based on the Contract Net Protocol. The exploited simulation framework is JAMES, a Java Based Agent Modeling Environment for Simulation, which aims at exploring the integration of the agents paradigm within a general modeling and simulation formalism for discrete-event systems. JAMES follows a formal approach for discrete-event simulation based on DEVS (Discrete Event Systems Specification) which allows to specify (atomic and coupled) models and execute them by sending typed messages between simulator objects. In [11] a logic based prototyping environment for multi-agent systems, CaseLP (Complex Application Specification Environment Based on Logic Programming) is presented. CaseLP integrates simulation tools for visualizing the prototype execution and for collecting the related statistics. The CaseLP visualizer tool provides documentation about events that happen at the agent level during the MAS execution. Developers according to their needs can instrument the code of some agents after it has been loaded by adding probes to the code of agents. In this way, events related to state changes and /or exchanged messages can be recorded and collected for on-line and/or off-line visualization. It is worth pointing out that from a

simulation point of view CaseLP is a time-driven centralized simulator with a global time known from all the agents in the system.

In this paper, we present a Java-based discrete-event simulation framework which supports the validation activity of agent-based and multi-agent systems which are modeled and programmed as a set of event-driven agents by means of the Distilled StateCharts formalism and the related programming tools [8]. The simulation framework is organized in four layers: (i) *low-level simulation framework*, which provides the basic mechanisms and classes to simulate general purpose systems; (ii) *agent platform*, which is built atop the low-level simulation framework and provides a distributed infrastructure formed by a network of interconnected agent servers; (iii) *ELA adapter*, which allows to map event-driven DSC-based lightweight agents onto the agent platform layer; (iv) *user*, which provides abstractions representing interacting users and users' behaviors. Using this framework, an agent-based complex system can be easily validated and evaluated by defining a simulator program along with suitable test cases and performance measurements.

The remainder of the paper is structured as follows. Section 2 overviews the Distilled StateCharts-based approach for the modeling and validation of agent-based system which adopts the proposed simulation framework as validation tool. In section 3, the simulation framework is described in detail whereas section 4 reports some results concerning with the performance evaluation of an agent-based e-Marketplace by means of the simulation framework. Finally conclusions are drawn and directions of future work delineated.

2. A Distilled StateCharts-based approach for the modeling and validation of agent-based systems: an overview

The Distilled StateCharts-based approach [5, 6], which aims at supporting the modeling and validation of agent-based and multi-agent systems, consists of the following phases (Fig. 1): High-Level Modeling, Detailed Design, Coding and Simulation.

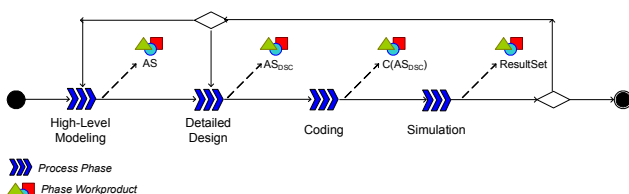


Figure 1. Process schema of the DSC-based approach.

The High-Level Modeling phase can be supported by well-established agent-oriented methodologies (such as the Gaia methodology [17]) which cover the phases of requirements capture, analysis and high-level design. The product of this phase is the agent-based system model (AS) defined as follows:

$$AS = \langle AT, LCL, act, serv, prot \rangle,$$

where:

AT (Agent Types) is the set of types of agents embodying activity, offering services and interacting with each other;

LCL (Logical Communication Links) is the set of logical communication channels among agent types which embody interaction protocols;

act: $AT \rightarrow activity\ description$ is the activity relation which associates one or more activities to an agent type;

serv: $AT \rightarrow service\ description$ is the service relation which associates one or more services to an agent type;

prot: $LCL \rightarrow interaction\ description$ is the protocol relation which associates an interaction protocol to a logical communication channel.

The Detailed Design phase is enabled by a Statecharts-based formalism, namely the Distilled StateCharts (DSC) [8], which supports the specification of the behavior of the agent types and the interaction protocols among the agent types of AS. In particular DSC allow for the specification of the behavior of lightweight agents (see §2.1) which are event-driven, single-threaded entities capable of transparent migration and executing chains of atomic actions. The DSC-based specification of an AS, denoted as AS_{DSC} , can be expressed as follows:

$$AS_{DSC} = \{Beh(AT_1), \dots, Beh(AT_n)\},$$

where $Beh(AT_i) = \langle S_{Beh}(AT_i), E_{Beh}(AT_i) \rangle$ is the DSC-based specification of the dynamic behavior of the i -th agent type. In particular, $S_{Beh}(AT_i)$ is a hierarchical state machine incorporating the activity and the interaction handling of the i -th agent type and $E_{Beh}(AT_i)$ is the related set of events to be handled triggering state transitions in $S_{Beh}(AT_i)$.

The Coding phase is carried out by using the Java-based Mobile Active Object Framework (MAO Framework) [8] and produces the work product $C(AS_{DSC})$ representing the code of AS_{DSC} . In particular, $Beh(AT_i)$ can be seamlessly translated into a composite object, which is the object-based representation of $S_{Beh}(AT_i)$, and into a set of related event objects representing $E_{Beh}(AT_i)$.

The Simulation phase is supported by MASSIMO, a Java-based discrete-event simulation framework for multi-agent systems (see §3). On the basis of the framework, a simulator program can be implemented and executed to obtain a ResultSet containing validation traces and performance parameter values. The validation of agent behaviors and interactions is carried out on execution

traces automatically generated, whereas the performance evaluation relies on the specific agent-based system to be analyzed; the performance evaluation parameters are therefore set ad-hoc. The ResultSet can also be used to feed back the High-level Modeling and Detailed Design phases.

2.1. The reference agent model

The agent model is based on the abstraction of event-driven state-based lightweight agent [7] which can be represented by the tuple:

$\langle \text{Id, Beh, DS, TC, EQ} \rangle$,

where:

- Id is the unique identifier of the agent;
- Beh is the DSC-based dynamic agent behavior;
- DS is the data space hierarchically organized of the agent;
- TC is the single thread of control supporting agent execution;
- EQ is the event queue of the agent containing received and to-be-processed events.

The event-driven state-based lightweight agent is programmed by specifying its Beh through the FIPA-compliant agent behavioral template [2], reported in Figure 2, which is a Distilled StateChart [8] consisting of a set of basic states (Initiated, Transit, Waiting, Suspended, and Active) and transitions labeled by events. In particular, the agent performs computations and interactions in the Active Distilled StateChart (ADSC) composite state, inside the Active state, which is to be refined by the agent programmer. The presence of the deep history connector (H^*) inside the Active state allows for a coarse-grained strong mobility-based agent migration [9]. An event reaction can produce computations, which can affect the DS, and/or the generation of one or more events, or a migration. While the reception of incoming events (or IN-events) is implicit and decoupled by the EQ, the transmission of events is explicitly carried out by means of the `generate(<event>(<parameters>))` primitive which allows to asynchronously raise outgoing events (or OUT-events). The execution semantics of the event-driven state-based lightweight agent are defined in terms of the Event Processing Cycle (EPC): the next available event is cyclically fetched from EQ and is passed to the Beh which can handle it so triggering one reaction. OUT- and IN-events are classified in:

- *internal* events, which can be defined at programming level for self-triggering active and/or proactive behavior. In the case of *internal* events, IN and OUT events coincide. In fact, an emitted *internal* event or

OUT-event is received as IN-event by the emitting agent itself.

- *management* events, which include requests and notifications of services at agent server level such as agent lifecycle management, creation, cloning, and migration.
- *coordination* events, which enable coordination acts between agents according to a specific coordination model. In this paper the considered coordination model is the asynchronous *Direct* model, even though the *Tuple-based* and the *Publish/Subscribe event-based* models could also be exploited as shown in [7].

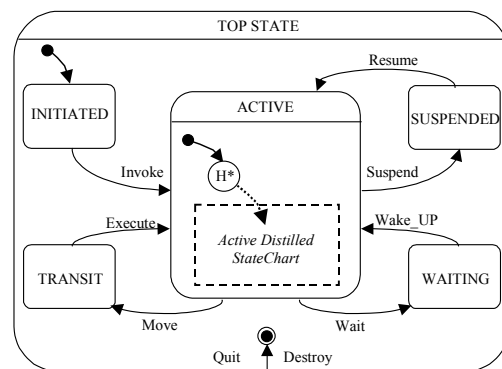


Figure 2. The FIPA-based template of the event-driven DSC-based lightweight agent.

In order to exemplify the DSC-based modeling of agent behavior, the specification of the ADSC of a mobile event-driven state-based lightweight agent is shown in Figure 3; Table 1 reports state variables, methods and events of the example agent specification. The agent overall goal is that of moving across a set of agent servers according to a predefined itinerary for monitoring a set of remote processes. In order to fulfill its goal, the agent alternates the following three phases:

- *Data acquisition*, which is performed by generating DataRequest coordination events targeting N different local agents which are controlling the local process. As soon as the monitoring data are collected (after the reception of all the DataReply coordination events), the internal Reply event is generated.
- *Data processing*, which is performed upon reception of the Reply event and carried out by means of the *process* method. It can also occur upon reception of the Process coordination event sent by another agent (e.g. the owner agent) if data are *enough* (the guard g holds), otherwise the agent returns in the substate of request which abandoned most recently.
- *Migration*, which depends on the data processing which, if successful, enables the agent to autonomously migrate to another site according to its itinerary; otherwise, the monitoring process is re-executed.

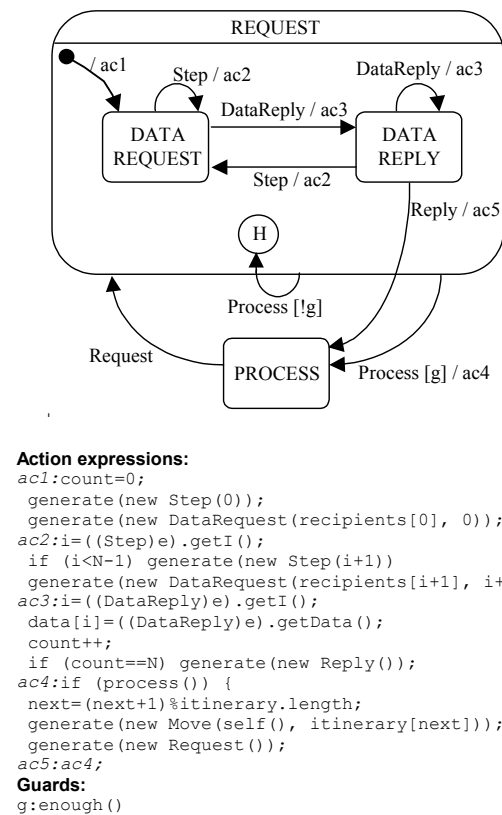


Figure 3. The Active Distilled StateChart of the example agent.

VAR	DESCRIPTION
<i>N</i>	Number of requests the agent issues to the local monitoring agents
<i>itinerary</i>	List of agent servers to be visited
<i>recipients</i>	List of identifiers of the interacting agents
<i>data</i>	Collector of the data coming from the replying agents
<i>next</i>	Index of the last visited agent server
<i>count</i>	Number of replies received in a monitoring cycle
<i>i</i>	Temporary integer variable
<i>e</i>	Reference to the last received event instance
METHOD	
<i>process</i>	Specific method for processing data which returns true if the processing was successful
<i>enough</i>	Specific method for evaluating if there are enough data for processing
<i>self</i>	Method which returns the identifier of the agent
EVENT	
<i>Step</i>	Internal event pacing the generation of <i>DataRequest</i>
<i>DataRequest</i>	Coordination event of the asynchronous Direct model sent by the agent to the local monitoring agents for requesting data
<i>DataReply</i>	Coordination event of the asynchronous Direct model sent by a local monitoring agent for replying to <i>DataRequest</i>
<i>Reply</i>	Internal event indicating data gathering completion
<i>Process</i>	Coordination event enabling a forced processing
<i>Request</i>	Internal event activating a monitoring cycle

Table 1. State variables, methods and events of the example agent.

3. MASSIMO: a discrete-event simulation framework for MAS

The Multi-Agent Systems Simulation framework (MASSIMO) is a Java-based discrete-event simulation framework which allows for the validation and evaluation of:

- the dynamic behavior (computations, interactions, and migrations) of individual and cooperating agents;
- the basic mechanisms of the distributed architectures supporting agents, namely agent platforms;
- the functionalities of applications and systems based on agents.

The architecture of MASSIMO (Fig. 4) is composed of four basic layers:

(i) *Low-level simulation framework*, which provides the basic mechanisms and classes to simulate general purpose systems;

(ii) *Agent platform*, which is built atop the low-level simulation framework and provides a distributed infrastructure formed by a network of interconnected agent servers;

(iii) *ELA adapter*, which extends the MAAF (Mobile Agent Adaptation Framework) [8] and allows to map event-driven DSC-based lightweight agents, provided by the MAO Framework, onto the agent platform.

(iv) *User*, which provides abstractions representing interacting users and users' behaviors.

In the subsections 3.1-3.4 the four layers are described in detail. In section 3.5, the basic structure of a MAS simulator program is exemplified.

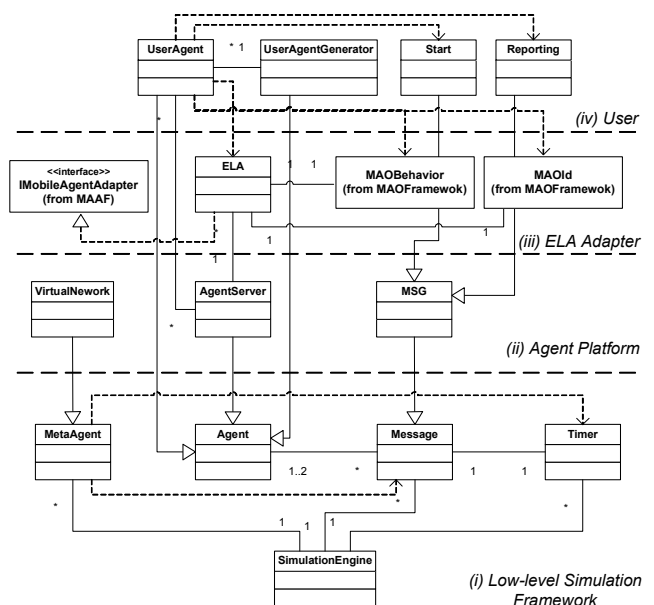


Figure 4. The architecture of MASSIMO.

3.1. The low-level simulation framework

The low-level simulation framework is composed of the following base Java classes which support agent-based programming and simulation of general-purpose systems:

- Agent, which represents a computational state-based agent communicating through asynchronous messages.
- MetaAgent, which represents a meta-level agent able to capture and constrain messages sent by computational agents or by other meta-agents.
- Message, which represents a message sent by an agent (source) to another agent (target).
- Timer, which is an object encapsulating a *Message* instance and a timeout. The message is delivered to its target at the timeout expiration.

The basic components of the simulation engine (Fig. 5) are:

- Global System Message Queue (GSMQ), which stores all the messages to be delivered.
- Global System Timer Queue (GSTQ), which stores all the timers ranked by timeout value.
- Simulation Clock (SC), which represents the simulation time. It is incremented every time that a timer expires.
- Filter (FT), which receives the messages generated by the computational agents and insert them into GSMQ if they are not subjected to the meta-level agent capture; otherwise FT forwards the messages to their associated meta-agents.
- Scheduler (SD), which cyclically extracts a message from GSMQ and dispatches it to the target agent. If there are not messages in GSMQ, SD forces a timer (the one with the lowest timeout) to fire and dispatches the associated message to its target.

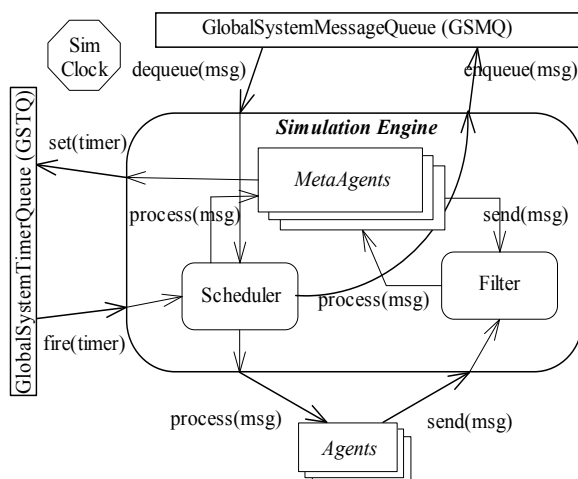


Figure 5. The architecture of the simulation engine.

3.2. The agent platform

The agent platform layer, which is built atop the low-level simulation framework, provides two basic abstractions: the AgentServer, which represents the infrastructure where event-driven lightweight DSC-based agents (ELAs) run, and the VirtualNetwork, which represents a network of hosts on which AgentServers can be mapped.

The AgentServer, which is an extension of the Agent class, provides the following functionalities:

- agent management lifecycle, which supports (de)registration and execution of ELAs;
- agent migration, which supports the migration of an ELA from one AgentServer to another;
- agent interaction, which supports the event-based interaction among ELAs;
- inter-agent-server service signaling.

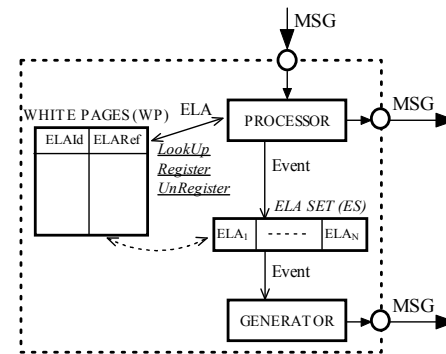


Figure 6. The architecture of the AgentServer.

The architecture of the AgentServer (Fig. 6) consists of the following components:

- *White Pages (WP)*, which keeps archived the ELAs running in the AgentServer. It consists of pairs $\langle \text{ELAId}, \text{ELARef} \rangle$, where *ELAId* is the ELA identifier and *ELARef* is either (i) the reference to the ELA identified by *ELAId* and belonging to the set of ELAs (*ES*) running in the AgentServer or (ii) the proxy of the ELA identified by *ELAId* and migrated to another AgentServer. A proxy is a triple $\langle \text{AS}, \text{MBX}, \text{active} \rangle$, where *AS* is the address of the AgentServer to which the ELA migrated, *MBX* is the ELA mailbox containing the events targeting the ELA during the ELA migration transitory, and *active* is a boolean variable indicating whether or not the forwarding activity of the proxy is on.
- *Processor*, which receives and processes incoming MSGs, extensions of the Message class, which can contain one of the following objects:
 - (i) *an Event targeting an ELA*. The ELA target of the Event is looked up and the Event passed to it if the ELA is present in the AgentServer; otherwise, the ELA

Proxy is returned and the Event is encapsulated in a MSG and the resulting MSG redirected to the AgentServer address contained in the proxy.

(ii) *a created ELA*. The created ELA is registered in the WP.

(iii) *an incoming migrating ELA*. The incoming ELA is registered in the WP. If it is not the first time that the ELA is hosted by the AgentServer, the previously left proxy is substituted by the incoming ELA.

(iv) *an outgoing migrating ELA*. The outgoing ELA is encapsulated in a MSG and the resulting MSG is transmitted to the target AgentServer. Finally, the outgoing ELA is unregistered from the WP and its associated Proxy is set.

(v) *an inter-AgentServer service message*. The basic service messages are those for the management of the MBX of the ELA proxy:

- GetMBX, which is a request issued by a remote AgentServer to activate the proxy and obtain the MBX of an ELA which migrated from the AgentServer to the remote AgentServer. Upon reception of GetMBX, the AgentServer first looks up the proxy of the ELA whose identifier is contained in the GetMBX; then, it retrieves the MBX associated to the ELA and, if the MBX is not empty, sends an MBX message containing the MBX to the remote AgentServer. Finally, the proxy forwarding is activated (active=true).
- MBX, which contains the mailbox of an ELA previously requested from a remote AgentServer by a GetMBX service request. Upon reception of an MBX message, the AgentServer looks up the ELA whose identifier is contained in the MBX message and, if the ELA is present in the AgentServer, encapsulates the events contained in the MBX message in Messages targeting the AgentServer itself and inserts them in the GSMQ. If the ELA is not present in the AgentServer, MBX is sent to the AgentServer where the ELA migrated if the proxy is on; otherwise, the events contained in the MBX message are inserted in the MBX of the ELA proxy.
- *Generator*, which processes the following events generated by the hosted ELAs:
 - (i) *Internal self-triggering Event*. The event is encapsulated in a MSG whose target is the AgentServer itself to which the MSG is then transmitted.
 - (ii) *External Event*. The event is encapsulated in a MSG whose target is the AgentServer hosting the ELA target of the event and the MSG is then transmitted to the target AgentServer.
 - (iii) *Creation Event*. The event contains the identifier and the dynamic behavior of an ELA created in the AgentServer. These parameters are used to create a new ELA agent which is then encapsulated in a MSG

whose target is the AgentServer itself to which the MSG is then transmitted.

(iv) *Timer Event*. The event is encapsulated in a MSG whose target is the AgentServer itself and the MSG then is encapsulated in a Timer which is set to the timeout contained in the timer event.

The VirtualNetwork, which is an extension of the MetaAgent class, is able to set Timers on transmitted MSGs. It relies on a graph-based network structure in which a network link is completely reliable and based on an end-to-end delay model by which the delay of event/message transmissions [3] and agent migrations [14] can be calculated. The calculated delay is used as timeout value of a Timer containing a MSG.

3.3. ELA adapter

The ELA (Event-drive Lightweight Agent) adapter (Fig. 7) allows to plug a MAOBehavior object encapsulating the DSC-based behavior of an event-driven lightweight agent into the simulation framework.

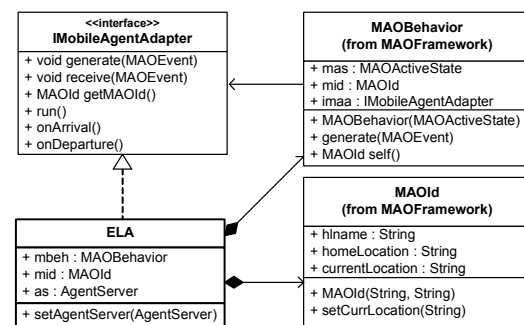


Figure 7. The ELA adapter layer.

The ELA class is an extension of the MAAF (Mobile Agent Adaptation Framework) [8] designed to provide the basic support for the adaptation of a MAOBehavior to a mobile agent class which is made available by a specific Java-based mobile agent platform. The ELA class implements the IMAOAgentAdapter interface and is associated with a MAOBehavior and a MAOId encapsulating the high-level agent identifier. IMAOAgentAdapter declares the following methods for adapting agent interaction, execution and migration:

- *receive*, which is invoked to pass MAOEvents to agents;
- *generate*, which interprets the MAOEvents generated within MAOBehavior and translates them into calls of platform-dependent methods;
- *run*, which is the method supporting agent execution;
- *onDeparture*, which is invoked just before the migration initiates;
- *onArrival*, which is invoked after the migration is completed.

To completely adapt an ELA to the agent platform layer the ELA class needs only to implement the methods *receive* and *generate*. The method *receive* is invoked by the AgentServer to deliver MAOEvents to ELAs. The method *generate*, which is invoked by the MAOBehavior, passes a MAOEvent to the AgentServer.

3.4. User

The User level makes it available two abstract classes UserAgent and UserAgentGenerator which are extensions of Agent. UserAgent represents a user directly connected to an AgentServer who can create, launch and interact with ELAs. UserAgentGenerator models the generation process of UserAgents. In particular, the UserAgentGenerator is able to create and start UserAgents according to a given logic (e.g. statistical distribution). Moreover, the Start message allows for the activation of a UserAgent or a UserAgentGenerator, whereas the Reporting message which targets a UserAgent contains a report sent from an ELA owned by the UserAgent.

3.5. Simulator programming

A MAS simulator can be programmed on the basis of the simulation entities described in the previous subsections: VirtualNetwork, AgentServer, ELA, UserAgent and UserAgentGenerator. A general simulator program can be constructed in the following steps:

1. creation of the VirtualNetwork;
2. creation of one or more AgentServers;
3. mapping of the created AgentServers onto distinct nodes of the VirtualNetwork;
4. creation of the ELAs that will not be created, directly or indirectly, by a UserAgent;
5. mapping of the created ELAs onto AgentServers;
6. creation of one or more UserAgentGenerators and/or one or more UserAgents. In the latter case, the created UserAgents are to be bounded to one or more AgentServers;
7. generation of the Start messages targeting the UserAgentGenerators and/or the UserAgents;
8. start of the simulation engine.

Figure 8 sketches the code of the simulator program of an example MAS. The MAS is composed of N stationary service agents (SA) distributed on N different AgentServer, a UserAgent (UA) which creates and launches a mobile agent (MA). MA travels along the N different AgentServers, interacts with the N SAs and, finally, comes back home by reporting to the UA.

```
//initialize the simulation engine
SimulationEngine.init();

//create an Homogeneous Small Network of N_AS+1 nodes
VirtualNetwork vn = new VirtualNetwork(N_AS+1, VirtualNetwork.HSN);

//add the VirtualNetwork to the set of MetaAgents
SimulationEngine.addMetaAgent(vn, MetaAgent.ALL_MSG);

//create N_AS agent servers
AgentServer [] ass = new AgentServer[N_AS];
String [] ass_url = new String[N_AS];
for (int i=0; i<N_AS; i++){
    ass_url[i] = "agentserver"+i;
    ass[i] = new AgentServer(ass_url[i], "typeX");
}

//map agent servers to network nodes
for (int i=0; i<N_AS; i++){
    vn.map(ass[i], i);
}

//create the service agents and map them to the agent servers
for (int i=0; i<NUM_AS; i++){
    ELA sa = new ELA(new MAOid(ass_url[i]+"#sa", null,
        ass_url[i]), new MAOServiceActiveState(100));
    Msg msg = new Msg(ass[i], ass_url[i], ass_url[i],
        Msg.AGENT_CREATION, sa );
    ass[i].process(msg);
}

//create the user agent and map it to the N_AS node
UserAgent ua = new UserItineraryAgent("useragentX", ass_url);
vn.map(ua, N_AS);

//send the Start message to the UserAgent
Agent.send(new Start(ua));

//start the simulation of a duration of 1000000
SimulationEngine.start(1000000);
```

Figure 8. An example MAS simulator program.

4. Performance evaluation of a consumer-driven agent-based e-Marketplace

An Agent-based e-Marketplace (AEM) is a distributed multi-agent system formed by both stationary and mobile agents which provide e-Commerce services to end-users within a business context. AEMs are distributed large-scale complex systems which require tools which are able to analyze not only the AEM at the micro level, i.e. behaviors and interactions of their constituting agents, but also the AEM at the macro level, i.e. the overall AEM dynamics. Although useful insights about AEM micro and macro levels can be acquired by playing e-Commerce simulation games and, then, analyzing the obtained results, or by evaluating real e-Commerce applications, discrete-event simulators are essential for evaluating how AEMs work on scales much larger than that achievable in games or in applications which involve humans. This section shows the application of the proposed discrete-event simulation framework to the analysis of micro level issues of a consumer-driven AEM, i.e. an e-Marketplace in which the exchange of goods is driven by the consumers that wish to buy a product.

4.1. An Agent-based Consumer Driven e-Marketplace model

The modeled AEM, inspired by the systems presented in [1] and [16], consists of a set of stationary and mobile agents which provides basic services for the searching, buying, selling, and payment of goods.

The identified types of agents are:

- *User Assistant Agent* (UAA), which is associated with users and assists them in: (i) looking for a specific product that meets their needs; (ii) buying the product according to a specific buying policy.
- *Access Point Agent* (APA), which represents the entry point of the e-Marketplace. It accepts requests for buying a product from a registered UUA.
- *Mobile Consumer Agent* (MCA), which is an autonomous mobile agent dealing with the searching, contracting, evaluation, and payment of goods.
- *Vendor Agent* (VA), which represents the vendor of specific goods.
- *Yellow Pages Agent* (YPA), which represents the contact point of the distributed Yellow Pages Service (YPS) providing the location of agents selling a given product. The organization of the YPS can be: (i) *Centralized* (C), each YPA stores a complete list of Vendor Agents; (ii) *One Neighbor Federated* (1NF), each YPA stores a list of VAs and keeps a reference to only another YPA; (iii) *M-Neighbors Federated* (MNF), each YPA stores a list of VAs and keeps a list of at most M YPAs.
- *Bank Agent* (BA), which represents a reference bank supervising money transactions between MCAs and VAs

The identified types of interactions between the agent types are described below by relating them to the system workflow triggered by a user's request:

1. *Request Input* (UAA→APA): the UAA sends a request to the APA containing a set of parameters selected by the user for searching and buying the desired product, i.e. the product description (*Prod_Desc*), the maximum product price (P_{MAX}) the user is willing to pay, and the type of buying policy (*BP*).
2. *Service Instantiation* (APA→MCA): the APA creates a specific MCA and provides it with the set of user's parameters, the type of searching policy (*SP*), and the location of the initial YPA to be contacted. Upon creation, the MCA moves to the initial YPA location.
3. *Searching* (MCA↔YPA): the MCA requests a list of locations of VAs selling the desired product to the YPA. The YPA replies with a list of VA locations and, possibly, with a list of linked YPA locations.
4. *Contracting & Evaluation* (MCA↔VA): the MCA interacts with the found VAs to request an offer (P_{offer}) for the desired product, evaluates the received offers, and selects an offer, if any, for which the price is acceptable (i.e., $P_{offer} \leq P_{MAX}$) according to the type of *BP*.

5. *Buying* (MCA↔VA↔BA): the MCA moves to the location of the selected VA and pays for the desired product using a given amount of e-cash (or bills) triggering the following money transaction: (i) the MCA gives the bills to the VA; (ii) the VA sends the bills to a BA; (iii) the BA validates the authenticity of the bills, disables them for re-use, and, finally, issues an amount of bills equal to that previously received to the VA; (iv) the VA notifies the MCA.

6. *Result Report* (MCA→UAA): the MCA reports the buying result to the UUA.

4.2. Agent behaviors

A model of MCA is defined on the basis of the tuple:
 $\langle SP, BP, TEM \rangle$,

where:

- *SP* is a searching policy in {ALL, PA, OS}:
 - a. *ALL*: all YPAs are contacted;
 - b. *Partial* (PA): a subset of YPAs are contacted;
 - c. *One-Shot* (OS): only one YPA is contacted.
- *BP* is a buying policy in {MP, FS, FT, RT}:
 - a. *Minimum Price* (MP): the MCA first interacts with all the VAs to look for the best price of the desired product; then, it buys the product from the VA offering the best acceptable price;
 - b. *First Shot* (FS): the MCA interacts with the VAs until it obtains an offer for the product at an acceptable price; then, it buys the product;
 - c. *Fixed Trials* (FT): the MCA interacts with a given number of VAs and buys the product from the VA which offers the best acceptable price;
 - d. *Random Trials* (RT): the MCA interacts with a random number of VAs and buys the product from the VA which offers the best acceptable price.
- *TEM* is a task execution model in {ITIN, PAR}:
 - a. *Itinerary* (ITIN): the *Searching* and *Contracting & Evaluation* phases are performed by a single MCA which fulfils its task by sequentially moving from one location to another;
 - b. *Parallel* (PAR): the *Searching* and *Contracting & Evaluation* phases are performed by a set of mobile agents in a parallel mode. In particular, the MCA is able to generate a set of children (generically called workers) and to dispatch them to different locations; the workers can, in turn, spawn other workers.

Thus, each one of the defined models implements the product buying service differently.

An MCA task execution model is chosen by the Access Point Agent (APA) when it accepts a user input request; the choice can depend on the $\langle SP, BP \rangle$ pair selected by the user and on the e-Marketplace characteristics. If the

chosen task execution model is of the *Parallel* type then the MCA is named PCA (*Parallel Consumer Agent*) otherwise if the chosen task execution model is of the *Itinerary* type then the MCA is named ICA (*Itinerary Consumer Agent*). Therefore, a PCA model is defined by a triple $\langle SP, BP, PAR \rangle$ whereas an ICA model is defined by a triple $\langle SP, BP, ITIN \rangle$.

The DSC-based behavior of the PCA models is reported in [4] whereas the DSC-based behavior of the ICA models, that can be seen as a particular case of the PCA behaviour, is described in detail in [6].

4.3. Simulation parameters and results

The goal for which the simulation phase was performed is twofold:

- to validate the behavior of each type of agent, the different models of MCA agents on the basis of the different YPS organizations, and the agent interactions.
- to gain a better understanding of the effectiveness of the simulation for evaluating MAS performances.

In order to analyze and compare the MCA models, the Task Completion Time (T_{TC}) parameter was defined as follows: $T_{TC} = T_{CREATION} - T_{REPORT}$ where, $T_{CREATION}$ is the creation time of the MCA and T_{REPORT} is the reception time of the MCA report. The simulation scenario was set up as follows:

- each stationary agent (UAA, APA, YPA, VA, BA) executes in a different agent server;
- agent servers are mapped onto different network nodes which are completely connected through links having the same characteristics. The communication delay (δ) on a network link is modeled as a lognormally distributed random variable with a mean, μ , and, a standard deviation, σ [3];
- each UAA is connected to only one APA;
- the price of a product, which is uniformly distributed between a minimum (PP_{MIN}) and a maximum (PP_{MAX}) price, is set in each VA at initialization time and is never changed; thus the VAs adopt a fixed-pricing policy to sell products;
- each YPA manages a list of locations of VAs selling available products.
- an UAA searches for a desired product, which always exists in the e-Marketplace, and is willing to pay a price P_{MAX} for the desired product which can be any value uniformly distributed between PP_{MAX} and $(PP_{MAX} + PP_{MIN})/2$.

Simulations were run by varying the organization of the Yellow Pages (C, 1NF and 2NFBT), the number of YPA agents in the range [10..1000] and the number of VA agents in the range [10..10000]. These ranges were chosen for accommodating small as well as large e-Marketplaces. The duration of the performed simulations were set so as

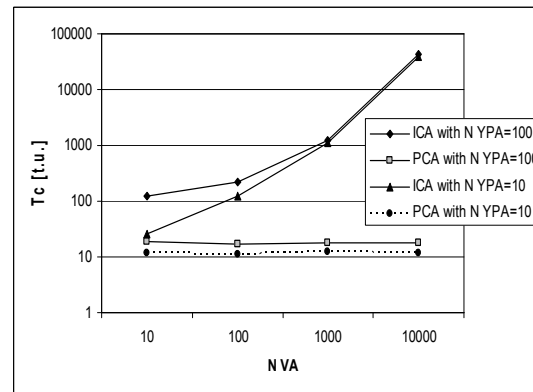


Figure 9. Performance evaluation of the $\langle ALL, MP, TEM \rangle$ models for an e-Marketplace with $YPS=2NFBT$, $N_{YPA}=\{10, 100\}$ and variable N_{VA} .

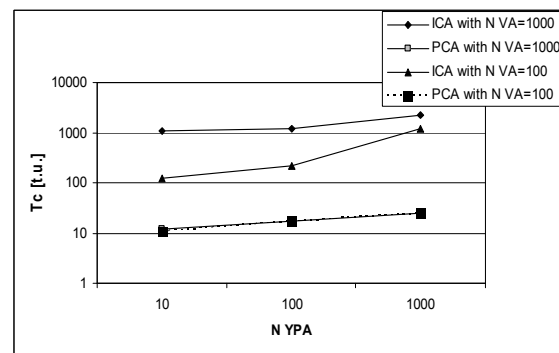


Figure 10. Performance evaluation of the $\langle ALL, MP, TEM \rangle$ models for an e-Marketplace with $YPS=2NFBT$, $N_{VA}=\{100, 1000\}$ and variable N_{YPA} .

to allow for the completion of the buying task carried out by the MCA. The results obtained from the simulations made it possible to:

(a) evaluate which task execution model is more appropriate with respect to SP and BP policies (see §4.2) and for the characteristics of the e-Marketplace;

(b) validate the analytical model proposed in [16] regarding the sequential and parallel dispatching of mobile agents.

With respect to point (a), the ICA performs better than the PCA in the following cases:

- $SP = \{ALL, PA, OS\}$, $BP = FS$, $YPS = \{C, 1NF\}$;
- $SP = \{PA, OS\}$, $BP = FS$, $YPS = 2NF$.

Thus, the APA can choose the itinerary task execution model if such cases occur.

With respect to point (b), the performance evaluation focused on $\langle ALL, MP, TEM \rangle$ models (see §4.2) as these are the only models of MCA which guarantee both a successful purchase and the best purchase since they are successful at identifying the VA selling the desired product at the minimum price. In order to compare the performances of PCA and ICA models, the results

obtained for the $\langle \text{ALL}, \text{MP}, \text{TEM} \rangle$ MCA models adopting a YPA organization of the 2NFBT type are reported in Figures 9 and 10. The results shown in Figure 9 were obtained with $N_{\text{YPA}} = \{10, 100\}$ and varying N_{VA} , whereas the results shown in Figure 10 were obtained with $N_{\text{VA}} = \{100, 1000\}$ and varying N_{YPA} . In agreement with the analytical model reported in [16], the PCA, due to its parallel dispatching mechanism, outperforms the ICA when N_{VA} and N_{YPA} are increased.

5. Conclusions

Validation tools for agent-based and multi-agent systems are highly required before such systems get completely deployed on distributed execution platforms. In order to support the validation phase of agent-based systems at different levels of granularity, from agent behaviors, protocols and services (micro-level) to global system behavior (macro-level), flexible and robust agent-oriented, discrete-event simulation frameworks should be carefully designed and developed. This paper has proposed a Java-based discrete-event simulation framework (MASSIMO – Multi-Agent Systems SIMulation framewOrk) which aims at supporting the validation activity of agent-based and multi-agent systems modeled and programmed by using an integrated approach based on the Distilled StateCharts formalism and the related programming tools. In particular, MASSIMO allows for the validation and the performance evaluation of the dynamic behavior (computations, interactions, and migrations) of individual and cooperating agents, the basic mechanisms of the distributed architectures supporting agents, namely agent platforms, and the functionalities of applications and systems based on agents. Finally, some results about the exploitation of MASSIMO for the validation of a consumer-driven agent-based e-Marketplace have been reported. Current efforts are being devoted to applying MASSIMO for the validation and performance evaluation of workflow instances enacted by agent-based enactment engines in the context of agent-based workflow management systems.

Acknowledgements

The work reported in this paper was partially supported by the M.I.U.R. (Italian Ministry of Instruction, University and Research) in the framework of the M.ENTE (Management of integrated ENTERprise) research project PON (N°12970-Mis.1.3).

References

- [1] D.J. Bredin, D. Kotz, and D. Rus. Market-based Resource Control for Mobile Agents. *Proc. of ACM Autonomous Agents*, May 1998.
- [2] FIPA Agent Management Support for Mobility Specification, DC00087C, 2002/05/10. <http://www.fipa.org>.
- [3] S. Floyd, V. Paxson, Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4), pp. 392-403, 2001.
- [4] G. Fortino, A. Garro, and W. Russo. An Integrated Approach for the Development and Validation of Multi Agent Systems. *Computer Systems Science & Engineering*, 20(4), pp.259-271, Jul. 2005.
- [5] G. Fortino, A. Garro, W. Russo. Modelling and Analysis of Agent-Based Electronic Marketplaces. *IPSI Transactions on Internet Research*, 1(1), pp. 24-33, Jan. 2005.
- [6] G. Fortino, A. Garro, W. Russo. E-commerce Services based on Mobile Agents. in Mehdi Khosrow-Pour, editor, *Encyclopedia of E-Commerce, E-Government and Mobile Commerce*, Idea Publishing Group, Hershey (PA), USA, 2006, to appear.
- [7] G. Fortino, W. Russo. Multi-coordination of Mobile Agents: a Model and a Component-based Architecture. *Proc. of the ACM Symposium on Applied Computing, Special Track on Coordination Models, Languages and Applications*, Santa Fe, New Mexico, USA, 13-17 Mar, 2005.
- [8] G. Fortino, W. Russo, and E. Zimeo. A Statecharts-based Software Development Process for Mobile Agents. *Information and Software Technology*, 46(13), pp. 907-921, Oct. 2004.
- [9] N.M. Karnik and A.R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(3), 52-61, 1998.
- [10] M. Luck, P. McBurney, and C. Preist. A Manifesto for Agent technology: Towards Next Generation Computing. *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 203-252, 2004.
- [11] M. Martelli, V. Mascardi, and F. Zini. Specification and Simulation of Multi-Agent Systems in CaseLP. *Proc. of Appia-Gulp-Prode Joint Conf. on Declarative Programming*, L'Aquila, Italy. M.C. Meo and M. Vilares-Ferro (eds), pp. 13-28, 1999.
- [12] M. Röhl, A. M. Uhrmacher. Controlled Experimentation with Agents - Models and Implementations. *Proc. of the 5th International Workshop "Engineering Societies in the Agents World"*, 20-22 October 2004, Toulouse, France.
- [13] C. Sierra, J. A. Rodríguez-Aguilar, P. Noriega, M. Esteva, and J. L. Arcos. Engineering Multi-agent Systems as Electronic Institutions. *Novática*, 170, July-August 2004.
- [14] M. Strasser and M. Schwehm, A Performance Model for Mobile Agent Systems, *Proc. of the Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, June 1997, 1132-1140.
- [15] A.M. Uhrmacher, M. Röhl, and B. Kullick. The Role of Reflection in Simulating and Testing Agents: An Exploration Based on the Simulation System James. *Applied Artificial Intelligence*, (9-10):795-811, October-December, 2002.
- [16] Y. Wang, K-L. Tan, and J. Ren. A Study of Building Internet Marketplaces on the Basis of Mobile Agents for Parallel Processing. *World Wide Web: Internet and Web Information Systems*, 5(1): 41-66, 2002.
- [17] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285-312, 2000.

On the Role of Simulation in the Engineering of Self-Organising Systems: Detecting Abnormal Behaviour in MAS

Luca Gardelli Mirko Viroli Andrea Omicini
DEIS, Alma Mater Studiorum–Università di Bologna,
via Venezia 52, 47023 Cesena, Italy
Email: {luca.gardelli, mirko.viroli, andrea.omicini}@unibo.it

Abstract—The intrinsic complexity of self-organising multi-agent systems calls for the use of formal methods to predict global system evolutions at early stages of the design process. In particular, we evaluate the use of simulations of high-level system models to analyse properties of a design, which can anticipate the detection of wrong design choices and the tuning of system parameters, so as to rapidly converge to given overall requirements and performance factors.

We take abnormal behaviour detection as a case, and devise an architecture inspired by principles from human immune systems. This is based on the TuCSoN infrastructure, which provides agents with an environment of artefacts—most notably coordination artefacts and agent coordination contexts. We then use stochastic π -calculus for specifying and running quantitative, large-scale simulations, which allow us to verify the basic applicability of our ID and obtain a preliminary set of its main working parameters.

I. INTRODUCTION

The trend in today information systems engineering is toward an increasing degree of complexity and openness, leading to rapidly changing requirements and highly dynamic environments. Since the cost of system management is becoming comparable to the cost of the system itself [1] we need new engineering methodologies and tools. In that sense social and natural sciences are recognised as rich sources of inspiration: e.g. the Autonomic Computing initiative tries to face complexity applying self-regulating mechanisms typical of biological processes [1], [2].

Self-organisation is a promising theoretical framework to reduce complexity of systems engineering. A system is said to be *self-organising* if it is able to re-organise itself upon environmental changes, by local interaction of its parts without any explicit pressure from the outside [3]. A system built according to this principle is usually able to perform complex tasks even though its components are far simpler when compared to a monolithic solution.

In this paper we continue along the line discussed in [4] in order to explore methodological aspects of the engineering of self-organising MASs. Because of the complexity inherent in these systems, and the difficulties in predicting their behaviour and properties, we find it crucial to exploit formal tools for simulating systems dynamics at the early stages of design. In the case of self-organising MASs, in fact, this approach appears to be almost unavoidable in order to nurture evolving

ideas and design choices, and to effectively tune parameters of the final system.

Among the various formal models to specify quantitative aspects of MASs we promote the use of the stochastic π -calculus process algebra [5]—see [4] for more details on that decision. This language is basically unexplored in the context of self-organising MASs: on the one hand, its simulation tools are relatively recent (see e.g. [6]), and on the other, it was primarily inspired by the need to model biological systems [7]. However, we show it can be fruitfully applied to the MAS paradigm as well: as far as stochastic aspects are concerned, the typical complexity of agent internal machinery can be suitably abstracted away, focussing instead on agent interactions and high-level activity changes.

For this purpose tools like SpiM (Stochastic PI-calculus Machine [6]) can be effectively used to track the dynamics of global system properties in stochastic simulations, validating design directions, inspiring new solutions, and determining suitable system parameters.

In this paper, we apply these ideas to the study of an intrusion detection (ID) infrastructure for open MASs. In particular our focus is on detecting anomalies in agents behaviour: the solution we describe here is inspired by principles of human immune system [8]. The infrastructure we devise is based on the TuCSoN technology [9]¹. This allows us to structure a MAS not only in terms of agents, but also with *tuple centres* [10] as *coordination artefacts* [11] and *agent coordination contexts* [12](ACC) as *boundary artefacts* [13]. Coordination artefacts are used to model resources in the environment on which agents act upon. ACCs specify and enact the access policies which each agent is subject to, and can be used to both (i) reify relevant information about the agent/artefacts interaction, and (ii) to deny malicious agents to access the MAS environment.

To evaluate the impact of different design choices and parameters of the ID infrastructure—such as inspection/detection rates, number of inspectors, and the like—we simulate the behaviour of different scenarios using SpiM specifications.

The rest of the article is structured as follows. In Section II we briefly highlight the main mechanisms and properties of intrusion detection and the human immune system. In Section

¹<http://tucson.sourceforge.net>

III we describe our general architecture for a MAS based on TuCSoN, and show how to develop an anomaly detection application. Section IV motivates the use of π -calculus and its stochastic extension, providing a simulation related to our ID domain using SpiM. Finally, Section V concludes by providing final remarks, and by listing some of the main directions for our future research.

II. INTRUSION DETECTION AND IMMUNE SYSTEM

In this section we first depict the main aspects of intrusion detection systems (IDSs), and then describe the structure and main principles regulating the human immune system. We are not concerned about accurately modelling or mimicking an immune system, instead we gather from there inspiration and principles for the engineering of secure self-organising applications.

A. Security and IDSs in Information Systems

There are several mechanisms used to protect information systems, but usually only the basic ones are implemented: (i) *authentication*, the identity is proved by the knowledge of a secret (e.g. password) or a physical unique property (e.g. fingerprint, retina, voice); (ii) *authorisation*: user actions on the system are constrained by its role and the policy linked to that role.

However, applications flaws typically cause these methods not to be sufficient alone [8]. For instance, protection at the host level is achieved using additional software such as firewalls, antivirus and many other specific tools. Furthermore authorisation policies cannot account for all possible sequences of actions, and a specific sequence might exhibit unexpected side-effects. In particular, it is in general too expensive and impractical (or even unfeasible) to intercept all emergent harmful paths at design-time.

Hence automated tools are a very useful support for the detection of malicious behaviour. In this direction, many efforts have already been spent in developing IDSs. An IDS tries to detect intruders and misuse of a target software system by observing users behaviour and deciding whether actions performed are symptomatic of an attack.

Efficiency of an IDS is evaluated by three parameters: *accuracy* (rate of false-alarms), *performance* (rate of audit processing), and *completeness* (rate of missed detection). *Misuse-based IDSs* try to detect intruders matching the actual user behaviour with known signatures of malicious behaviour. *Anomaly-based IDSs* try to detect behaviours that are different from what it is considered to be the normal activity. There has been already a lot of work for both approaches to deal with security issues either at the application, host and network level [8], [14], [15]. We are more concerned about the neat impact of such techniques, expressed in a stochastic manner, and how they can influence the design of a protection layer for a multi-agent system.

B. Human Immune System Overview

The human immune system protects the body against antigens, i.e. foreign molecules that trigger an immune response.

It is composed by reactive non-specific barriers such as the skin, and by active mechanisms, i.e. the *innate* and the *acquired* immune system. The innate immune system protects the body against known antigens, i.e. it is not adaptive, while the acquired immune system improves during individual life, discovering and memorising new antigens. The acquired immune system is composed of different types of cells: here we consider only *lymphocytes* since they are responsible for the main form of immune response. The mix of lymphocytes, which changes over time, defines the set of detectable antigens: this let the immune system cover a larger space of antigens—a phenomenon called *dynamic coverage*. Lymphocytes can become a “memory” if they bind to several antigens: this mechanism allow for a faster response if an antigen is met again.

It is easy to notice that we can define a parallel between human immune system and security for electronics systems. Static non-specific barriers are realised by authentication mechanism, firewalls etc. The innate immune system is mapped into authorisation policies, antivirus, trojan removers, and misuse-detection. The acquired immune system instead is mapped into anomaly-detection systems, which are able to discover new threats.

III. SECURITY IN MAS

In the following we describe our reference architecture for MASs, and discuss how to devise a security layer drawing useful concepts and techniques from previous works on IDS, as well as principles from the human immune system.

A. A General Architecture for MASs

In this section we describe a general architecture for MASs based on the TuCSoN coordination infrastructure [9], showing an approach to ensure security applying principles of the immune system.

We consider a system that provides agents with services encoded in terms of coordination artefacts, i.e. runtime abstractions encapsulating and providing a coordination service, to be exploited by agents in social contexts expressed by coordination rules and norms [11]. Following the general model for artefacts [16], a coordination artefact could be characterised by a usage interface, a set of operating instructions, and a coordination behaviour specification, which can be exploited by cognitive agents to rationally use a coordination artefact.

Accesses of agents to these resources is restricted by an authentication procedure. When an agent enters the system an authorisation policy limits its actions allowing the exploitation of a limited set of services and resources—e.g. those it has payed for. This is accomplished by the notion of Agent Coordination Context (ACC) [12], [13]. An ACC works as agent interface towards the environment: it is like a control room providing e.g. buttons and displays to an human, which are the only means by which he/she can interact with the environment. Thus, the ACC enables and rules the interaction between the agent and the environment [12], and it is then able to capture security and organisation aspects in MASs. In particular, the ACC is the right place to put authorisation

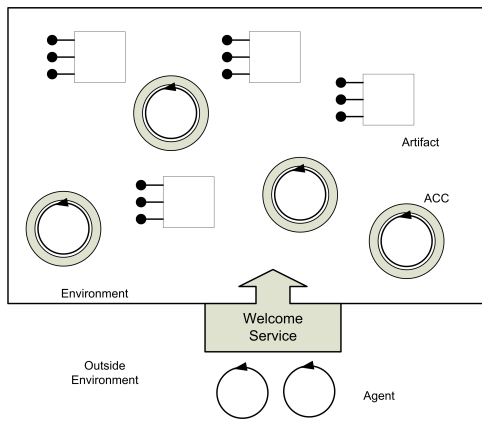


Fig. 1. A general architecture for a multi-agent system.

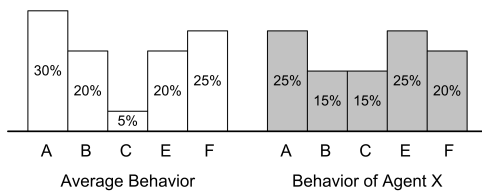


Fig. 2. The statistical approach for anomaly detection relies on the fact that the abnormal behaviour is distinguishable from the normal one. This can be restated in the *behaviour distribution of abnormal agents (right) is very different—at least for the critical actions—from the distribution of normal ones (left)*.

policies, typically specified using a Role Based Access Control model (RBAC) [17]. The whole architecture is depicted in Figure 1.

Usually the two mechanisms of authentication and authorisation are considered to guarantee a sufficient degree of protection. However we promote the idea—as pointed out in the intrusion detection community—that a dynamic system is better protected by additional dynamic mechanisms. Correspondingly, we introduce a layer aimed at detecting anomalies in agents behaviour inspired by the immune system as well as by previous works on IDSs [8], [14].

B. Anomaly Detection in MASs

Let us consider agents willing to exploit a specific artefact: we can trace their behaviour “for a while” and then create an average distribution of actions over that resource. We can consider that distribution to be the “normal” way for agents to interact with a particular artefact (Figure 2 left). From now on it is possible to observe an agent in order to build its particular distribution of actions: the deviation from the average distribution might be a symptom of intrusion or abnormal activity. For instance, if the action C is critical then the agent X (Figure 2 right) should be inspected to decide whether he is acting properly or might cause problems.

In order to support the process described above, a mechanism to observe the agent behaviour is needed, e.g. by using logging tools. This approach is valid under two hypotheses:

- 1) the number of traces is such that the data is statistically significant

- 2) the percentage of agents exhibiting abnormal behaviour is sufficiently low during the initial observation stage

The former hypothesis is quite straightforward and must hold true every time dealing with statistical data. The latter is very difficult to prove because one would have to check every action and in most situations this might be unfeasible or simply not affordable without automated tools. But we can reasonably argue that most of users of a system are interested in exploiting a resource rather than to hack it. Furthermore the second hypothesis affects the threshold value used to decide whether an agent is dangerous or not: this value is as reliable and accurate as the number of anomalies is low during the initial stage. Since we are performing the detection task on-line, the actions distribution might change over time, so we should also consider some tolerance ranges.

Referring to section III-A we describe now how the security layer fits into the general architecture. First we need a way to provide observability of the interaction agent-artefact: basically we have two choices:

- providing inspectors with access to ACCs
- reifying the action in a specific artefact for logging

Since we need two kinds of information, the average behaviour and the individual one, we can exploit both mechanisms: e.g. we can log the individual behaviour in the ACC while the average behaviour can be handled by another shared artefact. If agents privacy is not a main concern we can also reify both information using another artefact, which aggregates actions to define the average and individual signature.

Since observability mechanisms have been provided, now we need a set of agents whose goal is to observe periodically the use of resources. The actual *number of agents inspecting* and the *rate of inspection* are parameters of the security systems that should be dynamically tunable. When an agent detects abnormal behaviour, it should report it to the proper authority, which then decides whether to invalidate the ACC or not: invalidating the ACC means denying any further access to resources. This authority might be a human or artificial agent depending on the complexity and criticality of the decision process.

C. An Artefact for Logging Purpose

After we described the scenario, the architecture and the principles, we aim now at actually designing the artefact that could support the anomaly-detection task. For the sake of simplicity, our hypothesis is that there are no concerns of privacy for agents, i.e. it is not a problem to publish all agent actions: thus, in order to adopt the second approach described in section III-B, we only need to worry about the artefact design.

We use TuCSoN infrastructure [9] as our main source for artefacts: in particular, artefacts for logging are suitably-programmed ReSpecT tuple centres [10]. Hence what we need for a complete description of such an artefact are the templates for all the tuples used for the representation, and the ReSpecT specifications for handling the log tuples.

In particular, we only need three templates, respectively (i) for actions logging, (ii) for the individual behaviour signature, and (iii) for average behaviour signature.


```

%1) This reaction is executed only the first time the
% action identified by ActionID is performed for the
% first time over a specific artifact
reaction( out_r(action(AgentID, ActionID)), (
  in_r(action(AgentID, ActionID)),
  no_r(average_signature(ActionID, Count)),
  out_r(average_signature(ActionID, 1)),
  out_r(individual_signature(AgentID, ActionID, 1))
)).

%2) This reaction is triggered by an action identified by
% ActionID has already been performed from the agent
% identified by AgentID
reaction( out_r(action(AgentID, ActionID)), (
  in_r(average_signature(ActionID, Count)),
  in_r(individual_signature(AgentID, ActionID, Count)),
  in_r(action(AgentID, ActionID)),
  Count1 is Count + 1,
  Count1 is Count + 1,
  out_r(average_signature(ActionID, Count1)),
  out_r(individual_signature(AgentID, ActionID, Count1))
)).

%3) This reaction is triggered by an action identified
% by ActionID has already been performed from other
% agents, but it's the first time for the agent
% identified by AgentID
reaction( out_r(action(AgentID, ActionID)), (
  in_r(action(AgentID, ActionID)),
  in_r(average_signature(ActionID, Count)),
  no_r(individual_signature(AgentID, ActionID, Count)),
  Count1 is Count + 1,
  out_r(average_signature(ActionID, Count1)),
  out_r(individual_signature(AgentID, ActionID, 1))
)).

%4) This reaction should never be triggered in normal
% situation but it's useful to recover from inconsistencies
reaction( out_r(action(AgentID, ActionID)), (
  in_r(action(AgentID, ActionID)),
  no_r(average_signature(ActionID, Count)),
  in_r(individual_signature(AgentID, ActionID, Count)),
  Count1 is Count + 1,
  out_r(average_signature(ActionID, 1)),
  out_r(individual_signature(AgentID, ActionID, Count1))
)).

```

Fig. 3. ReSpecT specification for the logging artefact behaviour.

- 1) action(agentID, actionID)
- 2) individual_signature(agentID, actionID, count)
- 3) average_signature(actionID, count)

The system provides the first tuple each time an action is performed: this tuple triggers ReSpecT reactions which update signature tuples, then it is discarded. Each signature tuple is a counter for an action, which is incremented each time that action is performed by an agent: recording such information for each action makes it possible for an agent to build the actual signature.

Each time an artefact is introduced in the environment the signature of normal behaviour it is automatically built, which becomes significant only when the number of request exceeds a certain threshold. Figure 3 includes the whole specification².

Now that we have the anomaly detection support we must decide the parameters of the security systems: number of inspecting agents and rate of inspection—see section III-B for details. Given the computational cost of inspection and assuming a certain percentage of abnormally-behaving agents, we can simulate the system in order to predict the good values for system parameters. In the next section we describe Stochastic π -Calculus and how to exploit it for such purpose.

IV. SIMULATIONS IN π -CALCULUS

In this section we briefly introduce π -calculus [18] and its stochastic extension [5]. Then we present the results obtained by simulating a Stochastic Pi-Calculus specification using the Stochastic Pi Machine (SpIM) [6].

A. The π -Calculus

The π -calculus is a formal model developed to reason about concurrency [18]: it is a language for describing and analysing systems consisting of agents (or processes) which interact with each other. The basic entity is a *name*, which is used as an unstructured reference to a synchronous channel where messages can be sent and received. In its simpler version, a process is built from names according to the syntax:

$$P ::= 0 \mid \sum_{i \in I} \pi_i.P_i \mid (P|Q) \mid !P \mid (\nu x)P \quad (1)$$

0 is the empty process. The summation $\sum_{i \in I} \pi_i.P_i$ means that an agent might perform any prefix action π_i , and correspondingly continues as P_i behaviour: prefix forms π_i are of the kind $\bar{y}x$ (send the name x at channel y), $y(x)$ (wait for a name at channel y and rename it as x), and τ (perform a silent action). A composition $P|Q$ represents P and Q executed in interleaved concurrency. A replication $!P$ means that (infinitely) many copies of P can be executed concurrently (like $P|P|\dots$). Restriction $(\nu x)P$ creates the new name x and bounds its use in P . The semantics of π -calculus can be described by a transition system, where the transition relation $P \longrightarrow P'$ —a process P moving to P' by the occurrence of an inner interaction—is defined by operational rules [18].

B. On Stochastic Models

In general, each formal model whose semantics is given by a transition system can be extended to a stochastic version, resorting to the idea of Markov transition system. There, each transition $P \xrightarrow{r} P'$ is labelled with a *rate* r , a non-negative real value which describes how the transition probability between P and P' increases with time. Stochastic models allow for quantitative simulations, for rates can be used to express aspects such as probability, speed, delays, and so on.

However, from an engineering perspective the choice of which language is used for describing processes is a crucial one, for the system to be simulated is to be effectively represented in the language.

Three basic options are available: (i) automata, like finite-state ones, where the system is described by state changes and by supporting data structures (such as stacks); (ii) nets, like Petri Net, where the system is described by a marking of tokens spread over a graph; and finally (iii) process algebras, like π -calculus, where the system is described by a composition of interacting entities. We find the third approach to be the best suited for describing quantitative aspects of complex MASs—such as self-organising ones. On the one hand, differently from automata, process algebras allow to express concurrent activities (agents in this case). On the other hand, differently from nets, process algebras allow for full compositionality: this property is a particularly relevant one, as it allows to express agents (and artefacts) with different roles separately, and then simply reuse such definitions to express the whole system model by composition (parallel composition, summation and replication).

²The source code for the experiments in this paper can be downloaded from <http://www.alice.unibo.it/download/spim/woa2005.zip>.

C. Stochastic π -Calculus and π -Machine

Priami [5] introduced a stochastic extension to π -calculus. Each channel name is associated with an activity rate r : the delay of an interaction through that channel (representing the use of a resource [5]) is then a random variable with an exponential distribution defined by r . Exponential distributions are used because they enjoy the memoryless property, i.e. each transition is independent from the previous one. Given a channel name x , the probability p_i of a transition $P \xrightarrow{r_i} P_i$ representing an interaction through x is the ratio between its rate r_i and the sum of rates of the n transitions through x enabled by P :

$$p_i = \frac{r_i}{\sum_{j=1..n} r_j}, \quad 1 \leq i \leq n. \quad (2)$$

In the following, we consider the SpiM [6] implementation for the stochastic π -calculus interpreter.

D. Simulating Anomaly Detection

In this section we discuss how to exploit Pi-Calculus for simulating the previously described security system. We are not going to describe here how to write stochastic pi-calculus specifications since it has already been widely covered by the literature, e.g. see [7], [6]. In order to give an insight of the specification process, we have mapped an agent to a set of concurrent processes which are able to send/receive signal to/from other agents: a more detailed description of our approach is available in [4].

We consider an environment in which agents can enter and leave after being authenticated and authorised: since we want to keep the average number of agents constant we set the entering and leaving rates to be equal. The simulation parameters are (i) the number of agents within the system at $t = 0$, (ii) the “concentration” of normal vs. abnormal agents, (iii) the rates at which normal behaving agents enter the system, (iv) the rates at which abnormally-behaving agents enter the system, (v) the duration of the simulation.

Then we add inspector agents to the system, which observe the behaviour of external agents: each inspector performs the same task but independently from the others. The anomaly detection system parameters are

- the number of inspectors
- the rate of inspections

Both parameters should be dynamically adjustable, e.g. if the system is under attack it can raise its defences: for the whole duration of the simulation we consider them to be constant.

The results are plotted in Figure 4. With the chosen values for the parameters, this chart let us observe that the system is able to exclude agents behaving abnormally within about 400 time units. If that is acceptable—i.e. the system still working at the target quality level—we can choose those parameters value for tuning the actual system. Since these results have been obtained by simulation they must be validated by the actual system: this last step is going to be addressed soon in the future.

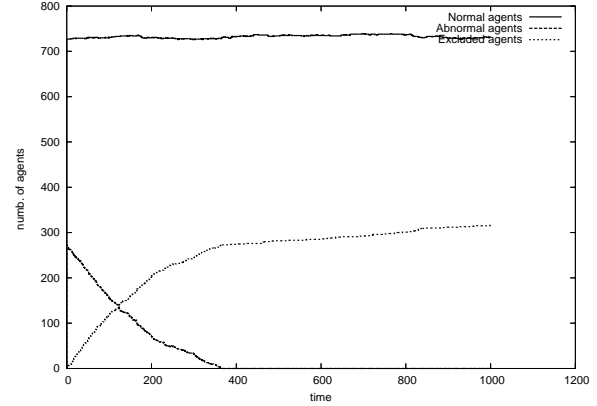


Fig. 4. A simulation of a simple system where normally- and abnormally-behaving agents can enter and leave. Inspectors limit the activity of abnormal agents.

V. CONCLUSION AND FUTURE WORKS

This paper is based on a previous work [4] where we started putting together the elements of a framework for engineering self-organising applications. While in [4] we focused on the applicability of stochastic pi-calculus, in this paper we have given more details on domain specific issues, programming artefacts to support anomaly detection for MAS and targeting the simulations to the specific case. We consider MASs composed by agents and artefacts [13], [11], and we build simulations in a stochastic process algebra setting able to tune system parameters at design time. We are developing an anomaly detection system for TuCSon and in parallel we consider this application as a case to assess the impact of simulation in engineering self-organising system.

The system depicted is based on the TuCSon coordination infrastructure, it features the remarkable notion of ACCs, which enable to control agent actions, reify information on action sequences (to be read by the infrastructure and/or other agents), prevent agent actions from a given point in time. For the architecture and general principles we took inspiration from the human immune system and previous works on IDSs. For the methodology, we relied on formal simulation and modelling via stochastic π -calculus, which—even though is a quite new language in the context of the MAS community—showed its effectiveness as a design tool.

Whereas our experiments need to be further detailed, we believe they generally emphasise the ability of the proposed approach to help the MASs developers to anticipate design decisions and strategies at the early stages of design—before actually developing prototypes and testing them.

We have a basic prototype of anomaly detection systems on top of TuCSon-based MASs, but we need to further detail and test it in order to validate simulation results. Other than testing security at the artefact level, we also plan to explore the implications of extending this approach to a network of nodes hosting the same sort of artefacts.

Finally, in this paper we have only been concerned with

self-organisation mechanisms. In future works we intend to explore the dynamics that causes system properties to emerge. For example, the uniqueness of the human immune system provide the human species with a greater probability to survive to a specific antigen: this emergent property could be very important for distributed system.

REFERENCES

- [1] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," IBM Corporation, Tech. Rep., 15 Oct. 2001. [Online]. Available: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf
- [2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003. [Online]. Available: <http://portal.acm.org/citation.cfm?id=642200>
- [3] F. Heylighen, "The science of self-organization and adaptivity," in *Knowledge Management, Organizational Intelligence and Learning, and Complexity*, ser. The Encyclopedia of Life Support Systems. EOLSS Publishers, 2003.
- [4] L. Gardelli, M. Viroli, and A. Omicini, "On the role of simulations in engineering self-organizing MAS: the case of an intrusion detection system in TuCSOn," in *3rd International Workshop "Engineering Self-Organising Applications" (ESOA 2005)*, S. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli, Eds., AAMAS 2005, Utrecht, The Netherlands, 26 July 2005, pp. 161–175.
- [5] C. Priami, "Stochastic pi-calculus," *Computer Journal*, vol. 38, no. 7, pp. 578–589, 1995.
- [6] A. Phillips, "The stochastic Pi machine (SPiM)," 2005. [Online]. Available: <http://www.doc.ic.ac.uk/~anp/spim/>
- [7] A. Phillips and L. Cardelli, "Simulating biological systems in the stochastic pi-calculus," Microsoft Research, Cambridge, UK, Tech. Rep., July 2004.
- [8] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer immunology," *Commun. ACM*, vol. 40, no. 10, pp. 88–96, 1997.
- [9] A. Omicini and F. Zambonelli, "Coordination for internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, 1999.
- [10] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001.
- [11] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination artifacts: Environment-based coordination for intelligent agents," in *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 286–293. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1018409.1018752>
- [12] A. Omicini, "Towards a notion of agent coordination context," in *Process Coordination and Ubiquitous Computing*, D. C. Marinescu and C. Lee, Eds. CRC Press, Oct. 2002, ch. 12, pp. 187–200.
- [13] A. Ricci, M. Viroli, and A. Omicini, "Agent coordination context: From theory to practice," in *Cybernetics and Systems 2004*, R. Trappl, Ed., vol. 2. Vienna, Austria: Austrian Society for Cybernetic Studies, 2004, pp. 618–623, 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), Vienna, Austria, 13–16 Apr. 2004. Proceedings.
- [14] A. Somayaji, S. Hofmeyr, and S. Forrest, "Principles of a computer immune system," in *1997 Workshop on New Security Paradigms (NSPW '97)*, T. Haigh, B. Blakley, M. E. Zurbo, and C. Meadows, Eds. New York, NY, USA: ACM Press, 23–26 Sept. 1997, pp. 75–82. [Online]. Available: <http://portal.acm.org/citation.cfm?id=283699.283742>
- [15] H. Inoue and S. Forrest, "Anomaly intrusion detection in dynamic execution environments," in *2002 Workshop on New Security Paradigms (NSPW '02)*, C. Serban, C. Marceau, and S. Foley, Eds. New York, NY, USA: ACM Press, 23–26 Sept. 2002, pp. 52–60. [Online]. Available: <http://portal.acm.org/citation.cfm?id=844112>
- [16] M. Viroli, A. Omicini, and A. Ricci, "Engineering MAS environment with artifacts," in *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, D. Weyns, H. V. D. Parunak, and F. Michel, Eds., AAMAS 2005, Utrecht, The Netherlands, 26 July 2005, pp. 62–77.
- [17] A. Omicini, A. Ricci, and M. Viroli, "RBAC for organisation and security in an agent coordination infrastructure," *Electronic Notes in Theoretical Computer Science*, vol. 128, no. 5, pp. 65–85, 3 May 2005, 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), 30 Aug. 2004. Proceedings.
- [18] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, part I/II," *Information and Computation*, vol. 100, no. 1, 1992.

A Methodology for Crowd Modelling with Situated Cellular Agents

Stefania Bandini, Mizar Luca Federici and Giuseppe Vizzari
 Dipartimento di Informatica, Sistemistica e Comunicazione
 Università degli Studi di Milano–Bicocca
 Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
 {bandini, mizar, vizzari}@disco.unimib.it

Abstract—This paper introduces a research activity aimed at the definition of a methodology to provide a solid conceptual framework for the development of simulation systems focused on crowd dynamics and based on the Situated Cellular Agent (SCA) model. After a brief introduction of the SCA Model, the general methodological approach is described. The main steps provide the definition of the spatial abstraction of the environment, the definition of its active elements, and the specification of types of mobile agents, the related behaviours with particular attention to their movement by means of the notion of utility. A case study is also briefly described in order to show how the methodology was applied in the modelling of crowd behaviour in a subway station.

I. INTRODUCTION

The Situated Cellular Agents (SCA) model [1] is a formal and computational framework for the definition of complex systems characterized by the presence of a set of autonomous entities interacting in an environment whose spatial structure represents a key factor in their choices on their actions and in determining their possible interactions. The model has been successfully applied in different contexts, and in particular its focus on the modelling of the environment as well as its inhabiting agents and their interactions, make it particularly suitable for simulation of actual physical systems. In particular, crowd modelling and simulation requires to model the autonomous behaviour of individuals interacting among themselves (e.g. because they compete over a shared resource, but also because of crowding effects) and the interaction among pedestrian and the environment. In fact in this situation the concept of perception must have a very precise meaning, and it influences the modelling activities as well as the design and implementation of the simulation system.

There are several computational approaches to crowd modelling, ranging from analytical ones, which generally consider humans as particles subject to various forces modelling interactions among pedestrian (see, e.g., [2]), to Cellular Automata (CA) based models which provide a discrete abstraction of the environment, whose cells encapsulate the possible presence of pedestrian and whose transition rules represent the rules that govern pedestrian movement (see, e.g., [3], [4]). Agent based models are more suited than the previous ones to be applied to situations characterized by a certain degree of heterogeneity and dynamism of agents and environment. Moreover, several indirect interaction models provide the possibility of situated

agents to leave marks in the environment to influence the behaviour of other mobile entities. This metaphor has been often exploited to model the movement of animals but also humans (see, e.g., [5]). Also situated agents were successfully applied in this context, and in particular the Situated Cellular Agents model (SCA) [6], but to our knowledge a methodology for the analysis, modelling and design of crowd simulations through situated agent models does not exist.

Several methodologies for the analysis and design of multi-agent systems have been defined (see, e.g., GAIA [7], IN-GENIAS [8] and SODA [9]) but they are more focused to the design of general purpose software systems analyzed and structured using the notion of agent, and thus they lack focus to simulate specific issues. Some specific methodological approaches to multi-agent simulation can also be found (see, e.g., [10]) but they are still very abstract and do not provide specific support to crowd modelling. In this paper the first proposal of a methodology for the modelling of SCA based crowd simulations is introduced. In particular, the methodology provides a set of phases for the definition of an abstraction of the structure of the simulated environment, the specification of active elements of the environment able to generate signals facilitating the movement of pedestrian, and types of agents, with the related perceptive capabilities and behavioural specifications. The following Section will briefly introduce the SCA model, while the proposed methodology is defined in Section III. A specific case study focused on the modelling of pedestrian in a subway station adopting the proposed methodology is described in Section IV. This scenario was chosen in the wider domain of crowd modelling and simulation because it presents very complex behaviours that are not easily found in other typical simulation scenarios such as room evacuation. Preliminary results and future developments will end the paper.

II. SITUATED CELLULAR AGENT MODEL

The Situated Cellular Agent model is a specific class of Multilayered Multi-Agent Situated System (MMASS) [11] providing a single layered spatial structure for agents environment and some limitations to the field emission mechanism. A thorough description of the model is out of the scope of this paper, and this aim of Section is to briefly introduce it to give some basic notion of the elements that are necessary to describe the methodology.

A *Situated Cellular Agent* system is defined by the triple $\langle Space, F, A \rangle$ where *Space* models the environment where the set *A* of agents is situated, acts autonomously and interacts through the propagation of the set *F* of fields and through reaction operations.

Space is defined as a not oriented graph of sites. Every *site* $p \in P$ (where *P* is the set of sites of the layer) can contain at most one agent and is defined by the 3-tuple $\langle a_p, F_p, P_p \rangle$ where:

- $a_p \in A \cup \{\perp\}$ is the agent situated in p ($a_p = \perp$ when no agent is situated in p that is, p is empty);
- $F_p \subset F$ is the set of fields active in p ($F_p = \emptyset$ when no field is active in p);
- $P_p \subset P$ is the set of sites adjacent to p .

A SCA agent is defined by the 3-tuple $\langle s, p, \tau \rangle$ where τ is the *agent type*, $s \in \Sigma_\tau$ denotes the *agent state* and can assume one of the values specified by its type (see below for Σ_τ definition), and $p \in P$ is the site of the *Space* where the agent is situated. As previously stated, agent type is a specification of agent state, perceptive capabilities and behaviour. In fact an agent type τ is defined by the 3-tuple $\langle \Sigma_\tau, Perception_\tau, Action_\tau \rangle$. Σ_τ defines the set of states that agents of type τ can assume. $Perception_\tau : \Sigma_\tau \rightarrow [N \times W_{f_1}] \dots [N \times W_{f_{|F|}}]$ is a function associating to each agent state a vector of pairs representing the *receptiveness coefficient* and *sensitivity thresholds* for that kind of field. $Action_\tau$ represents instead the behavioural specification for agents of type τ . Agent behaviour can be specified using a language that defines the following primitives:

- $emit(s, f, p)$: the *emit* primitive allows an agent to *start the diffusion of field f* on p , that is the site it is placed on;
- $react(s, a_{p_1}, a_{p_2}, \dots, a_{p_n}, s')$: this kind of primitive allows the specification a *coordinated change of state* among adjacent agents. In order to preserve agents' autonomy, a compatible primitive must be included in the behavioural specification of all the involved agents; moreover when this coordination process takes place, every involved agents may dynamically decide to effectively agree to perform this operation;
- $transport(p, q)$: the *transport* primitive allows to *define agent movement* from site p to site q (that must be adjacent and vacant);
- $trigger(s, s')$: this primitive specifies that an agent must *change its state* when it senses a particular condition in its local context (i.e. its own site and the adjacent ones); this operation has the same effect of a reaction, but does not require a coordination with other agents.

For every primitive included in the behavioural specification of an agent type specific preconditions must be specified; moreover specific parameters must also be given (e.g. the specific field to be emitted in an emit primitive, or the conditions to identify the destination site in a transport) to precisely define the effect of the action, which was previously briefly described in general terms.

Each SCA agent is thus provided with a set of sensors

that allows its interaction with the environment and other agents. At the same time, agents can constitute the source of given fields acting within a SCA space (e.g. noise emitted by a talking agent). Formally, a field type t is defined by $\langle W_t, Diffusion_t, Compare_t, Compose_t \rangle$ where W_t denotes the set of values that fields of type t can assume; $Diffusion_t : P \times W_f \times P \rightarrow (W_t)^+$ is the diffusion function of the field computing the value of a field on a given space site taking into account in which site (P is the set of sites that constitutes the SCA space) and with which value it has been generated. It must be noted that fields diffuse along the spatial structure of the environment, and more precisely a field diffuses from a source site to the ones that can be reached through arcs as long as its intensity is not voided by the diffusion function. $Compose_t : (W_t)^+ \rightarrow W_t$ expresses how fields of the same type have to be combined (for instance, in order to obtain the unique value of field type t at a site), and $Compare_t : W_t \times W_t \rightarrow \{True, False\}$ is the function that compares values of the same field type. This function is used in order to verify whether an agent can perceive a field value by comparing it with the sensitivity threshold after it has been modulated by the receptiveness coefficient.

III. METHODOLOGY FOR SCA-BASED CROWD MODELLING

In SCA agents' actions take place in a discrete and finite space. Entities populating the environment are classified in types, which represent templates for the specification of active elements of the environment. The latter are not only mobile entities, but also specific elements of the environment which the modeller wishes to exploit to influence the former (e.g. with attraction or repulsion effects). To model an agent type in SCA means to define the allowed states, perceptive capabilities and behavioural specification. In the proposed methodology, agent's states represent attitudes, in terms of perceptions (in fact, as previously introduced, the state determines the current agent's perceptive capabilities), but also conditions which determine its choices on actions to be selected and carried out. These actions include the definition of influences of the agent on other entities of the environment (e.g. crowding effects) by means of field emissions, the specification of its motory system and movement preferences by means of the notion of movement utility, but also the transitions from one state to another (i.e. a change of attitude towards the perception and action in the environment). While some agents related to active parts of the environment may present a very simple type specifications, mobile entities with different possible movement attitudes might require several states and complex behavioural specifications.

The diagram shown in Figure 1 shows the main phases of the methodology, while in the following subsections the steps that bring to the definition of a complete model for crowding simulations will be introduced. It must be emphasized that the first three steps lead to the development of a computational model which can be adopted in several experiments on an analogous scenario. The last two phases are those that effectively characterize the specific experiment. Section IV will

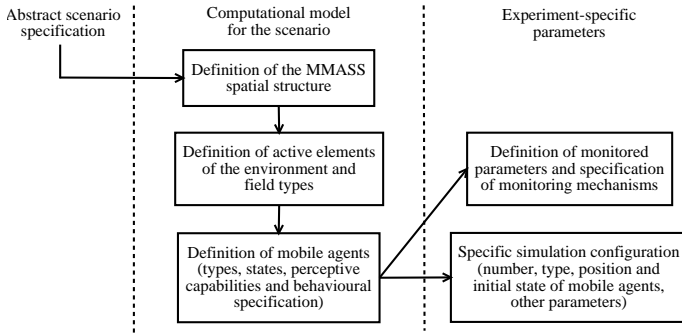


Fig. 1. A diagram showing the main phases of the methodology.

then present a concrete case study in which the proposed methodology has been applied.

A. SCA spatial structure

In order to obtain an appropriate abstraction of space suitable for the SCA model, we need a discrete abstraction of the actual space in which the simulation will take place. This abstraction is constituted of nodes connected with non-oriented arcs (i.e. a non oriented graph). Nodes represent the positions that can be occupied by single pedestrian once per time. Some of the nodes can be occupied by some agents that constitute part of the environment (doors, exits, shops etc), and that cannot be occupied by other individuals. Arcs connect nodes, representing the adjacency of one node to another. Individuals can move by single steps only from one node to other nodes that are in its immediate adjacency, so arcs and adjacency constraint agents' movement. However, as previously mentioned, the spatial structure of the environment also constraints field diffusion.

SCA space represents thus an abstraction of a walking pavement, but it has to be sufficiently detailed to be considered a good approximation of the real environment surface, and it allows a realistic representation of the movements and paths that individuals would follow. As for other crowd modelling and simulation systems we assume that a single node is associated to the space occupied by a still person [3], but the choice on the dimension of what must be considered the atomic element of the environment (a single cell) depends on the specific simulation scenario and on the related goals.

B. Active Elements of the Environment

As previously introduced, in this framework we assume that specific elements of the environment can be perceived as reference points influencing (or even determining) the movement of pedestrians. The SCA model provides a simple mean of generating at-a-distance effects that can be exploited to generate attraction or repulsion effects: the field diffusion-perception-action mechanism. However, only an agent can be the source of a field, and thus the proposed methodology requires the reification of objects or abstractions exploited to generate attraction/repulsion effects as immobile agents that are able to emit fields.

In this phase active objects in the environment have to be selected, and field types have to be assigned. Attention must be paid not only to physical objects of the environment which constraint agent movement (and that can thus be considered as reference points), but also to objects that somehow transmit conceptual information (e.g. exit signs or indications). This phase comprises two main operations:

- *selection of active elements*: the objects of the environment that are considered relevant for our simulation have to be identified. An element is considered relevant if, by a process of abstraction from reality, it can be considered as representing a target, or if it is possible to assume that it does exert an influence on the individuals that act in the environment;
- *assignment and design of field types*: the type of the fields emitted by the objects must be specified, in terms of emission intensity, diffusion and composition function, in relation to the desired extent of influence.

It must be noted that a field represents a signal and per se it does not imply an effect on agents' behaviours, in fact the possible reaction to the perception of a signal is provided by the agent type specification; moreover the actual behaviour of an agent is influenced by its current state. In this way, it is possible to model an environment as a source of different indications that are exploited in different ways by different agents to determine their paths. For instance, the window of a shop could be modelled as the source of a field diffusing outside the shop; such a field could cause a movement towards the shop for agents which consider interesting the represented goods, but could also be completely ignored by other agents. Moreover, different fields can be spread over the environment, and thus agents may perceive them and combine their effects according to a private criteria for action selection. In this way agents are not provided with a sort of script specifying their movement paths, a predefined map, or in general a strict behavioural specification, but they are provided a simple mean for evaluating the available actions against their current attitude in a more autonomous way.

C. Mobile Agents

Once the spatial abstraction has been defined, and the active elements of the environment and the related fields have been specified, the third phase of the methodology is to define the behaviour of the pedestrian. The model allows to define heterogenous agents thanks to the notion of agent type, which comprises the definition of related state, perceptive capabilities and behavioural specification. However, the modelled behaviour can be quite complex, as an individual may be endowed with several distinct attitudes towards movement and action selection that are activated in different contexts.

The behaviour of an agent type can thus be segmented in relevant states. The more complex is the behaviour that we want to capture, the higher will be the number of states that an agent can assume. This definition can be summarized in the two phases below:

- *definition of agent type's states*: in this phase of the modelling it must be established the number of states

that each agent can assume. Each state represents diverse priorities, and a different attitude of the agent. For each state must be determined the field emissions of the agent type (i.e. the influence of agents of this type towards other entities in the system), and the sensitivity to fields emitted by other agents. In addition to these elements, that are required for every SCA agent type, this methodology also provides the definition of the utility value for every field type, as a measure of the relevance of the perception of this field on agent's choices on its own movement;

- *definition of conditions for states transition*: the change of the state of an agent is related to the perception of a specific condition in its current context that determines a transition from a movement attitude towards a different one. These conditions must thus be carefully defined and modelled by means of a *react* or a *trigger* operation.

As previously introduced, in this framework possible attitudes of that type of agent are reified as states in order to specify the actions that can be selected only for agents. In these states its perceptive capabilities can be differentiated, but also its preferences on possible available moves. This can be modelled by means of a *utility function* which computes a sort of “desirability” value for every site in which the agent might move, in relation to its current state. Utility functions represent a flexible mean of combining different aspects influencing the selection of actions [12] and in this specific case these aspects are represented by different fields. In fact, fields are related to entities, either mobile (i.e. other pedestrian) or immobile (e.g. doors), that influence mobile agents' motion in a different way according to their context. It is thus necessary to specify, for each agent state, what is the impact of the perception of each field type on the desirability of the related place. The overall utility of a place is the aggregation of all these influences, that can have a positive, negative or null impact on the total value. The basic agent strategy for the choice on single movement action is thus to select the adjacent free place with highest utility value. According to the specific scenario, the possibility to remain still could be considered acceptable, penalized or even not allowed.

Before the conclusion of this paragraph we must specify that the utility values and the action modelling are not properly phases, but are activities that permeate the whole process of the construction of a simulation, and are subject to constant revision (a process that in some cases is referred to as simulation calibration).

D. Specific Simulation Configuration

The configuration for an experiment in a specific simulation scenario, not only in the case of crowd simulation, is a crucial phase that has to be performed carefully. In particular the effort of conceptualization carried out in the previous passages is wasted unless a realistic configuration for the experiment is defined. In fact, the data that are obtained through the execution of simulations are obviously strongly dependant on the starting conditions, as well as on the modelling of the simulated reality.

To configure a crowd simulation means to set the following parameters:

- *agents number and starting positions*: the number of the mobile agents that will populate the simulation must be decided in relation to the crowd scenery that is being represented; their positions must also be specified;
- *agents' initial states*: the initial state of every agent has to be specified. The decision to assign to an agent an initial state or another is taken in relation to the goals of the specific simulation: in fact, this parameter determines the initial movement attitude of the agent in the environment;
- *field emission intensity*: field emission intensity is a parameter that allows to modulate stronger or weaker influence effects; the choice on this parameter (together with the diffusion functions to be adopted for various field types) also determines the extent of the effects. The possibility to tune these parameters is a key factor in the definition of specific effects, both at individual level (e.g. amplifying or attenuating the field emission intensity of a specific agent) as well as on the collective scale (for instance modifying the intensity of fields related to elements of the environment).

E. Monitored Parameters and Mechanisms

This phase represents a formal statement of what is the goal of the simulation, a precise specification of what has to be observed and how. When simulating crowd dynamics in an evacuation scenario, the average number of turns required for agents to exit from a room is a crucial parameter to be monitored, while it can be of no interest when the goal of simulation is to observe the behaviour of pedestrian in a shopping centre. Other possible observable parameters could be average crowd density, average (or maximum) number of people waiting in a queue, occurrence of specific events, and many others dependant on the specific simulation context.

The variety of possibly monitored parameters, and thus also the number and heterogeneity of distinct monitoring mechanisms, does not allow to define specific guidelines for this phase.

IV. A CASE STUDY: THE UNDERGROUND STATION

An underground station is an environment where various crowd behaviours take place. In such an environment passengers' behaviours are difficult to predict, because the crowd dynamics emerges from single interactions between passengers, and between single passengers and parts of the environment, such as signals (e.g. current stop indicator), doors, seats and handles. The behaviour of passengers changes noticeably in relation to the different priorities that characterize each phase of their trips. That means for instance that passengers close to each other may display very different behaviours because of their distinct aims in that moment. In a crowd dynamic behaviours of the singles can also constitute a hindrance for the purpose of someone else. Passengers on board may have to get off and thus try to reach for the door, while other are instead looking for a seat or standing beside a handle. Moreover when trains stop and doors open very complex crowd dynamics happen, as people that have to get on the train have to allow the exit of passengers that are getting off. Passengers

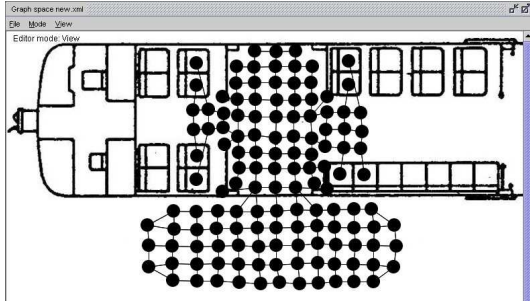


Fig. 2. Discretization of a portion of the environment

have to match their own priority with the obstacles of the environment, with the intentions of other passengers, and with implicit behavioural rules that govern the social interaction in those kind of transit stations, in a complex mixture of competition over a shared resource and collaboration to avoid stall situations. Given the complexity of the overall scenario, we decided to focus on a specific portion of this environment in which some of the most complex patterns of interaction take place: the part of platform in presence of a standing wagon from which some passengers are attempting to get off while other waiting travellers are trying to get on.

However the value of the realized simulation is not the main goal of this work, as our main aim is to show how the proposed methodology was applied in this case study. The goal of a complete simulation system in this context would be the possibility to support expert users in the detection of critical problems of the structure of the station, as bottlenecks, wrong disposition of the exits and so on, by offering the modelling instruments able to capture interaction between passengers and the environment, simultaneously on board and on the waiting platform. Such a tool would be of great aid for the prediction of security measure in situations of overcrowding or in presence of an unexpected hazard.

To build up our simulation we made some behavioural assumptions, now we will make some brief examples of the kind of behaviours we wanted to capture. Passengers that do not have to get off at a train stop tend to remain still, if they do not constitute obstacle to the passengers that are descending. Passengers will move only to give way to descending passenger, to reach some seat that has become available, or to reach a better position like places at the side of the doors or close to the handles. On the other hand in very crowded situations it often happens that people that do not have to get off can constitute an obstacle to the descent of other passengers, and they “are forced to” get off and wait for the moment to get on the wagon again. Passenger that have to get off have a tendency to go around still agents to find their route towards the exit, if it is possible. Passengers on the platform enter the station from the ingress points (station entrances) and tend to distribute along the threshold line while waiting for a train. Once the train is almost stopped they identify the entrance that is closer to them and move towards it. If they perceive some passenger bound to get off, they first let them get off and then get on the wagon.

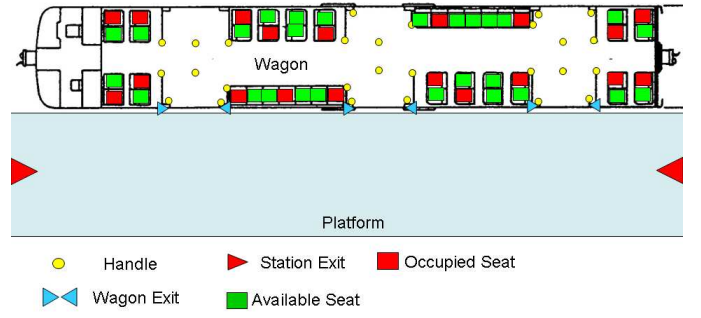


Fig. 3. Immobile active elements of the environment.

A. Environment Abstraction: a Metro Station

To build an environment suitable for SCA platform, first of all we need to define a discrete structure representing the actual space in which the simulation is set. In our case study we started from an available diagram of an underground wagon. A discrete abstraction of this map was defined, devoting to each node the space generally occupied by one standing person, as shown in figure 2. Arcs connecting nodes are not necessarily uniform across the space: in fact we decided to allow some specific movement opportunities to agents in critical positions of the environment. However a thorough analysis of the effects of this kind of heterogeneity in the spatial structure on field diffusion is needed, and will be the object of future works.

B. Active Elements of the Environment: Train and Station

In our simulation fields are generated by elements of the environment but also by agents that represent passengers. We identified the following objects as active elements of the environment: *Exits*, *Doors*, *Seats* and *Handles* (see figure 3 for their disposition). Now we give a brief description of the kind of fields that those static agents emit. Station exits emit fixed fields, constant in intensity and in emission, that will be exploited by agents headed towards the exit of the station. Exits could also constitute ingress points for agents that arrive on the platform. Agent-doors emit another field which can guide passengers that have to get off towards the platform, and passengers that are on the platform and are bound to get in the wagon. Seats may have two states: occupied and free. In the second state they emit a field that indicates their presence. An analogous field is emitted by handles, which however are sources of fields characterized by a minor intensity.

C. Mobile Entities: Passengers

We have identified the following states for agent of type passengers: *waiting* (w), *passenger* (p), *get-off* (g), *seated* (s), *exiting* (e). In relation to its state, an agent will be sensitive to some fields, and not to others, and attribute different relevance to the perceived signals. In this way, the changing of state will determine a change of priorities. A state diagram for passenger agents is shown in figure 4. State *w* is associated to an agent that is waiting to enter in the wagon. In this state agents perceive the fields generated by the doors as attractive, but they also perceive as repulsive the fields generated by

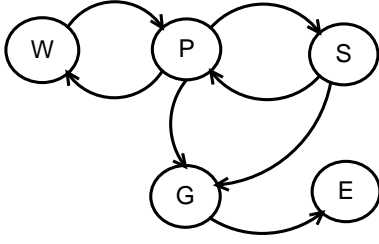


Fig. 4. A diagram showing various states of agent type passenger.

passengers that are getting off, in other words those in state g . In state w the agent “ignores” (is not sensitive to) the fields generated by other active elements of the environment, such as exits’ attractive fields, chairs attractive field and so on. Once inside the wagon, the agent in state w changes its state in p (passenger), through a trigger function activated by the perception of the maximum intensity of field generated by agent-door type. Agent in state passenger is attracted by fields generated by seats and handles, and repulsed by fields related to passengers that are getting off. It does not have any sensitivity for the attraction field of the doors. In state g the agent will instead emit a field warning other agents of its presence, while it is attracted by fields generated by the doors. Once passed through the wagon door, or in immediate proximity (detected by means of specific thresholds on related field intensity), the agent in state g changes its state to e (exiting) and its priority will become to find the exits of the station. The agent in state e is thus attracted by fields related to exits.

Table I summarizes the sensitivity of the passenger to various fields and it also sketches a first attribution of the utility of the presence of these field types on empty nodes considered as destination of a transport action. In particular, cells provide the indication of the fact that the related field is perceived as attractive or repulsive and the priority level (i.e. relevance) associated to that field type.

All passengers except those in state g emit a presence field that generally has a repulsive effect, but a lesser one with respect to the “exit pressure” generated by agents in get-off state.

V. PRELIMINARY RESULTS

A simulator implementing the previously introduced model was realized exploiting the Mmass framework [13] (please note that SCA is a particular class of Mmass model): only a subset of the overall introduced model was implemented, and more precisely active objects of the environment and passenger agents in state w , g , e , p . Figure 5 shows a screen-shot of this simulation system, in which waiting agents move to generate room for passenger agents which are going to get off the train. The system is synchronous, meaning that every agent performs one single action per turn; the turn duration is about one second of simulated time.

The goal of this small experimentation was to qualitatively evaluate the modelling of the scenario and the developed simulator. The execution and analysis of several simulations showed that the overall system dynamics and the behaviour

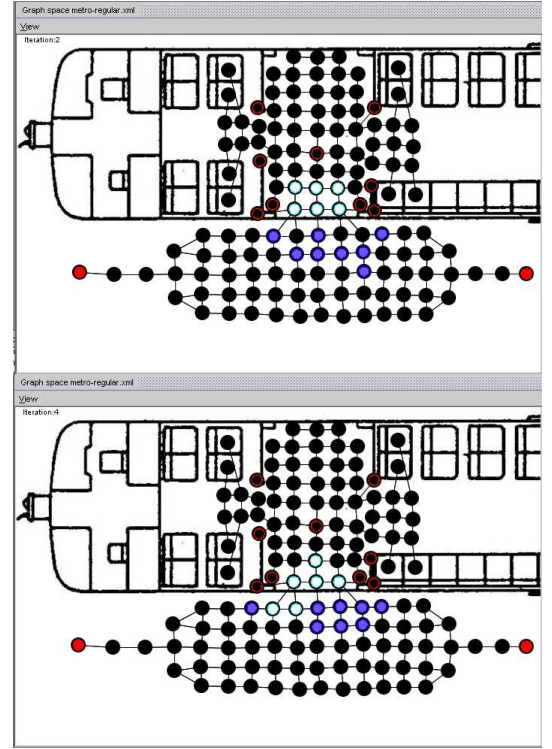


Fig. 5. Two screenshots of the metro simulation. On the first one light gray agents are inside the train and going to get off, while dark agents are standing outside and are going to get on. On the second, the latter have made some rooms for the former to get off.

of the agents in the environment is consistent with a realistic scenario, and fits with our expectations. In particular, we executed over 100 simulations in the same starting configuration, which provides 6 passengers located on a metro train in state g (i.e. willing to get off), and 8 agents that are outside the train in state w (i.e. waiting to get on). This simulation campaign is motivated by the fact that an agent having two or more possible destination sites characterized by the same utility value makes a non deterministic choice. In all simulations the agents achieved their goals (i.e. get on the train or get out of the station) in a number of turns between 40 and 80, with an average of about 55 turns.

Nonetheless we noticed some undesired transient effects and more precisely:

- *oscillations* and “forth and back” movements;
- static forms providing “groups” facing themselves for a few turns, until the groups dispersed because of the movement of a peripheral element.

These phenomena, which represent minor glitches under the described initial conditions, could lead to problems in case of high pedestrian density in the simulated environment. This points out the need of additional mechanisms correcting the movement utility. In particular, some possible improvements to the basic movement utility mechanisms are:

- introduce a notion of agent *facing*: the SCA model does not provide an explicit facing for agents because is not always relevant or even applicable (consider for instance the modelling of immune system cells [14]);

State	Exits	Doors	Seats	Handles	Presence	Exit press.
W (getting on)	not perc.	attr. (2)	not perc.	not perc.	rep. (3)	rep. (1)
P (on board)	not perc.	not perc.	attr. (1)	attr. (2)	rep. (3)	rep. (2)
G (getting off)	not perc.	attr. (1)	not perc.	not perc.	rep. (2)	not perc.
S (seated)	not perc.	attr. (1)*	not perc.	not perc.	not perc.	not perc.
E (exiting)	attr. (1)	not perc.	not perc.	not perc.	rep. (2)	not perc.

* = The door signal also conveys the current stop indication.

TABLE I

THE TABLE SHOWS, FOR EVERY AGENT STATE, THE RELEVANCE OF PERCEIVED SIGNALS.

however in this specific simulation scenario this is a relevant factor for agents' choices on their movement. Instead of modifying the general model, a possible way of introducing this notion is to allow agents to keep track of their previous positions, in order to understand if a certain movement is a step back. The utility of this kind of movement should be penalized, in order to discourage this choice;

- *penalize immobility*: in order to avoid stall situations, or simplify the solution of this kind of situation, an agent should generally move, unless it has attained the goal for a movement attitude (i.e. agent state), such as to be adjacent to a handle for an agent in state p . To achieve this effect, the memory of the past position, introduced in the previous point, could also be exploited to penalize the utility of the site currently occupied by the agent whenever it was also its previous position.

These correctives were introduced in the behavioural specification of mobile agents, and a new campaign of tests was performed to evaluate the effect of these modifications in the overall system dynamics. Once again, in all simulations the agents were able to achieve their goals, but the number of turns required to between 28 and 60, with an average of about 35 turns.

However, this reduction of time required for the completion of agents' movements, is not the only improvement obtained by introducing these correctives in agent behaviours. In fact another relevant part of the project in which this work has been developed provides the generation of effective forms of visualization of simulation dynamics to simplify its analysis by non experts in the simulated phenomenon. In particular, the developed simulator can be integrated with a 3D modelling and rendering engine (more details on this integration can be found in [15]), and a sample screenshot of the animation of the simulation dynamics is shown in Figure 6. In this kind of visualization the issues that were caused by the uncorrected movement utility specification brought to confusing, unrealistic and thus ineffective rendering of system dynamics. By introducing these correctives, the occurrence of oscillating agent movement was drastically reduced, and the penalization of immobility simplified the solution of stall situations among facing groups.

While these correctives can be easily modelled and implemented, to apply this approach to problems in larger scale scenarios, such as those related to malls or multi-floor buildings, it could be necessary to introduce some additional elements for the specification of agents' behaviours. In particular, in

order to endow agents with the possibility to select in a more autonomous way those signals that are relevant to direct their movement, it could be necessary to introduce some form of abstract map of the environment. However the introduced methodology is focused on supporting the modelling of situations in which there is a strong focus on specific spots of a spatial structure, such as a hall or a part of a building floor, in specific situations (e.g. evacuation).

VI. CONCLUSIONS AND FUTURE DEVELOPMENTS

This work has presented the first proposal of a methodology for the modelling of crowds through the SCA model. The main phases of this methodology were introduced, and in particular the first two provide the definition of an "active environment", able to support simple reactive agents in the navigation of its spatial structure according to their behavioural specification. A case study related to a complex modelling scenario was introduced in order to show how the proposed methodology can be applied in a concrete case study.

Future developments are aimed at refining both the methodology and the Mmass platform, in order to better support the modeller/user, in the construction of complex simulation scenarios. In particular the platform still does not provide specific user interfaces and modules aimed at supporting the definition of an active environment, and parameters for specific simulations. Moreover specific libraries for active objects and paradigmatic pedestrian behaviours could be defined after a thorough analysis of psycho/sociological studies of crowd behaviors.

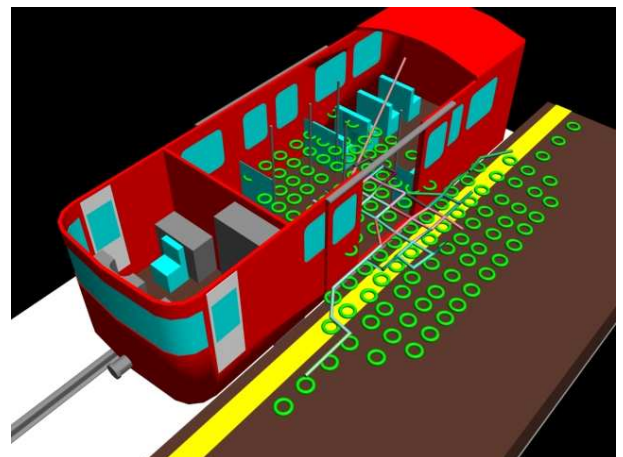


Fig. 6. A screenshot of the 3D modelling of the simulation dynamics.

ACKNOWLEDGEMENTS

This work is preliminary result of the Social Mobile Entities in Silico (SMES) project, and was partly funded by the New and Old Mobility Analysis and Design for the Information Society (NOMADIS) laboratory, in the context of the Quality of Life in the Information Society (QUA-SI) multi-disciplinary research programme in Information Society.

REFERENCES

- [1] S. Bandini, G. Mauri, and G. Vizzari, "Supporting Action-at-a-distance in Situated Cellular Agents", *Fundamenta Informaticae*, 2006 (in press).
- [2] D. Helbing, "A Mathematical Model for the Behavior of Pedestrians", *Behavioral Science*, no. 36, pp. 298–310, 1991.
- [3] A. Schadschneider, A. Kirchner, and K. Nishinari, "CA Approach to Collective Phenomena in Pedestrian Dynamics." in *Cellular Automata, 5th International Conference on Cellular Automata for Research and Industry, ACRI 2002*, ser. Lecture Notes in Computer Science, S. Bandini, B. Chopard, and M. Tomassini, Eds., vol. 2493. Springer, 2002, pp. 239–248.
- [4] J. Dijkstra, J. Jessurun, and H. J. P. Timmermans, *Pedestrian and Evacuation Dynamics*. Springer-Verlag, 2001, ch. A Multi-Agent Cellular Automata Model of Pedestrian Movement, pp. 173–181.
- [5] D. Helbing, F. Schweitzer, J. Keltsch, and P. Molnár, "Active Walker Model for the Formation of Human and Animal Trail Systems", *Physical Review E*, vol. 56, no. 3, pp. 2527–2539, January 1997.
- [6] S. Bandini, S. Manzoni, and G. Vizzari, "Situated Cellular Agents: a Model to Simulate Crowding Dynamics", *IEICE Transactions on Information and Systems: Special Issues on Cellular Automata*, vol. E87-D, no. 3, pp. 669–676, 2004.
- [7] F. Zambonelli, M. J. Wooldridge, and N. R. Jennings, "Developing Multiagent Systems: The GAIA Methodology", *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, pp. 317–370, 2003.
- [8] J. Pavón and J. J. Gómez-Sanz, "Agent Oriented Software Engineering with INGENIAS" in *CEEMAS*, ser. Lecture Notes in Computer Science, V. Marík, J. Müller, and M. Pechoucek, Eds., vol. 2691. Springer-Verlag, 2003, pp. 394–403.
- [9] A. Omicini, "SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems", in *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*, ser. Lecture Notes in Computer Science, P. Ciancarini and M. Wooldridge, Eds., vol. 1957. Springer-Verlag, 2001, pp. 185–193.
- [10] A. M. C. Campos, A. M. P. Canuto, and J. H. C. Fernandes, "Towards a Methodology for Developing Agent-Based Simulations: The MASim Methodology", in *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. Washington, DC, USA: ACM Press, 2004, pp. 1494–1495.
- [11] S. Bandini, S. Manzoni, and C. Simone, "Dealing with Space in Multi-Agent Systems: a Model for Situated MAS", in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. ACM Press, 2002, pp. 1183–1190.
- [12] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd ed.)*. Prentice Hall, 2002.
- [13] S. Bandini, S. Manzoni, and G. Vizzari, "Towards a Platform for Multilayered Multi-Agent Situated System Based Simulations: Focusing on Field Diffusion", *Applied Artificial Intelligence*, vol. 20, no. 4–5, 2006 (in press).
- [14] S. Bandini, F. Celada, S. Manzoni, R. Puzone, and G. Vizzari, "Modelling the Immune System with Situated Agents", in *International Workshop on Natural and Artificial Immune Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2005 (in press).
- [15] S. Bandini, S. Manzoni, and G. Vizzari, "Crowd Modelling and Simulation: Towards 3D Visualization", in *Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning*. Kluwer Academic Publisher, 2004, pp. 161–175.

Towards the Interpretation of Emergent Spatial Patterns through GO Game: the Case of Forest Population Dynamics

Stefania Bandini, Sara Manzoni, Stefano Redaelli
Dept. of Computer Science, Systems, and Communication
University of Milan–Bicocca
via Bicocca degli Arcimboldi 8, 20126 Milan, Italy
{bandini,manzoni,redaelli}@disco.unimib.it

Abstract—In this paper we present the preliminary results of an ongoing research that aims at supporting ecosystem management in the study of forest systems according to a distributed modeling and simulation approach. The Cellular Automata For Forest Ecosystems (CAFFE) project is an interdisciplinary research involving computer scientists of the Department of Computer Science, Systems and Communication of the University of Milano–Bicocca and urban planners, biologists and ecosystem managers of the System Research Department of Austrian Research Center (ARC). In particular, we focus here on the part of CAFFE project that concerns the design of a method to support the analysis step of simulations of forests according to a distributed approach (such as those based on Cellular Automata or Situated Multi-Agent Systems). To this aim an innovative analysis method inspired by the Chinese Go game is under design. The originality of the approach concerns the detection within system configurations of known patterns whose interpretations are well-known by expert Go players. In this paper, after a brief presentation of the CA-based model of forests, we focus first on the set of Go patterns that we currently studied, then we present some preliminary results on experiments we conducted to validate the proposed approach to spatial patterns interpretation.

I. INTRODUCTION

The CAFFE (Cellular Automata For Forest Ecosystems) project is an interdisciplinary research that involves computer science, biology, and ecosystem management. It started about one year ago and involves the Artificial Intelligence Lab (L.INT.AR.) of the Department of Computer Science, Systems and Communication of the University of Milano–Bicocca and the System Research Department of Austrian Research Center (ARC). The main aim of this ongoing research is the development of methods for sustainable afforestation and management of forests.

A central role is played in this project by computer supported simulations of the dynamics of the forest system. The modeling approach adopted by CAFFE for the forest system is based on Cellular Automata and describes the forest as the result of competition between heterogeneous vegetable populations. After a preliminary implementation of the CA-based model (see [1]), we are currently developing a software simulation platform for sustainable afforestation and management of forests in which both model improvements (i.e. a new model of the forest based on Multilayered Situated MAS [2], [3] is under design) and new software functionalities. One of

these functionalities will concern an innovative interpretation approach for patterns that can be detected as emerging from the dynamics of forest systems.

In this paper, we focus on the part of the CAFFE project that aims at designing a method to support the analysis step of software simulations of vegetable populations in the forest model. This goal is particularly relevant (and ambitious) due to the distributed modelling approach that is at the basis of the forest system simulation and modelling. According to [4], we refer here to distributed modelling approaches in order to indicate all those approaches that allow the representation of complex systems (and problems) whose evolution (and solution) results from the interactions between autonomous and interacting entities (more often indicated as based on Multi-Agent Systems - MAS). For this reasons, analyzing the dynamics of complex systems modelled and simulated according a distributed approach is still a challenging issue. In fact in a simulation of a forest composed by a lot of different species, we have a very complex behavioral dynamics, and it is very difficult to recognize all the collective emergent behaviors occurring during the simulation.

The work here presented concerns the model of a forest system according to a Cellular Automata (CA) approach. CA can be considered a simple case of MAS in which each cell of the automaton represents an agent and the CA dynamics is based on the behavior (change of state of cell) and on local interaction among cells (transition function of CA cells usually includes the state of adjacent cells). Obviously CA can only be considered as a very simple class of reactive MAS. However, they can be a suitable and very promising approach for the aims of CAFFE project (both for modelling and simulation, but also for the design of a novel analysis method). The CA-based model of forests that has been adopted by CAFFE project is derived by the one presented in [5], in which different plant species can inhabit the forest area and compete for the same resources (i.e. water, light, nitrogen, and potassium). The area is divided into cells and it is reproduced by the CA. The state of each cell of the CA is defined by a flag denoting whether or not it contains a tree, the amount of each resource present in the cell, and a set of variables defining the features of the tree (possibly) growing in it. The update rule of the automaton mainly depends on the presence of a tree in a cell. In case a

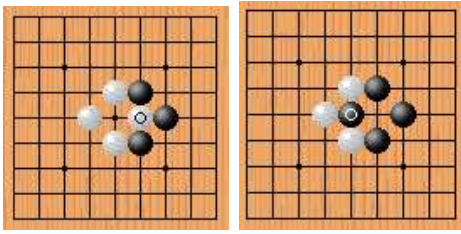


Fig. 1. The Ko rule in the Go game.

tree is present, part of the resources present in it (and in the neighboring ones, if the tree is large enough) are absorbed by the tree. Every cell also produces at each update step a given amount of each resource (that in any case cannot exceed a maximum threshold value). The production of resources in the cells is determined by a set of global parameters, and reproduces environmental factors such as rain, presence of animals in the area, and so on. The effect of the presence of a tree in a cell on the neighboring ones has been modelled by making resources flow from richer cells to poorer ones (that possess less resources since a part of them is consumed by the tree).

In the following Section, we introduce the proposal for a method for pattern detection in the dynamics of forest populations based on the detection of spatial patterns whose interpretation is suggested by the ones of similar spatial patterns occurring on a Go board during a game. In Section III we then show first experiments conducted to validate the proposal. Some remarks and future works conclude the paper.

II. GO-BASED INTERPRETATION OF SPATIAL PATTERNS

Most of the available approaches to analyze the behavior of complex systems are based on statistics and probability theory and they aim at deriving macro level interpretations by aggregating and correlating variables of the micro level(s). Within the context of forest ecosystems for instance, the dynamics of the forest (e.g. biomass) is computed aggregating the features of living trees taking into account different age and dimensions of trees [6]. Another common approach to complex system analysis concerns the detection and interpretation of recurring patterns [7]. These approaches are of course domain independent for what concerns pattern detection (usually the focus is in the searching of structural similarities within system configurations). On the other hand, when the model concerns a real world system and the analysis of its dynamics is oriented to verify or anticipate peculiar phenomena (e.g., in the forest ecosystem domain, deforestation of a given area), it is inevitable the necessity of domain dependent interpretation of the detected patterns. In order to reconcile the need of defining domain independent method with the detection and interpretation of a specific natural phenomenon, we started from the latter. In particular we noticed that several simulation scenarios provided recurrent dynamic configurations that were very similar to specific situations occurring in the Go game. Go game, due to the simplicity of its playing rules but also to the complexity of possible configurations and the

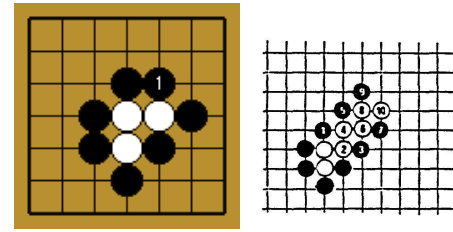


Fig. 2. Geta (on the left side) and Shicho (on the right side) in Go game.

consequent complexity of playing tactics and strategies, has already inspired several models (e.g. in economy, military, art, semiology, culture, and many others [8]). In the Go game there are two populations (i.e. black and white stones) that compete for survival in a territory with limited space and resources. During the game, black and white stones situated on the Go board cannot move, but they can be put onto a board site, survive or die as result of a metaphorical competition on the local territory with neighboring stones. The effect of the death of the stone is its removal from its site. A key concept for the survival of Go pieces is the notion of *liberty*: if a group of pieces have no liberties (i.e. none of its elements is adjacent to a free site), it is removed from the game.

The analogies between Go game and the CA-based model of forests above sketched are going to be formalized. Very broadly in the CA-based model of forests, trees of different species live in a territory, compete for limited territory resources, can born, die, and they cannot change their position. Moreover, the concept of liberty for plants can refer to favorable conditions for growth and reproduction. Starting from this analogies, our proposal suggests to exploit this game to study emergent patterns in the dynamics of complex systems, by studying some spatial patterns well-known by advanced Go players [9], and to verify whether their interpretation can be suitably and fruitfully applied to interpret similar spatial patterns occurring in dynamics configurations of the CA-based forest model.

We introduce now some common spatial patterns that can emerge during Go games and that are well-known by Go players. Each spatial pattern is interpreted by Go players in terms of game competition, and we briefly describe how the interpretation of Go patterns can be applied to interpret spatial patterns that emerge from the evolution of the CA-based model of forests.

A. Spatial Patterns Emerging in Go Game ...

- 1) **Ko pattern.** Ko is the configuration of Go stones such that a little free territory (i.e. a set of board positions not occupied by stones) belongs to the influence zone of two or more pieces of the same team (see Figure 1).
- 2) **Geta pattern.** Geta pattern corresponds to the local capture of a group of adversary pieces by a set of stones that surrounds it (see the left side of Figure 2).
- 3) **Shicho pattern.** Shicho is pattern in which a group of stones expands itself towards another side of the Go board. Shicho does not imply the movement of

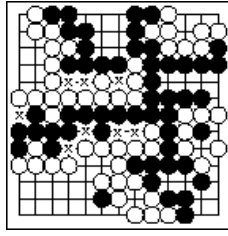


Fig. 3. Example of Go game. We can notice a lot of connected groups.

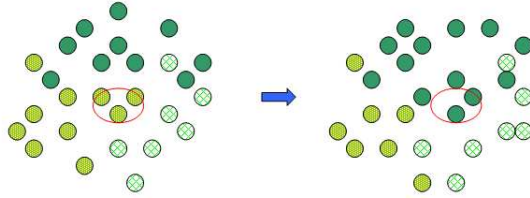


Fig. 4. Ko interpretation for forest ecosystems. Different species are competing for the domain in a little area

single stones that are part of the moving group. Group movement occurs in time-space that is, group movement is the result of stones removal from some positions and the positioning of others in adjacent ones (see the right side of Figure 2).

- 4) **Iki pattern.** A peculiar pattern that can emerge during a Go competition is Iki, a pattern that can not be captured by the adversary. Iki corresponds to a part of Go board surrounded by stones of the same color, with some free positions in its inner side to form two ‘eyes’, in Go jargon. This formation has two internal liberties that can not be occupied by adversary.
- 5) **The Tsugi pattern.** Tsugi is a Japanese word that means ‘connection’. Connections are very important in Go competition, because two stones connected to form a group are stronger than they alone (see Figure 3). In fact, it is more difficult for the adversary to build a group able to surround connected stones (i.e. it may require a lot of stones).

B. ... Their Application to Interpret Forest Population Dynamics

- 1) **Ko pattern.** In a CA configuration similar to a Ko pattern (see Figure 4), none of the the involved species (in the example we considered only two plant species) can control the territory in a stable way. In this type of situations, it is usually observed a continuous replacement of trees by others of another specie.
- 2) **Geta pattern.** Similarly to Go game, we can consider that if a group of trees is surrounded by plants of another specie and it is forced to be limited within a little and close territory zone, sooner or later it will die. In fact, trees and plants that are forced in a little zone have little

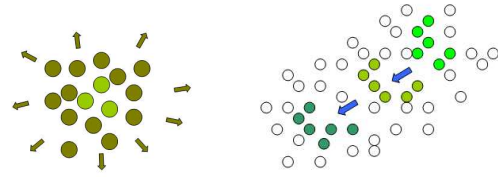


Fig. 5. Geta (on the left side) and Shicho (on the right side) interpretation for forest populations.

space to harvest the needed resources for survival and reproduction (see the left side of Figure 5).

- 3) **Shicho pattern.** In forests Shicho occurs when an homogeneous group of plants is situated in a zone where resources are not suitable related to population requirements, or in an area that is controlled by another specie (see the right side of Figure 5). Since new born trees are more likely to survive if they grow up on suitable areas, we can observe as emergent phenomenon, a group shifting in space-time.
- 4) **Iki pattern.** A spatial pattern in forest dynamics similar to Iki is characterized by a strong group that can survive for long time assuring part of the territory to its individuals (see Figure 6). This pattern is particularly strong because plants on the group border have a lot of space in the inner side and this guarantees the availability of space and resources for their survival and reproduction.
- 5) **Tsugi pattern.** Tsugi phenomenon occurs in a natural way also in forests. Each group of plants expands itself by reproduction, and when two groups expand toward one another, there is the possibility to create a connection between them. As in Go game, two connected groups of plants are stronger because they can support each other. When a plant dies neighboring ones can replace it, and when two groups are connected, neighbors increase in number. Beside this aspect, an isolated little group of trees can easily fall in the Geta phenomenon but if it is connected to another group, it is no more possible to surround it.

III. EXPERIMENTATIONS

In this section we present some experiments performed in order to validate the proposed approach and in particular to verify whether the hypotheses on the evolution of forest systems based on Go spatial patterns are confirmed by experimental simulations. To this aim, we exploited FORESTE [1], a simulation software developed according to the CA-based model of vegetable populations presented in [5].

In order to verify the occurrence of a spatial patterns in the experiments we adopted a method mainly based on the concepts of group and neighborhood. If we refer to neighborhood of a cell i as the set N_i , and we consider a trivial group composed by only one element the set $\{i\}$. We define a *connection* between to cells t and s (i.e. t is connected to s , and vice versa) if $t \in N_s$. We indicate connection between t and s with $t \rightarrow k$. Moreover, given a cell t and a group of

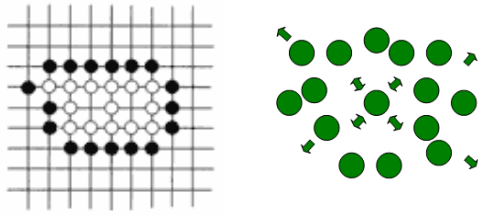


Fig. 6. Iki in Go game and its interpretation for forest ecosystems.

cells A , we say that t belongs to A (i.e. $t \in A$), if $\exists k \in A$ such that $t \rightarrow k$.

According to this model, the general aim of the experiments we conducted was to verify the correct evolution of Go-like patterns with reference to the forest simulation scenario. We define the starting conditions that represent a suitable situation for the occurrence of the pattern. Then we let start the simulation observing if the patterns evolve toward the final expected configuration. In this simulation experiment process we divide the patterns in two groups: static patterns (Geta and Iki), where it is easier to establish a good starting point for the phenomenon occurrence; and dynamic patterns (Ko, Shicho, Tsugi), where it is more difficult to find the initial situation that guarantees the occurrence of these phenomena.

A. Experiments on the Occurrence of Geta and Iki patterns

The aim of the first experiment we conducted was to verify the formation of Geta and Iki patterns. We consider the formation of Geta when it occurs the complete disappearance of surrounded specie. Geta pattern is perhaps the easiest pattern to study: starting from a given configuration, we can let evolve the CA and detect resulting configurations.

Three important elements can influence the Geta formation (i.e. involved populations, spatial dimensions of the pattern and resource distribution), and for each combination of them we conduct a set of simulations.

- *Involved populations:* we simulated Geta formation between both two groups of the same vegetal population and two heterogeneous species.
- *Spatial dimension of Geta pattern:* we considered a small (2×2), a medium (3×2) and a large (4×3) neighboring area. In general a small area is easier to attack, while a large one is more likely to survive.
- *Resources distribution on the territory:* we considered a uniformly favorable terrain and a less favorable one. The terrain resources state strongly influence the evolution of the specie competition.

According to the performed experiments we can conclude that Geta is a spatial pattern that occurs in all the studied cases. In Figure 7 we can see an example of simulations.

To verify the formation of Iki pattern, we started from an initial Iki pattern and we observe its evolution in time. We considered an Iki success if after 250 time steps we can establish that the specie involved has still a certain influence over the given territory area. We chose to consider 250 time steps because Geta complete its evolution in a maximum of

160 time steps. Therefore we think that this time period can be sufficient for a specie to take control over a territory. We consider four important elements that influence the Iki formation and we conducted experiments in all the conditions resulting from the combinations

As in the case of Geta, we considered important elements that can influence the Iki formation and we conducted a set of simulations for each combination of them.

- *Involved populations:* we simulated Iki between two homogeneous groups and between two heterogeneous species.
- *Not-Iki specie state:* we considered a first situation (referred as normal) where an Iki pattern is surrounded by a random distribution of trees of another specie, and a second one where Iki is not surrounded by another specie. In the first case we put in the simulation area a quantity of trees of the not-Iki specie that is approximately double respect to the number of trees involved in the Iki pattern.
- *Iki pattern spatial dimension:* we considered a small (5×5), a medium (5×6) and a large (6×7) neighboring area dimension.
- *Resources distribution on territory:* we considered two territory types, a uniformly favorable terrain and a terrain uniformly worse in the average.

In this experiments, we observed that the type of terrain has a great influence (better results occur when a specie is situated on a favorable terrain). Also in the cases where Iki pattern is destroyed in a given time period, the trees involved in the Iki pattern survive in their position for long (of course, due to the abundance of resources).

B. Experiments on the Occurrence of Ko, Shicho and Tsugi patterns

Simulation in the Ko case (turn over in a shared area) is more difficult because it is not guaranteed the formation of this pattern starting from a given initial situation. We considered the formation of a Ko pattern, when it can be observed a quick change of plant distribution in a local area occurs, while in the rest of the territory the same vegetal patterns do not change for long time.

In the case of Shicho pattern (a group shifts toward more suitable area), we defined a starting suitable situation and then we observed its evolution. We considered the formation of a Shicho pattern when the shift of the influence area of a given group can be observed.

To verify the occurrence of Tsugi pattern (formation of connections) starting from a suitable situation, we considered two groups of different species with their influence on opposite sides of the territory. We put a little group of the second specie in the territory of the first one and we observe whether the two groups of the same specie connect to each other and if this connection allows to save the influence over a given space portion. From the performed experiments, we observed that as one specie tries to connect two groups (reaching the Tsugi pattern) the other one tries to divide adversary groups with an infiltration in the middle. According to our Go game metaphor, a player tries always to divide enemy groups to surround them separately.

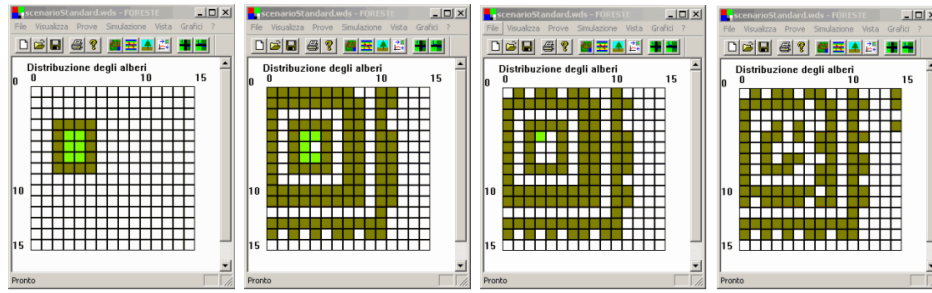


Fig. 7. An example of experimental simulation of Geta pattern: after some simulation time steps the surrounded specie is completely disappeared.

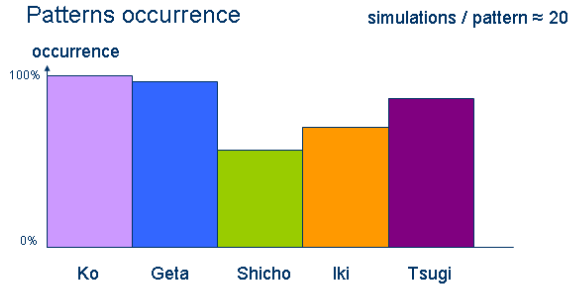


Fig. 8. Results of simulation experiments on pattern expected occurrence.

C. Simulation results

In accordance with experiments done we can conclude the Go-like patterns validity at least in the described domain. All the patterns described occur in a simulation scenario and they have the expected evolution. This first results encouraged us in continuing this research also looking for new other applications of this method. The scheme in Figure 8 shows the ratio of pattern occurrence, that in general very high. The results shown in the ratio are referred approximately to a 20 experiments for each pattern.

IV. CONCLUSIONS AND FUTURE WORKS

Detecting all patterns of emergent phenomena is very difficult for a human operator that analyzes the simulation, in particular for very large scenarios involving a lot of different species. Therefore the main future work that starts from these results is the realization of a detection algorithm capable of recognizing and interpreting the patterns during a simulation. A first prototype of this automatic detection method is already realized and it is based on the groups recognition method as we explained in section III. Our main efforts in the future is going toward the improvement of functionalities for the automatic detection method, and the implementation of a simulator with these automatic tools for pattern detection and analysis.

All these methods and tools will allow us to use the defined patterns for a meaningful interpretation of important phenomena in some simulation scenarios. In particular they can be useful, for example, in the case of artificial repopulation of forest in a given area with the introduction of new species.

In this case it is important to understand if the new specie can survive and what are the reactions of the other living species. But while the occurrence of Ko phenomenon means a good equilibrium between species, the frequency of Iki and Tsugi indicates the formation of a strong dominance presence in the area, and Geta and Shicho mean the disadvantage of a specie in comparison with the others. All these considerations can support the decision maker in the illustrated domain problem.

Surely another important future development of the this research can be the study of new patterns for the interpretation of other phenomena in different scenarios obtained, for example, with the introduction of other important elements in the system: the human presence (urbanization or pollution) or other natural interaction phenomena (desertification).

REFERENCES

- [1] S. Bandini and G. Pavesi, "A model based on cellular automata for the simulation of the dynamics of plant populations", in *Proceedings of the International Conference on Environmental Modelling and Software Society (iEMSs) - 14-17 June 2004 University of Osnabruck*, 2004, pp. 277-282.
- [2] S. Bandini, S. Manzoni, and C. Simone, "Heterogeneous agents situated in heterogeneous spaces", in *Cybernetics and Systems 2002, Proc. of the 16th European Meeting on Cybernetics and Systems Research*, R. Trappl, Ed. Vienna: Austrian Society for Cybernetic Studies, 2002, pp. 641-646.
- [3] Stefania Bandini and Sara Manzoni and Carla Simone, "Enhancing cellular spaces by multilayered multi agent situated systems", in *Cellular Automata, Proceeding of 5th International Conference on Cellular Automata for Research and Industry (ACRI 2002), Geneva (Switzerland), October 9-11, 2002*, ser. Lecture Notes in Computer Science, S. Bandini, B. Chopard, and M. Tomassini, Eds., vol. 2493. Berlin: Springer-Verlag, 2002, pp. 156-167.
- [4] J. Ferber, *Multi-Agent Systems*. Harlow (UK): Addison-Wesley, 1999.
- [5] S. Bandini and G. Pavesi, "Simulation of vegetable populations dynamics based on cellular automata", in *Cellular Automata, Proceeding of 5th International Conference on Cellular Automata for Research and Industry (ACRI 2002), Geneva (Switzerland), October 9-11, 2002*, ser. Lecture Notes in Computer Science, S. Bandini, B. Chopard, and M. Tomassini, Eds., vol. 2493. Berlin: Springer-Verlag, 2002.
- [6] D.G.Green, "Modelling plants in landscape", in *Plants to Ecosystem - Harek T. Michalewicz, ed. CSIRO, Lollingwood Ans.*, 1997.
- [7] S.Wolfram, "Cellular automata as models of complexity", in *Nature*, 311:419-424, 1984.
- [8] P. Reyssat, *Le Go: aux sources de l'avenir*. Chiron, 1994.
- [9] G. Soletti, *Note di Go*. FIGG (Federazione Italiana Giuoco Go).

Pervasive Pheromone-based Interaction with RFID Tags

Marco Mamei, Franco Zambonelli

Abstract—Despite the growing interest in pheromone-based interaction to enforce adaptive and context-aware coordination, the number of deployed systems exploiting digital pheromones to coordinate the activities of application agents is very limited. In this paper, we present a real-world, low-cost and general-purpose, implementation of pheromone-based interaction. This is realized by making use of RFID tags to store digital pheromones, and by having humans and robots to spread/sense pheromones by properly writing/reading RFID tags populating the surrounding environments. We exemplify and evaluate the effectiveness of our approach via an application for object-tracking. This application allows robots and humans to find "forgot-somewhere" objects by following pheromones trails associated with them. In addition, we sketch further potential applications of our approach in pervasive computing scenarios.

Index Terms—Pervasive computing, Pheromone-based coordination, RFID tags

I. INTRODUCTION

Pheromone-based interaction, exploited by social insects to coordinate their activities [BonDT99], has recently inspired a vast number of researches in pervasive and distributed computing systems [BabM02, MenT03, ParBS04, SveK04]. In these works, application agents (e.g., software agents, humans carrying on a PDA, or autonomous robots) interact in an indirect way by leaving and sensing artificial pheromones, digital analogues of chemical markers, in the environment. Pheromones, by encoding application-specific information in a distributed way and by uncoupling the activities of application agents, enable to enforce adaptive and context-aware coordination activities [Par97].

Despite the growing interest in pheromone-based interaction, the number of implemented systems exploiting pheromones for coordinating the activities of distributed applications situated in pervasive computing scenarios is very limited. The great majority of the proposals have only been simulated [BabM02, BonDT99, MenT03], only few of them

have been concretely implemented by deploying pheromones in shared virtual data spaces [ParBS04], other few realize pheromones by means of ad-hoc physical markers such as special ink or metal dust [SveK04]. In any case, none of them proposes valid solutions to actually spread pheromones in real-world everyday environments. Discarding centralized – not scalable – solutions, as well as power-hungry and costly sensor networks [EstC02, LiR03], it is not easy to find a suitable – cheap, not intrusive, and at the same time flexible – distributed infrastructure on which to store digital pheromones in pervasive environments.

Inspired by this challenge, we propose a novel approach exploiting RFID technology [Wan04] to enforce pheromone-based interaction in pervasive computing scenarios. The key idea of our approach is to exploit RFID tags dispersed in an environment as a sort of distributed memory in which to store digital pheromones. RFID readers, carried by humans or by robots, could deploy pheromone trails in the environment simply by writing pheromone values in the RFID tags around. Also, they could sense such pheromone-trails by simply reading pheromone values in in-range RFID tags. Clearly, such an approach is extremely low cost and not intrusive, as RFID tags will soon be present in any case, in any environment.

Relying on our simple yet flexible approach, a wide range of application scenarios based on pheromone interaction can be realized, ranging from multi-robot coordination [SveK04] to monitoring of human activities [Phi04]. Here, after having illustrated our approach, we detail and evaluate an application to easily find – by following proper pheromone trails – everyday objects forgot somewhere in our homes.

II. PHEROMONE-BASED INTERACTION

Ants and other social insects interact by spreading chemical markers (i.e., pheromones) as they move in the environment, and by being directed in their actions by the perceived concentration of pheromones. This simple mechanism enables ants to globally self-organize their collective activities in a seemingly intelligent way despite the very limited abilities of individuals of acquiring and processing contextual information in a cognitive way. For this reason, systems of social insects are said to be characterized by “swarm intelligence”, to emphasize the difference with “individual” intelligence [BonDT99, Par97].

Manuscript received November 2, 2005. This work was supported in part by the Italian MIUR and CNR in the “Progetto Strategico IS-MANET, Infrastructures for Mobile ad-hoc Networks”.

Marco Mamei is with the Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Italy; e-mail: mamei.marco@unimore.it.

Franco Zambonelli is with the Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Italy; e-mail: franco.zambonelli@unimore.it.

The classical example to show the power of pheromone-based interaction is ant foraging. Ants in a colony, when in search for food, leave the nest and start wandering around. When some food is found, they start spreading a pheromone and try to get back to the nest, thus creating a trail leading to the food source. When an ant is looking for some food, it can indirectly exploit the past experience of other ants by following an existing pheromone trail to reach previously discovered food sources. This also contributes to re-enforce the pheromone trail in that such ant spreads pheromones in its turn. To some extent, the environment becomes a sort of distributed repository of contextual information holding the paths' information to all the discovered food sources. The natural tendency of the pheromones to evaporate if not reinforced, allows the pheromone network to remain up-to-date and to adapt to changing conditions: when some ants discover a shorter path to food, longer paths tend to be abandoned and disappears; analogously, when a food source is extinguished, the corresponding pheromone trail disappears because no longer reinforced [BonDT99].

Despite its simplicity, pheromone-based interaction presents several features that makes it suitable in a lot of distributed and pervasive applications:

- it completely decouples agent (i.e., ant) interactions, which occur indirectly via the mediation of pheromones. This is a very desirable feature in open and dynamic scenarios where agents do not know each other in advance and can come and go at any time;

- it naturally supports application-specific context awareness, in that pheromones provide a representation of the environment in terms of paths leading to food sources;

- it naturally supports adaptation of activities, in that pheromones represent a contextual information that, when no longer updated, tends to vanish;

- the algorithms underlying pheromone-based interaction are simple and involve only local interactions (each ant locally deposits and follows pheromones without any clue – and associated burden/complexity – of being involved in a distributed task).

Given these features it is not surprising that several research proposals, in area as diverse as routing in networks [BonDT99], P2P computing [BabM02, MenT03], robotics [ParBS04, SveK04], self-assembly [SheS02], and (as in our approach) pervasive computing, incorporate and exploit pheromone-based interaction mechanisms.

III. DEPLOYING PHEROMONES WITH RFID TECHNOLOGY

At the core of our approach for deploying digital pheromones in an environment is the technology of Radio Frequency Identification (RFID). RFID tags are small wireless radio transceivers that can be attached unobtrusively to objects as small as a watch or a toothbrush. Tags can be purchased off the shelf, cost roughly €0.20 each and can withstand day-to-day use for years, in fact, being battery-free, they do not have power-exhaustion problems. Each tag is marked with a unique

identifier and provided with a tiny memory (up to some Kb) allowing to store data. Suitable devices, called RFID readers, can be interfaced with portable computers and can be used to access RFID tags by radio for read or write operations. The tags respond or store data accordingly using power scavenged from the signal coming from the RFID reader. RFID readers divide into short- and long-range depending on the distance within which they can access RFID tags, from a few centimeters up to some meters.

A. Scenario Assumptions

Our approach requires a scenario in which the operational environment is densely enriched with RFID tags. Tags can be attached at any – even small – object (we refer to these generically as *object-tags*). Also, tags are assumed to be attached at fixed locations (e.g. doors, corridors, etc.) and at unlikely-to-be-moved objects (e.g. beds, washing machines, etc.). We refer to these tags as *location-tags*. Tagging a location or a fixed object involves sticking a RFID tag on it, and making a database entry mapping the tag ID to a name and a spatial location. RFID readers accessing one of these tags can lookup the tag ID into the database and infer to be close to a specific object or location. Thus, beside pheromones, RFID can be used to enforce a simple yet effective localization mechanism for RFID readers (i.e., for the users or robots carrying them) [Hah04, Sat05].

It is worth emphasizing that current trends indicate that (i) within a few years, many household objects and furniture will be RFID-tagged before purchase and (ii) handheld devices provided with embedded RFID read and write capabilities will have an increasing diffusion (for instance, the Nokia 5140 phone can be equipped with a RFID reader). These factors will make our assumption become a de facto situation, and will make our approach become directly deployable at nearly zero cost.

B. Pheromone Deployment

As anticipated in the introduction, pheromones are created by means of data-structures stored in RFID tags. The basic scenario consists of human users and robots carrying handheld computing devices, provided with a RFID reader, and running an agent-based application.

The agent, unobtrusively from the user/robot, continuously detects in range *location-tags* to infer its current location as it roams across the environment. Moreover, the agent controls the RFID reader to write or read on need pheromone data structures (consisting at least in a pheromone ID) in the tags encountered. This process can create digital pheromone trails distributed across the *location-tags*.

More formally, let us call $L(t)$ the set of *location-tags* being sensed at time t . It is easy to see that an agent can infer that the user/robot is moving when $L(t) \neq L(t-1)$. Thus, if instructed to spread pheromone O , the agent will write O in all the $L(t)-L(t-1)$ tags as it moves across the environment.

For the majority of applications a pheromone trail consisting of only the pheromone ID is not very useful. Like

in ant foraging, most applications involve agents to follow each other pheromone trails to reach the location where the agents that originally laid down the pheromone were directed (or, on the contrary, to reach the location where they came from). Unfortunately, an agent crossing an-only-ID-trail would not be able to choose in which direction to go. To overcome this problem, for each pheromone O , agents store in the *location-tags* not only its ID but also an ever increasing hop-counter $C(O)$ associated with O . If an agent decides to spread pheromone O at time t , the agent reads also the counter $C(O)$ in the $L(t)$. If $C(O)$ is not present, the agent sets $C(O)$ to a fixed value zero. Upon a movement, the agent will store O and $C(O)+1$ in the tags belonging to $L(t+1)$ that do not have O or have a lower $C(O)$.

In addition, to support pheromone evaporation (as described later on), each pheromone has also an associated value $T(O)$ representing the time where the pheromone O has been stored.

The above pheromone data structures are stored in the limited memory of RFID tags. RFID tags, other than with an unchangeable unique identifier, are typically provided with an array of cells, each consisting of few bits. We organize such memory by allocating 3 slots for each pheromone. The first slot will hold the pheromone identifier. The second slot will hold the associated counter. The third will hold the above mentioned timestamp.

C. Pheromone Reading and Evaporation

To read pheromones, an agent trivially accesses neighbor RFID *location-tags* reading their memories. Given the result, the agent will decide how to act on the basis of the perceived pheromone configuration.

To realize pheromone evaporation, since the passive nature of RFID tags does not enable them to directly enforce evaporation, we have adopted the following solution. After reading a tag, an agent checks, for each pheromone O it reads, whether the associated timestamp $T(O)$ is, accordingly to the agent local time, older than a certain threshold T . If it is so, the agent deletes that pheromone from the tag. This kind of pheromone evaporation leads to two key advantages:

1. Since the data space in RFID tags is severely limited, it would be most useful to store only those pheromone trails that are important for the application at a given time; old, unused pheromones can be removed.
2. If an agent does not carry its personal digital assistant or if it has been switched off, it is possible that some actions will be undertaken without leaving the corresponding pheromone trails. This cause old-pheromone trails to be possibly out-of-date, and eventually corrupted.

In this context, it is of course fundamental to design a mechanism to reinforce relevant pheromones not to let them evaporate. With this regard, an agent spreading pheromone O , will overwrite O -pheromones having an older $T(O)$. From these considerations, it should be clear that the threshold T has

to be tuned for each application, to represent the time-frame after which the pheromone is considered useless or possibly corrupted.

IV. PHEROMONE-BASED OBJECT TRACKING

The application we present to exemplify our approach aims at facilitating the finding of everyday objects (glasses, keys, etc.) forgot somewhere in our homes. The application allows everyday objects to leave virtual pheromone trails across our homes to be easily tracked afterwards. Overall, the application works as follows:

1. As from the assumptions, the objects to be tracked are tagged with proper *object-tags*, distinguished from the *location-tags* identifying locations in the environment (object and location tags can be distinguished by their ID).
2. Users (or robots) are provided with a handheld computing device, connected to a RFID reader, and running the object tracking application.
3. The application can detect, via the RFID reader, *object-tags* carried on by the user. Exploiting the mechanism described in the previous section, it can spread a pheromone identifying such objects into the available memory of near *location-tags*.
4. This enables to spread pheromone trails associated with the objects across the *location-tags* of the environment.
5. When looking for an object, a user can instruct the agent to read in-range *location-tags* searching the object's pheromone in their memory. If such pheromone is found, the user can follow it to reach the object current location.
6. Once the object has been reached, if it moves with the user (i.e. the user grabbed it), the application automatically starts spreading again the pheromone associated with the object, to keep consistency with the new object location.
7. This application naturally suits a multi-user scenario where an user (or a robot), looking for an object moved by another user, can suddenly cross the pheromone trail left before by the object.

A. Spreading Object Pheromones

To spread pheromones, the application needs first to understand which objects are currently being carried (i.e. moved around) by the user. To perform this task unobtrusively, it accesses the RFID reader to detect in-range RFID tags once per second.

Let us call $O(t)$ the set of *object-tags* being sensed at time t , $L(t)$ the set of *location-tags* being sensed at time t . If the agent senses an *object-tag* O such that $O \in O(t)$, $O \in O(t-1)$, but $L(t) \neq L(t-1)$, then the agent can infer that the user picked-up the object O and the object is moving around. In this situation, the agent has to spread O pheromone in the new location. To this end, the agent writes O in the available memory space of all the $L(t)$ *location-tags* that do not already contain O . This

operation is performed, for every object O , upon every subsequent movement. Similarly, if the agent senses that an *object-tag* $O \in O(t-1)$, but $O \notin O(t)$, then the agent infers that the user left the object O . When this situation is detected the agent stops spreading the O pheromone.

These operations create pheromone trails of the object being moved around.

B. Tracking Objects

Once requested to track an object O the agent will start reading, once per second, nearby *location-tags* looking for an O -pheromone within the sensed *location-tags* $L(t)$. If such a pheromone is found, this implies that the user crossed a suitable pheromone trail.

There are two alternatives: either $L(t)$ contains only one *location-tag*, or $L(t)$ contains at least two *location-tags* having O -pheromones with different $C(O)$.

In the former case, the application notifies the user about the fact he has crossed a pheromone trail, but nothing else. In such situation, the user has to move in the neighborhood, trying to find higher $C(O)$ indicating the right direction to be followed (this is like dowsing -i.e. finding underground water with a forked stick – but it works!). We refer to this as *local-search*.

In the latter case, the agent notifies the user about the fact he has crossed a pheromone trail and it suggests to move towards those *location-tags* having the higher $C(O)$. In the following, we will refer to this as *grad-search*, since it is like following a gradient uphill. With this regard, it is important to emphasize that *grad-search* is likely to be available only with RFID readers with a range long enough to include in $L(t)$ at least two tags storing the pheromone trail. Moreover, since we do not require the presence of localization devices, the agent suggests the user to get closer to the location having higher $C(O)$, by naming the location – e.g., walk to the “front door” – and the user has to know how to get there without further help.

In either cases, following the agent advices, the user gets closer and closer to the object by following its pheromone trail, until reaching it.

V. EXPERIMENTS

To assess the validity of our approach and the effectiveness of the object tracking application, we developed a number of experiments, both adopting the real implementation and an ad-hoc simulation (to test on the large scale).

A. Real Implementation Set-Up

The real implementation consisted in tagging places and objects within our department (Figure 1a). Overall, we tagged 100 locations within the building (doors, hallways, corridors, desks, etc.) and 50 objects (books, laptops, cd-cases, etc.). Locations have been tagged with ISO15693 RFID tags, each with a storage capacity of 512 bits (each tag contains 60 slots, 1 byte each, thus it is able to store 20 pheromones). Objects have been tagged with ISO14443B RFID tags, each with a

storage capacity of 176 bits (each tag contains only the object ID).

For users, we exploited HP IPAQ 36xx PDAs, each running Familiar Linux 0.72 and J2ME (CVM – Personal Profile) and provided with a WLAN card and an Inside M21xH RFID reader (Figure 1b).

In addition, mobile robots have been realized by installing one PDA connected to a RFID reader onboard of a Lego Mindstorms robot (www.legomindstorms.com). The IPAQ runs an agent controlling both the RFID reader and the robot microprocessor (Figure 1c).

Finally, a wirelessly accessible server holds a database with the associations between tag IDs and places' and objects' description (i.e. ID 001 = Prof. Smith's office door). The IPAQ can connect, via WLAN, to the database server to resolve the tag ID into the associated description. Each IPAQ runs the described application



Figure 1. (a) Some tagged objects. (b) The test-bed PDA hardware. (c) The Lego Mindstorms robots with a PDA and an RFID reader mounted aboard.

B. Simulation Set-Up

To test more extensively and on the large scale, we realized a JAVA-based simulation of the above scenario. The simulation is based on a random graph of places (each associated to a *location-tag*), and on a number of objects (each associated to an *object-tag*) randomly deployed in the locations-graph. Each tag has been simply simulated by an array of integer values.

A number of agents are simulated wandering randomly across the locations-graph, collecting objects, releasing objects, and spreading pheromones accordingly. At the same time, other simulated agents are looking for objects in the environment eventually exploiting pheromone trails previously laid down.

For the sake of comparison, we tested both the *local-search* algorithm in which the agents perceive the pheromones in their current node, but cannot see the direction in which the pheromones increase, and the *grad-search* algorithm, in which the agents perceive pheromones together with the directions in which they increase. Also, these have been compared with a *blind-search* algorithm, in which agents wander randomly fully disregarding pheromones.

C. Results of the Experiments

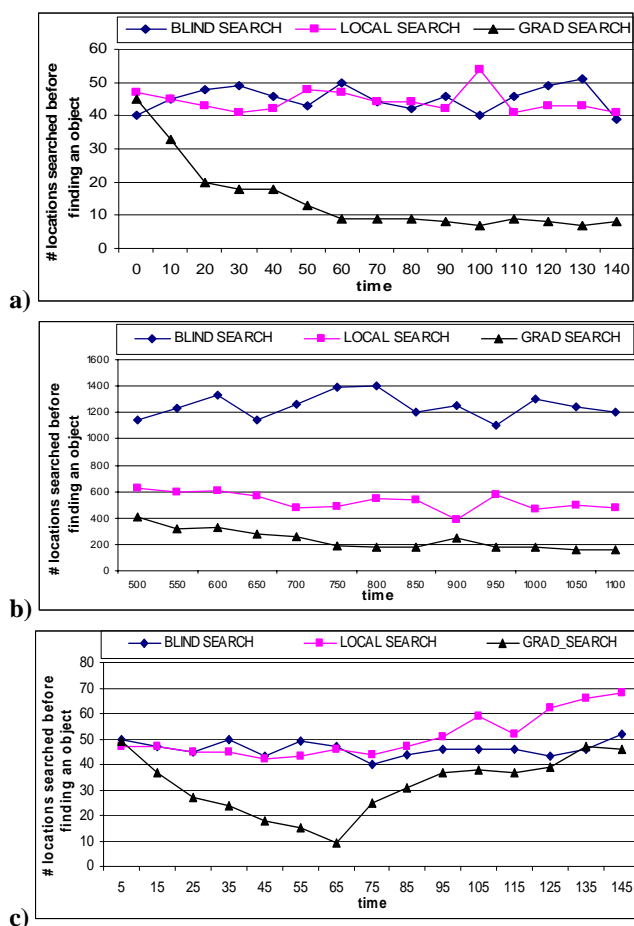


Figure 2. (a-b) Number of places visited before finding a specific object plotted over time in environments consisting of: (a) 100 tagged places, and (b) 2500 tagged places. (c) Number of visited places before finding a specific object plotted over time, when tags tend to saturate.

A first group of experiments aims at verifying the general effectiveness of our approach and of the object-tracking application. We report results from two different simulation

scenarios: the first consisting in 100 tagged places with 100 objects (Figure 2a); the second consisting in 2500 tagged places with 500 objects (Figure 2b). A number of 10 agents have been simulated to populate these environments wandering around moving objects and spreading pheromones and, at the same time, looking for specific objects. In the experiments, we report the number of places visited before finding specific objects, for different search methods, plotted over time. The reported results are average of about 300 simulations and qualitatively in line with the results obtained – on a smaller scale and on a more limited number of experiments – on the real implementation.

Starting from a scenario free of pheromones (time zero in Figure 2a), the more time passes the more pheromone trails get deployed. *Blind-search* does not take advantage of pheromone trails: objects are found after visiting on the average half of the places. *Grad-search* takes a great advantage of pheromones: after an initial period, and when several pheromone trails have been deployed less than 10% of the places need to be visited before finding the object. *Local-search*, at least in the small-scale scenario of Figure 2a, appears not to take any relevant advantage of pheromones. This is due to the cost of orienting in the environment to find the proper direction.

The situation changes when getting to larger-scale scenarios (as in Figure 2b, reporting experiments from a time past the initial transitory). There, both *local-search* and *grad-search* appears reasonably effective. The performance improvement of *local-search* is due to the fact that the cost of “orienting” in a local neighborhood becomes negligible when the environment is large. Thus, although the *grad-search* algorithm is always preferable, in large-scale scenarios our approach is effective even when using short-range RFID reader enabling to enforce the *local-search* algorithm only.

A second group of experiments aims at exploring the effects of RFID tag storage saturation upon pheromone spread. This of course represents a big problem, in fact, it can happen that pheromone trails can be interrupted, because there is not available space left on neighbor *location-tags*, while the object to be tracked moves away. This creates a broken pheromone trail leading to a place that is not the actual location of the object.

In Figure 2c, we report an experiment conducted in the 100-tagged-places-environment described before. This time the tag capacity has been fixed to 50 pheromones (150 bytes), and we plot the number of places visited before finding specific objects, for different search methods, over time. Let us focus on the *grad-search* behavior. It is easy to see that, when time is close to zero, *grad-search* works equal to *blind-search*, since no pheromone trails have been already laid down. After some time, *grad-search* works considerably better than *blind-search*, since pheromone trails drive agents. However, as time passes, tags capacity tend to saturate, the objects are moved, but no pheromone trails can be deployed. This situation rapidly trashes performance leading back to *blind-search* performance. In our real implementation (tags with a 512 bits

capacity) the above problem leads to broken trails when more than 30 objects are tracked.

In conclusion, it is rather easy to see that the limited storage capacity of the RFID tags represents a problem for our approach. Basically, if the number of objects to be tracked is greater than the available slots on the RFID tag, in the long run, the problem is unavoidable. Sooner or later, a new object will cross to an already full tag, breaking the pheromone trail. The pheromone evaporation mechanism that we implemented did not help this situation. In another set of experiments, we verified that performance remains more or less the same of Figure 2c, independently on the parameters used to control and tune the evaporation mechanisms.

We still do not have a solution for this problem. Our research is leading in two main directions: (i) we are currently researching more advanced pheromone evaporation mechanisms. (ii) We are considering the idea of spreading pheromone trails not only in *location-tags* but also on *object-tags*. The advantage would be that the more objects are in the system, the more storage space is available for pheromones, letting the system to scale naturally. The problem is how to manage the fact that *object-tags* containing pheromones can be moved around, breaking the pheromone trail structure. As a partial relief from this problem, it is worth reporting that recent RFID tags have a storage capacity in the order of several of KB, making possible to track hundreds of objects without changing our application.

VI. OTHER APPLICATION SCENARIOS

Pheromone interaction and stigmergy have attracted more and more researches due to their power in supporting agent coordination in a variety of scenarios. Thus, it is not surprising that even our proposal for RFID pheromone deployment could find a number of additional applications, beside the presented object-tracking.

In general, RFID tags in an environment and associated to objects can be used to improve *Context-Awareness*. The use of RFID tags as a simple tool for localization (i.e., for location-awareness) has already be outlined. More in general, RFID tags can be used to help users (as well as robots) in getting aware of what's in the environment more than their natural and artificial senses can do, by reading the additional information provided by tags. One of the most interesting work in this direction has been presented in [Phi04]: a software application is able to infer the users' daily activities on the basis of the objects he touches (e.g. if the user touches a teapot and a cup, the application can infer that he is preparing tea). All these facets of context-awareness – which mostly exploit information assumed to be already stored in tags, can be enriched by the ideas presented in this paper, suggesting to: (i) exploit RFID tags in the environment as a sort of distributed shared memory for writing contextual information; (ii) exploit pheromones to keep a traceable distributed track of past environmental activities. For instance, in the application for inferring daily activities, one could think

that – once the application recognize that some tea has been prepared, the teapot start spreading a pheromone trail leading to the fridge and indicating that some tea has already been prepared and is there to cool down.

In line with these ideas is the concept of *Pervasive Workflow Management*. Standard workflow management systems are rooted on a software engine keeping track of the status of the workflow being carried on. Workers notify to this engine the tasks being completed and the engine in turn notifies the subsequent tasks that have to be carried on. RFID tags and pheromone-based interaction could remove the need for a centralized engine in pervasive computing environments and lead to more situated and adaptive scenario. For instance, the RFID tags associated to the items to be processed could store a marker identifying the operations that the item undertook. Workers with RFID readers could simply read the state of the item and process it further. This approach could be employed in traditional manufacturing scenarios as well as in more mundane domestic workflow (e.g. store in the pet's collar a pheromone indicating if it has already eaten or not).

Of course, taking inspiration from the way pheromones are used by ants to let them coordinate their collective movements in an unknown environment toward food sources, we could think at exploiting RFID pheromones to enable a group of users and robots to coordinate on-the-fly their movement in an environment (consider, e.g., a rescue team in a disaster area). For instance, if users spread pheromones around as they walk and are instructed to stay away from existing pheromone trails, one can have reasonable guarantees that the whole environment is explored in an effective way by the group [SveK04].

In this context, it is important to remark that our approach clearly requires the presence of RFID tags before pheromones can be spread. Although RFID tags are likely to be soon densely present in everywhere (embedded in tiles, bricks, furniture, etc.), one cannot rely on this in sensible situations like in a disaster area. In these cases, however, it is possible to conceive solutions where users or robots physically deploy RFID tags on-the-fly while exploring the environment, to be used for subsequent coordination.

VII. CONCLUSIONS

While a preliminary prototype implementation shows the feasibility of our approach, a number of research directions are still open to improve its practical applicability. In particular, more experiments are required to verify the scalability of the proposed architecture to very large-scale scenarios, and more effective solutions must be found to the problem related to broken pheromone trails.

REFERENCES

- [BabM02] O. Babaoglu, H. Meling, A. Montresor, "A Framework for the Development of Agent-Based Peer-to-Peer Systems", Proceedings of the IEEE International Conference on Distributed Computing Systems, Vienna (A), IEEE CS Press, May 2002.
- [BonDT99] E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm Intelligence*, Oxford University Press (Oxford, UK), 1999.

- [EstC02] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", *IEEE Pervasive Computing*, 1(1):59-69, Jan.-March 2002.
- [Hah04] D. Hähnel, W. Burgard, D. Fox, K. Fishkin, M. Philipose, "Mapping and Localization with RFID Technology", Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona (ES), IEEE Press, April 2004.
- [LiR03] Q. Li, M. De Rosa, D. Rus, "Distributed algorithms for guiding navigation across a sensor network", Proceedings of the ACM Conference on Mobile Computing and Networking, San Diego, CA (USA), ACM Press, October 2003.
- [MenT03] R. Menezes, R. Tolksdorf, "A New Approach to Scalable Linda-systems Based on Swarms", Proceedings of the ACM Symposium on Applied Computing, Orlando, FL (USA), ACM Press, March 2003.
- [Par97] V. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems", *Annals of Operations Research*, 75:69-101, 1997.
- [ParBS04] V. Parunak, S. Brueckner, J. Sauter, "Digital Pheromones for Coordination of Unmanned Vehicles", Proceedings of the 1st International Workshop on Environments for Multi-agent Systems, LNAI 3374, Springer Verlag, 2004.
- [Phi04] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, D. Fox, H. Kautz, D. Hähnel, "Inferring Activities from Interactions with Objects", *IEEE Pervasive Computing*, 3(4):50-57, 2004.
- [Sat05] I. Satoh, "A Location Model for Pervasive Computing Environments", Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications, Kauai Island, HI (USA), IEEE CS Press, March 2005.
- [SheS02] W. Shen, B. Salemi, P. Will, "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots", *IEEE Transactions on Robotics and Automation*, 18(5):1-12, Oct. 2002.
- [SveK04] J. Svennebring, S. Koenig, "Building Terrain Covering Ant Robots: a Feasibility Study", *Autonomous Robots*, 16 (3):313-332, May 2004.
- [Wan04] R. Want, "Enabling Ubiquitous Sensing with RFID", *IEEE Computer*, 37(4):84-86, April, 2004.

Marco Mamei is an associate researcher at the University of Modena and Reggio Emilia, where he received the PhD in computer science. His current research interests include distributed and pervasive computing, swarm intelligence, and self organization. He is a member of the IEEE, AIIA, and TABOO. Contact him at Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Via Allegri 13, Reggio Emilia, Italy; mamei.marco@unimore.it.

Franco Zambonelli is a professor of computer science at the University of Modena and Reggio Emilia. His research interests include distributed and pervasive computing, multiagent systems, and agent-oriented software engineering. He received his PhD in computer science from the University of Bologna. He is a member of the IEEE, the ACM, AIIA, and TABOO. Contact him at Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Via Allegri 13, Reggio Emilia, Italy; franco.zambonelli@unimore.it.

CASMAS: An Agent-Based Support for Modulated Participation in Cooperative Applications

Federico Cabitza, Marco P. Locatelli and Marcello Sarini

University of Milano – Bicocca {cabitza, locatelli, sarini}@disco.unimib.it

Abstract—This paper proposes CASMAS: an agent-based model to design an environment of collaborative applications by taking into account the notion of community. Within this model, communities are characterized by declarative rules that express and shape the participative behavior of the community members. The degree of participation of each member can dynamically change according to her physical location and her position in the logical space of the applications used within the community. The paper shows how this approach can facilitate the design of collaborative applications that are community-aware, that is augmented with mechanisms by which to manage different levels of participation of the community members.

Index Terms—Computer-supported cooperative work, Multi-agent systems, Pervasive Computing

I. BACKGROUND AND MOTIVATIONS

AS widely recognized in the specialistic literature [1] the multi-agent approach makes easier to define a clear separation between the units of computation and the interactions among them in order to achieve some application goals through "separation of concern" and modularity [2]. Moreover agents can be conceived of as useful tools to describe (complex) systems from a systemic point of view. Because of the complexity of systems to design, it is impossible to predict (and design) in advance all the possible behaviors of the running system: hence agents are provided with simple behaviors and interaction capabilities and let interact within some computational environment so that the system is able to cope with unpredictable patterns of conditions by exhibiting an overall behavior that is an emerging property of the system itself [3]. The relevance of these approaches is also due to some important characteristics that they provide to designers: distributedness, openness, scalability, incremental design. In fact, agents are inherently distributed, and this makes the system more easily open, in terms of the possibility to add new elements given that they behave according to the established protocol; and robust, in terms of easy substitution of malfunctioning agents and of modification of incrementally designed agents.

More recently the characteristics of agent-based approaches

have been also considered in the light of the design of applications that support collaboration among people [4]. In the area of computer supported cooperative work (CSCW) cooperative applications pose strong requirements in terms of flexibility, adaptability, openness to environment in order to reflect the complexities of real work settings, that is of environments (or workplaces) where people work distributed in space, and can freely join and leave dynamically collaboration spaces, where collaborative behaviors can change according to the context. Agent-based approaches have been proposed to support different aspects of human collaboration: some of them are focused on the management of workflows which require adaptivity and dynamicity in dealing with a flow of work (representing either tasks to be accomplished or documents) among team members (see for instance [5]). Other agent-based approaches deal with coordination issues, ranging from support to not very structured interactions among members of small groups like the ones occurring in meetings [6] to more prescriptive interactions among distributed actors mediated by appropriate coordination mechanisms [7]. One of the most critical aspects concerning human collaboration is about how people act, learn, and interact together within the so-called communities of practice. In our view, the notion of community (in the sense initially proposed, and denoted as Community of Practice, by Wenger [8] and further articulated by Andriessen [9]) is a good mean to conceptualize how people mutually recognize, gather together, interact, collaboratively access and share resources, and move around to meet other people and exploit further resources. In fact, a community is spontaneously built, grows and evolves legitimating various degrees of participation of incoming members on the basis of its internal rules, conventions and practices: this is usually called "legitimate peripheral participation". The degree of participation of an actor is proportional to its distance from the center of the community, i.e., from the locus where the (physical and/or logical) ties which link its members together are stronger.

In our view, the possibility of considering different degree of participation of community members is a crucial aspect to be taken into account so as to design applications supporting

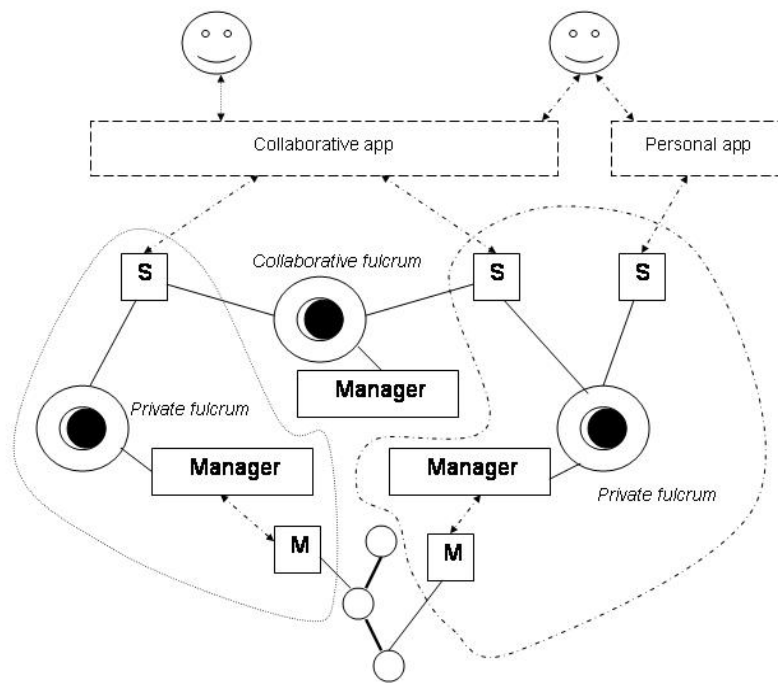


Fig. 1. The CASMAS model.

collaboration among community members or, in other words to build a new typology of applications which are more community-aware in terms of support based on their inner membership and participation mechanisms. With community-aware applications we hint to the fact that applications conceived as supportive of cooperative work can also play a significant role in supporting community life and community-oriented activities. We think this can be made possible if these applications are embedded within a network of interactions and information flows that occur between the human actors, their personal devices and the applications they use. In this common and shared information space (cf. later the concept of Fulcrum) the cooperative applications can then become aware of the way different levels of participation are managed with respect to the users' ability to get accustomed and align with the practices, conventions, artifacts, and knowledge sharing (learning) modalities of the community they belong to.

Taking into account the above considerations about agent-based approaches and the relevance of modulating community participation, we aim at defining an agent-based model, the CASMAS (Community-Aware Situated Multi-Agent Systems), which could be used to design community-aware applications. This model results from an extension of Santana, a framework for the management of distributed inference based on reactive behaviors programming [10], with the main features of MMass (Multi-layered Multi-Agents Situated Systems), a model that has been proposed for managing and modulating awareness information in cooperative applications [11, 12]. In fact, our model should be able to recognize and support modulated participation of members of a community where modulated participation calls for a notion of metrics and the latter for a notion of topological space. For this reason, the Santana framework, which is able to model

distributed computational capabilities together with the sharing of information and reactive behaviors through rules mobility, has been integrated with the MMass model in which the topological space and the consequent modulated diffusion of information are first-class objects.

Other agent-based models like Co-Field [13] implemented by TOTA [14] could be used to modulate the different degree of participation of community members since they take into account concepts of topological space, distance and propagation of information which is modulated within the space itself. But in those cases the modulation of information is influenced according to only a topological structure usually representing the underlying network architecture rather than one or more topological structures representing also logical aspects of the domain. Indeed, CASMAS allows for the definition of general criteria by which to establish the level of membership of people in a community through the notions of topological space and of field diffusion, which can represent a combination of both physical and logical aspects that dynamically characterize the community membership. Moreover, the same notions allow for the computation and modulation of different levels of participation. Hence by our model, we provide designers with a richer semantics in defining different metrics expressing possible levels of participation since the model makes possible to combine the mutual physical position of users, as well as their logical location, in order to define how the information can be modulated through the environment and according to the relationships among members. The paper is organized as follows. The next section presents the CASMAS (Community-Aware Situated Multi-Agent Systems) model, which integrates the main features of Santana and of MMass. Then, a high-level software architecture to implement the

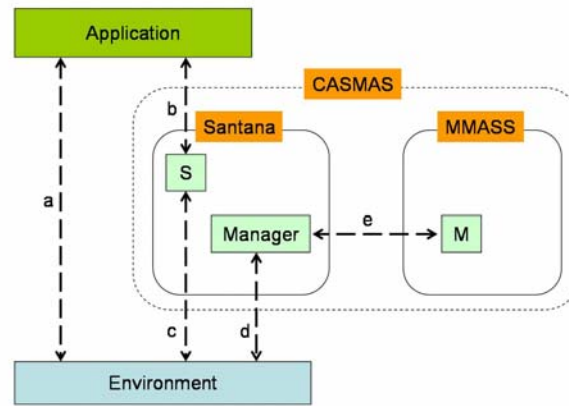


Fig. 2. The CASMAS' software architecture.

model is presented. Next the model is illustrated through a scenario. The state of its current implementation and its foreseen developments conclude the paper.

II. MMASS AND SANTANA IN A NUTSHELL

MMASS is a multi-agent model based on the *perception-reaction* paradigm. Agents are located on *sites* that constitute a *topological space* determining the agents mutual perception. In fact, agents can *directly interact* when they are located in close sites or can *remotely interact* when they are sensitive to the signals emitted by other agents. These signals within the MMASS model are called *fields* and their intensity is modulated by space according to a *diffusion function*, which takes into account the space topology. A *sensitivity function* characterizes each agent type and takes its current state as argument. The perception of a field by an agent triggers a reaction that can cause a *change of the perceiver's state or position or the emission of a field* by the perceiving agent. A system can be composed of several topological spaces (*multi-layers*), each characterized by its agents and their behaviors; layers communicate by means of *exported-imported fields*. Ad hoc layers that fictitiously represent applications can send information to the MMASS by means of imported fields and in so doing awareness information about those applications can be properly managed (more details on this architecture can be found in [12]).

Santana is a methodological framework conceived for the development of distributed inference systems in the Pervasive Computing application domain. The Santana framework is grounded on the *interconnection metaphor* in that any Intelligent Environment is conceived as a *web of computational sites* where devices of different computational and interactional capabilities interact. Interaction is realized (or better yet, mediated) through a *blackboard* mechanism, that is through a common space where devices share contextual information (called *facts*) as well as reactive behaviors (called *rules*), which can be acquired by or moved across the computational sites. In this way, the pervasive environment can reach an intelligent behavior as a result of synchronous inference activities exhibited by distributed

computational sites. Moreover, a blackboard approach makes the computational environment quite *flexible towards dynamic situations*: new devices, new actors leaving and joining the system, interaction patterns varying according to the context can be dynamically managed by means of suitable meta-rules that act as bridges between concepts (represented by declarative facts) and rules and that hence allow for the (de)activation of behaviors on an event-driven basis (i.e., the local, as well as the “global”, control flow is not completely predetermined by the programmers of the devices and applications involved in the same pervasive environment).

A. Their integration into CASMAS

Grounding on the two models outlined above, we then propose CASMAS: a model by which to conceive a “loose” integration between collaborative applications so that they can become more “community-aware”. To reach such high-level goal, CASMAS combines the MMASS functionality of modulating information between agents and the Santana functionality of supporting cooperating agents in sharing information and behaviors (e.g., tasks and ways to accomplish them). The combination of these two approaches fits the requirements of a *cooperative intelligent environment* that in CASMAS is interpreted as a *constellation of dynamically defined and interacting communities*. On one hand, cooperation requires the notion of agent as entity able to perceive context and propagate information on that context, as well as the notion of modulated mutual perception (awareness) among agents, that is a first-class concept of MMASS. On the other hand, cooperation in an Intelligent Environment requires the functionality of Santana to manage disparate and scattered devices, private and common information spaces as well as agents that are aware of context and endowed with behaviors that are adaptive and reactive to context [15]. Accordingly, the rationale behind CASMAS is to model a cooperative Intelligent Environment as composed of two main parts. First, a set of common information spaces, called *fulcra*, by which information and behaviors concerning communities practices and individual actors are managed (respectively, *cooperative fulcra* and *private fulcra* - see Figure 1). Each fulcrum is accessed by *S-agents*, one for each

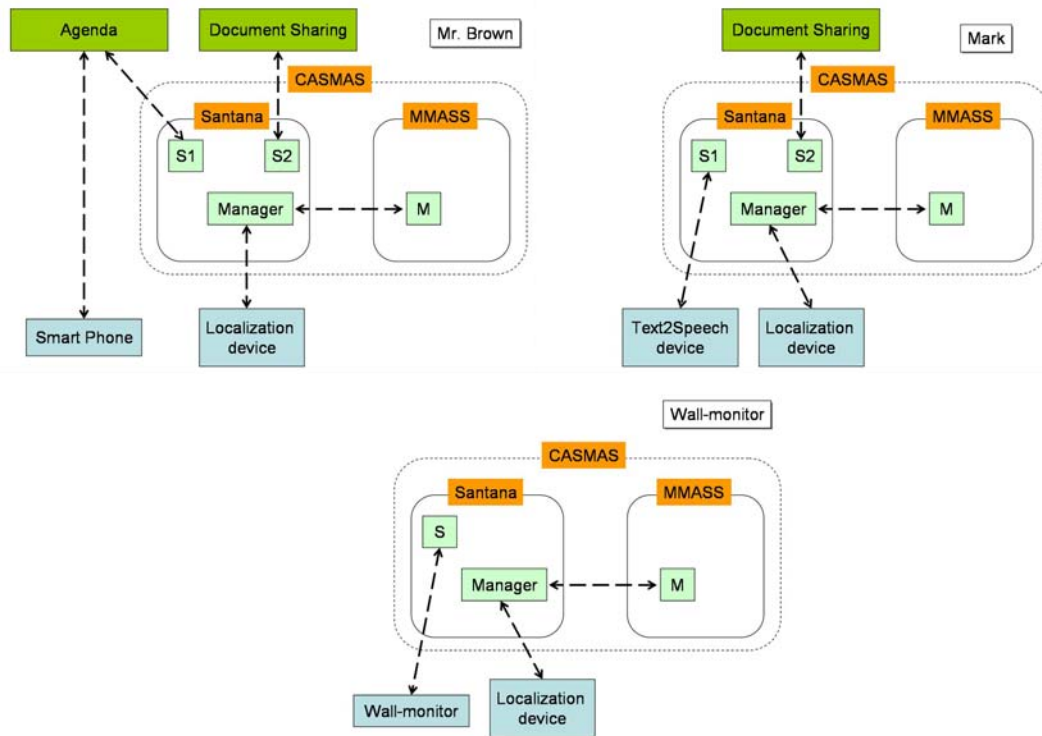


Fig. 3. Examples of configuration of the CASMAS architecture.

(human) actor involved in the (cooperative) application(s) in use by the community¹. Through the fulcrum, S-agents can share both declarative representations of context (facts) and reactive behaviors that characterize the community in terms of conventions, practices or shared knowledge. Accordingly, these behaviors are called *community rules* since, by being *shared* and *followed* by all the community members they literally make and demarcate the community. As a community-oriented specialization of Santana, CASMAS provides the designer with two transparent mechanisms to manage community rules that are implemented through suitable CASMAS *meta-rules*. The first is called *community enforcing* and it is used to manage inhibition of community rules, as well as updating and overwriting once they have been fetched within each S-agent. By means of this mechanism, community rules can dynamically change to reflect a more context-aware alignment of the community members towards common and ever-changing cooperative goals. The second mechanism is called *community participating*. Through this mechanism, according to the number of members that activate a community rule and the number of activations of that rule, the *salience* of that rule is dynamically changed (that is, it is modified the rule attribute expressing the probability of the rule to be chosen, activated and then executed at a certain contextual condition). This *participating* mechanism allows a community to change dynamically its nature and policies, also according to the contextual response of its members to these

policies. In this way a rule that has been first “injected” into a fulcrum with a low salience (and that hence can be seen as just a *suggestion*) can become a (shared) *practice* and then even a prescriptive *direction* according to its changing and growing salience.

All S-agents that stand proxy for a human actor² (e.g., A) in some collaborative fulcra are also connected with the private fulcrum associated to A: this allows a smooth interaction between private and cooperative tasks and information repositories, thus fulfilling a well known requirement of cooperation.

The second part of a CASMAS model encompasses a set of topological spaces that are “inhabited” by M-agents whose behavior is defined according to the MMAS model; CASMAS spaces can have a dynamic structure, a feature inherited by MMAS, but this feature is not used in the scenarios described later. Besides conveying contextual information, the role of M-agents is to compute the degree of participation of human actors in the communities that are built around the collaborative fulcra. The interplay between sensitivity to fields and fields propagation, which depends on M-agents state and position, “shapes” the M-agents mutual perception and computes how tight their mutual proximity is. This information flows towards the fulcra described above; these “react” to this flow by implementing the desired degree of participation through the adaptive behavior capability provided by Santana. This flow of information, modelled in

¹ The behavior of S-agents can either fully define the cooperative application associated to the collaborative fulcrum or, more realistically, define an interface between the cooperative application and the pervasive environment in which it is activated.

² In principle, when we refer to human actors also artificial ones could be considered. However, since the focus of the paper is on human cooperation, we will refer to human actors; if cooperation involves artificial actors as well, the extension of the illustrated mechanisms to them is immediate.

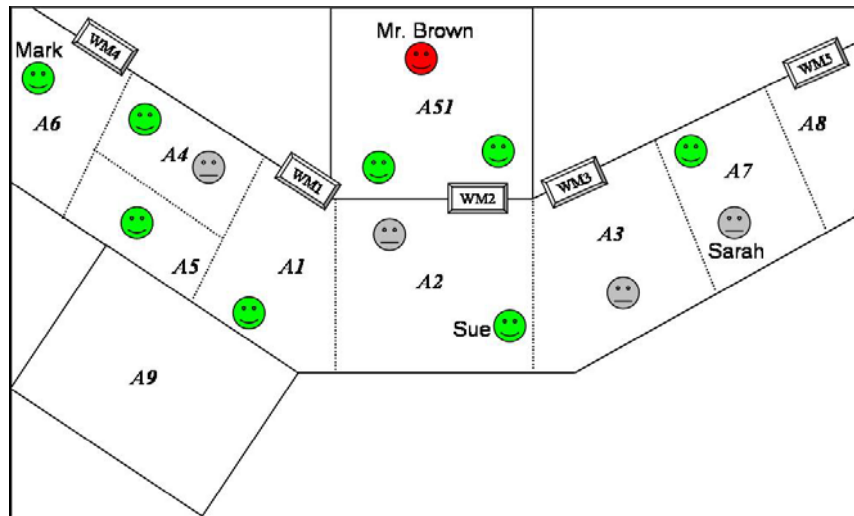


Fig. 4. The described scenario at the University. PCC workshop is located in room A51.

terms of exported fields, is the basic means by which the integration between Santana and Mmass is realized, and it makes the interaction with the external environment bidirectional. In order to realize this integration, while keeping the frameworks both fully decoupled and autonomous in their use and implementation, at each fulcrum is associated a special S-agent, called Manager. This agent is characterized by some *rendering rules*, that is rules that transform topological representations of the Mmass model into declarative representations (facts) by which the communication between the fulcrum and the M-agents populating the topological space(s) is managed. The following section illustrates the CASMAS functionalities through two scenarios: a simple scenario will show the communication patterns in some detail, while a more complex one will more clearly describe the CASMAS expressive power.

III. CASMAS SOFTWARE ARCHITECTURE

As stated in the previous sections, CASMAS is a model by which to conceive a “loose” integration between collaborative applications so that they can become more “community-aware”. Accordingly the model must be open to the software applications and to the environment as well. Due to these requirements, the points of interaction between the model architecture, the applications and the devices must be identified, so to characterize the high-level software architecture of CASMAS (see Figure 2). The architecture is composed of a Santana module, which includes S-agents (S in Figure 2), a Manager, a private fulcrum and the community fulcrum; and of a Mmass module, which includes M-agents (M in Figure 2) and a topological graph. In our view, the interaction between the environment and the CASMAS architecture is delegated to the S-agents in that they can interact both with the software applications and the environment (arrows *b* and *c* in Figure 2). Conversely, the Manager can only interact with the environment (arrow *d* in Figure 2) and specifically only with the localization devices in

order to acquire the physical location of the person that they are associated with. Software applications, which are entities outside the CASMAS architecture, can interact directly with the environment (arrow *a* in Figure 2).

The interaction between an S-agent (or the Manager) and the devices in the environment can be bidirectional and it is mediated by a proxy fact, i.e., a fact that represents the visible state of a device and that is declared in the private fulcrum associated to the device’s user. The S-agent that interacts with the device owns those rules that can be fired by changes in the proxy fact; by modifying the proxy fact, this agent is also able to modify the state of the corresponding device. In this way, the S-agent and the device are fully decoupled but the strict relationship between them is preserved by putting the rules only in the interested S-agent. This approach has several benefits: first, the device is potentially visible to all the S-agents linked to the private fulcrum; in this way, two S-agents can interact with two different non-overlapping functionalities of the same device. Secondly, the S-agent that owns the rules by which to interact with the device can delegate this interaction to another S-agent (linked to the same private fulcrum) simply by sending it the related rules; thirdly, the system is more fault tolerant in that, e.g., if the S-agent that interacts with the device stops working, another S-agent could manage the interaction with the device.

A. CASMAS at Work

To illustrate how the CASMAS model achieves its goal of supporting collaboration, we describe a scenario and the related CASMAS mechanisms.

The PerCom University is endowed with ID emitters that allow the identification of different zones of its building and with wall-monitors that show information about ongoing initiatives. Every member always carries at least one localization device (eventually embedded in something that the person carries always with her, e.g., the wrist-watch) that is able to perceive area IDs.

Today the University hosts a workshop entitled “PCC:

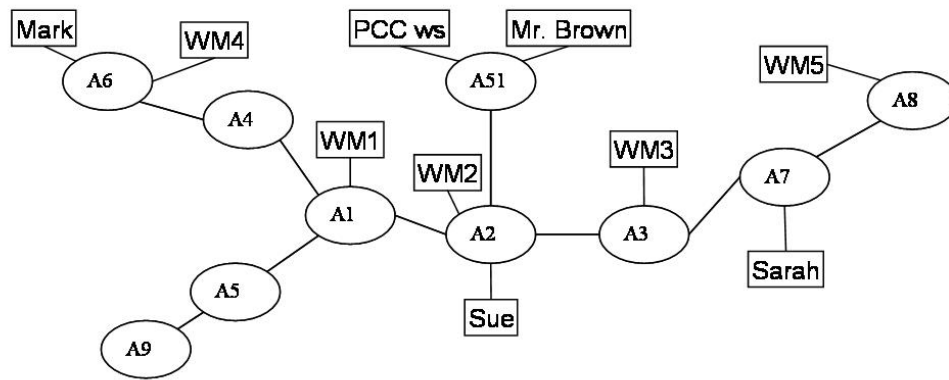


Fig. 5. Localization graph and M-agents located on it.

Pervasive Computing Challenges” that is scheduled for 10 a.m. Sue, Sarah and Mark (see configuration of the CASMAS architecture in Figure 3) are three people working at the PerCom University: Sue and Mark are interested in the PCC workshop. Currently Sue is in the corridor close the workshop room while Mark and Sarah are far from it (see Figure 4). Moreover, Mr. Brown (see configuration of the CASMAS architecture in Figure 3), the workshop speaker, is in the coffee room having a cup of good coffee.

Since Mr. Brown had previously set the commitment in his Agenda, at 9.45 a.m. his Smart Phone vibrates and shows a message reminding the scheduled event. When Mr. Brown arrives at the workshop room, his Agenda infers that the workshop is going to start and publishes this information. In order to reduce the information overload in the spirit of calm technologies [16], the information is showed on the wall-monitors (see configuration of the CASMAS architecture of a wall-monitor in Figure 3) close to the workshop room and notified only to the persons who are far from it. Therefore, Mark’s personal device perceives the information “workshop PCC is starting” while Sue and Sarah’s personal devices do not perceive it because of two different reasons: Sarah is not interested in the workshop, while Sue is interested but she is close the workshop room so she can see the notification on the wall-monitor.

When persons interested in the workshop approach the workshop room, they become member of the PCC workshop community (in this scenario the degree of participation to the community is limited to being or not member of the community, next we provide more information about modulated participation) and share rules and information that characterize it: for example, any “ringing device” owned by participants must be turned to silent mode. This happens to Mark and Sue when they enter in the workshop room. In addition, the PCC community states that the workshop speaker can publish his Curriculum Vitae (CV) and that members of the community can retrieve it if they like. Before the workshop begins, Mr. Brown publishes his CV to the community through his personal device; since Sue and Mark are members of the community their device either

automatically retrieves the speaker’s CV or asks them if they want it, according to their preferences, through the “Document Sharing” collaborative application (see Figure 3).

In order to model the illustrated scenario in CASMAS, a localization graph (see Figure 5) is needed to take into account the physical location of the different entities (people, devices, activities) and model the information modulation accordingly. The M-agent of the PCC workshop, which is an activity, emits a field on the localization graph to notify that the workshop is occurring.

Moreover, the model encompasses as many private fulcra as many human actors are using an instance of the Agenda application, and a single collaborative fulcrum that manages the workshop policies.

When Mark schedules in his Agenda that he will take part in the PCC workshop, his Agenda’s S-agent asserts in his private fulcrum the fact (see Figure 6)

1) X is interested in the PCC workshop

(where X is a parameter that represents the person) so that the Agenda’s Manager forwards this information to Mark’s M-agent through the filtering rule

2) if X is interested in the PCC workshop then send the external field “PCC workshop fields” to the X’s M-agent

The same holds for Sue.

According to the scenario, the sensitivity function of wall-monitors’ M-agents let them perceive only fields about workshops happening in the areas close to them.

When Mr. Brown enters the PCC workshop room, his M-agent perceives the PCC workshop field and emits the “PCC workshop is starting” field on the graph; Figure 7 illustrates the field diffusion and the perception by M-agents which represent persons and wall-monitors in their various locations. Hence, the wall-monitors close to the workshop room show this information, because their M-agents have perceived the field. This happens also to Mark because his M-agent communicates to the Manager of his private fulcrum that the

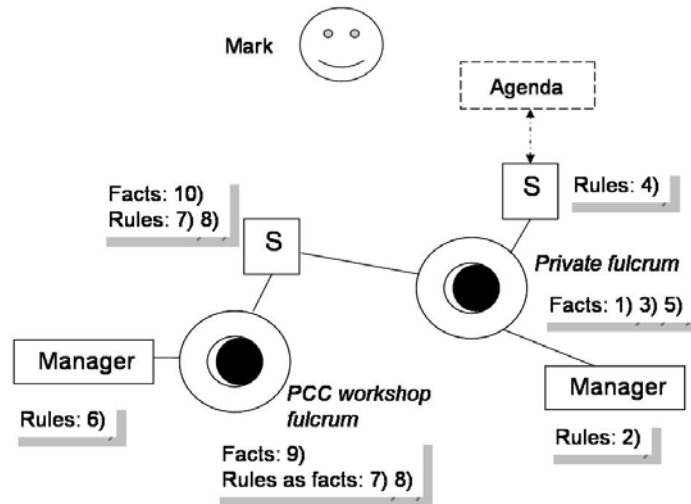


Fig. 6. Distribution of rules and facts on S-agents and fulcra.

PCC workshop is starting; the Manager declares the fact

3) the PCC workshop is starting

in the private fulcrum so the S-agent related to his Agenda can react to it, due to the rule

4) if the PCC workshop is starting and X is interested in the PCC workshop then send the message “the PCC workshop is starting” to the X’s Agenda

and the Agenda notifies Mark accordingly. Instead, Sue’s personal device does not inform her although her M-agent is sensible to the “workshop is starting” field, because she is near to the workshop room and the field intensity is lower than the perception threshold.

Becoming aware that the PCC workshop begins, Sue and Mark move to the workshop room. Since their M-agents perceive the workshop field with the highest intensity, each of them sends this information to the Manager of the private fulcrum; consequently, it infers that the person is participating to the workshop and asserts the fact

5) X is member of the PCC workshop community

This fact is transferred from the private fulcrum to the workshop fulcrum, through rules (provided by Santana) that allow exchanging facts between fulcra, and triggers the Manager of the PCC workshop fulcrum to add an S-agent for the new member to the workshop fulcrum, as stated by the community rule

6) if X is member of the PCC workshop community then create an S-agent for X in the PCC workshop fulcrum

As members of the workshop community, Sue and Mark’s proxies are endowed with the community rules

7) if X is member of the PCC workshop community then quiet all X’s “ringing devices”

8) if the speaker’s CV is available and X is interested in it then retrieve it

so their “ringing devices” automatically switch to silent mode; in addition, their S-agent acquire inferential rules to perceive and retrieve the CV of the speaker. When Mr. Brown publishes his CV, i.e. his S-agent asserts the fact

9) the speaker’s CV is available

in the PCC workshop fulcrum, rule 8) fires on the device of the members of the fulcrum; hence, Mr. Brown’s CV is retrieved by and presented on Sue and Mark’s device if they have declared an interest in it, through the fact

10) X is interested in the speaker’s CV

This scenario illustrates some central aspects of our approach to support collaboration. First, the environment is proactive, i.e., it is able to sense the location of the actors, make them aware of events and activate services accordingly. Secondly, to this aim the environment manages the (interaction with existing) single-user and cooperative applications as well as a mechanism to compute different degrees of participation in them. To better describe modulated participation –a first-class concept of CSMAS- let us come back to the previous scenario, which illustrates a basic use of the field diffusion mechanism, namely the joining of members to the workshop community when they enter the room. A more sophisticated use of field diffusion would use modulation to realize a more articulated notion of participation. In fact, the different values of a diffused field can trigger the activation of different behaviors of S-agents in

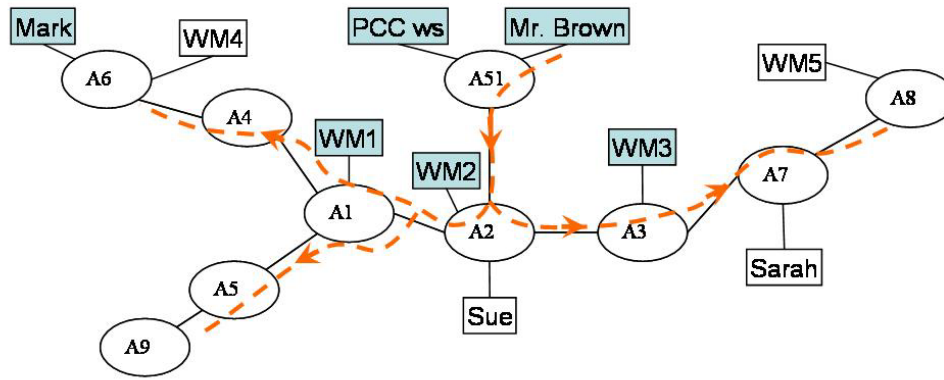


Fig. 7. Diffusion of the "PCC workshop is starting" field (hatched line) on the graph and perception by M-agents (colored agents perceive the field).

the receiving fulcra. This is realized through a mechanism that involves the M-agent linked to the graph where the field is diffused and the Manager. Manager owns rules to evaluate the degree of participation to the community based on the field perceived and exported by the M-agent; once they are executed, these rules assert into the private fulcrum facts that represent the degree of participation of the S-agent linked to the related community (e.g., rules sensible to the PCC workshop fields assert facts for the S-agent linked to the PCC workshop community). These facts are checked in the if-side of the community rules (already loaded in the corresponding S-agent); when the intensity of the field changes, new facts expressing the degree of participation are asserted; consequently different community rules can fire within the S-agent, and hence make it to participate to the community in a different way. In the scenario, the workshop field can be perceived with low intensity by agents located further away in the topological graph since the corresponding people are late and approaching the workshop room. In this case, they could be considered members of the community but with a more peripheral degree of participation: for example, they could listen on their Smart Phone to the voice of the speaker but with a limited access to the presented material or the speaker's CV; in this case, the Manager asserts the fact that represents a peripheral participation of the S-agent to the community, so that the rule that retrieves the CV is prevented from firing. The CV however will become available to them when they enter the room, because the perceived workshop field would get the highest intensity; consequently the fact that represents a full participation to the community is asserted, and then the rule to retrieve the CV can fire on it.

This example of modulated participation uses again the physical distance as a parameter. One could conceive situations in which the topology expresses logical distance between entities and modulates participation according to it. For example, suppose that the collaborative environment contains different fulcra that support a community in combination with different cooperative applications (e.g., workflow, co-authoring, shared repositories, distributed systems); moreover, S-agents that access those fulcra contain

rules that capture the interactions occurring in each of them between any two participating actors. This kind of information can be transmitted and organized in a topological space (usually called *social network* [17]) where the distance between the M-agents corresponding to any pair of actors expresses the degree of interaction among them. Moreover, M-agents own a sensitivity function expressing their availability to help an actor's request to solve an unexpected problem: for example, availability can be computed in terms of work overload or single actor's preferences, by means of the same rule-based mechanism described in the workshop scenario.

The environment, which could include several fulcra, a logical space associated to many of them as well as a physical one, can be modeled and managed by applying the integrated approach of CSMAS: the approach has the obvious advantage to manage uniformly the physical and logical features characterizing the environment and to support actors in their private and collaborative interaction by means of mutual perception and modulated participation to the applications that are available within the environment.

IV. CONCLUSION

This paper presented CSMAS, an agent based model for the design of collaborative community-aware applications. With community-aware we intend cooperative applications that - by means of the CSMAS constructs - are augmented with mechanisms managing different levels of participation of actors as members of communities. In order to let cooperative application become community-aware, the CSMAS model combines and integrates two models that were previously proposed within the multi-agent system research: Santana and MMAS. The former has been adopted for its ability to support the design of applications for Intelligent Environments, i.e., environments encompassing distributed and heterogeneous devices whose computational power can be combined together in order to build a context-aware environment that is able to react more aptly to the users' needs. CSMAS can be seen as an extension of Santana

aimed at supporting the design of cooperative applications that could be used also in those domains whose requirements are characterized within the Ambient Intelligence and Ubiquitous Computing research fields. The latter one is a multi-agent model that has been adopted for its ability to conceive of agents as entities situated on topological spaces representing both logical and physical aspects of a domain. The propagation of information among agents is modulated by means of the concept of field propagation, which is borrowed from Physics.

In the CASMAS model, primitives provided by Santana are used to let the various collaborative applications share those information and behaviors that concern and characterize the community of their users; MMASS is used to model how the level of participation of different community members can be modulated: this modulation occurs taking into account how the domain dependent information, which is relevant to affect the level of participation, is modulated through topological spaces representing either logical or physical aspects. As a result of the integration between Santana and MMASS, CASMAS provides designers with some additional mechanisms to let information be exchanged among Santana components and MMASS agents. Moreover, our proposal aims at making possible a seamless sharing of the rules regulating the community members' behaviors according to their current level of participation.

Currently, we are involved with the implementation of the Santana framework and of the MMASS model by means of the DJess platform [18], a middleware based on declarative programming by which distributed inference systems can share facts and rules through a blackboard interaction model. We are also investigating how to integrate the CASMAS model with other agent-based models: in particular with the ABACo Multi-Agent Framework [7] so that CASMAS fulcrum can become places where people are strongly supported in coordinating their activities.

REFERENCES

- [1] M. Wooldridge, "Agent-based Computing," *Interoperable Communication Networks*, vol. 1, pp. 71--97, 1998.
- [2] P. Ciancarini and M. J. Wooldridge, *Agent-Oriented Software Engineering*, vol. 1957 LNCS: Springer-Verlag, 2001.
- [3] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, pp. 139-159, 1991.
- [4] Y. Yiming and E. Churchill, "Agent Supported Cooperative Work," in *Multiagent systems, artificial societies, and simulated organizations*, G. Weiss, Ed.: Kluwer Academic Publishers, 2003.
- [5] S. Akinine and S. Pinson, "Managing Distributed Parallel Workflow Systems Using a Multi-agent Method," in *Agent Supported Cooperative Work*, Y. Yiming and E. Churchill, Eds.: Kluwer Academic Publishers, 2003.
- [6] C. Ellis, J. Wainer, and P. Barthelmess, "Agent-augmented Meetings," in *Agent Supported Cooperative Work*, Y. Yiming and E. Churchill, Eds.: Kluwer Academic Publishers, 2003.
- [7] M. Divitini, M. Sarini, and C. Simone, "Reactive Agents for a systemic approach to the construction of Coordination Mechanisms," in *Agents supported Cooperative work*, E. Churchill and Y. Yiming, Eds.: Kluwer Academic Press, 2003.
- [8] J. Lave and E. Wenger, *Situated Learning: Legitimate Peripheral Participation*: Cambridge University Press, 1991.
- [9] J. H. E. Andriessen, "Archetypes of Knowledge Communities," Second Communities & Technologies Conference (C&T2005), Milan, Italy, 2005.
- [10] F. Cabitza, B. Dal Seno, M. Sarini, and C. Simone, "Being at One with Things: The Interconnection Metaphor for Intelligent Environments," The IEE International Workshop on Intelligent Environments (IE05), University of Essex, Colchester, UK, 2005.
- [11] S. Bandini, S. Manzoni, and C. Simone, "Heterogeneous Agents Situated in Heterogeneous Spaces," *Applied Artificial Intelligence*, vol. 16, pp. 831-852, 2002.
- [12] C. Simone and S. Bandini, "Integrating Awareness in Cooperative Applications through the Reaction-Diffusion Metaphor," *Computer Supported Cooperative Work, The Journal of Collaborative Computing*, vol. 11, pp. 495-530, 2002.
- [13] M. Mamei, F. Zambonelli, and L. Leonardi, "Distributed Motion Coordination with Co-Fields: A Case Study in Urban Traffic Management," 6th IEEE Symposium on Autonomous Decentralized Systems (ISADS 2003), Pisa(I), 2003.
- [14] M. Mamei, F. Zambonelli, and L. Leonardi, "Tuples On The Air: A middleware for context-aware computing in dynamic networks," 23rd International Conference on Distributed Computing Systems (ICDCSW '03), 2003.
- [15] G. D. Abowd and A. K. Dey, "Towards a Better Understanding of Context and Context-Awareness," Workshop on The What, Who, Where, When, and How of Context-Awareness - Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands, 2000.
- [16] M. Weiser and J. S. Brown, "Designing calm technology," *PowerGrid Journal*, vol. 1, 1996.
- [17] B. Wellman and S. D. Berkowitz (eds.), "Social structures: A network approach." Cambridge: Cambridge University Press, 1988.
- [18] F. Cabitza, M. Sarini, and B. Dal Seno, "DJess - A Context-Sharing Middleware to Deploy Distributed Inference Systems in Pervasive Computation Domains," IEEE International Conference on Pervasive Services 2005 (ICPS'05), Santorini, Greece, 2005.

ANEMONE - A Network of Multi-Agent Platforms for Academic Communities

G. Armano, P. Baroni, G. Cerchi, M. Colombetti, A. Gerevini, M. Mari, A. Poggi,
C. Santoro, E. Tramontana, M. Verdicchio

Abstract—This paper presents ANEMONE, a multi-agent platforms network that provides services for the academic community implemented by using the JADE agent development framework. In particular, ANEMONE provides a set of services to support i) academic people in some of their recurrent activities (fix an appointment, organize a meeting and search documents on the Web, ii) students in getting information about courses and iii) information technology people (including students) in getting information on documents and people that may help them to solve their programming problems. Moreover, it also provides a set of system-oriented services for the management of agent platforms and services and for the realization of new types of service.

Index Terms—Multi-agent systems, cooperative systems, user-oriented services

I. INTRODUCTION

ONE of the main reasons to use autonomous software agents is their ability to interact to show useful social behaviors rapidly adapting to changing environmental conditions. But the most interesting applications require that large and open societies of agents are in place, where collaborating and competing peers are able to interact effectively. In a context where a number of possible partners or competitors can appear and disappear, agents can highlight their ability to adapt to evolving social conditions, building and maintaining their networks of trust relations within a global environment.

The first effort to create such a large and open society of autonomous software agents was Agentcities [1]. This project developed a network of agent platforms spanning over the whole globe and a number of complex agent-based

applications were deployed on the network. OpenNet is an evolution of Agentcities whose goal is to integrate agent platforms with Web Services / Semantic Web platforms [2]. In this paper, we present ANEMONE, an agent platform network developed inside the OpenNet initiative.

II. ANEMONE

ANEMONE is a multi-agent platforms network that provides services for the academic community (professors, researchers and students) implemented by using the JADE agent development framework [3].

ANEMONE offers both system-oriented and user oriented services. System oriented services allow the management of the network and the realization of new user-oriented services through their extension and composition. User-oriented services have the goal of helping academic users and can be used through a simple Web browser.

III. SYSTEM-ORIENTED SERVICES

System-oriented services have the goal to provide support to the management of agent platforms and services and provide reusable components to realize new types of service.

A. Platforms and Services Management

Platform and service management services are based on the services provided by the JADE agent development software and a set of services to register and search platforms (Agent/Service Platform Directory Services), agents (Agent Directories) and services (Service Directories) in an open network of agent platforms. Using these services, it is possible to connect a new platform to the openNet network, making it visible to others, and deploy own naming, directory and monitoring services.

Moreover, a set of monitoring services (Agent/Service Platform Monitoring Services) allow to monitor the platform's status and its ability to communicate with others.

B. Agent Interaction

As we introduced above, the platforms in the ANEMONE network are developed by using JADE. Agent interaction in JADE systems is based on message exchange. Thus, the adopted agent communication language has a crucial role. JADE agents' messages follow the standard proposed by the Foundation for Intelligent and Physical Agents (FIPA [10]), which is the most complete proposal to date.

Manuscript received November 3, 2005. This work is partially supported by the "Ministero dell'Istruzione, dell'Università e della Ricerca" through the COFIN project ANEMONE.

M. Mari and A. Poggi are with the Dipartimento di Ingegneria dell'Informazione, University of Parma, Italy (e-mail: mari@ce.unipr.it, poggi@ce.unipr.it).

P. Baroni is with the Dipartimento di Elettronica per l'Automazione, University of Brescia, Italy (e-mail: baroni@ing.unibs.it).

G. Armano and G. Cerchi are with the Dipartimento di Ingegneria Elettrica ed Elettronica, University of Cagliari, Italy (e-mail: armano@diee.unica.it).

C. Santoro is with the Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, University of Catania, Italy (e-mail: csanto@diit.unict.it).

E. Tramontana is with the Dipartimento di Matematica e Informatica, University of Catania, Italy (e-mail: tramontana@dmf.unict.it).

M. Colombetti and M. Verdicchio are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy (e-mail: colombet@elet.polimi.it, verdicch@elet.polimi.it).

Still, the FIPA semantics shows some shortcomings that inevitably affect the systems whose communication is based on such standard. Our aim is to provide a different semantics to tackle these problems. The FIPA proposal and ours share the assumption that agent communication should be dealt in terms of communicative acts, a special action type aiming at allowing information interchange. We part from FIPA guidelines when it comes to defining the semantics of such acts. The FIPA semantics exploits the Belief, Desire, Intention (BDI, [5]) model, which views communicative acts as events that change agents' mental states. Instead of analyzing changes in the state of the internal architecture of agents, our approach focuses on the external social state holding among agents. We describe communicative acts as actions performed by agent to change their commitments towards the others. We rewrote the FIPA Communicative Act Library according to our perspective to have a benchmark for the two approaches. The advantages of dealing with social states rather than mental ones have surfaced in the analysis of the Contract Net protocol, in which an agent, in a need for a specific service, issues a call for proposal to other agents. To check whether the contract has been fulfilled, the current FIPA protocol prescribes an inform message from the service provider itself. This procedure is effective only under very strict assumptions about the sincerity of the agents, which we cannot afford when we deal with open multi-agent systems. In a commitment-based approach, on the contrary, each message exchange of the Contract Net protocol leads to changes in the social dimension of the multi-agent system, in that, commitments are proposed, and such proposals are accepted or rejected. Commitments are public and reflect an objective state of affairs between agents. They can be stored for further reference and thus they offer an effective way to check whether agents have fulfilled their commitments. The FIPA communicative act library in terms of commitments and the results of the analysis of the Contract Net protocol are formally presented in [6].

We provide the ANEMONE network and, more generally, every JADE-based multi-agent system such commitment-based communication system in the form of an agent that is called Notary. The Notary is responsible for examining the content of the messages that are exchanged over the system and creating public structured data items reporting the relevant commitments between agents. The Notary makes use of witness agents the communicating agents have agreed upon to check whether the commitments have been fulfilled or violated. The witnesses reply to queries by the Notary, and thus increase its knowledge base. The Notary exploits a JESS (Java Expert System Shell, [7]) inference engine to reason about its own knowledge base and verify whether a commitment has been fulfilled or not. To support the commitment generation and manipulation processes, the exchanged messages need to be carrying more information than as prescribed by the usual FIPA standard. To maximize backward compatibility, we have chosen not to add fields to the original FIPA message structure, but to enrich and

standardize its content field by means of XML. The Notary is provided with XML parsing capabilities thanks to a SAX (Simple API for XML, [8]) module.

The Notary-enhanced communication system introduces significant overhead in the message interchange process, which may not suit the needs for lightweight application in such environments like PDAs or mobile phones. This service is offered as an option when agents need a trusted third-party to guarantee for their communication process, e.g. in electronic auctions or business transactions. Agents only need to put the Notary among the messages' addressees to obtain its service.

C. Automated Reasoning

Domain-independent automated reasoning services, based on stand-alone software tools previously developed in the context of other research activities, are made available to the community by a wrapper agent, which is in charge of receiving requests from other agents specifying reasoning tasks to be carried out, of exploiting the suitable software system to produce the relevant solutions, and of returning them to the requestor agents.

The wrapper agent and the related agents devoted to registration and brokering of the available reasoning services are implemented according to the FIPA specification "Agent software integration" [10].

Two reasoning services are currently being integrated into the ANEMONE network, namely an argumentation system and a planning system.

Argumentation theory is a framework for practical and uncertain reasoning, where arguments supporting conclusions are progressively constructed in order to identify the set of conclusions that should be considered justified according to the current state of available knowledge. The use of argumentation has been advocated both at the level of interaction among agents to support dialogue and negotiation and at the level of an agent's internal reasoning (see [14] for a survey).

Since the construction of arguments proceeds by exploiting incomplete and uncertain information, conflicts between them may arise: the conflict relations between arguments are formally represented by a structure called defeat graph. The core problem is then to compute the "defeat status" of the arguments, namely to determine which arguments emerge undefeated from the conflict: several semantics have been proposed to this purpose in the literature.

A reasoning task in this case consists in the specification of a defeat graph and the solution provided is the defeat status assignment for the arguments included in the graph. The solution may be produced according to the well-known grounded [13] semantics or to the recently introduced CF2 [9] semantics.

As to the planning system, this reasoning service receives requests to solve plan generation problems specified using the recent standard PDDL2.2 language [12], and computes plans solving such problems (assuming they are solvable and not too hard for the integrated planner). PDDL2.2 is an expressive

planning language supporting the representation of domains involving numerical quantities, actions with durations, predictable exogenous events and domain axioms. The integrated planning system is LPG [11], an efficient, state-of-the-art, fully-automated planner which received two awards at the last International planning competition.

IV. USER-ORIENTED SERVICES

ANEMONE provides a set of services to support i) academic people in some of their recurrent activities (fix an appointment, organize a meeting and search documents on the Web, ii) students in getting information about courses and iii) information technology people (including students) in getting information on documents and people that may help them to solve their programming problems.

A. Agenda Management

An agenda management system called MAgentA (Multi-Agent Agenda) has been developed. The system, besides managing users' personal agendas, provides a specific support to meeting organization: through a process of automated negotiation agents are able to determine the temporal location of a meeting which best fits the preferences of their owners, while satisfying some constraints specified by the meeting proposer.

The MAgentA system, implemented using the JADE agent development environment, consists in the following agents:

- a user management (UM) agent, in charge of managing the authentication of authorized users;
- a meeting management (MM) agent, in charge of coordinating the negotiation of a meeting among users' agents and of managing a database of meetings;
- a set of personal agenda (PA) agents, which represent individual users and maintain information about their scheduled activities and their preferences over their possible temporal allocation. PA agents are expected to be continuously running and available to receive meeting organization requests from other agents;
- a set of GUI agents, in charge of managing the interaction with the MAgentA users through a graphical interface. A GUI agent is activated only when necessary, i.e. during a user working session.

In a typical use scenario, a user, after authentication, interacts with a GUI agent to express her/his preferences about temporal locations of requested meetings and possibly to insert some personal scheduled activities within her/his agenda. The GUI agent communicates this information to user's PA agent which will use them when negotiating the organization of a meeting.

Moreover, using the GUI, a user may initiate the organization of a meeting by specifying:

- some temporal constraints about the temporal location of the meeting;
- the minimum and maximum duration of the meeting;
- a list of expected participants, partitioned into necessary participants and optional participants.

Once a request of a meeting organization has been formulated by the initiator user, it is submitted to the MM agent which tries to identify a solution, namely a suitable temporal location of the meeting, through a negotiation process consisting in the following steps.

First of all, using the FIPA contract-net protocol, the PA agent of every participant is solicited to propose a set of possible solutions compatible with the meeting temporal constraints, and to specify the user's preferences about the proposed solutions.

Then the MM agent verifies whether there exists a temporal location where all participants are available and, if one or more of them exists, it proposes them to the initiator user, ordering them on the basis of participants' preferences. Otherwise, the solutions where at least the necessary participants are available are searched for. Again, if these "weaker" solutions are found, they are proposed in an order consistent with the preferences specified in the agendas of the involved users; otherwise, a final search for (possibly less-satisficing) solutions is carried out, where personal activities included in participants' agendas are ignored. The list of the solutions found or a message of failure is then provided to the initiator user.

If a list of solutions has been found, the initiator user selects and confirms one of them: a notification is then sent to the MM agent and to the involved PA agents, which add the meeting to their databases.

In case of failure, it is up to the user to define a new request with different constraints and to initiate a new negotiation process.

B. Supporting Students in their University Activities

DIEE has developed an e-service devised to support graduated and undergraduated students in their activities. It is built upon a generic multi-agent architecture, designed to support the implementation of applications aimed at: (i) retrieving heterogeneous data spread among different Internet sources (i.e., generic web pages, news, and forums), (ii) filtering and organizing information according to personal interests explicitly stated by each user, and (iii) providing adaptation techniques to improve and refine throughout time the profile of each selected user. The generic architecture has been called PACMAS, standing for Personalize, Adaptive, and Cooperative MultiAgent System, and encompasses four main levels (i.e., information, filter, task, and interface), each being associated to a specific role that agents can play. The communication between adjacent levels is achieved through suitable middle agents, which form a corresponding mid-span level. Each level is populated by a society of agents, which are autonomous and flexible, and can be personalized, adaptive and cooperative depending on the role they assume in the implemented application. PACMAS agents belong to one of the following categories:

- information agents, which access information sources, and are able to collect and manipulate such information [19];

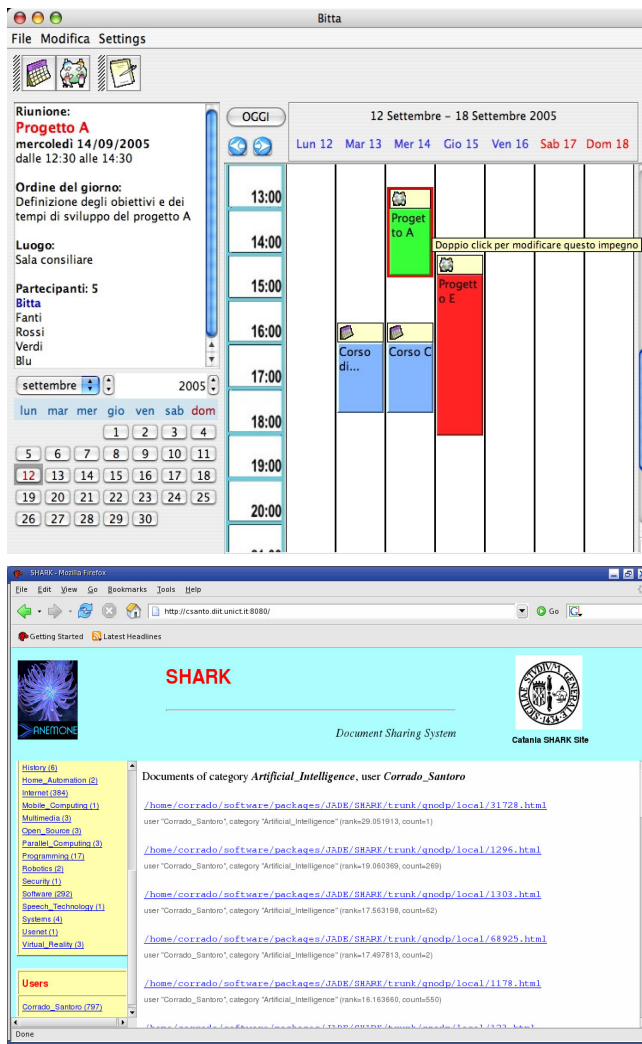


Figure 1. The GUIs of the four user-oriented

- filter agents, able to process information according to user preferences [16] ;
- task agents, which help users to perform tasks by solving problems and exchanging information with other agents [17];
- interface agents, devised to facilitate the interaction between the user and other agents [18];
- middle agents, which are in charge of establishing communication among requesters and providers.

Let us consider a typical University Department. It generally makes available the information about courses, seminars, exams, professors, and students on different areas: web sites, forums, and news (NNTP) servers. All relevant information is not directly available but it is usually spread on the department portal, on the web site of each course, and on the personal page of each professor. Furthermore, each professor might activate her/his news and forum service. Some of the information potentially interest all students, such as lesson timetables, exam dates, taxes, and student tutoring. On the other hand, students belonging to different courses are interested in different lessons and exams. Typically, a student in search of relevant information about her/his University

activities browses web sites, and reads announcements from forum and news services. This is often a repetitive and boring task that can be automated. From our perspective, personalization and adaptation represent the added value of such an automated system.

To provide an e-service able to support students in their activities, a prototype based on the PACMAS architecture has been implemented, using JADE [3] as the underlying framework. Supporting students involves several activities: information extraction, information retrieval and filtering, information processing, and result presentation. Each activity corresponds to a suitable level of the PACMAS architecture. Information extraction is carried out at the information level by information agents that play the role of wrappers, specialized for dealing with a specific information source. Information retrieval and filtering is carried out at the filter level, populated by two kind of agents: generic and personal. Generic filters are specifically aimed at removing all non relevant information retrieved from the involved information sources, whereas personal filters are devoted to select the information according to the personal needs, interests, and

preferences of the corresponding user. Information processing is carried out at the task level, where agents are customized for a specific task (e.g. lesson timetable, seminars, and exams scheduling). Result presentation is carried out at the interface level, through agents that interact with the users. A suitable graphical interface - personalized for each user - that can run on a web browser, is available to allow communication among interface agents and the user.

The prototype has been tested on the information system of the Department of Electrical and Electronic Engineering (DIEE) at the University of Cagliari.. The system is able to learn specific user's interests to retrieve, filter, and show only the information deemed relevant by her/him. A beta version of the web service is available at: <http://iascw.diee.unica.it/PacmasWWW>.

C. Documents Search

SHARK is a multi-agent P2P document sharing system aiming to provide users with a more effective tool to find documents and promote collaborations among them [20]. Each SHARK agent (such as Categoriser, Searcher, UserProfiler, etc.) autonomously performs a small task, such as document categorisation, finding, user profiling, etc. and communicates its results to other agents.

The hosts in a SHARK network are given different roles. A client host provides users with a few services, such as user profiling and document analysis, and allow users to log in to an AgentCities [21] host. AgentCities hosts are servers, connected to each other, each running a FIPA-compliant agent platform and handling data related to SHARK users and their documents.

1) SHARK Agents

In the following we describe the agents that constitute SHARK. Agent Cruncher analyses shared documents and extracts from each a set of keywords. For this, Cruncher uses filters to recognise and remove HTML, LaTeX, RTF and PDF tags that are used only to format the text; then it removes the stop words, and, for all the remaining words, it extracts the appropriate stems. The output is a list of word stems ranked by the number of occurrences found [22][23].

Agent Categoriser, on the basis of extracted word stems, associates categories to documents. Categoriser holds a knowledge base, containing, for each known category: its name, the list of keyword stems and the respective frequency. Taking as input a list of word stems, Categoriser calculates the "distance" between such a list and the known categories, by using the dot product. The category that minimises the distance is chosen as the category to which the document belongs.

Agent UserProfiler detects the activities that a user operating with a web browser performs, and analyses the shared documents in order to continually update his/her profile. The user profile consists of the list of categories corresponding to shared documents or visited web pages. Each category is associated with a score, which reflects the degree of interest, measured on the basis of the number of shared documents and visited web pages.

Agent Searcher runs on an AgentCities host and holds the list of categories identified for the local shared documents, for each category the list of documents and the user providing each document. Given a user-provided query (as a list of keywords), Searcher looks for matching categories and returns the list of corresponding documents with the user providing each. The query is then propagated to the other AgentCities hosts, where local Searchers will perform analogous activities.

Agent Correspondent handles document download requests originating from other users.

Agent Advertiser periodically checks user profiles in order to find a partial match. Whenever the matching degree is above a given threshold, the users with common interests are notified with an email message. It is then up to the users to find the opportunity for a collaboration.

The instances of the agent classes described above run on different hosts. The user host is equipped with Cruncher, UserProfiler and Correspondent; AgentCities servers host Categoriser, Searcher and Advertiser.

2) Using SHARK

Users interact with SHARK by means of a web interface, of which we highlight here two important features. The first one is the searching facility: once a user has performed a query, by typing a set of keywords into a web form, this is sent to the Searcher on the AgentCities server the user is connected with. The results of Searcher are sorted so that the more relevant document is that exhibiting the highest frequency (in percentage with respect to all the document's keywords) of the keyword queried-if only one keyword is provided. If more than one keyword is given, the total relevance is computed as the average of each single keyword relevance.

The second feature is the collaboration facility. This is connected with searches and consists of providing a list that reports the name of the users who have, in their user profile, the keyword(s) queried. Names are ranked according to the relevance of the user profile with respect to the keyword(s) queried. Relevance is computed using the same method employed for documents.

D. Software Development

RAP (Remote Assistant for Programmers), is a Web and multi-agent based system to support remote students and programmers during common projects or activities based on the use of the Java programming language [24].

1) RAP Agents

In this section we describe the agents that compose the RAP system. Personal Agents allow the interaction between the user and the different parts of the system and, in particular, between the users themselves. Moreover, these agents are responsible of building the user profile and maintaining it when the user is "on-line". User-agent interaction can be performed in two different ways: through a Web based interface or through emails (if the user is not on-line). User Profile Managers are responsible of maintaining and updating the profile of system users. Answer Managers maintain the answers provided by users during the life of the system and they find the

appropriate answers to the new queries of the users. Besides providing an answer, these agents update the score of the answer and forward the vote to the User Profile Manager for updating the user profile. Document Managers find the appropriate documents to answer the queries submitted by system users. E-mail Managers are responsible of the communication between the system and the off-line users. Starter Agents are responsible for activating a Personal Agent when either a user logs on or another agent requests it. Directory Facilitators are responsible to inform an agent about the address of the other agents active in the system (yellow pages service).

2) *Profile Management and Open Communities*

The management of user and document profiles is performed in two different phases: an initialization phase and an updating phase. In order to simplify and reduce the possibility of inaccuracy due to people's opinions of themselves and to incomplete information, we decided to build the initial profile of the users and documents in an automated way. Profiles are represented by vectors of weighted terms whose values are related to the frequency of the term itself in the user's documents. Document and user profiles are computed by using "term frequency inverse document frequency" (TF-IDF) [24] algorithm. Each user profile is built by user's Personal Agent through the analysis of the software she/he wrote. This is only the initial user's profile, it will be updated when the user writes new code or interacts with the system answering some queries.

An important requirement that has guided the design of RAP has been the support for open and distributed communities. RAP structure is open, since new users can register and access the system, and a registered user can acquire new skills or produce new software. The community beneath RAP is distributed: the whole system can consist of a dynamic group of local communities. Each community can operate isolated, but can also decide to join a group of communities, sharing experts and documents repositories.

The open and distributed nature of the system entails some significant problems in the evaluation of information: the evaluation of both experts and documents is strongly dependent on the actual composition of the community group. For example, if a user is rated as the maximum expert to answer a query, he is rated considering only the users registered in the system at that moment. As a matter of fact, TF-IDF algorithm can be easily used in a centralized system where all the profiles and the data are managed, while our context is more complex. For these reasons, each profile component of RAP is associated with two elements: an absolute element and a TF-IDF weighted element. The absolute one depends only on the user (or document) profile, instead the TF-IDF element is related to both the user profile and the whole community profiles. Moreover, while the absolute element is stored in a database, the weighted one is maintained in memory and it is recalculated when necessary.

V. CONCLUSION

In this paper, we presented ANEMONE, a multi-agent platforms network that provides services for the academic community (professors, researchers and students).

The ANEMONE network and services are the result of a project involving five Italian universities (University of Parma, University of Brescia, University of Cagliari, University of Catania and "Politecnico di Milano" Technical University) and the realized network is composed of five nodes deployed in the different universities. However, the ANEMONE network can interoperate with agent platforms deployed in different parts of the world. In fact, ANEMONE project takes part of the OpenNet initiative [2] that is a project dedicated to facilitating collaboration between research projects developing, applying and above all deploying Agent, Semantic Web, Web Services, Grid and similar networked application technologies in large-scale open environments such as the public Internet. In particular, the core partners of this initiative deployed a backbone network of agent platforms, including a platform at the University of Parma. This backbone network has the goal to be the interconnection network among the systems and prototypes belonging to the initiative (currently different projects are running in different part of the world and different tens of agent platform are active).

REFERENCES

- [1] "Agentcities: A Worldwide Open Agent Network" Steven Willmott, Jonathan Dale, Bernard Burg, Patricia Charlton and Paul O'brien. Short article in Agentlink News Issue 8, November 2001.
- [2] OpenNet initiative Home Page. Available from <http://x-opennet.org/>.
- [3] F. Bellifemine, A. Poggi, G. Rimassa, Developing multi agent systems with a FIPA-compliant agent framework. *Software Practice & Experience*, 31:103-128, 2001
- [4] Foundation for Intelligent Physical Agents, "Agent Communication Language Specifications", <http://www.fipa.org/repository/aclspecs.html>, 2002
- [5] M. Wooldridge, "Reasoning about rational agents", MIT Press, 2000
- [6] M. Verdicchio, M. Colombetti, "A Commitment-based Communicative Act Library", *Proceedings of the Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 05)*, vol. 2, p755-761, Utrecht, 2005
- [7] Ernest Friedman Hill, "JESS in Action", Manning Publications, 2003
- [8] D. Megginson, "SAX", <http://www.saxproject.org>, 2004
- [9] P. Baroni, M. Giacomin, G. Guida, SCC-recursiveness: a general schema for argumentation semantics, accettato per la pubblicazione su *Artificial Intelligence*, 2005
- [10] FIPA Specification 00079, *Agent Software Integration*, 2001, <http://www.fipa.org/specs/fipa00079/>
- [11] A. Gerevini, A. Saetti, I. Serina, Planning through Stochastic Local Search and Temporal Action Graphs in LPG, *Journal of Artificial Intelligence Research (JAIR)*, 20, 2005, 239-290
- [12] J. Hoffmann, S. Edelkamp, The Deterministic Part of IPC-4: An Overview, to appear in *Journal of Artificial Intelligence Research (JAIR)*, 2005
- [13] J. Pollock, How to Reason Defeasibly, *Artificial Intelligence*, 57(1), 1992, 1-42
- [14] H. Prakken and G. A. W. Vreeswijk, Logics for Defeasible Argumentation, in Dov M. Gabbay and F. Guenther Eds., *Handbook of Philosophical Logic*, Kluwer, 2001
- [15] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 578-583, 1997.

- [16] A. Falk and I. Jossion. Paws: An agent for www-retrieval and filtering. In Proceedings of Practical Application of Intelligent Agents and Multi-agents Technology (PAAM-96), pages 169.179, 1996.
- [17] J. Giampapa, K. Sycara, A. Fath, A. Steinfeld, and D. Siewiorek. A multi-agent system for automatically resolving network interoperability problems. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1462.1463, 2004.
- [18] H. Lieberman. Autonomous interface agents. In Proceedings of the ACM Conference on Computers and Human Interface (CHI-97), pages 67.74, 1997.
- [19] P. Maes. Agents that reduce work and information overload. Communications of the ACM, 37(7):31.40, 1994
- [20] A. Di Stefano, G. Pappalardo, C. Santoro, E. Tramontana. "SHARK, a Multi-Agent System to Support Document Sharing and Promote Collaboration". In Proceedings IEEE Hot P2P Workshop. Volendam, Holland. October, 2004.
- [21] WWW. www.agentcities.net, 2005.
- [22] H. Lieberman. Letizia: An Agent That Assists Web Browsing. In International Joint Conference on Artificial Intelligence, Montreal, August 1995.
- [23] H. Lieberman, P. Maes, and N. Van Dyke. Butterfly: A Conversation-Finding Agent for Internet Relay Chat. In International Conference on Intelligent User Interfaces, Los Angeles, January 1999.
- [24] L. Lazzari, M. Mari, A. Negri, A. Poggi: Support Remote Software Development in an Open Distributed Community. In AAMAS05 workshop Agent-Based System for Human Learning (ABSHL), Utrecht, 2005.
- [25] Salton, G.: Automatic Text Processing. (1989), Addison-Wesley.

MAgentA: Un Sistema Multi Agente per la Gestione di Agende e Riunioni

Pietro Baroni Alfonso Gerevini Paolo Toninelli
 Dipartimento di Elettronica per l'Automazione
 Università degli Studi di Brescia
 via Branze 38, I-25123 Brescia, Italy
 {baroni, gerevini}@ing.unibs.it

Abstract—In questo articolo viene presentato MAgentA, un sistema multi-agente sviluppato nel contesto del progetto interuniversitario ANEMONE allo scopo di gestire agende e pianificare riunioni tra gli utenti in modo semiautomatico. Ogni agenda è gestita da un agente che conosce gli impegni personali e le preferenze di un particolare utente, mentre le riunioni vengono organizzate da un apposito agente in grado di contattare le agende e ragionare sulle preferenze degli utenti e sulla loro disponibilità nel tempo. MAgentA, che è stato sviluppato utilizzando la piattaforma JADE e lo standard FIPA, si caratterizza rispetto ad altri sistemi analoghi poichè è in grado di discretizzare il tempo in modo guidato dalle preferenze oltre che dalla disponibilità degli utenti contattati ed utilizza criteri che cercano di ottimizzare anche la durata della riunione, entro i limiti stabiliti dall'utente che ne ha fatto richiesta.

I. INTRODUZIONE

Le tecnologie ad agenti forniscono strumenti che permettono di modellare ed implementare sistemi software sempre più articolati, in cui sono possibili astrazioni espressive come agenti, società di agenti ed ambiente [1].

In questo contesto si colloca il progetto interuniversitario ANEMONE (A NEtwork of Multi-agent OpeN Environments). Il progetto si propone come obiettivo da una parte la realizzazione di una rete di piattaforme multi-agente in grado di offrire servizi dedicati alla comunità accademica ed interoperabile con altre piattaforme attive in diverse parti del mondo, e dall'altro la ricerca di soluzioni ad alcuni problemi comuni nella realizzazione di piattaforme multi-agente. Una delle funzionalità che il progetto si propone di realizzare è la gestione semiautomatica degli impegni dei docenti che fornisca un supporto all'organizzazione di riunioni. Il problema è di particolare rilevanza pratica ed è appropriato per l'applicazione della tecnologia ad agenti, come confermato dall'esistenza di svariati lavori sul tema [2], [3], [4], [5], [6].

Nell'ambito del progetto ANEMONE, tale servizio viene fornito da un sistema ad agenti denominato MAgentA (Multi Agent Agenda) e sviluppato tramite l'utilizzo di JADE (Java Agent DEvelopment Framework) [7], un framework che semplifica l'implementazione dei sistemi multi-agente attraverso un middle-ware conforme alle specifiche FIPA [8]. Alcune caratteristiche di MAgentA lo differenziano da lavori analoghi preesistenti. In particolare il metodo utilizzato per la definizione dei possibili intervalli temporali in cui pianificare una riunione. Tali intervalli vengono infatti individuati senza

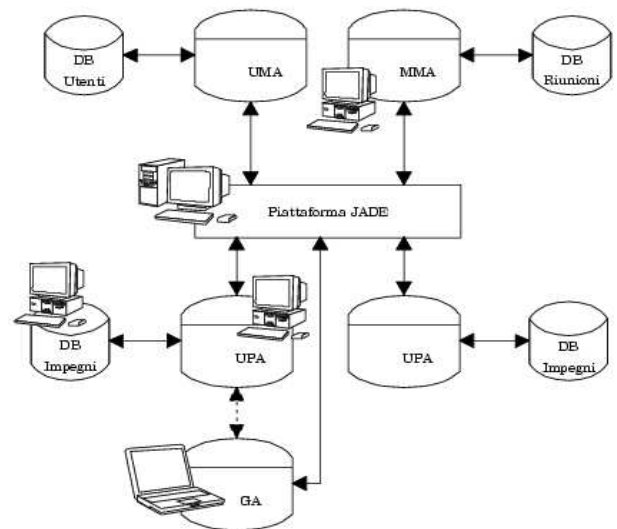


Fig. 1. Architettura del sistema. Agenti, database e connessioni.

la necessità di suddividere il tempo in intervalli discreti di dimensione fissa, tramite una algoritmo guidato dalle preferenze degli utenti ed in grado di ignorare intervalli poco rilevanti.

L'articolo descrive le funzionalità e la struttura del sistema MAgentA ed è organizzato nel modo seguente: nelle sezioni II e III verrà presentata una panoramica sull'architettura del sistema, gli agenti che lo compongono, il modo in cui essi comunicano e le funzionalità offerte; nella sezione IV si vedranno quali sono i compiti dell'agente che implementa l'agenda di un utente; nella sezione V verranno esposte le caratteristiche dell'agente incaricato della gestione degli utenti del sistema; nella sezione VI sarà esaminato l'agente che permette all'utente di interagire con la propria agenda tramite l'interfaccia grafica; nella sezione VII verranno esposte le caratteristiche dell'agente incaricato di pianificare nuove riunioni su richiesta degli utenti, tenendo in considerazione le loro preferenze. La sezione VIII conclude l'articolo sintetizzando i risultati ottenuti e menzionando i possibili sviluppi futuri di questo lavoro.

II. ARCHITETTURA E COMPONENTI DI MAGENTA

Il sistema è costituito da un insieme di agenti dotati di diverse capacità ed in grado di cooperare. Ognuno di essi

ricopre uno specifico ruolo ed è in grado di accedere ad un database in cui vengono mantenuti i dati di cui l'agente ha bisogno per assolvere ai propri compiti. I dati sono organizzati in diversi database:¹

- Un database degli utenti registrati nel sistema.
- Un database delle riunioni fissate tra gli utenti registrati.
- Un database riservato ad ogni singolo utente, in cui sono memorizzati i suoi impegni personali.

Gli agenti che costituiscono il sistema sono (figura 1):

- *User Personal Agent (UPA)*: un agente in grado di accedere al database personale dell'utente che ne è proprietario ed assumere il ruolo di partecipante durante la pianificazione di una riunione. Questo agente è in grado di recuperare tutti gli impegni dell'utente, valutare in quali intervalli possa essere considerato libero e gestire eventuali preferenze espresse dall'utente in modo da fornire all'iniziatore di una riunione le informazioni necessarie a costruire le funzioni obiettivo utilizzate per collocarla opportunamente.
- *User Management Agent (UMA)*: un agente per la gestione degli utenti che ha il compito di avviare le agende e di distribuire loro le informazioni necessarie ad effettuare l'accesso al proprio database personale. L'agente interviene inoltre durante il login degli utenti, fornendo ad ognuno l'indirizzo della propria agenda. L'UMA è il solo agente autorizzato ad accedere al database degli utenti.
- *GUI Agent (GA)*: un agente per la gestione della GUI, il cui compito fondamentale consiste nel fare da intermediario tra l'agenda e l'interfaccia utente, traducendo gli eventi della GUI in messaggi per l'UPA e viceversa.
- *Meeting Management Agent (MMA)*: un agente gestore delle riunioni a cui è consentito l'accesso al database delle riunioni ed il cui compito principale consiste nell'assumere il ruolo di iniziatore durante la pianificazione di una riunione. Questo agente è in grado di ragionare sui vincoli temporali delle riunioni e sugli impegni presenti nelle agende dei partecipanti per pianificare la nuova riunione in modo ottimale.

A sistema avviato esistono, oltre all'UMA ed al MMA, un UPA per ogni utente ed un GA per ogni utente che abbia effettuato il login sul proprio UPA.

Ogni agente "vive" all'interno di un *container* a sua volta contenuto in una piattaforma JADE (Java Agent Development Framework) [10] e provvede a registrarsi presso il DF (Directory Facilitator conforme alle specifiche FIPA) [8] messo a disposizione della piattaforma.² Tale agente permette di ottenere informazioni circa gli agenti registrati e viene utilizzato per recuperare l'indirizzo di UMA e MMA ogni volta che un'agenda abbia bisogno di contattare uno di questi agenti.

¹I database possono essere distribuiti su più server. La connessione è stata realizzata tramite JDBC, mentre per la permanenza dei dati si è utilizzato JPOX[9], un'implementazione dello standard JDO. In questo modo sia il formato dei database che il tipo di server possono essere configurati.

²Ogni container può essere lanciato su una diversa macchina in modo da distribuire gli agenti che compongono il sistema. Il corretto indirizzamento degli agenti viene gestito tramite il DF.

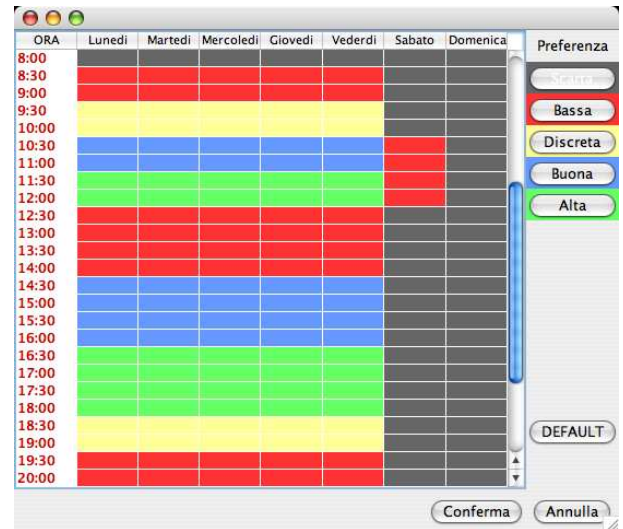


Fig. 2. Configurazione delle preferenze di un utente.

Ogni volta che viene iniziata una conversazione tra gli agenti, l'iniziatore genera un identificativo (*Conversation-ID*) univoco da assegnare a tutti i messaggi che verranno scambiati durante la conversazione. In questo modo ogni agente può essere impegnato in più conversazioni senza che queste interferiscano tra loro.

III. FUNZIONALITÀ

Nella versione attuale il sistema offre alcune funzionalità che permettono ad un utente di configurare la propria agenda esprimendo delle preferenze circa eventuali riunioni a cui dovrà prendere parte, gestire i propri impegni personali ed interagire con altre agende allo scopo di pianificare una riunione. Più in dettaglio, le funzionalità attualmente disponibili sono:

- Configurazione delle preferenze: ogni utente ha la possibilità di esprimere un *grado di disponibilità* (preferenza) a partecipare ad eventuali riunioni in un determinato intervallo di tempo all'interno di una settimana tipo. Tali preferenze possono essere espresse con una granularità di mezz'ora tramite l'utilizzo di cinque livelli di disponibilità: basso, discreto, buono, medio e scartato (figura 2). L'ultimo livello viene considerato come un impegno fittizio che impedisce all'utente di partecipare a qualsiasi riunione che vi si sovrapponga. Gli altri livelli corrispondono invece a dei valori numerici che quantificano le preferenze dell'utente.
- Inserimento e modifica di impegni personali: ogni utente può agire sul proprio database personale tramite un'interfaccia che permette di aggiungere, rimuovere e modificare gli impegni traducendo ogni azione in opportuni messaggi da inviare all'UPA ed aggiornando quanto presentato a video all'utente.
- Richieste di riunione: ogni utente può assumere il ruolo di iniziatore definendo dei vincoli per una nuova riunione

ed inviando una richiesta al *MMA*, il quale interagirà con le agende degli utenti invitati a prendervi parte.

- Ricerca e ordinamento soluzioni: l'agente gestore delle riunioni è in grado di valutare le possibili collocazioni temporali individuate per una nuova riunione allo scopo di determinarne la qualità in relazione alle preferenze espresse dalle agende degli utenti invitati. Ciò permette di ordinare le collocazioni individuate in modo da identificare quelle che meglio soddisfano la richiesta.
- Selezione di una soluzione, memorizzazione e notifica agli utenti: gli intervalli di tempo selezionati e valutati possono essere analizzati dall'iniziatore allo scopo di verificarne la qualità. L'utente potrà quindi selezionarne uno da assegnare alla riunione che si vuole fissare ed inviare una notifica alle agende degli utenti invitati.

IV. UPA: AGENTE PERSONALE DI UN UTENTE

L'*UPA* rappresenta l'agenda vera e propria di un utente. Questo agente è in grado di accedere al database personale del suo proprietario per inserirvi nuovi impegni ed estrarre o modificare quelli esistenti. I compiti principali di questo agente sono:

- rispondere alle richieste provenienti dal *GA*, in modo da permettere all'utente di visualizzare ed interagire con la propria agenda;
- interagire con il *MMA* assumendo il ruolo di partecipante durante le conversazioni effettuate tramite il protocollo FIPA-Contract-Net, allo scopo di fornire all'iniziatore gli intervalli di tempo disponibili per pianificare una nuova riunione in accordo con gli impegni del proprio utente;
- valutare tali intervalli tenendo in considerazione gli impegni personali, le riunioni e le loro eventuali ricorrenze;
- valutare le eventuali preferenze dell'utente traducendole in un opportuno insieme di valori numerici.

Nel momento in cui l'*UPA* viene contattato dal *MMA* allo scopo di fissare una nuova riunione, esso si attiva valutando le possibilità e le preferenze del suo proprietario. Le possibilità dell'utente vengono definite analizzando la lista di tutti i suoi impegni ed estraendo tutti gli intervalli in cui l'utente risulta libero. Per quanto riguarda le preferenze l'agente analizza la configurazione effettuata dall'utente su di una settimana tipo e la applica alle possibilità dell'utente. In figura 2 è riportata la finestra che permette all'utente di configurare le proprie preferenze.

V. UMA: AGENTE GESTORE DEGLI UTENTI

Il primo compito dello *UMA* consiste nell'avviare le agende, prelevando le informazioni necessarie dal database degli utenti registrati. Per ogni utente sono definite le seguenti informazioni:

- il nome dell'utente, che deve essere unico in quanto verrà utilizzato come identificativo;
- il nome dell'agenda di cui l'utente è proprietario, ovvero il nome da assegnare allo *UPA* di quel particolare utente;
- il nome del database di cui l'utente è proprietario;

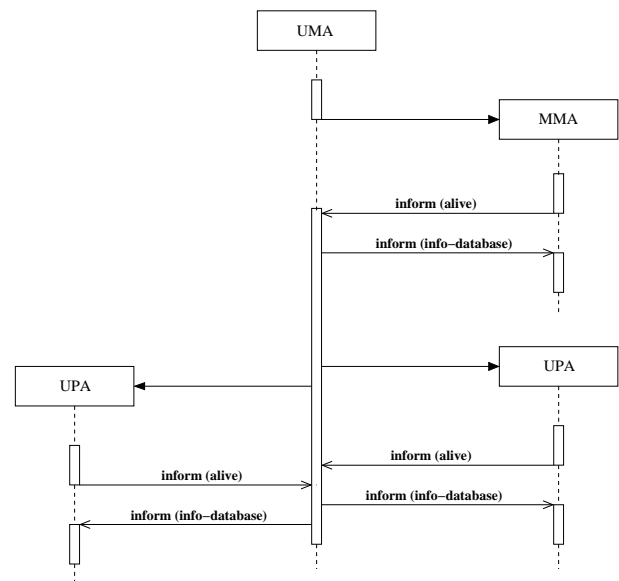


Fig. 3. Scambio di messaggi all'avvio delle agende.

- il nome utente e la password che l'agenda in questione dovrà utilizzare per poter accedere al proprio database personale;
- la categoria a cui l'utente appartiene

Il meccanismo di avvio delle agende coinvolge tutti gli agenti (tranne il *GA*, che esiste solo nel momento in cui un utente effettua un login sulla propria agenda) ed è costituito dai seguenti passi (figura 3):

- 1) lo *UMA* recupera dal proprio database la lista degli utenti, avvia un *MMA* e si mette in attesa di un suo messaggio di conferma;
- 2) ricevuta una conferma (*inform (alive)*) dal *MMA* lo *UMA* avvia tutte le agende (*UPA*) di cui gli utenti prelevati dal database sono proprietari e attende conferma da ognuna;
- 3) ricevuto un *inform (alive)* da parte di un'agenda, lo *UMA* invia allo *UPA* il nome del proprio database personale ed il nome utente e la password che ne consentono l'accesso;

Durante il meccanismo di avvio vengono effettuati dei controlli per impedire che il sistema si trovi in situazioni anomale. Tali controlli impediscono ad esempio che un'agenda venga avviata qualora nel sistema non vi sia nessun agente che offra il servizio di gestione delle riunioni.

Una volta avviate le agende, ogni *UPA* si pone in uno stato di attesa per eventuali messaggi provenienti dal *MMA* o da un eventuale *GA*, che può cercare di contattare un'agenda per effettuare il login.

VI. GA: AGENTE GESTORE DELLA GUI

Il *GA* deve essere lanciato dall'utente che vuole effettuare un login sulla propria agenda. Questo agente avvia un'interfaccia grafica tramite la quale l'utente può interagire con il proprio

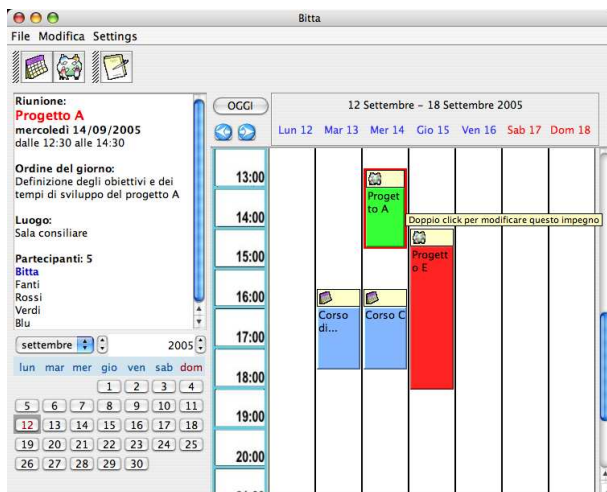


Fig. 4. Interfaccia presentata all'utente dal GUI Agent.

UPA, inserendo o modificando degli impegni. I compiti di tale agente sono:

- 1) permettere il login su di un UPA, contattando lo UMA allo scopo di identificare l'utente e l'agenda di cui egli è proprietario;
- 2) fornire un servizio di traduzione delle azioni eseguite dall'utente in opportuni messaggi da inviare allo UPA affinché l'azione richiesta venga eseguita;
- 3) fornire un servizio di traduzione dei messaggi provenienti dallo UPA in opportune azioni dell'interfaccia che permettano di visualizzare le informazioni contenute nell'agenda;

In figura 4 è mostrata l'interfaccia che viene presentata all'utente dopo che è stato effettuato il login.

A. Login/logout di un utente

Una volta avviato, il GA provvede a registrarsi presso il DF e richiede all'utente di immettere nome e password per effettuare il login. Inserite queste informazioni vengono effettuati i passi seguenti (figura 5):

- 1) il GA contatta il gestore degli utenti, UMA, inviando il nome utente e chiedendo che gli venga fornito l'indirizzo dell'agenda di cui l'utente è proprietario
- 2) lo UMA identifica l'utente ed il suo UPA, cerca tale agente tra quelli registrati nella piattaforma ed invia il suo indirizzo al GA; nel caso in cui non esista un'agenda per l'utente specificato, il gestore degli utenti risponde con un failure che termina il protocollo;
- 3) il GA contatta lo UPA inviando una richiesta di login. Se l'utente risulta già loggato l'UPA rifiuta la richiesta, altrimenti chiede al GA la password di accesso e verifica l'identità dell'utente confrontando nome e password con le informazioni memorizzate nel proprio database personale;
- 4) se il confronto ha successo il login viene accettato, viene registrato l'indirizzo del GA che lo ha effettuato il login e a quest'ultimo viene inviato un messaggio di conferma

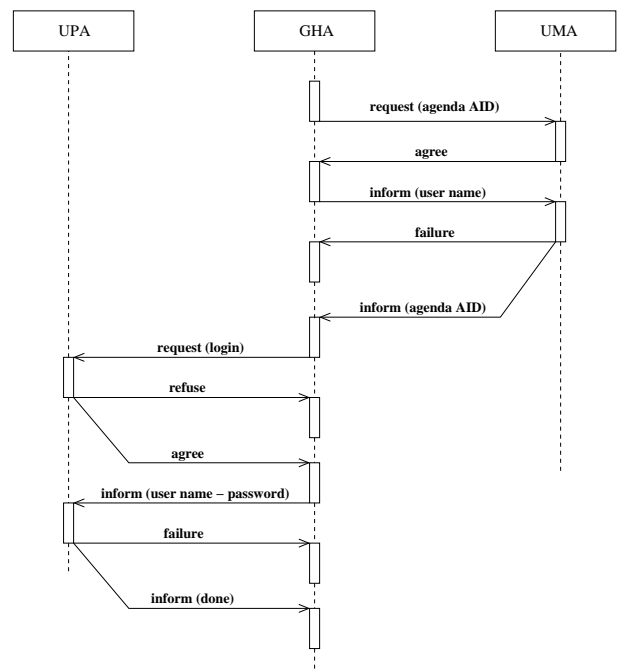


Fig. 5. Messaggi scambiati dagli agenti durante il meccanismo di login di un utente.

(inform (done)), in caso contrario viene inviato un messaggio di fallimento (failure);

Al termine del meccanismo di login lo UPA conosce l'indirizzo del GA e si pone in attesa di eventuali richieste provenienti da questo agente.

Il meccanismo di logout viene invece effettuato senza che venga coinvolto il gestore degli utenti. In questo caso il GA invia la richiesta di logout direttamente all' UPA, il quale deregistra tale agente e si pone in uno stato in cui non possono essere accettate richieste provenienti dal GA.

B. Interazione con l'agenda

Dopo avere effettuato il login, il sistema permette agli utenti di inserire, rimuovere e modificare impegni personali all'interno della propria agenda. Tutte le funzionalità vengono realizzate inviando dei messaggi di richiesta allo UPA di proprietà dell'utente.

Le conversazioni tra il GA e l'UPA vengono effettuate utilizzando una versione modificata del protocollo FIPA-Request. Il protocollo utilizzato prevede un inform aggiuntivo la cui utilità consiste nel poter inviare al destinatario delle informazioni necessarie a portare a termine la richiesta una volta che questa sia stata accettata.³ Il protocollo utilizzato è riportato in figura 6:

- 1) il GA invia allo UPA una richiesta in cui viene specificato il tipo di azione che si vuole venga effettuata;

³Ad esempio una tipica richiesta della GUI verso l'agenda è costituita da un messaggio request che contiene l'azione che si vuole venga eseguita (es. richiesta di invio degli impegni), mentre tutti i parametri necessari ad eseguirla (es. periodo di tempo di cui si vogliono gli impegni) vengono forniti nell'inform successivo.

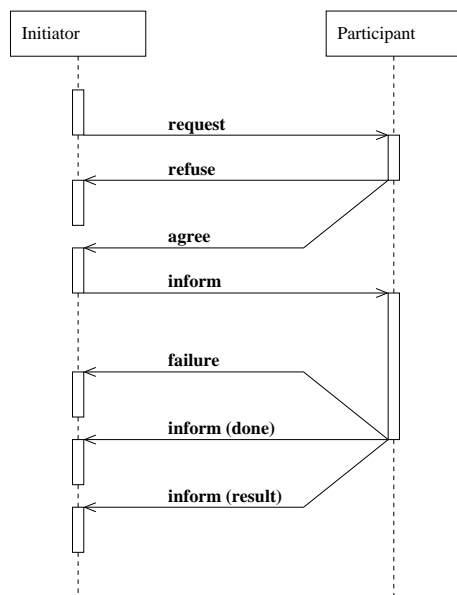


Fig. 6. Protocollo FIPA-request con l'aggiunta di un messaggio inform

- 2) lo *UPA* decide se accettare o rifiutare la richiesta ed invia un messaggio di conferma o di rifiuto al *GA*;
- 3) se la richiesta viene accettata, il *GA* invia un *inform* contenente eventuali parametri necessari a soddisfare la richiesta;
- 4) lo *UPA* esegue l'operazione richiesta: in caso di successo viene inviato al *GA* un *inform* contenente il risultato dell'operazione, in caso contrario viene inviato un messaggio di fallimento (*failure*) che conclude la conversazione.

L'interazione con il gestore della GUI da parte dell'agenda viene effettuata da un *behaviour* strutturato come una macchina a stati finiti costituita dai seguenti stati (figura 7):

- 1) stato di accettazione delle richieste: l'agenda accetta qualsiasi messaggio di richiesta proveniente dal gestore della GUI che ha effettuato il login, ricevuto un messaggio l'agenda si porta nello stato successivo;
- 2) stato di identificazione della richiesta: il messaggio ricevuto viene utilizzato per identificare l'operazione richiesta; in questo stato l'agenda decide se accettare o meno la richiesta ed invia un messaggio di conferma o di rifiuto al *GA*;
- 3) uno stato di rifiuto in cui l'operazione richiesta non viene eseguita e la conversazione viene terminata inviando un *refuse* al *GA*;
- 4) uno stato per ogni possibile operazione: una volta identificata ed accettata la richiesta l'agenda si pone in uno dei possibili stati, in attesa di eventuali parametri da parte del *GA*; una volta ricevuto il messaggio *inform* contenente tali parametri l'operazione viene portata a termine;
- 5) stato conclusivo: vengono inviati al *GA* i risultati dell'operazione (oppure un messaggio di fallimento se essa

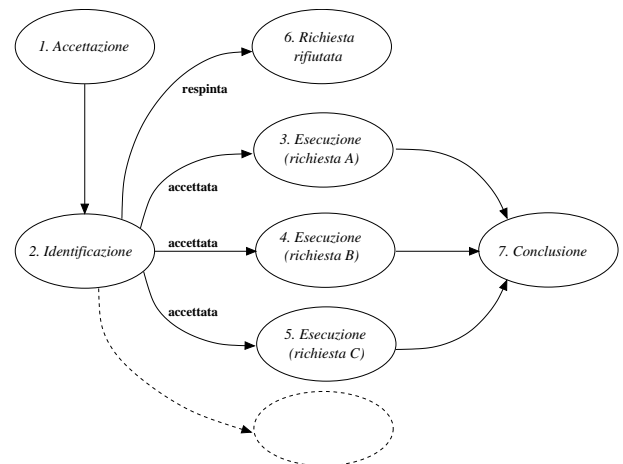


Fig. 7. Macchina a stati finiti che implementa l'interazione con la GUI da parte dell'agenda. Lo stato tratteggiato rappresenta l'esecuzione di ulteriori possibili richieste.

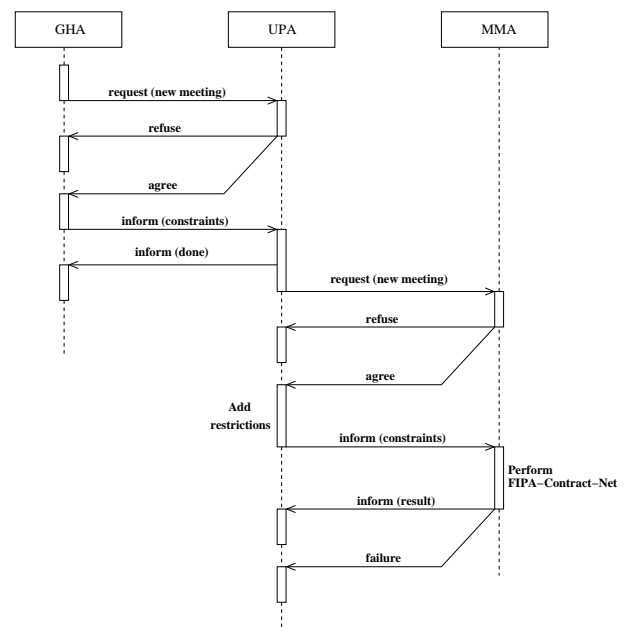


Fig. 8. Messaggi scambiati durante la richiesta di una nuova riunione da parte di un utente.

non è andata a buon fine) e si riporta la macchina allo stato di accettazione;

VII. MMA: GESTIONE DELLE RIUNIONI

Il *MMA* è costantemente in attesa di richieste provenienti dagli *UPA* ed è in grado di fornire due servizi:

- 1) inviare ad un *UPA*, qualora ne faccia richiesta, le riunioni a cui l'utente che ne è proprietario deve partecipare (perchè possano essere mostrate all'utente oppure per definire gli intervalli in cui egli è libero da impegni);
- 2) assumere il ruolo di organizzatore di nuove riunioni, in seguito ad una richiesta di nuova riunione da parte di un *UPA*.

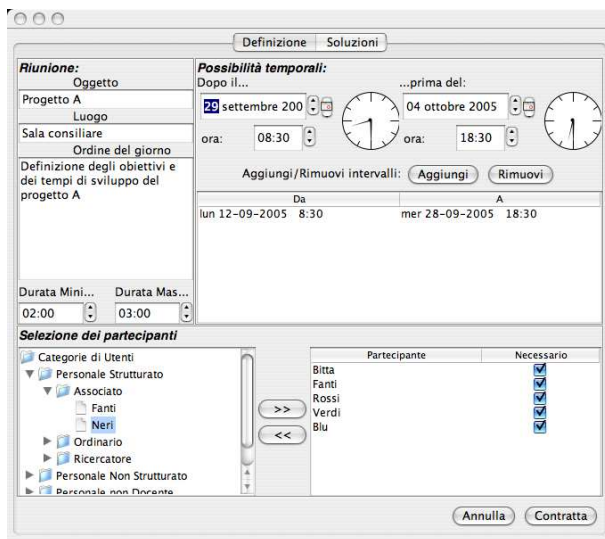


Fig. 9. Inserimento dei vincoli per una nuova riunione.

Ogni utente può richiedere che venga fissata una riunione (figura 9) e definire dei vincoli temporali sulla sua collocazione. La richiesta dell'utente viene inviata dal GA allo UPA che provvede ad attivare il MMA, il quale si assumerà l'incarico di contattare le agende di tutti gli utenti invitati a prendere parte alla riunione e di elaborare delle possibili soluzioni che rispettino i vincoli temporali che sono stati imposti. I messaggi scambiati dagli agenti in questa fase sono riportati in figura 8. La richiesta di fissare una nuova riunione può essere rifiutata dal MMA.⁴ La conversazione tra il GA e lo UPA viene sospesa non appena lo UPA abbia ricevuto i vincoli definiti dall'utente, in attesa che il MMA elabori delle possibili soluzioni. Durante la conversazione con il MMA lo UPA provvede ad aggiungere altri vincoli (determinati dagli impegni dell'utente che ha iniziato la richiesta). Spetta invece al MMA il compito di soddisfare la richiesta. Per fissare una nuova riunione l'utente deve fornire:

- la descrizione della riunione da pianificare, ovvero l'oggetto della riunione, il luogo in cui si dovrà tenere e opzionalmente una descrizione degli argomenti da trattare;
- uno o più intervalli temporali alternativi entro i quali vincolare la collocazione della riunione;
- la durata minima ed eventualmente quella massima da assegnare alla riunione. Nel caso in cui la durata massima non venga fornita essa verrà posta uguale a quella minima e si cercherà di fissare la riunione esattamente con quella durata;
- gli utenti invitati a partecipare. Essi possono essere selezionati da una struttura ad albero in cui sono mostrati tutti gli utenti registrati, organizzati in categorie. Inoltre l'iniziatore può marcare alcuni utenti come *non neces-*

⁴Attualmente, ad esempio, essa viene rifiutata nel caso in cui essa coinvolga utenti che sono impegnati a fissare un'altra riunione, poiché il sistema non è in grado di gestire in modo corretto richieste concorrenti. L'inserimento di tale funzionalità potrà essere valutato in futuro.

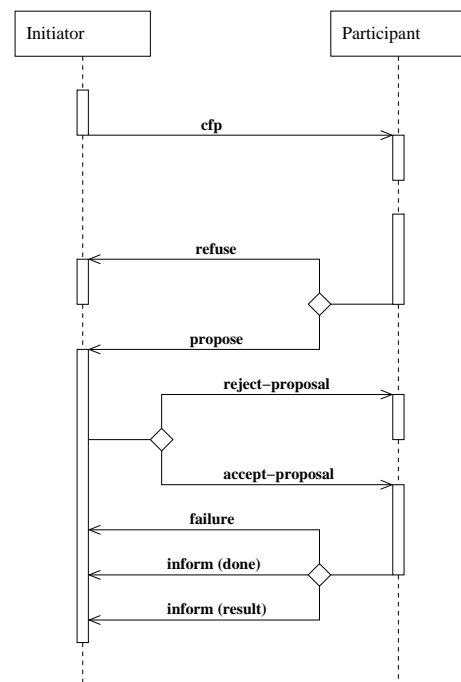


Fig. 10. Protocollo FIPA-Contract-Net

sari, permettendo al sistema di tenere conto di questa indicazione se ciò può essere di aiuto all'identificazione di possibili soluzioni.

A questi vincoli si aggiunge quello costituito dalla *non sovrapposizione* con eventuali impegni dell'utente iniziatore. Tali informazioni vengono inviate al MMA, il quale inizia una conversazione con le agende degli utenti invitati utilizzando il protocollo FIPA-Contract-Net (figura 10):

- 1) il MMA invia allo UPA di ogni utente invitato un *cfp* contenente tutte le informazioni circa la riunione che si vuole fissare;
- 2) se l'UPA è già impegnato a fissare un'altra riunione risponde al MMA con un *refuse* allo scopo di evitare il caso di richieste concorrenti. Altrimenti l'UPA esamina i propri impegni e costruisce una lista di intervalli in cui il proprio utente può essere considerato libero; questa lista (eventualmente vuota)⁵ viene inviata al MMA tramite un *propose*, assieme alle preferenze che l'utente ha espresso configurando la propria agenda;
- 3) il MMA esamina le risposte ricevute estraendo una lista di possibili collocazioni per la riunione;
- 4) se tale lista è vuota il MMA invia ad ogni UPA contattato un messaggio *reject-proposal* che conclude la conversazione; in caso contrario viene inviato un *accept-proposal*;
- 5) lo UPA invia un *inform* di conferma che conclude la conversazione.

⁵il MMA è in grado di identificare anche collocazioni temporali in cui non tutti gli utenti sono disponibili e di valutare la qualità di tali collocazioni in base alla percentuale di utenti che possono effettivamente partecipare alla riunione.

Al passo 3 il *MMA* analizza le risposte ricevute dalle agende, da cui estrae delle *possibili soluzioni* per la richiesta di riunione. Tali soluzioni sono definite come segue:

Definizione 7.1: Una **possibile soluzione** per una richiesta di riunione è un qualsiasi intervallo di tempo che soddisfi tutti i vincoli temporali definiti dall'iniziatore (compreso quello di non sovrapposizione con gli impegni di tale utente).⁶ Nelle prossime sezioni verrà illustrato il modo in cui l'*MMA* identifica le possibili e soluzioni e ne valuta la qualità.

A. Funzioni obiettivo e valutazione

Prima di identificare le possibili soluzioni il gestore delle riunioni esamina le risposte delle agende invitate e costruisce le *funzioni obiettivo* che permetteranno di valutare la qualità degli intervalli individuati. Tali funzioni vengono utilizzate anche per la selezione di un sottoinsieme delle possibili soluzioni, allo scopo di limitarne il numero (che altrimenti potrebbe essere infinito) ed evitare di considerare soluzioni di scarsa rilevanza.

Per ogni utente i invitato alla riunione vengono costruite, sulla base della risposta ricevuta dal suo *UPA*, due funzioni, che chiamiamo *disponibilità* ($d_i(t)$) e *preferenza* ($p_i(t)$) dell'utente i -esimo, dove t rappresenta il tempo.

Definizione 7.2: La **funzione di disponibilità** $d_i(t)$ dell' i -esimo utente assume i seguenti valori

$$d_i(t) = \begin{cases} 1 & \text{utente non impegnato al tempo } t \\ 0 & \text{utente impegnato al tempo } t \end{cases}$$

La funzione di preferenza $p_i(t)$ viene costruita a partire dalle preferenze espresse dall' i -esimo utente su di una settimana tipo. Esse vengono tradotte in un insieme di *valori di preferenza*, ovvero valori discreti all'interno di un range predefinito.

Definizione 7.3: La **funzione di preferenza** $p_i(t)$ assume il valore di preferenza espresso dall'utente i -esimo per il giorno della settimana e l'ora in cui ricade l'istante t .

A partire dalle funzioni di disponibilità e preferenza degli utenti invitati alla riunione, vengono costruite due funzioni obiettivo: la **disponibilità media** $d(t)$ e la **preferenza media** $p(t)$ dei partecipanti alla riunione. La disponibilità media $d(t)$ degli utenti invitati è definita nel seguente modo:

$$d(t) = \frac{1}{n} \sum_{i=1}^n d_i(t) \quad (1)$$

mentre $p(t)$ è definita come:

$$p(t) = \frac{1}{n} \sum_{i=1}^n p_i(t) \quad (2)$$

dove n rappresenta il numero di utenti invitati a partecipare alla riunione.

Le possibili soluzioni identificate per soddisfare la richiesta di riunione, costituite da intervalli temporali per la sua collocazione, vengono valutate in base alle funzioni obiettivo che

⁶Sovrapposizioni con impegni di altri utenti sono in generale permesse, tuttavia verranno in seguito esaminate ed utilizzate allo scopo di valutare la qualità degli intervalli individuati.

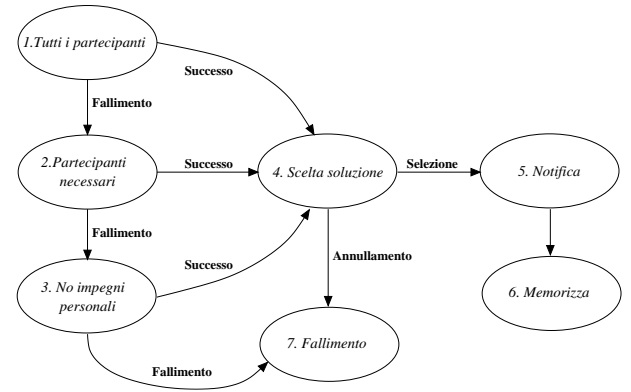


Fig. 11. Macchina a stati finiti per l'identificazione degli intervalli da esaminare per la ricerca delle soluzioni rilevanti.

sono state definite. Ad ogni soluzione vengono associati due indici α e β ottenuti integrando le funzioni obiettivo all'interno dell'intervallo da valutare. In particolare, data una soluzione $\omega = [t_0, t_1]$, si ha

$$\alpha_\omega = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} d(t) \delta t \quad (3)$$

$$\beta_\omega = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} p(t) \delta t \quad (4)$$

dove la normalizzazione viene effettuata per rendere confrontabili intervalli di diversa durata.¹

Le soluzioni individuate vengono quindi ordinate sulla base degli indici α e β e della loro durata. Il criterio di ordinamento tra due intervalli $\omega_1 = [t_0, t_1]$ e $\omega_2 = [t_2, t_3]$ è il seguente:

$$\omega_1 \prec \omega_2 \begin{cases} \alpha_1 > \alpha_2 \vee \\ \alpha_1 = \alpha_2 \wedge \beta_1 > \beta_2 \vee \\ \alpha_1 = \alpha_2 \wedge \beta_1 = \beta_2 \wedge D_1 > D_2 \vee \\ \alpha_1 = \alpha_2 \wedge \beta_1 = \beta_2 \wedge D_1 = D_2 \wedge t_0 < t_2 \end{cases} \quad (5)$$

dove con D_1 e D_2 sono state indicate rispettivamente la durata dell'intervallo ω_1 e quella di ω_2 . Si procede confrontando ordinatamente α , β ed infine la durata delle soluzioni in esame. Nel caso in cui questi parametri risultino tutti coincidenti si rispetta l'ordinamento temporale definito dall'istante iniziale di ogni soluzione.

B. Individuazione delle possibili soluzioni

Le soluzioni, ovvero gli intervalli in grado di soddisfare alla richiesta, vengono individuate tramite una macchina a stati finiti (riportata in figura 11) che attualmente è costituita da tre fasi di ricerca di soluzioni:

¹Se gli indici α e β non venissero normalizzati rispetto alla durata di ω , intervalli molto lunghi ma di scarsa qualità rispetto alle funzioni obiettivo potrebbero essere privilegiati rispetto a intervalli più brevi ma migliori dal punto di vista delle funzioni di disponibilità e preferenza.

- 1) si tengono in considerazione gli impegni di tutti gli utenti invitati: viene costruita una lista costituita dagli intervalli temporali in cui tutti gli utenti invitati risultano liberi da impegni;
- 2) in caso di fallimento si passa a considerare solo gli impegni di tutti gli utenti invitati come partecipanti necessari: viene costruita una lista costituita da intervalli temporali in cui tutti gli invitati necessari risultano liberi da impegni;
- 3) se anche lo stadio 2 si conclude con un fallimento si cerca una soluzione considerando solo gli impegni dell'iniziatore e le riunioni che coinvolgono i partecipanti necessari: viene costruita una lista costituita dagli intervalli temporali in cui l'iniziatore risulta libero da qualsiasi impegno e gli invitati necessari risultano liberi da altre riunioni in cui la loro presenza sia necessaria.
- 4) nel caso in cui non si riesca a definire alcun possibile intervallo in cui pianificare la riunione, il procedimento termina avvertendo del fallimento l'iniziatore, il quale potrà eventualmente modificare la richiesta ridefinendone i vincoli temporali.

Nel caso in cui tutte le fasi falliscano la macchina termina in uno stato di fallimento e la richiesta non viene soddisfatta. In caso contrario, la lista L degli intervalli che sono stati individuati e le funzioni obiettivo $d(t)$ e $p(t)$ vengono utilizzati per costruire un insieme di *istanti rilevanti* da utilizzare per identificare le possibili soluzioni.

Definizione 7.4: Chiamiamo T l'insieme degli istanti rilevanti. Un istante t_k è rilevante se

- t_k coincide con l'istante iniziale o finale di uno degli intervalli contenuti in L ,
- t_k coincide con un punto di salto di almeno una delle funzioni obiettivo $d(t)$ e $p(t)$.⁷

La lista di intervalli L e l'insieme T degli istanti rilevanti, vengono analizzati dall'*Algoritmo di Identificazione delle Soluzioni Rilevanti* (figura 12) allo scopo di estrarne un insieme di possibili soluzioni. Tale algoritmo rappresenta un elemento di novità rispetto alle soluzioni proposte in [2], [3], [5], [6]. Anzichè discretizzare il tempo in intervalli predefiniti l'algoritmo individua degli intervalli che sono rilevanti al fine di ottimizzare la collocazione di una riunione rispetto alle funzioni obiettivo. Vengono considerati tutti e soli gli intervalli temporali che sono candidati a rappresentare dei massimi locali per gli indici α e β . Questo permette di ottenere un numero finito di possibili soluzioni con il vantaggio di una maggiore flessibilità.

Esempio: Nella situazione riportata in figura 13 si ha un intervallo da analizzare e tre istanti temporali rilevanti t_0 , t_1 e t_2 , di cui t_0 e t_2 rappresentano l'istante iniziale e finale dell'intervallo da analizzare, mentre t_1 rappresenta un punto di salto di una delle funzioni obiettivo. L'algoritmo di individuazione delle soluzioni inizia individuando una soluzione di lunghezza minima posizionata a partire dall'istante t_0 . Successivamente

Algoritmo di Identificazione delle Soluzioni Rilevanti

input: un insieme L di intervalli da esaminare, la durata minima e massima d_{min} e d_{max} , un insieme di *istanti rilevanti* T

output: Un insieme R di soluzioni.

```

1.  $R \leftarrow \emptyset$ 
2. for each  $I \in L$  do
3.    $\omega = [I_{start}, I_{start} + d_{min}]$ 
4.   while  $\omega_{end} < I_{end}$  do
5.      $R = R + \omega$ 
6.     if  $(\omega_{end} - \omega_{start}) < d_{max}$  then
7.        $\omega_{end} = \min \{next(T, \omega_{end}), (\omega_{start} + d_{max})\}$ 
8.     else
9.       if  $(next(T, \omega_{end}) - \omega_{end}) < (next(T, \omega_{start}) - \omega_{start})$  then
10.         $\omega_{start} = next(T, \omega_{end}) - d_{max}$ 
11.      else
12.         $\omega_{start} = next(T, \omega_{start})$ 
13.       $\omega_{end} = \omega_{start} + d_{min}$ 
14. return  $R$ 

```

Fig. 12. Algoritmo per l'identificazione delle soluzioni. I pedici *start* e *end* indicano rispettivamente l'istante iniziale e finale di un intervallo. La funzione $next(T, t)$ restituisce il primo istante temporale successivo a t tra quelli contenuti in T .

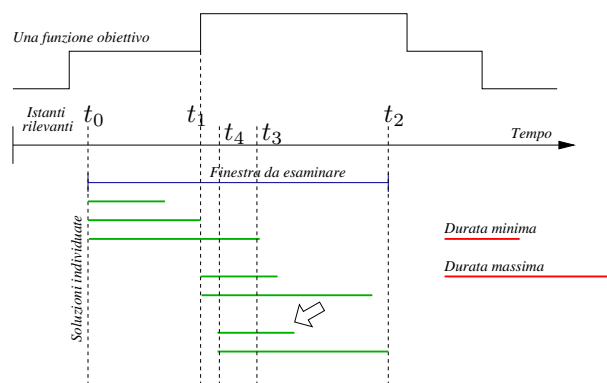


Fig. 13. Esempio di individuazione delle soluzioni rilevanti all'interno di uno dei possibili intervalli da esaminare. La soluzione indicata dalla freccia è quella individuata al passo 10 dell'algoritmo.

l'algoritmo espande la durata, individuando una soluzione che inizia in t_0 e termina in t_1 . Cercando nuovamente di espandere la durata della soluzione non è possibile individuare altri istanti rilevanti che precedano t_3 , il quale rappresenta l'istante finale della soluzione a durata massima con inizio in t_0 . Dopo aver memorizzato la soluzione compresa tra t_0 e t_3 si procede cercando di spostare l'istante iniziale. Esso viene posizionato in t_1 , e l'algoritmo riparte considerando la soluzione di durata minima e cercando di espanderla sino alla durata massima. Infine l'istante iniziale viene posizionato in t_4 , calcolato in modo da poter espandere la durata della soluzione sino al massimo valore possibile restando nei limiti imposti dall'intervallo analizzato (la soluzione a durata massima con istante iniziale in t_4 termina in t_2 , ultimo istante utile nell'intervallo a cui ci si è vincolati). Tutte le soluzioni individuate vengono valutate, ordinate e comunicate all'agente *GA* dell'iniziatore, che le presenta all'utente (figura 14) il quale ne può selezionare

⁷Tali funzioni sono discontinue a tratti per costruzione

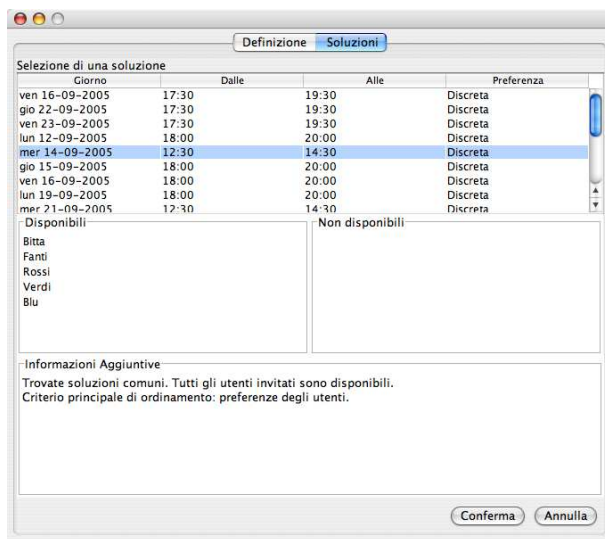


Fig. 14. Possibili soluzioni presentate all'utente iniziatore.

una da assegnare definitivamente alla nuova riunione. Tale scelta potrà essere effettuata esaminando le caratteristiche di ciascuna soluzione (qualità stimata, massimo numero di utenti non disponibili, identità di tali utenti e percentuale di utenti disponibili in media). L'intervallo selezionato sarà quindi assegnato alla riunione che potrà essere notificata agli utenti invitati e memorizzata nel database.⁸

VIII. CONCLUSIONI E SVILUPPI FUTURI

In questo articolo abbiamo descritto MAgentA, un sistema multi agente che offre un servizio di gestione degli impegni degli utenti con la possibilità di pianificare nuove riunioni in modo semiautomatico. Ogni utente può inserire e modificare a piacimento i propri impegni personali ed inviare al sistema richieste per pianificare nuove riunioni. Tali richieste sono gestite trovando collocazioni temporali per una riunione che soddisfino le disponibilità degli utenti invitati, o almeno di un loro sottoinsieme, secondo alcuni criteri che cercano di minimizzare i conflitti con altri impegni e riunioni, oltre che di massimizzare il soddisfacimento delle preferenze dei partecipanti. La bontà delle soluzioni ottenute viene stimata in base al numero effettivo di utenti disponibili in ogni possibile soluzione ed alle preferenze espresse dagli utenti.

Il sistema MAgentA presenta alcuni aspetti che lo differenziano dai lavori preesistenti sulla stessa tematica (come ad esempio [2], [3], [5], [6]). In particolare:

- 1) MAgentA utilizza lo standard FIPA [8] per la comunicazione tra gli agenti: le conversazioni sono state realizzate utilizzando messaggi conformi allo standard FIPA-ACL [11], mentre per l'identificazione di possibili

intervalli in cui posizionare una riunione è stato utilizzato il protocollo FIPA-Contract-Net; questo rende il sistema aperto all'integrazione con piattaforme conformi allo standard;

- 2) nella pianificazione di una riunione il tempo non viene discretizzato in intervalli fissi e noti a priori, ma suddiviso in intervalli dipendenti dall'andamento di funzioni obiettivo costruite per soddisfare una particolare richiesta;
- 3) le possibili collocazioni temporali di una riunione vengono identificate da un algoritmo guidato da funzioni obiettivo, capace di considerare soluzioni di diversa durata e di scartare le possibilità di scarsa rilevanza.

Il sistema è stato valutato sperimentalmente costruendo un database di agende e posizionando gli impegni di ciascuna in modo tale da verificare il comportamento del software in diverse situazioni. I risultati ottenuti in questa analisi sperimentale preliminare hanno mostrato la capacità di MAgentA di pianificare riunioni in modo efficiente rispetto alle possibilità degli utenti ed alle loro preferenze. In futuro si prevede di effettuare una fase di sperimentazione più approfondita che permetta di verificare il comportamento, le prestazioni ed i limiti del sistema proposto all'interno di un contesto realistico.

Alcuni possibili sviluppi futuri del sistema sono: il raffinamento dei vincoli (es. controllo sul numero legale di partecipanti prima di memorizzare la riunione); la possibilità di esprimere vincoli di ordinamento rispetto ad altre riunioni; la gestione di richieste concorrenti e di spostamento di riunione; lo sviluppo di un'interfaccia WEB; l'integrazione con tecnologie standard per la gestione di calendari.

REFERENCES

- [1] N.R.Jennings and M.Wooldridge, *Applications of Intelligent Agents*. University of London: Queen Mary and Westfield College, 1998, in *Agent Technology: Foundations, Applications and Markets*, 3-28.
- [2] X. D. Ahlem Ben Hassine and T. B. Ho, *Agent Based Approach to Dynamic Meeting Scheduling Problems*. AAMAS'04, 2004, vol. 3, pp.1132-1139.
- [3] E. Crawford and M. Veloso, *Mechanism Design for Multi-Agent Meeting Scheduling Including Time Preferences, Availability, and Value of Presence*. Beijing, China: in *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, September 2004.
- [4] —, "Learning Dynamic Time Preferences in Multi-Agent Meeting Scheduling," School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Tech. Rep., 2005.
- [5] N.R.Jennings and A.J.Jackson, *Agent Based Meeting Scheduling: A Design and Implementation*. Electronics Letters, The Institution of Electrical Engineering, 31 (5), March 1995, pp. 350-352.
- [6] S. Sen, *An automated distributed meeting scheduler*. IEEE Expert, July/August 1997, vol. 12, no. 4, pp. 41-45.
- [7] JADE web site. <http://jade.cselt.it/>.
- [8] FIPA web site. <http://www.fipa.org>.
- [9] JPOX web site. <http://www.jpox.org>.
- [10] F. B. et al., *JADE Administrator's guide*, January 2005, <http://jade.cselt.it/doc/index.html>.
- [11] FIPA ACL specifications. <http://www.fipa.org/repository/aclspecs.html>.

⁸Gli utenti che si trovano "on line" in quel momento riceveranno un messaggio di notifica che potranno confermare o rifiutare. In ogni caso la riunione verrà memorizzata, ma in caso di rifiuto l'iniziatore verrà avvertito dell'assenza dell'utente. Gli utenti "off line" accettano per default tutte le notifiche.

Improving Aglets with Strong Agent Mobility through the IBM JikesRVM

Giacomo Cabri, Luca Ferrari, Letizia Leonardi, Raffaele Quitadamo, *Member, IEEE*

Abstract — Agents are problem-solving entities that, thanks to characteristics such as autonomy, reactivity, proactivity and sociality, together with mobility, can be used to develop complex and distributed systems. In particular, mobility enables agents to migrate among several hosts, becoming active entities of networks. Java is today one of the most exploited languages to build mobile agent systems, thanks to its object-oriented support, portability and network facilities. Nevertheless, Java does not support strong mobility, i.e., the mobility of threads along with their execution state; thus developers cannot develop agents as real mobile entities. This paper reports our approach for Java thread strong migration, based on the IBM Jikes Research Virtual Machine, presenting our results and proposing an enrichment of the Aglets mobile agent platform in order to exploit strong agent mobility.

Index Terms — Mobile Agents, JikesRVM, Aglets, strong mobility.

I. INTRODUCTION

AGENTS are autonomous, proactive, active and social entities able to perform their task without requiring a continue user interaction [23]; thanks to the above features, the agent-oriented paradigm is emerging as a feasible approach to the development of today's complex software systems [18]. Moreover agents can be *mobile*, which means they can migrate among different sites/hosts during their execution.

Mobility is an interesting feature for agents, since they are able to move among networks to find out data and information, to perform load balancing activities, and so on. The exploiting of mobile agents can simplify different issues in the design and implementation of applications and enables developers to quickly build distributed and parallel systems.

Mobile agent execution is hosted by a special software layer, called Mobile Agent Platform (MAP) that enables also the agent migration and allows security checks over agents. In order to enable agents to migrate among different platforms and, thus, architectures, portable technologies and languages must be adopted to develop MAPs and the corresponding agents. Thanks to its portability and network facilities, Java is today the most exploited language to develop mobile agents, and in fact several Java-based MAPs exist [17, 3, 28].

With regard to mobility, we distinguish [34] *strong*

mobility, which enables the migration of code, data and execution state of execution units (for instance, *threads*), from *weak* mobility, which migrates only code and data [14]. Some distributed operating systems [32] go even further and make it possible for entire processes to be migrated with also their kernel mediated state, comprising I/O descriptor, alarm timers and others. This extremely transparent migration is known as *full migration* [16]. Unfortunately, current standard Java Virtual Machines (JVMs) do not support thread migration natively, and thus a Mobile Agent Platform running on top of them cannot provide strong mobility of agents. Moreover, the Java language itself [15] does not support constructs or mechanisms for thread serialization and migration: that is, there is no way, using the standard Java language and JVMs, to enable agents to exploit strong mobility. Even if this does not represent a problem for many applications based on mobile agents, such as those of automated booking [13], it does not allow using the agent paradigm to develop more complex and distributed systems, such as those for load balancing [25].

To overcome this limitation, this paper proposes an approach to support strong thread migration for Java MAPs, based on the IBM JikesRVM [5], which is a Java virtual machine with very interesting features. Our approach significantly differs from other proposals, since it requires neither any modification to the JVM, nor it exploits any pre-processing, but it simply defines an appropriate Java library.

The paper is organized as follows: section II presents the state of the art, explaining the existing approaches and pointing out their limitations. Section III introduces the features of JikesRVM and explains how they can be exploited to build a library for supporting strong mobility. Section IV presents the Aglets platform and shows how strong mobility can be designed and implemented in such a platform by using our mobility library. Finally, Section V reports our first performance measures and Section VI concludes the paper.

II. STATE OF THE ART

The approach presented in this paper aims at implementing strong thread migration in Java, which is not a new idea. Several approaches have been proposed so far and they can be, typically, split into two categories, depending on the fact that they require to modify the JVM (*JVM-level approach*) to support an advanced thread management or exploit some kind of bytecode instrumentation (*Application-level approach*) to

track the state of each thread.

Approaches that modify the JVM (such as Sumatra [2], ITS [10], Merpati [29], Jessica2 [33] and JavaThread [9]) often introduce the problem of the management of the virtual machine itself. First of all, it is worth noting that they are applied to JVM that are at least one (or even more) version older than the SUN production one. Second, the adoption of a modified JVM can introduce problems of trust and security bugs. Third, virtual machines are usually written in a language different from Java (e.g., C++), thus suffering from portability problems.

Instead, approaches that exploit bytecode manipulation (e.g. JavaGoX [26] or Brakes [30]) or Java source code manipulation [16], even if based on a pure Java technique (and thus really portable), do not provide a full thread management and suffer from problems related to performances. In fact, the idea of these approaches is to transparently place a few control instructions, similar to recovery-points, which allow a thread to deactivate itself once it has reached one of them. Recovery-points are quite similar to entry points used in most Java MAPs (i.e., methods that are executed when an agent is reactivated at the destination host), even if the former ones enable a finer grain control than entry points. Unfortunately, a thread cannot deactivate (or reactivate) itself outside of these recovery-points, which are also not customizable, thus a thread cannot really suspend itself in an arbitrary point of the computation. Moreover, the use of bytecode manipulation produces low performances, thus these techniques are not appropriate for those applications where speed represents a strong requirement.

In general, all existing strongly mobile systems have to deal with the problem of locating object references when they want to migrate a thread with all its set of stack-referenced objects: they force the use of some “type inference” mechanism [9, 33], either at execution or at compilation time, thus introducing a significant performance overhead in threads execution. In order to tackle the drawbacks of strong mobility while saving its clarity and power, some interesting algorithms have been proposed [8] that translate transparently the apparently “strongly mobile code” into a “weakly mobile” form, with the above mentioned benefits of weak mobility. Starting from the above considerations, we have decided to design and implement a thread migration system able to overcome all the problems of the above-explained approaches. In particular, it is written entirely in Java, thus portable as much as possible and it grants high performances even without modifying the JVM. In fact, every single component of the migration system has been designed and developed to be used as a normal Java library, without requiring rebuilding, changing or patching the virtual machine, in specific, the IBM JikesRVM. Programmers and users do not have to download a modified, untrustworthy, version of JikesRVM, but can import the implemented mobility package into their code and execute it on their own copy of JikesRVM. Therefore, our JikesRVM-based approach can be classified as a midway approach between the above-mentioned *JVM-level* and *Application-*

level approaches.

III. FROM WEAK TO STRONG MOBILITY: A JIKESRVM-BASED APPROACH

A Mobile Agent Platform realizes an environment for the execution of agents, featured with a bent for mobility. The support for mobility is often one of the first design choices when implementing such a platform, since it has a great impact on the remainder of the design.

A. Weak vs. Strong Mobility

Current execution environments for programming languages (e.g., the Java Virtual Machine [22] and the Common Language Runtime, embedded into Microsoft .NET Framework [1]) are usually not suited or not capable of providing the required level of mobility to the execution units (i.e. the threads) that they host. They all lack an explicit support for the mobility of their execution units and, in order to overcome this lack, MAP designers must choose between two directions [14]:

- Adopting one of the techniques explained in the previous section so that threads can suspend their execution locally and resume elsewhere transparently (*strong mobility*).
- Introducing a further abstraction level above the thread concept: the *weak mobile* agent, which is explicitly thought as a serializable representation of an execution unit.

From the complexity point of view, weak mobility is quite simple to implement using well-established techniques like network class loading or object serialization [27]. However, weak mobility systems, by definition, discard the execution state across migration and hence, if the application requires the ability to retain the thread of control, extra programming is required in order to manually save the execution state. The transparency of the migration offered by strong mobility systems has instead a twofold advantage: it reduces the migration programming effort to the invocation of a single operation (e.g. a `migrate()` method), and requires a size of the migrated code smaller because it does not add artificial code.

Despite these advantages, most of the mobile agent systems support only weak mobility and the reason lies mainly in the complexity issues of strong mobility and in the insufficient support of existing JVMs to deal with the execution state. It is a common idea that strong mobility should be convenient only in load balancing contexts or when thread persistence is needed to build fault-tolerant applications [9].

Recently, an innovative project is drawing researcher’s attention to the benefits that a virtual machine written in the Java language can offer. The main features of this open-source project, called *JikesRVM* [5], are outlined in the following subsection: for the sake of brevity, we will focus on those

aspects that make JikesRVM an ideal execution environment for strongly mobile agents, overcoming the drawbacks and the limitations of many existing solutions.

B. The Jikes Research Virtual Machine

JikesRVM began life in 1997 at IBM T. J. Watson Research Center as a project with two main design goals: supporting high performance Java servers and providing a flexible research platform “where novel VM ideas can be explored, tested and evaluated” [4]. JikesRVM is almost totally written in the Java language, but with great care to achieving maximum performance and scalability exploiting as much as possible the target architecture’s peculiarities. The all-in-Java philosophy of this VM makes very easy for researchers to manipulate or extend its functionalities. Further, JikesRVM source code can be built, with a prior custom compilation, both on IA32 and on PPC platforms [19], but the bulk of the runtime is made up of Java objects portable across different architectures.

The first step toward the development of our MAP based on JikesRVM has been the implementation of the strong Java thread mobility. Threads embody concurrent flows of execution within an instance of the JVM and are represented by the `java.lang.Thread` object [22], used by the Java programmers disregarding any knowledge of their underlying physical implementation. In JikesRVM, threads are full-fledged Java objects and are designed explicitly to be as lightweight as possible [4]. Many server applications need to create new threads for each incoming request and a Mobile Agent Platform has similar requirements since thousands of agents may request to execute within it. While some JVMs adopted the so-called *native-thread model* (i.e. the threads are scheduled by the operating system that is hosting the virtual machine), JikesRVM designers chose the *green-thread model* [24]: Java threads are hosted by the same operating-system thread, implemented by a so-called *virtual processor*, through an object of class `VM_Processor` [6]. Each virtual processor manages the scheduling of its virtual threads (i.e., Java threads), represented by objects of the class `VM_Thread`. The scheduling of virtual threads was defined *quasi-preemptive*, since it is driven by the JikesRVM compiler. What happens is that the compiler introduces, within each compiled method body, special code (*yield points*) that causes the thread to request its virtual processor if it can continue the execution or not. If the virtual processor grants the execution, the virtual thread continues until a new yield point is reached, otherwise it suspends itself so that the virtual processor can execute another virtual thread.

The choice of using virtual processors not only allows JikesRVM to reduce the number of threads the operating system is in charge of, but also allows it to perform an efficient and well-controlled thread-switch. As a consequence, this allows elegantly addressing the problem of precisely locating object references when a garbage collection occurs.

JikesRVM uses *type-accurate* collectors [31] that build the so-called *reference maps* automatically at compile-time, unlike conservative collectors, which attempt somehow to infer whether a stack word is a reference or not. These reference maps are periodical snapshots of the situation of references in each method frame.

The tracks of object references used to speed up the JikesRVM *type-accurate* garbage collectors can be exploited by MAP designers to collect stack-referenced objects for strong thread migration. This eliminates the need for “type inference” mechanisms required by existing strongly mobile systems.

In general, many JVMs do not permit the programmer to access the execution state (i.e. the stack and the context registers), in order to enforce the security model of the Java language. As a consequence, they do not allow strong mobility. Instead, JikesRVM provides, once again, a built-in facility to extract correctly the execution state of a suspended thread. This facility is an efficient implementation of the *On-Stack Replacement* (OSR) technique, originally developed for the Self language [35]. It enables a method to be automatically replaced by the system while it is executing. In particular, the system replaces the runtime stack activation frame of the method with that of the new version, and continues execution at the same point within the new version. JikesRVM exploits the OSR mechanism [12] in order to enable the dynamic optimization of methods. The Adaptive Optimization System (AOS) [7] samples the execution of programs to identify frequently executed (i.e. “hot”) methods and, when their optimization is predicted to be beneficial, the system compiles the method with JikesRVM *optimizing compiler* [11]. The old less-optimal frame is discarded and a new optimized frame is placed, initialized with the current state of the method (i.e. the value of the local variables and stack operands, together with the current bytecode index).

This mechanism has been successfully exploited to quickly get a complete and portable representation of the serialized call stack. The structure of the *OSR scope descriptor* [12] inspired the idea of the `MobileFrame`: an object representing the current state of the method execution in a format that should be understandable by any JVM since it refers purely to bytecode-level entities (bytecode program counter, locals and stack operands). Our mechanism applies the capturing to all user frames in the stack of the serialized thread and, on the one hand, offers the advantage of the portability of the frames and, on the other hand, exploits a fully integrated component of the JVM. The latter aspect is crucial from both the reliability and the performance point of view, since no unsafe manipulations are carried out on the JVM code to force the externalization of the execution state of the thread.

The presented features of JikesRVM allow the addition of strong thread migration, without modifying the virtual machine, but simply extending it. The entire system is available as a library comprised in a Java package that can be imported as usual into the application code. This means that

the implemented JikesRVM extension does not affect the performance of other applications, since no permanent modifications have been made to the VM itself.

IV. STRONG MOBILITY IN AGLETS

A. Overview Of The Aglets Workbench

The Aglets Workbench [3] is a project originally developed by the IBM Tokyo Research Laboratory with the aim of producing a platform for the development of mobile agent based applications by means of a 100% Java library. The Aglets Workbench provides developer with applet-like APIs [20], thus creating a mobile agent (called Aglet) is a quite straightforward task. It suffices to inherit from the base class Aglet and to override some methods transparently invoked by the platform during the agent life. Weak mobility is provided through the Java serialization mechanism, and a specific agent transfer protocol (ATP) has been built on top of such mechanism [21]. Each Aglet can exploit the special method `dispatch(...)` to move to another host; such method is the equivalent of the generic `migrate(...)` previously mentioned.

As many other Java MAPs, Aglets exploits weak mobility, that means, from a programming point of view, that each time an agent is resumed at a destination machine, its execution restarts from a defined entry point, that is the `run()` method call. Due to this, dealing with migrations is not always trivial, and developers have to adopt different techniques to handle the fact an agent will execute several times the same code but on different machines. Even if the Aglets library provides a set of classes that helps dealing with migrations, the code will appear like the one shown in the simple example of Figure 1. There, in case of a single migration, the migrated flag is used to select a code branch for the execution either on the source or destination machine.

```
public class MyAgent extends Aglet{
    protected boolean migrated = false;
    // indicates if the agent has moved yet
    public void run(){
        if( ! migrated ){
            // things to do before the migration
            // ...
            migrated = true;
            try{
                dispatch(new URL("atp://nexthost.unimore.it");
            }catch(Exception e){ migrated = false; }
        }
        else{
            // things to do on the destination host
            // ...
        }
    }
}
```

Figure 1. An example of Aglet with a single migration.

The code of Figure 1 is just a simple example, but similar agents can be written for other agent platforms. The point here

is that with weak mobility, which is the one provided by the Java language and the most existing MAPs, it is as the code routinely performs rollbacks. In fact, looking at the code in Figure 1, it is clear how, after a successful `dispatch(...)` method call that causes the agent migration, the code does not continue its execution in the `run()` method from that point. Instead, the code restarts from the beginning of the `run()` method (on the destination machine, of course), and thus there is a code rollback. The fact that an agent restarts its execution always from a defined entry point, could produce awkward solutions, forcing the developer to use flags and other indicators to take care of the host the agent is currently running on.

B. Designing Strong Mobility

In Section III.B we have presented the innovative features of JikesRVM that can be exploited to strongly migrate threads. Now we apply these features to the Aglets to realize the idea of an Aglet as a strong migrable thread. Instead of using one of the pre-created threads to execute methods of the aglets, JikesRVM makes feasible to have a single independent thread for each aglet. As already mentioned, this is possible because of the lightweight implementation of Java threads in that JVM, being targeted to server architectures, where scalability and performance are key requirements. Further, having a separate thread for each aglet ensures a high level of *isolation* between agents: consider, for example, the case where an agent wants to sleep for some time, without being deactivated (i.e. serialized on the hard disk). Using the classical `sleep()` method on the `java.lang.Thread` object will produce strange effects on the current Aglets implementation platform (such as locking the message passing mechanism). These shortcomings are due to the aforementioned thread sharing among multiple agents through the pool of threads. Instead, potentially dangerous actions by malicious (or bugged) aglets do not affect the stability of our platform, allowing possibly a clean removal of the dangerous agent without the need of a MAP reboot.

Message handling or events are implemented using the quasi pre-emptive JikesRVM scheduler, described earlier. Yield points are used to let the running aglet/thread extract messages from its message queue and handle them. Thus, for example, the aglet can process a *dispatch* message even in the middle of its execution (i.e. while the `run()` method is still in the stack) and strongly migrate to the destination site, where it will resume transparently restarting from the last execution point. The programmer gets rid of the burden of saving intermediate results into serializable fields and of structuring its code with entry points (such as methods) from which the agent execution is restarted each time it arrives at a new host, as mentioned above.

The conceptual model of our prototype was thought as intuitive and understandable as possible in this development stage: we took inspiration from the fantastic world of space

travels through *black holes*. According to this model, a mobile agent (i.e. "the traveller") invokes the services offered by (i.e. "gets himself absorbed by") a black hole on one host (i.e. "planet") to move through the network (i.e. "the space") and arrives at the destination host (i.e. "another planet on a distant universe"), being extracted from the other side of the black hole.

C. Implementing Strong Mobility

After having tested the mobility library building a simple prototypal framework whose classes manage the departure and arrival of the mobile threads, we now in our research are integrating the mobility support in the Aglets framework to have a full-fledged MAP endowed with strong mobility.

The implementation of the *black hole* model is based on JikesRVM and embedded into the Aglets runtime to make available the migration services to agents. In Figure 2, our system is described using the classical notation of *queuing networks*. The software components added by our approach are highlighted with boxes and it can be clearly seen how these parts are dynamically integrated into JikesRVM scheduler, when the programmer opens a black hole to enable migration services: no JVM manipulations are performed, therefore a non invasive extension is carried out.

Agents are classified into three main categories:

1. *Incoming agents*, coming from the outside world and requesting execution on the current host. They are read from a network socket and re-established in the local execution context, to be scheduled there.
2. *Outgoing agents*, which are leaving the scheduler queues to be transferred on another machine. They invoked a `dispatch()` method and got queued into the hole's migration queue.
3. *Stationary agents*, not interested or affected by the migration facilities of our mechanism.

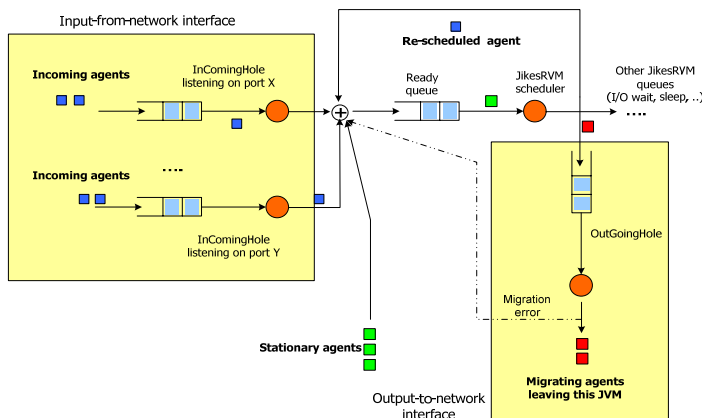


Figure 2. The queuing network model of the mobility framework

The black hole provides two kinds of services: a sending service, for agents exiting the local JVM, and a receiving

service, for incoming execution requests. The former service is implemented by a server thread created with `BlackHole` instantiation, started when the `BlackHole` gets opened. This thread, instance of the `OutGoingHole` class, tests a migration queue in an endless loop, until the application closes its parent `BlackHole`, and analyzes every extracted mobile agent: its execution state is retrieved using OSR built-in state capturing and the thread object, together with the chain of all the stack frames, are written into the socket established with another peer host. In more details, the JikesRVM thread/agent is suspended before the state capturing can occur and the stack is walked back from the last pushed frame to the first one (i.e. the `run()` method). At every step, the corresponding physical frame is analyzed invoking the OSR extraction service and the OSR descriptor is produced; but this intermediate form is not yet fully portable, mainly because it has been conceived only to refer to structures that are supposed to stay in the local memory: in particular, we are talking about the compiled methods in method area and the corresponding program counters (the so called *return address* of each frame) in the machine code body. So, the next essential stage performed by our mechanism is to retrieve a return address as much portable as possible: the bytecode index corresponding to the machine code index of the method. The mapping between the two indexes is, once again, granted by JikesRVM compilers and can be calculated in very little times. Local variable and stack operands are converted also into portable objects and stored into the `MobileFrame` for each method in the stack, as shown in Figure 3.

It must be pointed out that this representation of the serialized thread is a very general one, as it uses only bytecode level entities (e.g. bytecode indexes as program counters, local variables and stack operands and so on) and this grants high portability of the state. Dataspace objects are packed into the mobile frames (or in the thread object) and serialized as well. When all the necessary frames are successfully captured, the system can send them all to the destination host.

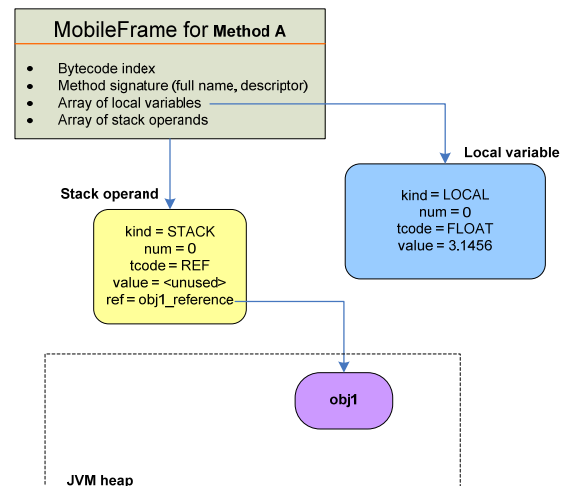


Figure 3. A MobileFrame object

To let external mobile agents enter a local environment, a group of `InComingHole` threads are created and started at `BlackHole` opening time. Each `InComingHole` opens a server socket bound to a specified TCP port and waits, in an endless loop, for incoming connection requests. When a connection is accepted and established, the `InComingHole` reads all the information about the state of the agent and, when finished reading, resumes the agent in the local instance of `JikesRVM`. At the arrival the aglet rebuilding is performed following some essential steps:

1. the aglet object is read from the network stream into the memory;
2. a new thread is created for this aglet or an existing one acquired from the pool, if available;
3. this agent is notified the arrival event and its execution is temporarily frozen;
4. the physical frames, produced by the `MobileFrame` objects, are injected on the fly into its stack;
5. the execution of the thread/aglet is transparently resumed.

The injection task is performed by a *frame installer* component, which adds each frame to a newly allocated stack, adjusting thread context registers and frame pointers. Frames are constructed in compliance with the baseline layout of the target platform. We have currently implemented a working frame installer for the IA32 architecture, but we are planning to complete the system with the PPC frame installer.

The migrated aglet will be, by default, destroyed in the source JVM and its associated thread added back to the thread pool, for a possible future reuse. Nevertheless, the *dispose* message can be explicitly intercepted by the programmer so that the aglet can continue executing, thus realizing a form of “agent cloning”.

Referring to the code example of Figure 1, the adoption of strong thread mobility overtakes the mentioned drawback, since the code restarts at the destination machine from the same point it stopped at the source one. Thus the code shown in Figure 1 becomes the one of Figure 4.

```
public class MyAgent extends Aglet{
    public void run(){
        // things to do before the migration
        try{
            migrate(new URL("atp://nexthost.unimore.it");
        }catch(Exception e){ ... }
        // things to do after migration
    }
}
```

Figure 4. An example of Aglet code using our approach.

As readers can see, the code is simpler (no flags and branches are required) and shorter than the previous one.

V. PERFORMANCE AND OPEN ISSUES

At the current stage of our research, the thread serialization mechanism, integrated into the Aglets framework, has been

successfully tested, focusing mainly on the state capturing and restoring of the threads executing the aglet.

First of all, we made some first performance tests to discover possible bottlenecks and evaluate the cost of each migration phase. The times measured are expressed in seconds and are average values computed across multiple runs, on a Pentium IV 3.4Ghz with 1GB RAM on `JikesRVM` release 2.4.1. We tested the serialization with increasing stack sizes (5, 15 and 25 frames) and found a very graceful time degradation. These times are conceptually divided into two tables, where Table 1 refers to the thread serialization process, while Table 2 refers to the symmetrical de-serialization process at the arrival host.

	5 frames	15 frames	25 frames
Frame extraction	1.78E-5	1.89E-5	1.96E-5
State building	3.44E-5	3.75E-5	3.43E-5
Pure serialization	2.49E-3	7.32E-3	1.50E-2
Overall times	2.54E-3	7.38E-3	1.51E-2

Table 1. Evaluated times for thread serialization (sec.)

	5 frames	15 frames	25 frames
Pure deserialization	4.46E-3	5.33E-3	7.06E-3
State rebuilding	5.45E-4	5.27E-4	5.06E-4
Stack installation	1.53E-3	1.60E-3	1.71E-3
Overall times	6.54E-3	7.46E-3	9.28E-3

Table 2. Evaluated times for thread rebuilding (sec.)

Considering how these times are partitioned among the different phases of each process, we can see that the bulk of the time is wasted in the pure Java serialization of the captured state, while the extraction mechanism (i.e. the core of the entire facility) has very short times instead. The same bottleneck due the Java serialization may be observed in the de-serialization of the thread. In the latter case, however, we have an additional overhead in the stack installation phase, since the system has often to create a new thread and compile the methods for the injected frames.

VI. CONCLUSIONS AND FUTURE WORK

This paper has introduced our approach to support Java thread strong mobility based on the IBM `JikesRVM` virtual machine, and has outlined how this mechanism is being integrated in the Aglets Mobile Agent Platform in order to exploit such approach. Thanks to the support to thread serialization, agents will be simpler in terms of code, and, at the same time, the code will be easier to be read since a single execution flow will be followed from the beginning to the end.

Our approach represents an extension of `JikesRVM` but does not change any part of this JVM. Rather, it exploits some

interesting facilities provided by that JVM to avoid many of the drawbacks of the presented solutions. OSR facility also allowed us to capture the state in a very portable (i.e. bytecode-level) format. Thanks to the scheduling policy of the JikesRVM, which enables the support of thousands of Java threads, our approach will keep the thread management efficient, and allows having one thread for each agent, overcoming the limitation of the current implementation of the Aglets system.

With regard to future work, we will perform a comparison test between the current Aglets release (with weak mobility) and our JikesRVM-based version (with strong mobility). This comparison will be performed also under critical conditions (such as a large number of agents). From the first results reported in section V, we can draw the conclusion that the prototype can be further optimized with respect to the Java serialization bottleneck, in particular trying to reduce the size of the thread state data to be serialized. This perhaps will allow us to reduce strongly the unavoidable gap between a weak agent serialization and a strong one.

ACKNOWLEDGMENT

Work supported by the Italian MIUR and CNR within the project "IS-MANET, Infrastructures for Mobile ad-hoc Networks" and by the European Community within the project "CASCADAS".

REFERENCES

- [1] ECMA TC39/TG3. The CLI Architecture. Technical Report, ECMA, October 2001.
- [2] A. Acharya, M. Ranganathan, J. Saltz, "Sumatra: A Language for Resource-aware Mobile Programs". 2nd International Workshop on Mobile Object Systems (MOS'96), Linz, Austria, 1996
- [3] The Aglets Mobile Agent Platform website <http://aglets.sourceforge.net>
- [4] B. Alpern, C.R. Attanasio, D. Grove and others, "The Jalapeno virtual machine", IBM System Journal, Vol. 39, N°1, 2000
- [5] B. Alpern, S. Augart, S.M. Blackburn, M. Butrico, A. Cocchi, P. Cheng, J. Dolby, S. Fink, D. Grove, M. Hind and others, "The Jikes Research Virtual Machine project: Building an open-source research community", IBM Systems Journal, Vol. 44, No. 2, 2005
- [6] B. Alpern, D. Attanasio, J. J. Barton, A. Cocchi, S. F. Hummel, D. Lieber, M. Mergen, T. Ngo, J. Shepherd, S. Smith, "Implementing Jalapeño in Java.", ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '99), Denver, Colorado, November 1, 1999
- [7] Matthew Arnold, Stephen Fink, David Grove, Michael Hind, and Peter F. Sweeney, "Adaptive Optimization in the Jalapeño JVM", ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2000), Minneapolis, Minnesota, October 15-19, 2000
- [8] L. Bettini and R. De Nicola, "Translating Strong Mobility into Weak Mobility", MA2001, pages 182-197, number 2240, Springer, 2001.
- [9] S. Bouchenak, D. Hagimont, S. Krakowiak, N. De Palma and F. Boyer, "Experiences Implementing Efficient Java Thread Serialization, Mobility and Persistence", I.N.R.I.A., Research report n°4662, December 2002
- [10] S. Bouchenak, D. Hagimont, "Pickling Threads State in the Java System", Technology of Object-Oriented Languages and Systems Europe (TOOLS Europe'2000) Mont-Saint-Michel/Saint-Malo, France, Jun. 2000
- [11] G. Burke, J. Choi, S. Fink, D. Grove, M. Hind, V. Sarkar, M.J. Serrano, V.C. Sreedhar, H. Srinivasan, "The Jalapeno Dynamic Optimizing Compiler for Java", ACM Java Grande Conference, June 1999
- [12] Stephen Fink, and Feng Qian, "Design, Implementation and Evaluation of Adaptive Recompilation with On-Stack Replacement", International Symposium on Code Generation and Optimization San Francisco, California, March 2003
- [13] M. 13chetti, "Tireless travel agent Special Report: The Rise Of E-Business/Wheeling And Dealing", available at http://domino.research.ibm.com/comm/wwwr_thinkresearch.nsf/pages/travel199.html
- [14] A. Fuggetta, G. P. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol 24, 1998
- [15] J. Gosling, B. Joy, G. Steele, G. Bracha, "The Java Language Specification, second edition", SUN Microsystem
- [16] M. Hohlfeld and B.S. Yee, "How to Migrate Agents", Unpublished, available at <http://www.cse.ucsd.edu/~bsy/>, 1998.
- [17] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, "JADE - A White Paper", EXP in Search of Innovation, TILAB, vol. 3, 2003
- [18] N. R. Jennings, "An agent-based approach for building complex software systems", Communications of the ACM, Vol. 44, No. 4, pp. 35-41 (2001)
- [19] The JikesRVM project site: <http://jikesrvm.sourceforge.net>
- [20] D. B. Lange, M. Oshima, G. Karjoth, K. Kosaka, "Aglets: Programming Mobile Agents in Java", in the Proceedings of the International Conference on Worldwide Computing and Its Applications (WWCA), 1997
- [21] D. B. Lange, Y. Aridor, "Agent Transfer Protocol (ATP)", IBM=TRL, draft number 4, 19 March 1997
- [22] T. Lindholm, F. Yellin, "The Java Virtual Machine Specification, second edition", SUN Microsystem
- [23] M. 23, P. McBurney, C. Preist, "Agent Technology: Enabling Next Generation Computing – A Roadmap for Agent Based Computing", AgentLink, <http://www.agentlink.org/roadmap>
- [24] Scott Oaks and Henry Wong, "Java Threads, 2nd edition", Oreilly, 1999
- [25] The 25 Project web site: <http://25.sourceforge.net/>
- [26] T. Sakamoto, T. Sekiguchi, A. Yonezawa, "A bytecode transformation for Portable Thread Migration in Java", 4th International Symposium on Mobile Agents 2000 (MA'2000), Zurich, Sep. 2000.

- [27] "The Java Object Serialization Specification", Sun Microsystems, 1997
- [28] D. Sislak, M. Rollo, M. Pechoucek, "A-globe: Agent Platform with Inaccessibility and Mobility Support", in Cooperative Information Agents VIII , n. 3191, Springer-Verlag Heidelberg, 2004
- [29] T. Suezawa, "Persistent Execution State of a Java Virtual Machine", ACM Java Grande 2000 Conference, San Francisco, CA, USA, Jun. 2000
- [30] E. Truyen, B. Robben, B. Vanhaute, T. Coninx, W. Joosen, P. Verbaeten, "Portable support for Transparent Thread Migration in Java" 4th International Symposium on Mobile Agents 2000 (MA'2000), Zurich, Switzerland, Sep. 2000
- [31] Paul R. Wilson, "Uniprocessor Garbage Collector Techniques", in the Proceedings of the International Workshop on Memory Management (IWMM92), St. Malo, France, September 1992
- [32] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS distributed operating system", In Proceeding of the Ninth Symposium on Operating Systems Principles, pages 49-70, ACM 1983.
- [33] W. Zhu, C. Wang, F. C. M. Lau, "JESSICA2: A Distributed Java Virtual Machine with Transparent Thread Migration Support". IEEE Fourth International Conference on Cluster Computing, Chicago, USA, September 2002
- [34] G. Vigna, G. Cugola, C. Grezzi and G.P. Picco, "Analyzing Mobile Code Languages", Mobile Object Systems n. 1222, Springer, 1997.
- [35] C. Chambers, "The Design and Implementation of the Self Compiler, an Optimizing Compiler for Object-Oriented Programming Languages", PhD thesis, Stanford University, Mar. 1992. Published as technical report STAN-CS-92-1420.

Secure, Trusted and Privacy-aware Interactions in Large-Scale Multiagent Systems

Federico Bergenti

Dipartimento di Ingegneria dell'Informazione

Università degli Studi di Parma

Parco Area delle Scienze 181/A, 43100 Parma, Italy

Email: bergenti@ce.unipr.it

Abstract—One of the inherent problems of large-scale, open multiagent systems is the lack of mechanisms and tools to guarantee legally valid interactions. Agents are supposed to perform crucial tasks autonomously and on behalf of humans; however, (i) they are not legal persons on their own, and (ii) of a full legal corpus for the virtual world and its inhabitants is yet to come. Therefore, the ultimate responsible for the actions of an agent is its developer. In this paper we address an innovative model of interaction between agents that leads to an increase of the level of security and trust in privacy-aware, interaction-intensive multiagent systems. In particular, after a brief introduction, we focus in Section II on some common problems related to trust and security in real-world, liable interactions. In Section III, we address these problems and outline some abstractions that we use to guarantee a sound level of security and privacy-awareness in interactions with third-party (possibly unknown) agents, whether human or not. Then, in Section IV we describe the design of an API that we implemented to provide developers with a general-purpose, reusable means to realize secure, trusted and privacy-aware multiagent systems. To conclude, in Section V we briefly discuss our model and outline directions of future development.

I. INTRODUCTION

Agent technology is quickly evolving towards the realization of complex societies of agents. Just to cite one recent example, the aims and scope of the IST project CASCOS [4] show how agents are becoming more and more relevant in important sectors, e.g., healthcare and personal data management. This evolution is not yet matched by an equivalent legal development. The lack of a legal substrate capable of grounding the interactions between agents ultimately means that every aspect of interactions (e.g., see [11]) with other (possibly unknown) third-party agents must be explicitly treated by the developer. Moreover, for a legal point of view, the developer is the ultimate responsible for the actions of its agents. This situation is then exacerbated by the impossibility of tracing all actions agents perform: if we cannot guarantee traceability [19] of the actions of individual agents, no law would be sufficient to prevent and punish mendacious agents (whether human or not). Obviously, traceability does not guarantee that agents could not misbehave; anyway, if they do so, other agents would have the possibility of demonstrating the misbehaviour.

Having said this, the ultimate goal of our work is to provide mechanisms and tools to support agents in interacting:

- 1) In a secure, traceable and privacy-aware way; and

- 2) With guarantees of a desired level of security and trust, exploiting the minimum possible number of trusted parties.

Our study of these issues is concretized in the realization of a model capable of representing a secure, trusted and privacy-aware interaction between two agents. The generalization of this model to multi-party interactions is quite straightforward, but its exhaustive description is out of the scope of this paper.

Our work is based on the introduction of two closely-related abstractions, *Validation-Oriented Ontologies (VOOs)* and *Guarantors*. A VOO is a signed set containing an ontology [1] and all runtime tools needed to assert that a particular individual of the world actually belongs to a certain family of individuals. Guarantors are agents that, in some sense, play the role of middleman in interactions. Guarantors are trusted by all interacting parties and they are in charge of supporting interactions by providing (under their responsibility) all necessary VOOs.

This paper is organized as follows: next section describes some crosscutting problems of two-party interactions and it motivates why we need something more than available techniques and tools to guarantee security, trust and privacy in multiagent systems. Section III briefly describes our model and introduces the notions of VOOs and Guarantors. Section IV shows how we support our model and its new abstractions by describing an API that we realized to support developers in their everyday work. Finally, Section V briefly discusses our model to point some interesting direction of development and to show its wide applicability in real-world scenarios.

II. PROBLEMS IN TWO-PARTY INTERACTIONS

The first assumption that we take in the discussion of our model is that, from the point of view of security, trust and privacy, we can always reduce any two-party interaction to the act of signing of a contract. Then, we assume that proposals and agreements between interacting parties are exchanged in the form of individuals of known ontologies. This assumption allows agents to manage the information contained in proposals and agreements in a friendly way, e.g., to reason about proposals and to assert the formal validity of proposals against the constraints of the ontology.

All in all, the assumption of modelling interactions as contracts that are individuals of known ontologies is ab-

solutely general and has some remarkable advantages. The most interesting advantage that we see is the possibility of combining simple ontologies into complex models of proposals and agreements. We can compose simple ontologies into complex descriptions of proposals and agreements, thus avoiding duplication of definitions and possible ambiguities.

The second advantage that we see in our working assumption is that it greatly simplifies the creation and validation of proposals and agreements. The creation of a proposal is reduced to the creation of one or more individuals of known ontologies, with properties set accordingly to given values (potentially specified in external policies). Controlling the suitability of a proposal simply reduces to checking whether a candidate proposal actually belongs to the family of admissible proposals described in the referenced ontology.

Finally, ontologies expressed in common formats are easily mapped into human readable documents for a subsequent inspection of the agreements that software agents may have autonomously signed.

A. Problem 1. Trusting Ontologies

Ontologies seem to be a suitable means for describing agreements, but any attempt to use them in real-world scenarios immediately encounters a problem: How an agent could trust a new ontology? Suppose that a seller of bandwidth requires to negotiate agreements with potential customers using an ontology available in some public repository. This ontology may model some property as being “required by local laws.” How could customers trust this requirement if they have no trust relationship with the seller that pointed it to this ontology? Could a customer (in some sense) validate the ontology to decide whether to trust it or not?

Another facet of this problem occurs in the case of an ontology that is partially non-disclosed. Let us suppose that the aforementioned seller creates its ontology and splits it into two parts: a public part describing valid proposals and agreements, and a private part used to model the policies that it employs to enforce bandwidth reservations, i.e., the policies that it uses to reason on proposals. This last part contains background knowledge on the marketing strategies of the seller and it is vital not to disclose this knowledge to potential competitors. In this case, a full fledged reasoning on the ontology could be done only by accessing the whole ontology, and only partial reasoning is possible for customers.

Moreover, we have to take into account a third (very serious) facet of this problem: there is no way to validate the adherence of the ontology to real-world laws, without involving highly specialized jurists. Obviously, no potential customer would be in the position of performing this sort of validation.

In the end, all these exemplified facts of the same problem, i.e., trusting the ontology, show that trust cannot be given to an ontology per se: it must be accorded to its signer. Ontologies used to model formal agreements and contracts must be provided by trusted and liable signers.

B. Problem 2. Trusting Identities

The ultimate aim of our model of interaction is to guarantee legal validity. Therefore, the problem of checking the identities of involved agents is obviously critical. Unfortunately, a simple static control of identities by means of certificates [6], [7] is inadequate because, e.g., certificates can be revoked or keys can be stolen. This inadequacy should not be surprising because it is very common also in human interactions. The identification of agents in a secure, trusted and privacy-aware multiagent system can be performed only through a set of runtime tools capable of validating certificates, and thus realizing a trusted source of identification.

The problem of checking identities is closely related to the representation of identities. The identification code is the only means that we have to validate the identity of a legal person (physical or not). Therefore, one of the very basic issues that we have to tackle is how to represent identities in an agent-processable way. In our model, we decided to design an ontology describing legal persons and their attributes and to associate this ontology with a set of general-purpose tools for addressing the majority of problems related to identification. The connection between this ontology and its tools is reinforced by the necessity of a common trusted signer.

It is worth noting that in order to fully exploit the possibility of having runtime tools capable of providing some sort of guarantees regarding sensible tasks on an ontology, both the ontology and its associated tools must have the same levels of trust and security. Let us consider these two examples to clarify this point.

1) *Case 1. Trusted ontology - Untrusted tools:* Suppose that two negotiating agents trust the same ontology (i.e., they trust the publisher of the ontology) but they use untrusted services to perform validations. They exchange proposals until an agreement is reached and they mutually check their identities using an untrusted tool. Since they do not trust the identity verification tool, they can both suppose that they are signing an agreement with an unknown party.

2) *Case 2. Untrusted ontology - Trusted tool:* Suppose that we have an identity-verification tool that receives in input an ontology and an identity, and verifies the identity in a database. What happens if someone gives formally valid (compliant with the ontology), but legally void identity? Since the given identity matches the record in the database of identities, the tool would return an affirmative answer, but this identity is legally void and therefore unusable in signing formal agreements.

These two examples show that both ontology and runtime tools must be trusted and secure. If any of the two has not a suitable level of trust and security, the combined use of them will not result in a secure and trusted interaction.

III. TRUST, CONTRACTS AND GUARANTORS

The analysis of the two-party interaction outlined in the previous section allows to introduce two abstractions that

we can generally use to model secure, trusted and privacy-aware interaction between two agents. These closely-related abstractions, namely Validation-Oriented Ontologies (VOOs) and Guarantors, are briefly described in this section.

The problem of defining trust has been addressed in many different ways [15]. While we recognize the importance of cognitive models [5] to quantify trust, we start from the definition given in [9] to provide a probabilistic interpretation of trust. In particular, if “*Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends*”, it is reasonable to model trust in terms of an estimation of the real probability by which B would perform the target action. Many factors contribute to this estimation [11], [13]; nonetheless we use a blackbox approach, in which trust is modelled as a random variable in an interval $[p_{a,x}, p_{b,x}]$. The only assumption that we take is that such an estimation is a reasonable approximation of the real value of the quantity.

Our probabilistic model of trust is out of the scope of this paper, and we simply enumerate the quantities that we exploit in our treatment:

- 1) $p_{k,x}$, the probability that the information k provided by agent X is correct;
- 2) $p_{c,x}$, the probability that agent X would adhere to all the obligations stated in contract c ;
- 3) $t_{c,x,y}$, the level of trust agent X has in agent Y with respect to contract c ;
- 4) $p_{a,x}$, the minimum value of trust in an estimation, i.e., the lower bound of the probability distribution function of trust;
- 5) $p_{b,x}$, the maximum level of trust in an estimation, i.e., the upper bound of the probability distribution function of trust.

Since trust expresses the estimate of a probability, it is clear that p_a and p_b are both between zero and one. The assumption that $p_b \geq p_a$ is not restrictive.

As stated in the previous section, we assume that all interactions between two agents can be reduced, from the point of view of trust and privacy, to the action of signing a contract. While it is reasonable to think of a number of different (and very complex) contracts [2], we adopt a very simple contract model. It involves only two signers, and it is totally described by two triples, one known to each signer. Each triple, that we call *subjective evaluation* of the contract, is made of a *reward*, an *investment* and a *penalty*. This triple summarizes the contract, and its effects, for the agent to which the triple belongs. Being subjective values, it is not possible to assess any mathematical relation between values of two different subjective evaluations, even though they refer to the same contract.

The subjective evaluation of contract c given by agent X is written as follows:

- 1) $R_{c,x}$ is the reward that agent X receives upon success of contract c ;
- 2) $I_{c,x}$ indicates the investment that agent X makes in

contract c ; i.e., a certain granted value that it renounces to, when signing contract c ;

- 3) $P_{c,x}$ is the penalty of the contract, i.e., the value that agent X receives if the contract fails because of the other party.

The contract gives to its signers the absolute security of receiving the stated values, i.e.:

- 1) if the contract is respected, agent X receives $R_{c,x}$ with probability one;
- 2) if the contract fails because of agent Y , agent X receives $P_{c,x}$ with probability one.

Another assumption that we take concerns the order between reward, investment and penalty in a subjective evaluation. We are interested in contracts whose parameters are ordered as follows:

$$P_{c,x} \leq I_{c,x} \leq R_{c,x} \quad (1)$$

This inequality expresses the fact that contracts are advantageous but risky. This, in turn, implies that an agent signs a contract in the hope that it would be respected by the other signer, since in case of failure it would experience the following loss:

$$I_{c,x} - P_{c,x} \quad (2)$$

Furthermore, each agent does not consider its own failure probability, since it will only consider contracts that it can reasonably respect; nevertheless the uncertainty about the other signer remains.

Taking this probabilistic model that we briefly described here, and that is subject for an in-depth investigation in a future paper, we can provide a probabilistic description of a two-party interaction. From the point of view of security, trust and privacy, such an interaction can take advantage of the presence of a Guarantor that plays (in some sense) the role of middleman in the interaction. In order to provide a synthetic description of the aims and scope of the notion of Guarantor, we need to introduce another accessory abstraction, namely Validation-Oriented Ontology.

A *Validation-Oriented Ontology* (VOO) is a signed set containing:

- 1) An ontology that models a domain;
- 2) A set of runtime tools capable of asserting properties of individuals of this ontology.

Runtime tools are intended to provide a means for validating assertions on the domain described by the ontology without requiring a full-fledged reasoning on the domain. As we have seen in the previous section, this is essential from the point of view of security and trust for real-world applications.

One very important advantage of the introduction of VOOs is that they reduce the amount of distributed trust, since in a single signed object lay both the semantic description of thing and a set of related actions.

Moreover, VOOs promote software reuse and help standardization, since many ontology-related tasks are performed from external bodies (the tools of VOO) in a standard, well-defined, trusted and secure way. It is worth noting that the

concrete technology used to realize the tools of the VOO is not mandatory: they could be Web or Grid services [8], as well as RMI invocations, as long as they are projected and signed together with their ontology. In this way it is possible to achieve platform independence by including in the VOO a description of the invocation procedure of its tools.

VOOs are not sufficient to address all issues related to real-world agreements because we need to trust both the VOO and the signer of the VOO, as discussed in the previous section. In fact, if we go back to the human world, the proper way to stipulate contracts is through a notary public. This happens because only legal person trusted by the State can perform critical tasks (e.g., querying databases containing privacy-critical information). This is the reason why we introduce the abstraction of Guarantor, and we say that an agent is a Guarantor for an interaction between two other agents if it can sign a VOO that the two other agents can use in their interaction.

We can be more precise in this definition by rephrasing the auditing principle of [2] for a validation case:

If Role 1 cannot witness the truthfulness of an assertion about Role 2, another Role 3 should testify the condition of Role 2 if the party playing Role 2 is not trusted by the party playing Role 1. This document must be received by Role 1 before the execution of its primary activity, and the party playing Role 3 should be trusted by the party playing Role 1.

According to this principle, we suppose that both agents involved in a two-party interaction trust a common agent, playing Role 3, that we call Guarantor. This agent is supposed to be responsible for the exactness of the information provided by itself and by its tools. Unlike other agents, the Guarantor of the interaction can easily check ontologies, tools and other Guarantors, to provide tools that can operate on other Guarantors' certificates, ontologies, etc. Therefore, the introduction of the Guarantor allows agents to put their trust in a single entity, thus simplifying greatly the decisions related to according or revoking trust.

In summary, in our model the Guarantor is responsible for the following tasks:

- 1) Provide identity certificates;
- 2) Provide signed ontologies compliant with real-world laws;
- 3) Provide signed runtime tools for its ontologies and/or certifying external tools under its responsibility.

Then, if we remember that identity certificates are provided as signed instances of concepts of an ontology, and if we go back to the previous definition of VOO, these three responsibilities of the Guarantor can reduce to a single responsibility: *provide VOOs*.

The Guarantor takes the responsibility of catalyzing the trust of an interaction in various ways, e.g., through:

- 1) A signed list of trusted tools;

- 2) A certified public key whose private key is provided only to trusted tools;
- 3) A certified set of services that could access the Guarantor's database and whose use could be detected by the tools' user.

IV. AN API FOR SECURE, TRUSTED AND PRIVACY-AWARE COMMUNICATIONS

The abstractions of VOOs and Guarantors must be adequately supported by some development tools in order to implement them correctly in real-world MASs. This is the reason why we developed an API for JADE capable of providing a direct support to developers in the realization of secure and trusted MASs.

The API we developed focuses primary on the double-Guarantor model, because it is general enough to subsume the single-Guarantor model, but it is simple enough to allow an in depth evaluation and study. Furthermore the double-Guarantor model is probably the most frequent case.

We designed our API to match a set of fundamental requirements, that resulted in strict development guidelines.

- 1) Privacy. All communications must be encrypted and directed to trusted parties.
- 2) Traceability and security. Invocations involving tools of VOOs must be signed by the caller, while responses from such tools must be signed, directly or indirectly, by a Guarantor. The API transparently checks this property and provides a transparent tracing service that logs all invocations and responses.
- 3) Locality. The number of trusted parties involved in any interaction must be kept at minimum. This means that an operation performed by a given Guarantor in a mutual recognition case, must be delegated upwards and not delegated immediately to the other Guarantor's tools.
- 4) Transparency. The invocation procedures of VOO tools must be transparent to the user, i.e., the user is not directly involved in the use of the tools that the VOO provides.
- 5) Ease of use. The API must provide high level procedures to perform common tasks, as well as low level, more specific procedures devoted to fine-grained (and less common) tasks.
- 6) Standardization. Information exchange, including certificates and proposals, must be performed using well-known formats.

These guidelines are completed with the following use cases and result in a first set of requirements that we used in the realization of our API.

The design of the API is split into two views: (i) a client view that shows the classes that a client agent can use to access the services of the security and privacy subsystem, and (ii) a Guarantor view that describes the components that the Guarantors use to implement their functionality. Such views are connected through the *Guarantor* interface, that plays the logical role of a remote interface that Guarantors implement and that client exploit by means of proxies.

```

public interface Guarantor {
    public SessionToken signOn(Credentials c) throws SignOnFailed;
    public boolean signOff(SessionToken st); /* true on successful sing-offs */

    public Object directInvocation(
        DistributedTimeStamp dts,
        ServiceDescriptor sd,
        Object[] parameters
    );

    public DelegationToken createDelegationToken(
        ServiceDescriptor sd,
        DelegationDescriptor dd
    );

    public Object indirectInvocation(
        DistributedTimeStamp dts,
        DelegationToken dt,
        Object[] parameters
    );
}

public interface ServiceDelegationDescriptor {
    public Certificate delegator();
    public Certificate delegated();

    public long getNumberOfInvocations(); /* max number of invocations */
    public long getDeadline(); /* in millis from generation time */
}

public interface Token {
    public String getCanonicalString(); /* UUencoded */
    public long getExpirationDate(); /* in millis from generation time */
}

public interface SessionToken extends Token {}

public interface DelegationToken extends Token {}

```

Fig. 1. Client view of the API

It is worth noting that the client view represents a mandatory interface, on the contrary, the Guarantor view is only a suggestion of a possible internal design of Guarantors. Obviously, client view plays a substantially more important role in this design.

A. Client View

For the sake of clarity and readability, Figure 1 collects the interfaces of the client view in terms of Java interfaces.

The central interface of the client view is Guarantor. It encapsulates all methods that Guarantors expose to clients. Such methods are accessed remotely through an encrypted channel and they are available after an initial mutual recognition phase through the *signOn()* method. Client wishing to use the services of a Guarantor, invoke this method and pass their credentials. The type of requested credentials depends on the Guarantor: simple username/password may be sufficient in certain cases, or more complex X.509 certificates may be needed in other cases. Guarantors will provide their specific subclass of interface Credentials to have clients provide the required information.

If the Guarantor intends to serve the client that issued the *signOn()* request, it will respond with a *SessionToken* that the client will use for subsequent invocations on the Guarantors interface. A *SessionToken* is a particular sort of *Token*. Just like all tokens, it has an expiration date and it can be converted in a canonical string.

signOn() requests are invoked on some kind of secure channel, e.g., HTTPS, and subsequent services are requested on the same secure channel. A client can issue request for services until the client itself signs off (through the *signOff()* method) or until the session expires.

Once a client is authenticated with a Guarantor, it can perform two kinds of requests:

- 1) Direct requests, i.e., requests for services whose outcome is used by the client itself;
- 2) Indirect requests, i.e., requests that are performed on behalf of some other client.

Direct requests are performed simply through the *directInvocation()* method. These are ordinary requests for services except for the following two constraints:

- 1) Parameters and result value are transported on a secure channel;
- 2) The Guarantor is responsible for tracing the request to guarantee non-repudiability;
- 3) The client is responsible for providing a distributed timestamp to allow for traceability of complex interactions.

The *directInvocation()* method is the only mechanism that Guarantors offer to have services performed in this way. Other methods of the Guarantor interface (or of any of its subclasses) are not guaranteed to respect the aforementioned constraints. All in all, the *directInvocation()* method plays the role of the Dynamic Invocation Interface of CORBA objects, or of the *Method.invoke()* method of Java reflection. It is the preferred way to handle secure services. Indirect requests are a delegation mechanisms that allows a client (A, delegated) to have a service performed on behalf of another client (B, delegator). This process is made of the following steps:

- 1) Client A requests the Guarantor to grant indirect requests to client B;
- 2) If the Guarantor can honour this request, it will accept requests from B and serve them as if they were requested by A;
- 3) The Guarantor stops serving indirect requests from B if the delegation has expired, e.g., because the maximum number of requests from B is reached.

The first step of this process is performed when A invokes the *createDelegationToken()* method on the Guarantor interface. This method needs the following parameters:

- 1) A ServiceDescriptor that identifies which service(s) of the Guarantor the client is willing to delegate;
- 2) A DelegationDescriptor that provides the Guarantor with all information needed to actually perform the delegation, e.g., who is the delegated client, for how long the delegation will last.

If the Guarantor can grant the delegation of the service to B, the return value of *createDelegationToken()* is a globally unique token that identifies the delegation. The delegated client B will use this token to finally access the services of the Guarantor through a call to *indirectInvocation()*. This method has exactly the same meaning and constraints of the *directInvocation()* method, except for the fact that it can be invoked by delegated clients that are not currently authenticated with the Guarantor.

If a Guarantor needs additional, application-specific information, to grant indirect requests, it can provide its own sub-


```

public interface SensitiveDataStore {
    public SessionToken signOn(Credential c) throws SignOnFailedException;
    public boolean signOff(SessionToken st); /* true on successful sing-offs */

    public ResultSet query(SessionToken st, QueryStatement q) /* result set */
        throws IllegalSessionToken, IllegalStatement;
    public long insert(SessionToken st, InsertStatement o) /* number of additions */
        throws IllegalSessionToken, IllegalStatement;
    public long update(SessionToken st, UpdateStatement o) /* number of updates */
        throws IllegalSessionToken, IllegalStatement;
    public long delete(SessionToken st, DeleteStatement o) /* number of deletions */
        throws IllegalSessionToken, IllegalStatement;
    public boolean create(SessionToken st, CreateStatement o) /* creations */
        throws IllegalSessionToken, IllegalStatement;

    public void setUsernameForNotifications(SessionToken st, String username)
        throws IllegalSessionToken, IllegalUsername;
}

public interface InvocationTracer {
    public void traceIndirectInvocation(
        DistributedTimeStamp dts,
        DelegationToken dt,
        Object[] parameters,
        Object result
    );

    public void traceDirectInvocation(
        DistributedTimeStamp dts,
        ServiceDescriptor sd,
        Object[] parameters,
        Object result
    );
}

```

Fig. 2. Guarantor view of the API

classes of classes *ServiceDescriptor* and *DelegationDescriptor* interfaces.

Indirect requests is the preferred way to allow a third party having a service done without explicitly requesting sensitive data. For example, let's consider a buyer A and a seller B. Normally, the seller will request the details of As credit card in order to:

- 1) Check the validity of the credit card;
- 2) Perform the withdrawal of the exact amount of the requested payment.

B would be able to perform exactly such operations if buyer A would instruct its bank (the Guarantor) to serve this two requests from B as if they were issued by A itself. This approach has the great advantage of allowing A to buy from B without revealing any sensitive information. The delegation token that allows B to perform the withdrawal is a sort of anonymized view of the sensitive data of A. Formally, this delegation token is a *one-time password* for logical access control.

B. Guarantor View

The Guarantor view of the architecture describes how a Guarantor may implement a general-purpose infrastructure for providing its services with requested level of security and privacy. This architecture is not mandatory because every Guarantor may decide its own optimized approach to provide services. Anyway, the quality of Guarantors in performing tasks related to security and privacy, e.g., the global uniqueness of the generated tokens, or the correct tracing of invocations, are important metrics for clients to put trust of Guarantors. Therefore, the Guarantor view is highly recommended as it helps clients estimating the reputation of Guarantors. Figure 2 shows the Java interfaces that make the Guarantor view of the architecture.

One of the principal interfaces that build the Guarantor view of the architecture is *SensitiveDataStore*. This is an abstract view of a data store that is meant to allow for a seamless treatment of sensitive data. It is worth remembering that every Nation in the European Community adopted laws to provide guarantees to citizens regarding the treatment of their sensitive data.

Such laws are all rooted in a note of the European Commission and they all contain strict technical requirements that databases of sensitive data must follow. As an example, the following are examples of the requirements of the Italian law on privacy:

- 1) The password of the manager of the data store must be of 8 alphanumeric characters, at least;
- 2) The password of the manager of the data store must be changed every 3 months;
- 3) If any access credential to the data store has not been used for more than 3 months, it must be revoked.

Any implementation of the *SensitiveDataStore* interface will wrap existing technologies for storing data, e.g., JDBC or JNDI, and it will add the support for any requirement to make it compliant with a particular legislation (at a particular time). Any *SensitiveDataStore* allows a direct management of the data it contains thought methods: query, insert, update, delete and create. These are wrapper to the underlying storing technology and their statement parameters are concrete subclasses that provide all necessary information to concretely perform requested operations. Not all such methods are always permitted to allow for accommodating different levels of management of the data store, e.g., query may be always possible, but creation and deletion of database table is possible only when the Guarantor is not online. Any attempt of violating such application-specific constraints will generate an *InvalidStatement* exception.

Methods *setUserNameForNotifications()* is used to instruct the data store to actively provide information on compliancy. For example, a particular data store may decide to notify the manager of any credential that no longer complies with laws, or it may decide to notify the administrator of any clean-up of old data. This unusual behaviour of a *SensitiveDataStore* is needed to allow data stores suspending operations on a session when, for some reason, the session does no longer comply with laws. So, for example, a properly implemented sensitive data store would stop functioning when the password of the person in charge is more than 3 months old.

The second interface that may be used to design Guarantors is *InvocationTracer*. This interface provides all methods for tracing direct and indirect requests served or rejected by the Guarantor. Such requests are stored in a *SensitiveDataStore* that would save all (context) information regarding requests. It is worth noting that the invocation *DistributedTimeStamp* property allow to correlate logs of different Guarantors, and therefore it allows backward tracing the execution of complex actions.

V. DISCUSSION

The central focus of this paper is on the motivated introduction of two abstractions, VOOs and Guarantors, that we can use to provide general-purpose mechanisms to realize secure and trusted MASs. The need of these abstractions should be clear if we go back to the very general issues related to security and trust that we identified for two-party interactions. Obviously, the introduction of these abstractions is not the only way we can think to tackle such issues, but we believe that our approach has two interesting properties:

- 1) Concentrated trust. Guarantors are sorts of trust catalysts that we use to keep trust concentrated on the minimum number of parties. From the point of view of interacting agents, this is good because the number of operations related to according or revoking trust is minimized.
- 2) Pragmatic interactions. The strict coupling between an ontology and a set of tools capable of performing general-purpose, critical tasks on the individuals of this ontology (i.e., the idea of VOO) guarantees the possibility of performing secure and trusted interactions also to agents with minimal reasoning capabilities.

In conclusion, we believe that the introduction of VOOs and Guarantors provides a solid ground for the concrete development of trusted and secure MASs. Many issues related to these properties are encapsulated by these abstractions and we believe that their in-depth study can lead to a better understanding of the subtle behaviours of these complex systems in real-world situations.

ACKNOWLEDGEMENTS

This work is partially supported by project CASCOM (FP6-2003-IST-2/511632). The CASCOM consortium is formed by DFKI (Germany), TeliaSonera AB (Sweden), EPFL (Switzerland), ADETTI (Portugal), URJC (Spain), EMA (Finland), UMIT (Austria), and FRAMeTech (Italy). This article reports on joint work that is being realised by the consortium. The authors would like to thank all partners for their contributions.

REFERENCES

- [1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P.F., (Eds.), *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, 2003.
- [2] Bons, R.W.H., *Designing Trustworthy Trade Procedures for Open Electronic Commerce*, Ph.D. Dissertation, 1997, EURIDIS and Faculty of Business Administration, Erasmus University.
- [3] Casati, F., Shan, E., Dayal, U., and Shan, M.-C., *Service-Oriented Computing: Business-Oriented Management of Web Services*. Communications of the ACM, 46:10, October 2003.
- [4] CASCOM Web site <http://www.ist-cascom.org>
- [5] Castelfranchi, C., and Falcone, R. Principles of Trust for MAS: Cognitive Anatomy, Social Importance, and Quantification. In *Proceedings of The International Conference of Multi-agent Systems (ICMAS)*, 72–79, 1998.
- [6] Ellison, C., *SPKI Requirements*. IETF RFC 2692, September 1999.
- [7] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T., *SPKI Certificate Theory*. IETF RFC 2693, September 1999.
- [8] Foster, I., Kesselman, C., and Tuecke, S., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Int'l Journal of Supercomputer Applications, 15(3), 2001.
- [9] Gambetta, D. (Ed.), *Trust: Making and Breaking Co-operative Relations*, Basil Blackwell, Inc., UK, 1985.
- [10] JENA Web site <http://jena.sourceforge.net>
- [11] Jennings, N. R., Parsons, S., Sierra, C. and Faratin, P., *Automated Negotiation*, in *Procs. of the 5th Int'l Conference on the Practical Application of Intelligent Agents and Multi-Agents Systems, PAAM-2000*, Manchester, UK.
- [12] Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, D., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K., *Bringing Semantics to Web Services: The OWL-S Approach*, in *Procs. of the 1st Int'l Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, July 2004, San Diego, USA.
- [13] Marsh, S. *Formalising Trust as a Computational Concept*. Ph.D. diss., Department of Mathematics and Computer Science, University of Stirling, Stirling, UK, 1994.
- [14] Meyer, B., *Object Oriented Software Construction, Second Edition*, Prentice-Hall, NJ, 1997.
- [15] MINDSWAP, *A Definition of Trust for Computing with Social Networks* Technical report, University of Maryland, College Park, February 2005.
- [16] OWL Web site <http://www.w3.org/2004/OWL>
- [17] Poggi, A., Tomaiuolo, M., Vitaglione, G., *Do Agents Need Certificates? Distributed Authorization to Improve JADE Security*, in *Procs. of the 6th Int'l Workshop on Trust, Privacy, Deception, and Fraud in Agent Societies, AAMAS 2003*, July 2003, Melbourne, Australia.
- [18] Racer Web site <http://www.sts.tu-harburg.de/~r.f.moeller/racer/>
- [19] Szomszor, M., and Moreau, L., *Recording and reasoning over data provenance in web and grid services*, in *Procs. of the Int'l Conference on Ontologies, Databases and Applications of Semantics (ODBASE'03)*, LNCS 2888, November 2003, Catania, Italy.

An Implemented Prototype of Bluetooth-based Multi-Agent System

Volha Bryl

Department of Information
and Communication Technology
University of Trento,
via Sommarive 14,
38050 Povo (TN), Italy
Email: volha.bryl@unitn.it

Paolo Giorgini

Department of Information
and Communication Technology
University of Trento,
via Sommarive 14,
38050 Povo (TN), Italy
Email: paolo.giorgini@dit.unitn.it

Stefano Fante

ArsLogica Lab,
IT Laboratories BIC,
Viale Trento, 117,
38017 Mezzolombardo (TN), Italy
Email: stefano.fante@arslogica.it

Abstract—People tend to form social networks within specific geographical areas. This is motivated by the fact that the geographical locality corresponds generally to common interests and opportunities offered by the people active in the area (e.g. students of a university could be interested to buy or sell textbooks adopted for a specific course, to share notes, or just to meet together to play basketball). Cellular phones and more in general mobile devices are currently widely used and represent a big opportunity to support social communities. We present an application of multi-agent systems accessible via mobile devices (cellular phones and PDAs), where Bluetooth technology has been adopted to reflect users locality. We illustrate an implemented prototype of the proposed architecture and we discuss the opportunities offered by the system.

I. INTRODUCTION

Being widespread and ubiquitous, cellular phones are recently used not only as the means of traditional communication. They are also supposed to satisfy the information needs of their users, e.g. to support information search and filtering or electronic data exchange. Users equipped with mobile devices, such as cellular phones or PDAs, can form so called mobile virtual communities [1], which make possible the collaboration and the information exchange between their geographically distributed members. Such communities are inherently open, new users can join and existing ones can leave anytime. Our aim is to build a general architecture for open distributed systems that can facilitate the interaction and the collaboration among members of co-localized groups of users via their mobile devices.

We adopted Bluetooth [2] technology to connect mobile devices to servers where virtual communities based on multi-agent systems are formed and allow users to interact with one another. Bluetooth is a cheap and a widely used wireless communication technology that can connect Bluetooth-enabled devices located in a range of 100 meters.

A number of multi-agent applications to mobile devices environments have been proposed in literature. [3] presents a multi-agent system named KORE where a personal electronic museum guide provides to visitors (with Java-enabled mobile devices) information about artistic objects they are currently looking at. Information is filtered and adapted to the user

profile. Bluetooth technology is used to detect the user position. In [4] MobiAgent is proposed, an agent-based framework that allows users to access various types of services (from Web search to remote applications control) directly using their cellular phones or PDAs. Once the user sends the request for a specific service an agent starts to work on her behalf on a centralized server. The user can disconnect from the network and the agent will continue to work for her. When the request has been processed, the user is informed via Short Message Service and she can decide to reconnect the network to download the results. MIA information system [5] is another example that provides personalized and localized information to users via mobile devices.

What is still missing in the above architectures is the interaction and the collaboration between the members of the virtual community. Just few proposals in the literature introduce domain-specific collaborative environments where interacting and collaborative agents act on the behalf of their users. For instance, [6] describes a context-aware multi-agent system for agenda management where scheduling agents can execute on PCs or PDAs and assist their users in building the meeting agenda by negotiating with the other agents. ADOMO [7] is an agent-based system where agents running on mobile devices sell the space on the device's screen to commercial agents for their advertisements. Agents on behalf of their users negotiate and establish contracts with neighbors via Bluetooth.

There exist a number of multi-agent platforms that can be used on mobile devices. Taking into account the limited computational and memory resources, it could be very problematic to run a multi-agent platform on such mobile devices as cellular phones. A possible solution is either to avoid running multi-agent platform on mobile devices, as for example in [7], or to use portal multi-agent platforms [8] where agents are executed not on the device itself but on the external host.

In this paper we present a general architecture based on this last option. The architecture proposes independent servers where multi-agent platforms can be installed and where agents can act on behalf of their users. Each server proposes one or more specific services related to the geographical area in which it is located (e.g. a server inside the university could

offer the service of selling and buying text books, renting an apartment, etc.) and users can contact their personal agents using their Bluetooth mobile phones. The main advantage of the proposed framework with respect to the above described architectures is that the system is domain independent (it does not depend on the specific services offered by the servers) and independent from the multi-agent technology adopted (we can use different technologies on each server).

The paper is organized as follows. Section II describes a motivating example of our system. The general architecture of the system is introduced in Section III, while Section IV provides some architectural details and describe the implemented prototype. Section V concludes the paper and provides some future work directions.

II. MOTIVATING EXAMPLE

Let's consider three places in a town: university, railway station and bar. People staying for some time in one of these places may have some common interests and needs. For instance, students at the university might want to buy or to sell secondhand textbooks, to find a roommate, or to form study groups. People at the bar could be interested in the latest sport news (especially in Italian bars), or they could just be looking for someone to chat with. Passengers waiting at the railway station may want to know some details about the trip they are going to have — what cities their train goes through, or what the weather is like at the destination point. They may want also to find someone with common interests to chat with during the trip.

Let's suppose also that people cannot or do not want to spend their time on examining announcements on the bulletin boards, or questioning people around them, or searching for the information office. They would prefer to enter the requests they have into their mobile phones and wait for the list of available proposals.

To support interests and needs of such groups of co-localized users a server is placed at each of the three meeting points. Servers can provide a certain number of services to people equipped with mobile phones or pocket computers (hereinafter referred as users). A user can have access to the services when she is close enough (depending on her Bluetooth device) to one of the three servers — at the bar, in the waiting room of the station, or at the main hall of the university.

Let's suppose that among the available services we have the following ones. University server can be used for buying and selling used books, or for looking for a roommate. At the bar sport news service is available, as well as the service which helps to find interesting people around. Railway station server gives a possibility to get information about trips (including touristic information).

Users interaction and collaboration is the base for the satisfaction of their needs. To sell a secondhand textbook, one should find a buyer and agree on the price. To find someone in the bar to chat with, one should look for the person with similar interests. Each server recreates the group of co-localized human users in a virtual community of personal

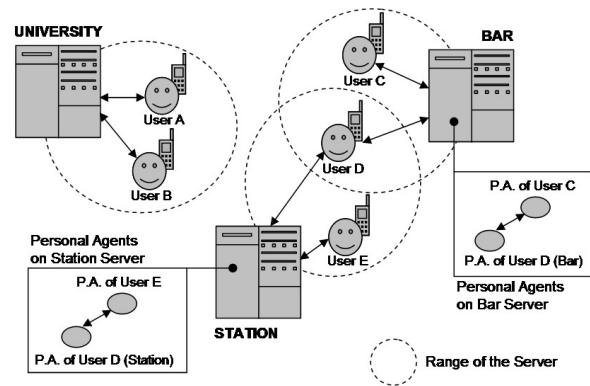


Fig. 1. Users, Servers, Virtual Communities of Personal Agents

agents (Figure 1) able to interact and collaborate with one another. Users formulate their requests and forward them to their personal agents.

Personal agents interacting with the other available agents (they may also negotiate, not just interact, as in the case of selling or buying books) produce results that will be sent back to the users. The main idea is to have a distributed system composed of a number of open virtual communities that evolve and act autonomously on the behalf of human communities.

III. SYSTEM ARCHITECTURE

In this section we describe the general architecture of the system. We start from the requirements and then we illustrate the various sub-components and their interaction.

A. System Requirements

We can summarize the requirements of the whole system in the following objectives.

- Allow the user to express her interests and choose the services she wants to access.
- Provide access to the requested services when the mobile device and the appropriate server are co-localized (i.e. the Bluetooth connection is feasible).
- Allow the user to retrieve pending results. Results should be accessible both in the case the user is still in the Bluetooth range and in the case she is out of the range.

B. System Components

The architecture of the system includes four main types of components: mobile device, PC, server and services database.

The PC component provides an interface for the user's registration to the system, for getting and choosing available services, and building requests for the chosen services. Also the pending results can be retrieved via PC. The mobile device is used to send the user's requests to the servers and to get back the results. Each server within the system provides a list of predefined services. The server runs a multi-agent platform with personal agents representing single users, a database where results are archived, and an interface responsible for establishing connections with mobile devices and PCs, and for

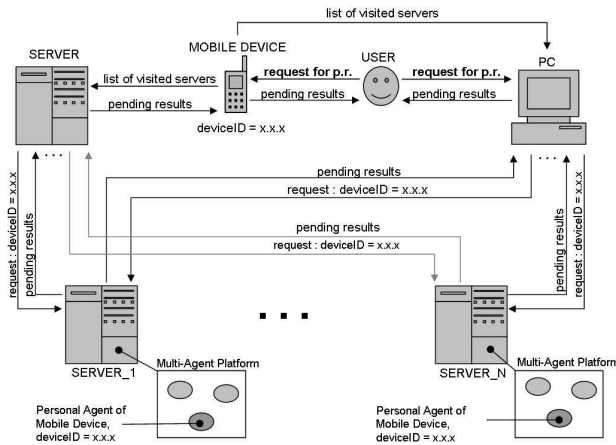


Fig. 4. Retrieving Pending Results

device within the platform. If not, new personal agent is created. Personal agent communicates and collaborates with other agents in order to find "a partner" which will satisfy its request. Interaction protocols and collaboration mechanisms are domain (services) dependent.

IV. IMPLEMENTATION ISSUES

In this section we present the details of the implemented prototype. Basically, the system is a first implementation of the architecture presented in Section II and focuses on a number of servers spread around the university campus (faculties, libraries, departments, etc.). Each server offers only the service for selling and buying books. We are currently working on a number of other services including services available on servers located outside of the university campus (e.g. train station, museums and places close to touristic attractions).

A. On-line registration and services selection

To start working with the system, the user has to register. She can fill the on-line registration form where she needs to put her personal info such as name, birth date, e-mail, Bluetooth address and phone number of her mobile device, and password. The registration, basically, allows the system to identify the user and the mobile device she is going to use. Password is used to access the information about servers and related services and to upload/update the user information (e.g. the user can decide to use different mobile device or just to change her data such as telephone number or e-mail address). All this information is stored in the services database. Registered users obtain the rights to download the software for the PC and the mobile device components (which are two jar files), and the XML file containing all available servers with corresponding services.

After the registration (or login), the user can start selecting services to use. Using the Java GUI interface shown in Figure 5, she can explore all the available services using filtering criteria such as server location (e.g. we can have servers located in different cities or in different places in the same city),

Fig. 5. Request Input Form

```
<server>
  <ip> 192.168.2.151 </ip>
  <name> UnitnServer </name>
  <location> Trento, via Sommarive, Povo </location>
  <service>
    <description> Buy/sell new and used books </description>
    <parameters>
      <param>
        <paramname> Book title </paramname>
        <paramtype> String </paramtype>
        <paramvalue> Lord of the Rings </paramvalue>
      </param>
      <param>
        <paramname> Desired price </paramname>
        <paramtype> int </paramtype>
        <paramvalue> 15 </paramvalue>
      </param>
      <param>
        <paramname> Maximum price </paramname>
        <paramtype> int </paramtype>
        <paramvalue> 20 </paramvalue>
      </param>
    </parameters>
  </service>
</server>
```

Fig. 6. Configuration File

type or category of the service (e.g. buy/sell books, exchange courses' notes, or meet people), and keywords (e.g. books, course, etc.). The list of the selected services is managed by the PC component that allows the user to customize these services with the specific requests (e.g. title of the book to buy or to sell, the desired price, minimal or maximal price).

The list of services (with related servers' addresses) are stored in a XML configuration file, which is uploaded via Bluetooth in the mobile device. Figure 6 shows an example for the "sell/buy books" service.

B. Accessing the services

To access the services, the user needs to run the Bluetooth application in her mobile device. The application is written in Java and uses JSR-82 [9], which is Bluetooth API for Java. The application starts a continuous search for the Bluetooth-enabled devices in its neighborhood, and whenever it finds a server with the services specified in the configuration file, the mobile device sends the user's requests to the server. Figure 7 shows the protocol we use for the interaction among the different components.

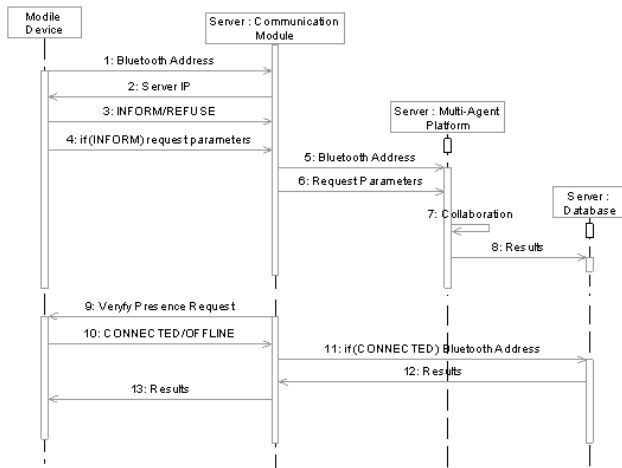


Fig. 7. Getting Access to Services

A specific communication module on the server is responsible for managing the interaction with the mobile device. It receives the list of requests from the mobile device and checks whether in the platform (running in the server) already exists a personal agent assigned to that mobile device (the Bluetooth address is used to map the mobile device with the personal agent). If there is no personal agent for the user, the communication module connects to the central services database and verify whether the user is registered to the system. Only in case of a positive answer, it creates a new agent and assigns it to the mobile device (user). Then, the communication module forwards all the user's requests to the personal agent.

Now, the personal agent starts the interaction with the other agents on the platform trying to satisfy all the user's requests. In our example the personal agent receives one or more requests for buying and/or selling books (with specified title, desired price, maximum and minimum prices, etc.). If the agent reaches an agreement with another agent about their users' requests, it can decide either to send the results back to the user or store them locally in the server database. This depends on the retrieval modality that the user has defined in the configuration file.

C. Results retrieval

Whenever a new connection between a server and a mobile device is established, the communication module sends to the mobile device the IP-address of the server. The mobile device stores the IP addresses of all the visited servers in an XML list (Figure 8-a), that is used later to retrieve all pending results. The format of the results produced by the personal agent is shown in Figure 8-b. It may contain the request identifier, the contacts (e.g. phone number) of the user interested to buy or sell the book, the actual agreed price, etc.

As discussed in Section III, the user has three different modalities to retrieve results: get the results immediately, get pending results using the mobile device, and get pending results using the PC. Each of these modalities has to be defined

```

<iplist>
  <ip> 192.168.2.148 </ip>
  <ip> 192.168.2.151 </ip>
...
</iplist>
(a)

<responses>
  <response>
    IP server: 192.168.2.151,
    Name: UnitnServer,
    Message: Buyer: Stefano Fante,
    Phone: 230-5658821,
    E-mail: stefano.fante@arslogica.it,
    Book Title : The Lord of the Rings,
    Price: 20 euros
  </response>
  ...
</responses>
(b)
  
```

Fig. 8. XML Formats. (a) List of IP Addresses of Visited Servers. (b) List of the Responses

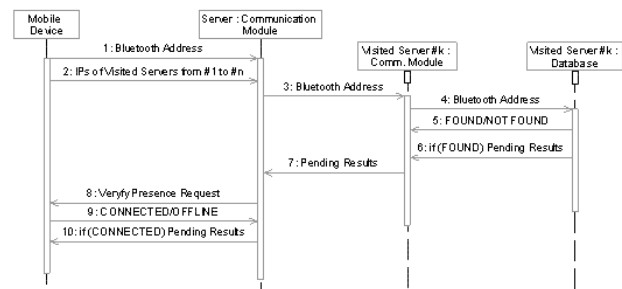


Fig. 9. Pending Results from the Mobile Device.

in advance by the user and can be changed at runtime by means of the mobile device application.

Choosing the first option, the user can receive the results immediately in her mobile device. Of course, she can receive the results if and only if she is still at a Bluetooth distance from the server. The communication module checks the availability of the mobile device and sends to it the results obtained from the corresponding personal agent.

Figure 9 shows the interaction protocol of retrieving the pending results via mobile device. Consider for example the situation in which a user is near to the server of the central library. After the connection has been established, the mobile device sends the list of IP-addresses of all previously visited servers (e.g. faculty servers, departments servers, etc.) to the library server. The communication module of the server sends then the Bluetooth address of the mobile device to all listed servers. In turn, the communication module of each server extracts from the internal database all the stored results related to the user and sends them to the requester server. All the results are collected by the communication module and finally sent to the mobile device. If the mobile device is no longer connected to the server (e.g. the user has left the library), the retrieval process will fail and the results will be cancelled (they are still available on the original servers).

Figure 10 shows the interaction protocol of retrieving the pending results via PC. The user connects her mobile device to

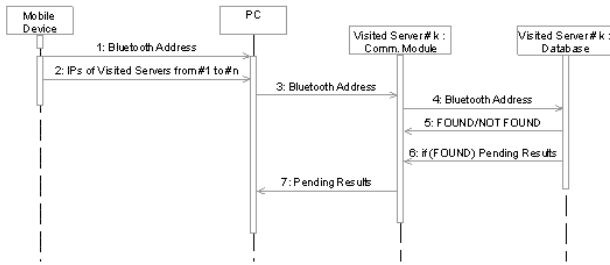


Fig. 10. Pending Results from PC

the PC via Bluetooth and sends the list of all visited servers to the PC component. Now, the user can decide either to retrieve the results from all the servers or she can just select some of them. An interface on the PC allows the user to connect to the servers and then view or download the pending results.

D. Agents interaction

As we said in this first prototype we implemented just one kind of service, namely the "buy/sell books" service. The multi-agent system has been implemented in JADE (Java Agent DEvelopment framework) [10]. The interaction mechanism is very simple. The point here is that we do not pay particular attention to the multi-agent interaction since we are mainly focused on the design and the implementation of the whole infrastructure.

Figure 11 presents the implemented interaction protocol used by the agents in the case of the "buy/sell books" service. Buyer's personal agent broadcasts the request of looking for a specific book (information about title, desired price, etc. are specified in the message). If in the platform there is another agent that is selling the requested book, it responds to the buyer with the price it wants for the book. If the price is greater than the maximum price specified by the buyer, the interaction continues with a discount request from the buyer agent. The seller responds either with the discounted price or with the initial proposed price (in case it does not want to give the discount). If this price is less than maximum price for the buyer, it accepts the deal. After that, the buyer and seller personal agents exchange their users' data, form the agreed proposals and send them to the server's database. The proposals are then forwarded either to mobile device, or to the PC as described in Section IV-C.

We tested the system using Nokia 6260 cellular phones and PC/Server equipped with Tecom Bluetooth adapter. Bluetooth communication has been implemented using Blue Cove [11] which is an open source implementation of the JSR-82 Bluetooth API for Java.

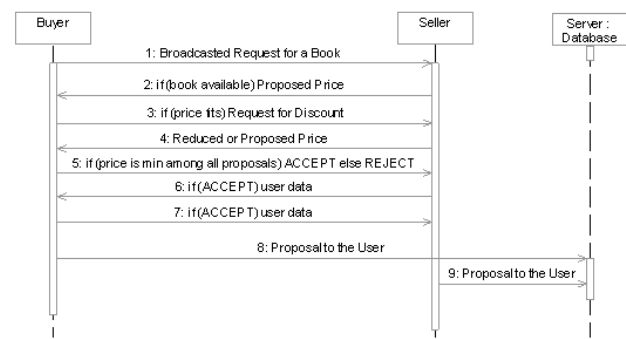


Fig. 11. Agent Interaction

V. CONCLUSIONS

In this paper we have presented an implemented prototype where multi-agent systems and Bluetooth wireless communication technology are combined together to support co-localized communities of users. We have discussed the general architecture of the system and we have presented using the buy and sell books example some implementation issues related to the prototype we have built.

A lot of work has to be done to make the system working in a real-life environments, including the implementation of various multi-agent systems able to provide different kinds of services. We are currently working with ArsLogica s.r.l. in the development of a real scenario where to apply the system.

ACKNOWLEDGEMENT

We thank ArsLogica s.r.l. for the collaboration and the support to this project. This research also is partially supported by COFIN Project "Integration between learning and peer-to-peer distributed architectures for web search (2003091149 004)".

REFERENCES

- [1] A. Rakotonirainy, S. W. Loke, and A. Zaslavsky, "Multi-agent support for open mobile virtual communities," in *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2000) (Vol 1)*, Las Vegas, Nevada, USA, 2000, pp. 127–133.
- [2] The official Bluetooth website — <http://www.bluetooth.com/>.
- [3] M. Bombara, D. Cali, and C. Santoro, "Kore: A multi-agent system to assist museum visitors," in *Proceedings of the Workshop on Objects and Agents (WOA2003)*, Cagliari, Italy, 2003, pp. 175–178.
- [4] L. Vasii and Q. H. Mahmoud, "Mobile agents in wireless devices," *Computer*, vol. 37, no. 2, pp. 104–105, February 2004.
- [5] MIA project — <http://www.uni-koblenz.de/~bthomas/MIA.HTML>.
- [6] O. Bucur, P. Beaune, and O. Boissier, "Representing context in an agent architecture for context-based decision making," in *Proceedings of the Workshop on Context Representation and Reasoning (CRR'05)*, Paris, France, 2005.
- [7] C. Carabelea and M. Berger, "Agent negotiation in ad-hoc networks," in *Proceedings of the Ambient Intelligence Workshop at AAMAS'05 Conference, Utrecht, The Netherlands*, 2005, pp. 5–16.
- [8] C. Carabelea and O. Boissier, "Multi-agent platforms on smart devices: Dream or reality?" in *Proceedings of the Smart Objects Conference (SOC03)*, Grenoble, France, 2003, pp. 126–129.
- [9] JSR-82: Java APIs for Bluetooth — <http://www.jcp.org/en/jsr/detail?id=82>.
- [10] Java Agent DEvelopment Framework website — <http://jade.tilab.com/>.
- [11] Blue Cove project — <http://sourceforge.net/projects/bluecove/>.

Designing and Implementing Electronic Auctions in a Multiagent System Environment

Davide Roggero[†], Fioravante Patrone[‡], Viviana Mascardi[†]

[†]DISI, Università di Genova,

Via Dodecaneso 35, 16146, Genova, Italy

davide@unige.it, mascardi@disi.unige.it,

[‡] DIPTeM, Università di Genova,

P.le Kennedy - Pad D, 16129, Genova, Italy

patrone@diptem.unige.it

Abstract—Agent-Mediated Electronic Commerce is gaining a wide consensus both from the academia and from the industry, since it provides the right abstractions, models and tools to face the challenges that electronic commerce raises. According to C.Sierra, e-commerce can be described as *organization + mechanism + trust*, where mechanism is concerned with the rules that govern the interaction among agents in such a way that certain properties can be guaranteed.

This paper describes the design and implementation of a library of customizable agents for simulating auction mechanisms. The purpose of the library is to provide a support to the correct engineering of mechanisms in the e-commerce setting, by providing a flexible tool for the quick prototyping of realistic auctions to the auctions' developers. The auction mechanisms that are included in our library respect the *Revenue Equivalence Theorem*, one of the most important theorems of the formal theory of auctions.

Keywords. Auction Theory, Electronic Auction, Multiagent System

I. INTRODUCTION

Information and Communication Technology (ICT) is currently considered as one of the forces that can deeply influence and transform human society. Many people agree on the important role played by ICT in productive growth and international competitiveness, thanks to reduction of transaction costs, support to efficient management, and exchange of a wide amount of information. This happens especially for commerce, radically changing the way enterprises and companies work. For example, large on-line selling enterprises use the Internet strategically to improve service quality, process speed and for cost savings, whereas small enterprises use electronic commerce (e-commerce) primarily to increase their customer base and make themselves known. In order to offer answers suitable to the currently open challenges in the e-commerce area, like business process outsourcing, marketing of agricultural exports and online dispute resolution, new technologies are required. Agent-Mediated Electronic Commerce (AMEC) is the most recent (and one of the most promising) technology born with the purpose of facing the e-commerce challenges.

In his paper "*Agent-Mediated Electronic Commerce*" [1], C.Sierra asserts that e-commerce can be described by the

following equation:

$$e\text{-Commerce} = organization + mechanism + trust$$

In this paper we deal with the second element of the sum: *mechanism*. Mechanism design is concerned with establishing the rules that govern the interaction among agents in such a way that certain properties (such as stability, or equilibrium) can be guaranteed. The definition of the rules of the game determines how the interaction will take place and, based on the assumption of rationality for the agents, tries to achieve a desired behaviour by them, possibly corresponding to dominant strategies.

In order to provide a support to the correct engineering of mechanisms in the e-commerce setting, we have developed a *library of customizable agents for simulating auction mechanisms* with the goal of providing a flexible tool for the quick prototyping of realistic auctions. The design of the auction mechanism exploits the AUMML language (<http://www.aumml.org/>) for defining the interaction protocols between a bidder and an auctioneer, while the prototyping phase is carried out by exploiting the tools offered by the DCaseLP environment [2], [3]. We have designed and implemented the auction mechanisms that are included in our library, that can be downloaded from the DCaseLP home page, <http://www.disi.unige.it/person/MascardiV/Software/DCaseLP.html>, in such a way that they respect the mathematical theory behind auctions. In fact we have based our work on the results obtained in the Auction Theory area, and in particular on the well-known Revenue Equivalence Theorem (RET, described in [4], [5], [6]). By carrying out many experiments run under different initial conditions, we have experimentally validated that the mechanisms developed as part of our library respect the RET.

The paper is structured in the following way: Section II summarizes the main mathematical results behind auction theory; Section III describes the design of the mechanisms that we provide in our library, while Section IV illustrates their implementation and shows that it respects the RET. Section V concludes.

II. AUCTION THEORY

The typical situation where an auction is suitable to allocate some goods can be described in this way: on one side of the market (the offering side) a monopolist wants to sell some goods; on the other side there are two or more potential buyers. It is implicitly assumed that the monopolist will choose the procedure (or mechanism) to allocate the goods, but this does not necessarily mean that he can extract the entire surplus, because he does not know the buyers' true evaluation of the goods.

There are many different auction mechanisms that can be classified according to their features [7], [8]. The first distinction can be made between *open* and *sealed-bid* auctions. In the *open* auction mechanisms, the seller announces prices or the bidders call out the prices themselves, thus it is possible for each agent to observe the opponents' moves. The most common type of auction in this class is the *ascending* (or *English*) auction, the well-known procedure typical of artwork auctions, where the price is successively raised until no one bids anymore and the last bidder wins the object at the last price offered. Another diffused type, the *descending* or *Dutch* auction works in exactly the opposite way: the auctioneer starts at a high price and then lowers it continuously (notice that this kind of auction essentially belongs to the "sealed bid" type). The first bidder that accepts the current price wins the object at that price. The *sealed-bid* auction mechanisms are characterized by the fact that offers are only known to the respective bidders (as the name suggest, offers are submitted in sealed envelopes). In the *first-price* sealed-bid auction each bidder independently submits a single bid without knowing the others' bid, and the objects is sold to the bidder who made the best offer. First-price auction are especially used in government contract. Another widely used and analyzed auction in this class is the *second-price* sealed-bid auction, that works exactly as the first-price one except that the winner pays the second highest bid. This auction is sometimes called Vickrey auction after William Vickrey, who wrote the seminal paper on auctions [9].

An auction mechanism is said to be *efficient* if and only if the offered object is always given to the buyer with the highest valuation for it. The four basic mechanisms just described (English, Dutch, first price sealed-bid, and second-price sealed-bid) are all efficient (assuming that bidders are rational and their bids are in equilibrium). Assuming that an auction mechanism is efficient, an interesting problem is to establish which procedure can guarantee the maximum revenue to the seller. Economic theory provided some fundamental and surprising results on the equivalence (at equilibrium) of the expected revenues of various auction mechanisms. Vickrey provided the earliest conceptualization and results in [9], which was, together with [4], a major factor in his 1996 Nobel prize. Myerson [5] and Riley and Samuelson [6] showed that Vickrey's results apply very generally.

The Revenue equivalence theorem is stated in the following way:

(Revenue equivalence theorem) Assume that

- there are N risk-neutral potential buyers (i.e., they are indifferent between, for example, playing a lottery which gives 0 euro with probability $1/2$ and 1000 euro with probability $1/2$, and gaining 500 euros for sure);
- the independent private-value model applies (i.e. each bidder: 1) has a private evaluation of the object, unknown to the other bidders; 2) believes that the other bidders' evaluation of the object can be described by a probability distribution that is identical for all the bidders; 3) believes that there is statistical independence between the individual evaluation);
- the buyers are symmetric (i.e. they cannot be distinguished one from the other).

Then all the efficient auction mechanisms guarantee to the seller the same expected revenue, and each bidder makes the same expected payment as a function of his valuation.

This theorem implicitly defines a wide class of equivalence of auction mechanisms (in terms of expected revenue) and both the first-price and the second-price sealed auction belong to this class. This could be surprising: in the first-price sealed auction, the winner pays the price that he called while in the second-price one the winner pays a price equal to the highest bid made by the other players. The fact is that the players' best strategy in the second-price auction is to bid their true valuation while in the first-price auction the bidders face a trade-off between lowering the offer (thus obtaining a better payoff in case of success) and getting higher probability of success (but paying more for the object). It is optimal for a bidder in a first-price auction to bid his valuation minus a discount: the revenue equivalence theorem states that this discount compensates exactly (in expected value) the reduction of payment caused by the second-price mechanism.

III. ANALYSIS AND DESIGN OF THE AUCTION MECHANISMS

Considering that the Dutch auction mechanisms is completely equivalent under any value model to the first-price sealed-bid auction, we have implemented the remaining three standard mechanisms described in Section II: English, first-price sealed-bid and second-price sealed-bid mechanism. Since the English mechanism is the most complex (and interesting) one among the three, in this section we concentrate on it, by analyzing the communication protocol that governs the interaction between auctioneer and bidders, and by describing the design of the agents' behavior. The details on the sealed-bid mechanisms can be found in [10].

Each auction mechanisms require two types of agent at least:

- 1) The *Auctioneer* agent that puts items on sales, receives offers, distributes information on what is going on and decides the auction winner

- 2) The *Bidder* agents that try to buy the items on sale by evaluating newly acquired information and sending offers

We implemented an English auction mechanism for a single indivisible object. Our analysis of this auction led us to the definition of the interaction protocol in Figure 1. This protocol is described using AUML that extends UML with agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates.

In the registration phase, p Bidder agents ask to be registered in the Auction by sending a message with a communicative act `request`, the Auctioneer can accept the request (sending back a message of `confirm` to each accepted agent) or deny the request (with a `refuse` communicative act).

Once the registration time is over, the Auctioneer sends an `inform` message to the n registered agents specifying its reservation price, this warns the Bidder about the minimal acceptable offer. Then the Auctioneer sends another `inform` communicative act to start the offering phase.

In the offering phase, the Bidder agents send `propose` messages that contain offers: every time a Bidder x offers a bid that is better than the highest received bid, the Auctioneer sends an `inform` message back to x to notify that is winning the object. Then the Auctioneer has three possibilities:

- 1) to broadcast to all n participants what is the new highest offer
- 2) to broadcast to all n participants that there is an extension to the original auction span
- 3) to declare the end of the offering phase

All these possibilities are communicated by `inform` messages and each of them causes different behaviors of the Bidder agents: the first two messages leave to the Bidders the chance to make new offers (shown in the Figure 1 by the loop back arrows), while the last message moves the communication protocol to the next phase, the object attribution.

The object attribution phase of an English auction mechanisms with continuous bidding is simple because the evaluation of the best bid is completely done in the offering phase, so the winner agent is already determined once that phase is finished. Thus, the Auctioneer broadcasts an `inform` message with the name of the winner, then wait for a `confirm` message.

The next step in our analysis was to describe the behavior of Auctioneers and Bidders of each auction mechanism, and we decided that Pascal pseudo-code was the right tool for this activity. Since, for space constraints, we cannot include the pseudo-code that we have defined for both the auctioneer and the bidders, we only include - as an example - a portion of the code defining the core activity of the bidder's offering stage. This is the activity of the Bidder after receiving the information about the current bid from the Auctioneer. The bidder 1) updates its value model according to it; 2) evaluates its new offer, according to the (updated) value model; and 3) offers a new bid, if its new offer is better than the current one.

```
if (receive('inform','present-bid(Bid)',auctioneer) and not
I-win)
```

```
then
```

```
Present-bid:=Bid;
update-value-model(Bid,[],Value-model);
New-bid:=eval-offer(Present-bid,Value-model,
Bidder-number,End-auction);
if(better(New-bid,Present-Bid)
then
    send('propose','offer(New-bid)',auctioneer);
endif
```

```
...
```

IV. IMPLEMENTATION OF THE LIBRARY OF AUCTION MECHANISMS

Each auction mechanism in our library is constituted by the implementation of the code of the auctioneer, and the code for the bidders. The files that contain the auctioneer code are named '`AUCT.type.pl`' while the files with the bidders code are named '`GX.type.pl`', where *type* refers to the auction mechanism and X is an integer. The agents are implemented in tuProlog in the DCaseLP environment based on the JADE platform (<http://jade.tilab.com/>). The code of the agents can be downloaded from the DCaseLP home page (<http://www.disi.unige.it/person/MascardiV/Software/DCaseLP.html>), together with the packages that constitute DCaseLP, and the Master thesis by D. Roggero [10] (in English) that describes the application.

The characteristics of the auctioneer that can be customized by the user are:

Registration time. It is the duration of the registration phase in minutes.

Acceptance of registration. The user can customize the predicate that define the rules by which an agent can be accepted as a bidder. These rules can be private of the auctioneer or depend on an external reputation system.

Auction time. It is the duration of the offering phase in minutes.

Alarm time. Only in the English auctions. It is the interval of time at the end of the offering phase, during which any new offer will trigger the extension of the auction time.

Extension time. Only in the English auctions. It is the interval of time that the auctioneer adds to the auction time if any offer has arrived during the alarm time.

Wait time. It is the interval of time that the auctioneer waits for a message of confirmation from the winning bidder.

Reservation price. The reservation price is the lowest bid accepted by the auctioneer to sell the object.

Bid comparison. The auctioneer must choose if a new bid is better than another. The user can customize this feature to reflect the preferences of the auctioneer over offers.

Attribution of the object. In case the auction ends with two or more bidders owning the best offer, the auctioneer must decide who is the real winner using a lottery whose definition can be customized.

As far as the bidders are concerned, the characteristics of the bidders that can be customized by the user are:

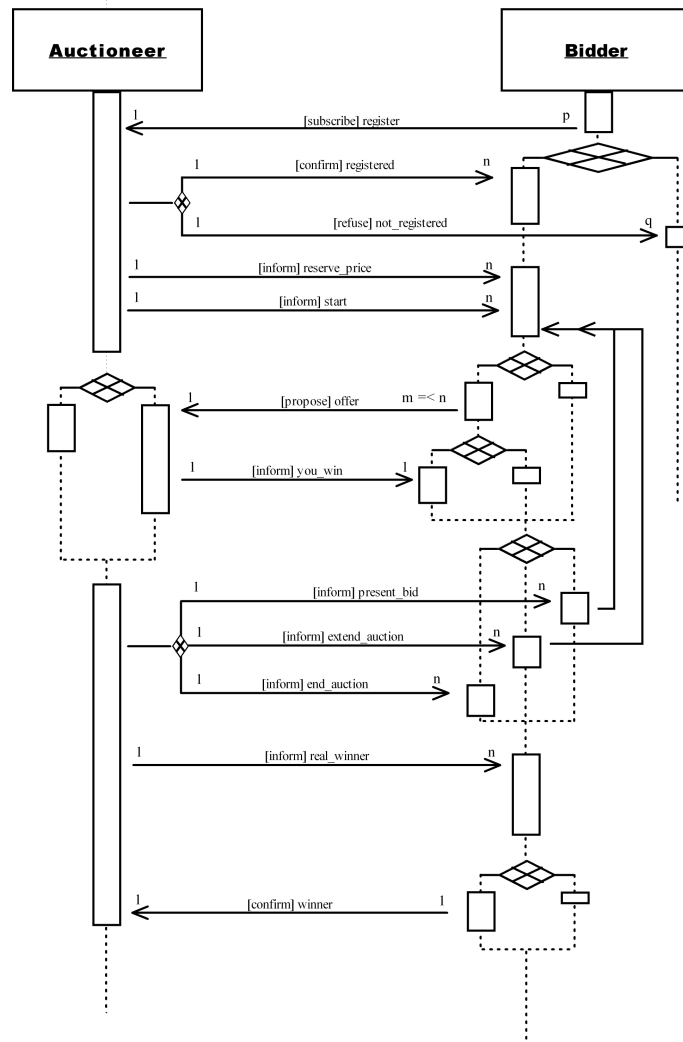


Fig. 1. English auction mechanism with continuous bidding

Value Model. The value model of a bidder determines its object's monetary worth. The default value model implemented in our bidders is the private one: the bidder asserts a static value for the object. The user can customize this feature, defining a predicate that calculates the object's worth for the bidder using both private and public information. Notice that the software works independently of the assumption of private value, so that other agents' bids can be informative.

Strategy. The strategy of a bidder determines the value and the time of its offers. It depends mainly on the value model and on other bidder's behavior, but other aspects can be considered, like time and information from sources external to the auction.

In order to test our implementation, we ran all the mechanisms of our library with the same parameters to show that they satisfy the RET discussed in Section II.

In each auction, the name of the auctioneer agent is of the form 'auct_mech' where *mech* is the type of auction

mechanism while the names of the bidder agents are of the form 'gX_mech' where *X* is an integer in the interval $[1, 4]$ and *mech* is the type of auction mechanism. The complete address will be name@Vento:1099/JADE since all the agents are deployed on a single computer called 'Vento'. In the text output, each agent's output can be recognized by its address at the beginning of the line.

In the following, we only discuss the outcomes of our experiments with the English auction; a complete account of the implementation of the sealed-bid mechanisms can be found in [10]. Game theory suggests that, in an English auction with private values, the best strategy for any bidder is to remain in the competition, making small raising, until the price reaches his evaluation of the object, then drop out of the auction: in this way the winner will get the object at a price just a little higher than the second-highest private value. In our implementation, each bidder uses this strategy (implemented in Prolog):

```
eval_offer(New_bid) :-
    object.value(Value), present_bid(Present_bid),
    Present_bid < Value, New_bid is Present_bid + 1, !.
```

The bidder makes the evaluation of a new offer each time the auctioneer inform all the participants that the present winning price is changed. With this strategy, every bidder (except the one who made the last winning bid) makes the same offer as soon as they get the message: being in an English auction with continuous bidding, the auctioneer will accept the first arrived offer as the temporary winning bid (in fact, it bested the old one by 1 euro¹) and discard all the subsequent identical offer made by other bidders.

The messages on lines 318, 319, 320, 321 of Figure 2 are inform messages that contain the present temporary best offer. As soon as they get this message, all the bidders (except g1 who was the present winner) send propose messages (lines 322, 323, 324) containing new identical offers calculated with the eval_offer predicate seen before. The auctioneer gets the first offer (line 322), sees that it is better than the last winning bid and take it as the new winning bid (at line 325): when the auctioneer examines the other offers, it finds that they are *equal* to the present winning bid, so it discards them.

In Figure 3, we can see that, at the end of the auction, agent g4_eng_c wins with an offer of 301. We can also note that the auction time was expired before the real conclusion of the competition and this has triggered the extension mechanism that permits to establish the final price of the object; in fact, at the end of the auction time the winner was agent g3_eng_c with a bid of 300.

From the theoretical point of view, if an English auction has no limits of time, the best selling price will emerge for sure, but a more realistic approach suggests to limit the duration of the auction, like we did. This can create consequences: for example, if the private value of at least two bidder is much bigger than the reservation price, the extended time could expire before the competition is over, thus denying the individuation of the best offer and not attributing the object to the bidder with the highest private value. This fact is inevitable but we realized that, in this implementation, the order in which the bidders register to the auction influences the order in which they bid, thus giving advantage to a bidder that registered earlier than another: this leads to an unfair attribution of the object. Hence, we decided to implement a round-based version of the English auction that could change this unfair behavior. The English auction with rounds behaves like the continuous one, apart from the attribution stage, where a fair approach to determine the winner is adopted (more details can be found in [10]).

We run all four auction mechanisms implemented under common conditions to verify the RET. Examining all the auction run, we can notice that every one of them terminated with agent g4_eng_r as winner, thus demonstrating to be

efficient auctions. The two sealed bid mechanisms individuated an auctioneer's revenue of 300, while for the two English mechanisms the revenue was of 301: this difference is caused by the *discrete bidding* strategy that our bidders use. In fact, if the strategy in the English auctions had been to raise the last winning price by 0.1, then the difference between the revenues would have been not 1 but 0.1; if the strategy had been to raise the price by 0.01, then difference would have been 0.01; and so on. Thus, we can say that our implementation verifies the RET.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have described the work done to develop a library of agents for simulating auction mechanisms. We have analyzed and implemented four different mechanisms:

- the first-price sealed-bid auction mechanism,
- the second-price sealed-bid auction mechanism,
- the open English auction mechanism with continuous bidding,
- the open English auction mechanism with rounds.

For each auction mechanism, the interaction between auctioneer and bidder has been analyzed and an Interaction Protocol has been produced. In the design phase, the internal behavior of each type of agent has been studied and their customizable features have been highlighted. Each agent's behavior has been written down in a pseudo-Pascal listing. Finally, each agent has been implemented with tuProlog in the DCaseLP environment, thus achieving the goal of providing customizable tools for simulating auction mechanisms. For example, by modifying the reservation price of the English auctioneer and the value model of the related bidders, it is possible to simulate English multi-dimensional auctions. Moreover, DCaseLP and JADE supply many tools for analyzing message exchange and debugging agent behaviors, thus helping the user in the analysis of the bidders' strategy.

We have ran all the implemented mechanism using risk-neutral bidders with independent private value taken from a uniform distribution. Under these hypothesis, Game Theory demonstrated that there exist an optimal bidder's strategy for each of the implemented mechanism: we programmed our test bidders with these strategies and we verified that all the simulated auctions gave the same revenue to the auctioneer and the same payoff to the bidders. The fact that RET is satisfied (up to some error clearly due to discretization) can be seen as a check for the correctness of the implementation.

As far as the related work is concerned, today there are many commercial and research applications for implementing real electronic auctions, or simply for simulating them. The Trading Agent Competition (<http://www.sics.se/tac/>), for example, is carried out every year, in order to promote and encourage high quality research into the trading agent problem, while the well-know electronic commerce portals, eBay (<http://www.ebay.com/>) and Amazon (<http://www.amazon.com/>), demonstrate the commercial applicability of the research on agent-mediated auctions.

¹We assume that the granularity of the bid is 1 euro.

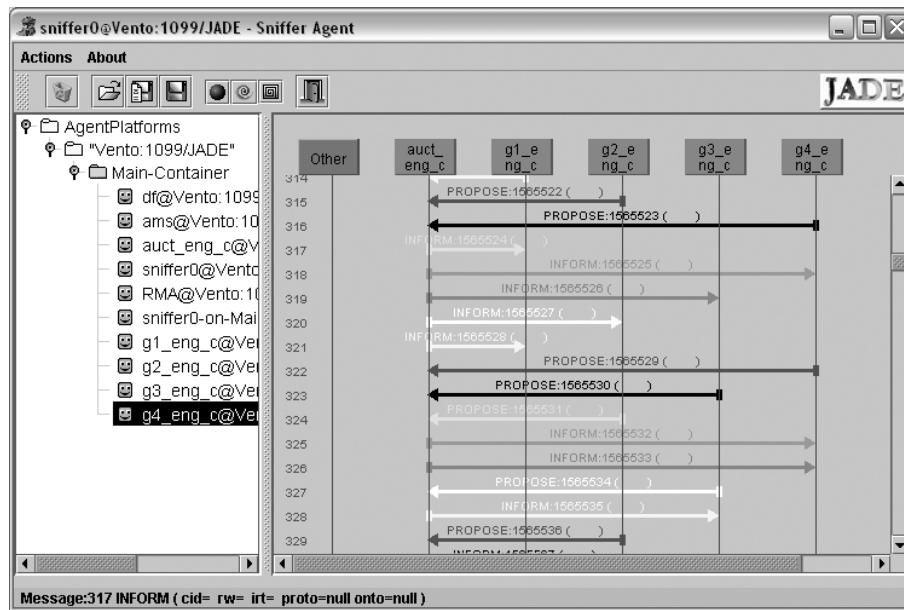


Fig. 2. English auction with continuous bidding: offering phase

```

C:\WINDOWS\system32\cmd.exe - java jade.Boot -gui
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
277' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
278' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
279' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
280' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
281' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
282' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
283' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
284' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
285' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
286' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
287' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
288' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
289' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
290' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
291' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
292' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
293' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
294' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
295' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
296' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
297' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
298' di 'g3_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
299' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
300' di 'g3_eng_c@Vento:1099/JADE'

'auct_eng_c@Vento:1099/JADE': 'Estendo l'asta'

'auct_eng_c@Vento:1099/JADE': 'Offerta vincente: '
301' di 'g4_eng_c@Vento:1099/JADE'
'auct_eng_c@Vento:1099/JADE': 'Asta finita'

'Uince l'asta l'agente 'g4_eng_c@Vento:1099/JADE' con offerta di '301
'Fine'

```

Fig. 3. English auction with continuous bidding: shell output.

Despite the wide range of available applications, we decided to implement and distribute our own, in order to implement (as part of our future work) some extensions to the basic auction mechanism that may benefit from the reasoning capabilities provided by our tuProlog agents. In particular, we would be interested in:

- analyzing and implementing other less common but interesting auction mechanisms, like double auctions and all-pay auctions. The last kind of auctions is quite common, often at a non formalised level: just considering lobbying activities, or competition for a given (potential) boy/girl friend...
- building a society of agents, with “advertising” agents that contain information (like starting and ending time, type of object on sale, type of auction mechanism) on the auctions that are going to be held and “searcher” agents that look for interesting auction using user’s preferences and informs the bidder.
- implementing a reputation system, where reliable “notarial” agents calculates the reputation of the subscribers using other agents’ opinions and past behaviors and making it public to the agent community.

REFERENCES

- [1] C. Sierra, “Agent-mediated electronic commerce,” *Autonomous Agents and Multi-Agent Systems*, vol. 9, pp. 285–301, 2004.
- [2] E. Astesiano, M. Martelli, V. Mascardi, and G. Reggio, “From Requirement Specification to Prototype Execution: a Combination of a Multiview Use-Case Driven Method and Agent-Oriented Techniques,” in *Proc. of SEKE’03*, 2003, pp. 578–585.
- [3] I. Gungui and V. Mascardi, “Integrating tuProlog into DCaseLP to engineer heterogeneous agent systems,” in *Proc. of CILC 2004*. Available at <http://www.disi.unige.it/person/MascardiV/Download/CILC04a.pdf.gz>.
- [4] W. Vickrey, “Auction and bidding games,” in *Recent advances in Game Theory*. Princeton University Conference, 1962, pp. 15–27.
- [5] R. Myerson, “Optimal auction design,” *Mathematics of Operations Research*, vol. 6, pp. 58–73, 1981.
- [6] J. Riley and W. Samuelson, “Optimal auctions,” *American economic review*, vol. 71, pp. 381–92, 1981.
- [7] P. Klemperer, *Auctions: Theory and practice*. Princeton University Press, 2004.
- [8] V. Krishna, *Auction Theory*. Academic Press, 2002.
- [9] W. Vickrey, “Counterspeculation, auctions and competitive sealed tenders,” *Journal of Finance*, vol. 16, pp. 8–37, 1961.
- [10] D. Roggero, “Aste elettroniche in ambiente multi-agente,” Master’s thesis, DISI, University of Genoa, Italy, 2005.

Social roles, from agents back to objects

Matteo Baldoni and Guido Boella
Dipartimento di Informatica
Università degli Studi di Torino
Email: {baldoni,guido}@di.unito.it

Leendert van der Torre
CWI Amsterdam and Delft university of Technology
Email: torre@cwi.nl

Abstract—In this paper we introduce a new view on roles in Object Oriented programming languages. This view is based on an ontological analysis of roles and attributes to roles the following properties: first, a role is always associated not only with an object instance playing the role, but also to another object instance which constitutes the context of the role and which we call institution. Second, the definition of a role depends on the definition of the institution which constitutes its context. Third, this second property allows to endow players of roles with powers to modify the state of the institution and of the other roles of the same institution. As an example of this model of roles in Object Oriented programming languages, we introduce a role construct in Java. We interpret these three features of roles in Java as the fact that, first, roles are implemented as classes which can be instantiated only in presence of an instance of the player of the role and of an instance of the class representing the institution. Second, the definition of a class implementing a role is included in the class definition of the institution the role belongs to. Thirdly, powers are methods of roles which can access private fields and methods of the institution they belong to and of the other roles of the same institution.

I. INTRODUCTION

The concept of role is used quite ubiquitously in Computer Science: from databases to multiagent systems, from conceptual modelling to programming languages. According to Steimann [18], the reason is that even if the duality of objects and relationships is deeply embedded in human thinking, yet there is evidence that the two are naturally complemented by a third, equality fundamental notion: that of roles. Although definitions of the role concept abound in the literature, Steimann maintains that only few are truly original, and that even fewer acknowledge the intrinsic role of roles as intermediaries between relationships and the objects that engage in them. There are three main views of role:

- Names for association ends, like in UML or in Entity-Relationship diagrams.
- Dynamic specialization, like in the Fibonacci [2] programming language.
- Adjunct instances, like in the DOOR programming language [22].

The two last views are more relevant for modelling roles in programming languages. Both of them have pros and cons. For example, dynamic specialization captures the dynamic relation between a class and a role which can be played by it (e.g., a person can become a student), but it less easily models the intuition that roles can have their own state (e.g., an employee has a different phone number than the person playing that

role). In contrast, roles as adjunct instances can obviously have their own state, but they may pose problems when role instances are detached from the object which plays the role.

There is a wide literature on the introduction of the notion of role in programming languages. However, most works, starting from Bachman and Daya [3]’s revision of database models, extend programming languages with roles starting from practical considerations. In contrast, the research question of this paper is the following: How to introduce in an Object Oriented programming language a notion of role which is ontologically well founded? We refer to the ontological analysis of the notion of role made in [7], [5], [6]. According to that proposal, roles have the following properties:

- Roles are always associated both to an object instance playing the role, and to another object instance which constitutes the context of the role and which we call the *institution*.
- The definition of a role depends on the definition of the institution which constitutes its context.
- This second property allows to endow players of roles with powers to modify the state of the institution and of the other roles of the same institution.

For example, the role *student* has a *person* as its player and it is always a student of a *school*, a *president* is always the president of an *organization*, a *customer* can be played by a *person* or an *organization*, and it is always a *customer* of an *enterprise*. In contrast, almost all current approaches focus only on the relation between the role and its player.

The methodology we follow is to introduce a new programming construct in a real programming language, Java, one of the most used Object Oriented languages and one of the most principled. To prove its feasibility, we translate the new language, called *powerJava*, to pure Java by means of a precompilation phase.

The role construct we introduce in Java promotes the separation of concerns between the core behavior of an object and its context dependent behavior. In particular, the interaction among a player object, the institution and the other roles is encapsulated inside the role the object plays.

In Section II we summarize the ontological definition of roles while, in Section III, we introduce roles Java with *powerJava*. Related work and conclusion end the paper.

II. FOUNDATION, DEFINITIONAL DEPENDENCE, AND POWERS

The distinguishing features of roles in [7], [5], [6] are their foundation, their definitional dependence from the institution they belong to, and the powers attributed to the role by the institution. Consider the roles student and teacher. A student and a teacher are always a student and a teacher of some school. Without the school the roles do not exist anymore: e.g., if the school goes bankrupt, the actors (e.g. a person) of the roles cannot be called teachers and students anymore. The institution (the school) also specifies the properties of the student, which extend the properties of the person playing the role of student: the school specifies its enrollment number, its email address, its scores at past examinations, and also how the student can behave. For example, the student can give an exam by submitting some written examination. A student can make the teacher evaluate its examination and register the mark because the school defines both the student role and the teacher's role: the school specifies how an examination is evaluated by a teacher, and maintains the official records of the examinations. Otherwise the student could not have an effect on the teacher. But in defining such actions the school *empowers* the person who is playing the role of student: without being a student the person has no possibility to give an examination and make the teacher evaluate it.

This example highlights the following properties that roles have in our model [7], [5], [6]:

- *Foundation*: a (instance of) role must always be associated with an instance of the institution it belongs to (see Guarino and Welty [10]), besides being associated with an instance of its player.
- *Definitional dependence*: The definition of the role must be given inside the definition of the institution it belongs to. This is a stronger version of the definitional dependence notion proposed by Masolo *et al.* [14], where the definition of a role must use the concept of the institution.
- *Institutional empowerment*: the actions defined for the role in the definition of the institution have access to the state and actions of the institution and of the other roles: they are powers.

Moreover, as Guarino and Welty [10] notice, contrary to natural classes like person, roles lack *rigidity*: a player can enter and leave a role without losing its identity; a person can stop being a student but not being a person. Finally, Steimann [19]'s highlights that a role can be played by different kinds of actors. For example, the role of customer can be played by instances both of person and of organization, i.e., two classes which do not have a common superclass. The role must specify how to deal with the different properties of the possible actors. This requirement is in line with UML, which relates roles and interfaces as partial descriptions of behavior.

This last property compels to avoid modelling roles as dynamic specializations as, e.g., [2], [9] do. If customer were a subclass of person, it could not be at the same time a subclass of organization, since person and organization are disjoint

classes. Symmetrically, person and organization cannot be subclass of customer, since a person can be a person without ever becoming a customer.

III. INTRODUCING ROLES IN JAVA: POWERJAVA

Roles are useful in programming languages for several reasons, from dealing with the separation of concerns between the core behavior of an object and its interaction possibilities, to reflecting the ontological structure of domains where roles are present, from modelling dynamic changes of behavior in a class to fostering coordination among components.

In our proposal, we model roles as instances of role classes, which can be associated at runtime with objects which can play a role. However, roles are special kind of objects, and instances of role classes do not exist on their own, but they always require to be associated with an object instance of its player and an object instance of the related institution. The relations of a role with these two instances are different. Concerning the former relation, the player of the role is an object whose properties and behavior are extended when it is seen under the perspective of the role. Moreover, the role does not affect the core behavior. In contrast, concerning the latter relation, the object instance which represents the institution which the role belongs to gives the role powers: the role is enabled to access the institution's own state and the state of the other roles via its methods; thus, role's behavior can effect the institution's behavior. Accessing the institution's state is possible only if the classes defining it and its roles are connected. This is what it is called definitional dependence and it requires that the role class belongs to the namespace of the institution class.

Analogously to classes and interfaces in OO, we distinguish the role implementation in an institution from the role definition (both powers and requirements). A role implementation should implements the role powers definition while a player should implements a role requirements definition.

Finally, the constraint of foundation requires that the creation of a role instance involves both an institution instance and an object instance. A power can be invoked from a role only by specifying the role which the player had to play. Note that an object can play not only several roles, but also the same role in different institutions at the same time. Hence, the role under which a player is seen must be specified using not only the role but also the institution instance.

In this paper we extend Java with these desired features of roles in OO programming languages. In summary, in our proposal, first a role is defined specifying what is requested to play a role and what is offered by a role by an abstract definition similar to a Java interface. Second, since Java inner classes allow a class to belong to the namespace of another class, we use them to give powers to roles in institutions. Moreover, implementing a role definition as an inner class of an outer class defining an institution parallels exactly the definitional dependence. Third, the association of a role instance with an institution instance can be dealt with the implicit reference in Java of an inner class from its outer class. So we are left only to deal explicitly with the association of a

```

interface StudentReq //Student's requirements
{ String getName();
  int getSocialSecNumber(); }

role Student playedby StudentReq // Student's powers
{ String getName();
  void takeExam(int examCode, HomeWork hwk);
  int getMark(int examCode); }

interface TeacherReq // Teacher's requirements
{ String getName();
  int getSocialSecNumber();
  int getQualificationNumber();
  int read(HomeWork hwk); }

role Teacher playedby TeacherReq // Teacher's powers
{ String getName();
  int evalHomeWork(HomeWork hwk); }

```

Fig. 1. Definition of roles and their requirements.

role instance with a player instance, to complete foundation. Finally, seeing an object under a role is paralleled with type casting in Java.

A. The definition of roles

The definition of a role has to specify both what is required to play the role and which powers the player have in the institution the role will be implemented. In order to make role systems reusable, it is necessary that a role is not played by a class only. For Steimann and Mayer [20], roles define a certain behavior or protocol demanded in a context independently of how or by whom this behavior is to be delivered (and, we add, roles also empowers the player in the context). Thus, roles must be specified independently of the particular classes playing the role, so that the objects which can play the role might be of different classes and can be developed independently of the implementation of the role. This is a form of polymorphism. In order to achieve such polymorphism we associate with a role descriptions of classes listing the signatures of the methods which are requested to and object in order to play a role. We, thus, have that a role definition must express, first, the methods required to objects playing the role: *requirements*. For the instances of a class to play a role, the class must offer some methods. These are specified by the role as an interface. Second, the methods offered to objects playing the role: *powers*. If an object of a class offering the requirements, plays the role, it is empowered with these new methods. The definition of a role using the keyword `role` is similar to the definition of an interface; it is the specification of the powers acquired by the role in the form of abstract methods signatures. The only difference is that the role definition by means of the keyword `playedby` refers also to another interface, that in turn specifies the requirements which an object playing the role must satisfy.

In Figure 1, the definitions of the roles `Student` and `Teacher` are introduced. The roles specify, like an interface, the signatures of the methods that correspond to the powers that are assigned to the objects playing the role. For example, returning the name of the `Student` (`getName`), submitting

an homework as an examination (`takeExam`), and so forth. Moreover, we couple a role definition with the specification of its requirements by the keyword `playedby`. This specification is given by means of the name of a Java interface, e.g., `StudentReq`, imposing the presence of methods `getName` and `getSocialSecNum` (his social security number).

B. Institutions and definitional dependence

In [7], [5], [6] roles are always associated with an instance of, and are definitionally dependent on, an institution. Roles add powers to objects playing the roles. Power means the possibility to modify also the state of the institution which defines the role and the state of the other roles defined in the same institution. In our running example, we have that the method for taking an exam in the school must be able to modify the private state of the school. For example, if the exam is successful, the grade should be added to the registry of exams in the school by the teacher. Analogously, the student's method for taking an exam can invoke the teacher's method of evaluating an examination. Powers, thus, seems to violate the standard encapsulation principle, where the private variables are visible to the class they belong to only. However, here, the encapsulation principle is preserved: all roles of an institution depend on the definition of the institution; so it is the institution itself which gives to the roles access to its private fields and methods. Since it is the institution itself which defines its roles, there is no risk of abuse by part of the role of its access possibilities. Enabling a class to belong to the namespace of another class without requiring it to be defined as friend is achieved in Java by means of the inner class construct. Thus, we extend the notion of inner class to allow roles to be implemented inside an institution (the outer class). The inner class construct is extended with the keyword `realizes` which specifies the name of the role definition the inner class is implementing. An institution is simply a class with an inner class realizing roles in the very same way as a class implements an interface. In Figure 2, `StudentImpl` (`TeacherImpl`) realizes the role definition `Student` (`Teacher`), inside the institution `School`. Note that, a role (implementation) could itself be an institution with its own role implementations, it could enact other roles and, analogously, an institution could play a role. Moreover, roles can be implemented in different ways in the same institution.

Since the behavior of a role instance depends on the player of the role, in the method implementation, the player instance can be retrieved via a new reserved keyword: `that`. So this keyword refers to *that* object which is playing the role at issue, and it is used only in the role implementation. The value of `that` is initialized when the constructor of the role implementation is invoked. The referred object has the type defined by the role requirements or a subtype. We do not need a special expression for creating instances of the inner classes implementing roles, because we use the Java inner classes syntax: starting from an institution instance (or from a class name in case of static inner classes), the keyword `new` allows the creation of an instance of the role as an instance of the

```

class School {
    private int[][] marks;
    private Teacher[] teachers;
    private String schoolName;
    public School (String schoolName) {
        this.schoolName = schoolName;
        ...
    }

    class StudentImpl realizes Student {
        private int studentID;
        public int getStudentID() {
            return studentID;
        }
        public void takeExam(int examCode; HomeWork hwk) { }
        marks[studentID][examCode] =
            teachers[examCode].evalHomeWork(hwk);
    }
    public String getName() {
        return that.getName() +
            ", student at " + schoolName;
    }
}

class TeacherImpl realizes Teacher {
    private int teacherID;
    public int getTeacherID() { return teacherID; }
    public int evalHomeWork(HomeWork hwk) { ...
        mark = that.read(hwk); ...
        return mark;
    }
    public String getName() {
        return that.getName() + ", teacher at "
            + schoolName;
    }
}

class Person implements StudentReq {
    private String name;
    private int socialSecNumber;
    public Person(String name, int socialSecNumber) {
        this.name = name;
        this.socialSecNumber = socialSecNumber; }
    public String getName() { return name; }
    public int getSocialSecNumber() {
        return socialSecNumber;
    }
}

class QualifiedPerson extends Person
    implements TeacherReq {
    private int qualificationNumber;
    public QualifiedPerson(String name,
        int socialSecNumber,
        int qualificationNumber) {
        super(name, socialSecNumber);
        this.qualificationNumber = qualificationNumber;
    }
    public int getQualificationNumber() {
        return qualificationNumber;
    }
    public int read(HomeWork hwk) { ... }
}

```

Fig. 2. Definition of an institution and its role implementations.

```

class TestRole {
    public static void main(String[] args) {
        Person chris = new Person("Christine", 1234);
        Person george =
            new QualifiedPerson("George", 5678, 9876);
        School harvard = new School("Harvard");
        School mit = new School("MIT");
        harvard.new StudentImpl(chris);
        harvard.new TeacherImpl(george);
        mit.new TeacherImpl(george);
        String x =
            ((harvard.StudentImpl) chris).getName();
        String y =
            ((harvard.TeacherImpl) george).getName();
        String z =
            ((Teacher)(mit.TeacherImpl) george).getName();
        ((harvard.StudentImpl) chris).takeExam(..., ...);
    }
}

```

Fig. 3. Using roles.

inner class, e.g., `harvard.new StudentImpl(chris)` in Figure 3. Note that, all the constructors of role implementations have at least a (implicit) parameter which must be bound to the player of the role and become the value of `that`.

In order for an object to play a role it is sufficient that it conforms to the role requirements. Since the role requirements are a Java interface, it is sufficient that the class of the object implements the methods of such an interface. In Figure 2, the class `Person` can play the role `Student`, because it conforms to the interface `StudentReq` by implementing it.

C. Exercising the powers of a role

A role represents a perspective on an object. An object has different (or additional) properties when it is seen in the perspective of a certain role, and it can perform new activities, which we call powers, as specified by the role definition. In Steimann [18]'s terminology, a role is a type specifying behavior.

When an object is seen under the perspective of a role, we want that the object has a specific state for it. This state is different from the player's one, it is specific to each role in each institution, and it can evolve with time by invoking methods on the roles (or on other roles of the same institution as we have seen in the running example). This state is given by a role instance which is associated with the player. Since a role represents the perspective on an object, the object playing the role should be able to invoke the role's methods without any explicit reference to the instance of the role. In this way the association between the object instance and the role instance is transparent to the programmer. The object should only specify in which role it is invoking the method. For example, if a person is a student and a student can be asked to return its enrollment number, we want to be able to invoke the method on the person as a student without referring to the student role instance.

The same methods will have a different behavior according to the role which the object plays when they are invoked. On the other hand, methods of a role can exhibit different

behaviors according to whom is playing it. So a method of student returning the name of the student together with the name of the school returns different values for the name according to whom is playing the role of student. This is possible since the implementation of methods representing powers uses the methods required by the role to its player in order to play the role. These required methods obviously can access the state of the player since they are part of the implementation of the player.

Roles are always roles in an institution. Hence, an object can play at the same moment the same role more than once, albeit in different institutions. Instead, we do not consider the case of an object playing the same role more than once in the same institution. An object can play several roles in the same institution. In order to specify the role under which an object is referred, we evocatively use the same terminology used for casting by Java: we say that there is a casting from the object to the role. However, to refer to an object in a certain role, both the object and the institution where it plays the role must be specified. We call this methodology *role casting*. Type casting in Java allows to see the same object under different perspectives while maintaining the same structure and state. In contrast, role casting views an object as having a different state and different behaviors when playing different roles. So, the last syntactic change in powerJava is the introduction of *role casting expressions* extending the original Java syntax for casting. A role cast specifies both the role and the instance of the institution the role belongs to. For example, in `(harvard.TeacherImpl) george`, in Figure 3, the person `george` is casted to its role `harvard.TeacherImpl` of type `School.TeacherImpl`. It is important to observe that role casting is done to the inner class implementing the role but the role instance can always be type casted to the role as well as it can be done with Java interfaces: `((Teacher)(harvard.TeacherImpl) george).getName()`. While in the previous case it was possible to use all the methods of the specific implementation, in this case, only the methods that are specified in the role definition can be applied.

IV. TRANSLATING ROLES IN JAVA

In this section we provide a translation of the role construct into Java. This is done by means of a precompilation phase, as, e.g., Guillen-Scholten *et al.* [11] propose for introducing components and channels in Java, or in the way inner classes are implemented in Java. The precompiler has been implemented by means of the tool `javaCC`, provided by Sun Microsystems [1]. The translation of the example is shown in Figures 4–7.

The role definition is simply an interface (see Figure 4) to be implemented by the inner class defining the role. So the role powers and its requirements form a pair of interfaces used to match the player of the role and the institution the role belongs to. The relation between the role interface and the requirement interface is used in the constructor of an inner class implementing a role. The requirement interface is used

```
interface Student {
    String getName();
    void takeExam(int examCode, HomeWork hwk);
    int getMark(int examCode);
}

interface Teacher {
    String getName();
    int evalHomeWork(HomeWork hwk);
}
```

Fig. 4. Translation of role definitions.

```
class School {
    private int[][] marks;
    private String schoolName;

    class StudentImpl implements Student {
        StudentReq that; // Added by the precompiler
        public StudentImpl (StudentReq that) {
            this.that = that; // Added the by precompiler
            ((ObjectWithRoles)this.that).
                setRole(this, School.this);
        }
        // role's fields and methods ...
    }

    class TeacherImpl implements Teacher {
        TeacherReq that; // Added by the precompiler
        public TeacherImpl (TeacherReq that) {
            this.that = that; // Added by the precompiler
            ((ObjectWithRoles)this.that).
                setRole(this, School.this);
        }
        // role's fields and methods ...
    }
    // institution's fields and methods ...
}
```

Fig. 5. Translation of an institution.

to constrain the creation of role instances relatively to players that conform to the requirements.

When an inner class implements a role (see Figure 5), the role specified by the `realizes` keyword is simply added to the interfaces implemented by the inner class. The correspondence between the player and the role object, represented by the construct `that`, is precompiled in a field called `that` of the inner class. If the inner class implements the role `Student` the variable is of type `StudentReq`. This field is automatically initialized by means of the constructors which are extended by the precompiler by adding a first parameter to pass the suitable value. The constructor adds to its player `that` also a reference to the role instance (by means of `setRole` method). The remaining link between the instance of the inner class and the outer class defining it is provided automatically by the language Java (`School.this` in our running example).

To play a role an object must be enriched by some methods and fields to maintain the correspondence with the different role instances it plays in the different institutions (see Figure 6). Since every object can play a role, it is worth noticing that the ideal solution would be that the `Object` class offered directly these features.

Every object can play many roles simultaneously. This is obtained by adding, at precompilation time, to every class a

```

interface ObjectWithRoles {
    public void setRole(Object pwr, Object inst);
    public Object getRole(Object inst, String pwr);
}

class Person implements StudentReq,
    TeacherReq, ObjectWithRoles {
    /** Added by the precompiler: BEGIN */
    private java.util.Hashtable roleslist =
        new java.util.Hashtable();
    public void setRole(Object pwr, Object inst) {
        roleslist.put(inst.hashCode() +
            pwr.getClass().getName(), pwr);
    }
    public Object getRole(Object inst, String pwr) {
        return roleslist.get(inst.hashCode() +
            inst.getClass().getName() + "$" + pwr);
    }
    /** Added by the precompiler: END */
    private String name;
    private int socialSecNumber;

    public String getName() {
        return name;
    }
    public int getSocialSecNumber() {
        return socialSecNumber;
    }
}

```

Fig. 6. Translation of players.

structure for book-keeping its role instances. This structure can be accessed by the methods whose signature is specified by the `ObjectWithRole` interface. The two methods that are introduced by the precompiler are `setRole` and `getRole` which respectively adds a role to an object specifying where the role is played and returns the role played in the institution passed as parameter. Further methods can be added for using single institutions, leaving a role, transferring it, *etc.*

We present one possible implementation of these methods which is supported by a private hashtable `rolelist`. As key in the hashtable we use the institution instance address and the name of the inner class. Role casting is precompiled using the `getRole` method. The expression referring to an object in its role (a `Person` as a `Teacher`, e.g., `(harvard.TeacherImpl) george`) is translated into the selector returning the reference to the inner class instance, representing the desired role with respect to the specified institution. The translation will be `george.getRole(harvard, "TeacherImpl")` (see Figure 7). The string `"TeacherImpl"`, that is the name of the inner class that implements the role inside the institution `School`, is provided because in our solution it is used as a part of the index and, therefore, it is necessary in order to retrieve the proper definition of the role.

Note that, the interfaces that implement the requirements of a role extend the interface `ObjectWithRoles` (see Figure 6). This interface requires that the players implement the methods for book-keeping their roles. Observe that if the Java `Object` class supplied these features, this extension would not be necessary.

```

Person chris = new Person("Christine");
Person george = new Person("George");
School harvard = new School("Harvard");
School mit = new School("MIT");
harvard.new StudentImpl(chris);
harvard.new TeacherImpl(george);
mit.new TeacherImpl(george);
String x = ((School.StudentImpl) chris.
    getRole(harvard, "StudentImpl")).getName();
String y = ((School.TeacherImpl) george.
    getRole(harvard, "TeacherImpl")).getName();
String z = ((Teacher)(School.TeacherImpl) george.
    getRole(mit, "TeacherImpl")).getName();
...
((School.StudentImpl) chris.
    getRole(harvard, "StudentImpl")).takeExam(...);

```

Fig. 7. Translation of the use of roles.

V. CONCLUSIONS AND RELATED WORK

In this paper we introduce a new view on roles in OO programming languages based on an ontological analysis of the notion of role. We introduce this model of roles in an extension of Java, called *powerJava*. Many works on the introduction of roles in programming languages [2], [9], [8], [16] consider roles as dynamic specializations of classes, e.g., a customer is seen as a specialization of the class `person`. This methodology does not capture the fact that a role like customer can be played both by a person and by an organization (that is not a person). Roles as specializations prevent realizing that a role is always associated not only with a player, but also to an institution, which defines it. This intuition sometimes emerges also in these frameworks: in [16] the authors say “a role is *visible* only within the scope of the specific application that created it”, but context are not first class citizens like institutions are in our model.

Some other works adopt a closer methodology: roles are seen as instances which are associated with objects. Wong *et al.* [22] introduce a parallel role class hierarchy connected by a “played-by” relationship to the object class hierarchy. However, they fail to capture the intuition that a role depends on the context defining it. Moreover, the method lookup as delegation they adopt has a troublesome implication: when a method is invoked on some object in one of its roles, the meaning of the method can change depending on all the other roles played by the object. This is not a desired feature in a language like Java.

In [13] it is recognized that a role depends on its player and that the properties of the role are present only due to the perspective the role is seen from. However, they consider roles as a form of specialization, albeit one distinguishing the role as an instance related to but separated from its player. As a consequence, the properties of the role include the properties inherited from its player. This idea conflicts with our position, which we adopt from Steimann [21], of roles as interfaces: roles are partial descriptions of behavior, they shadow the other properties of their players, rather than inheriting them.

Our approach share the idea of gathering roles inside wider

entities with languages like Object Teams [12] and Caesar [15]. However, these languages emerge as refinements of *aspect oriented* languages aiming at resolving some of their practical limitations. Aspects fit our conceptual model as well: e.g., when the execution of methods gives raise, by advice weaving, to the execution of a method of a role, in our model this means that the actions of an object playing a role “count as” actions executed by the role itself. Finally, our notion of role, as a double-sided interface, bears some similarities with Traits [17] and Mixins. However, they are different as, with a few exceptions, e.g., [4], they are not used to extend instances, like roles do, but classes.

REFERENCES

- [1] “Java compiler compiler [tm] (javaCC [tm]) - the java parser generator,” Sun Microsystems, <https://javacc.dev.java.net/>.
- [2] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini, “An object data model with roles,” in *Procs. of VLDB’93*, 1993, pp. 39–51.
- [3] C. Bachman and M. Daya, “The role concept in data models,” in *Procs. of VLDB’77*, 1977, pp. 464–476.
- [4] L. Bettini, V. Bono, and S. Likavec, “A core calculus of mixin-based incomplete objects,” in *Procs. of FOOL Workshop*, 2004, pp. 29–41.
- [5] G. Boella and L. van der Torre, “An agent oriented ontology of social reality,” in *Procs. of FOIS’04*. Torino: IOS Press, 2004, pp. 199–209.
- [6] —, “Attributing mental attitudes to roles: The agent metaphor applied to organizational design,” in *Procs. of ICEC’04*. IEEE Press, 2004.
- [7] —, “Regulative and constitutive norms in normative multiagent systems,” in *Procs. of KR’04*. AAAI Press, 2004, pp. 255–265.
- [8] M. Dahchour, A. Pirotte, and E. Zimanyi, “A generic role model for dynamic objects,” in *Procs. of CAiSE’02*, ser. LNCS, vol. 2348. Springer, 2002, pp. 643–658.
- [9] G. Gottlob, M. Schrefl, and B. Rock, “Extending object-oriented systems with roles,” *ACM Transactions on Information Systems*, vol. 14(3), pp. 268 – 296, 1996.
- [10] N. Guarino and C. Welty, “Evaluating ontological decisions with ontoclean,” *Communications of ACM*, vol. 45(2), pp. 61–65, 2002.
- [11] J. Guillen-Scholten, F. Arbab, F. de Boer, and M. Bonsangue, “A channel based coordination model for components,” *ENTCS*, vol. 68(3), 2003.
- [12] S. Herrmann, “Object teams: Improving modularity for crosscutting collaborations,” in *Procs. of Net.ObjectDays*, 2002.
- [13] B. Kristensen and K. Osterbye, “Roles: Conceptual abstraction theory and practical language issues,” *Theory and Practice of Object Systems*, vol. 2(3), pp. 143–160, 1996.
- [14] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino, “Social roles and their descriptions,” in *Procs. of KR’04*. AAAI Press, 2004, pp. 267–277.
- [15] M. Mezini and K. Ostermann, “Conquering aspects with caesar,” in *Procs. of the 2nd International Conference on Aspect-Oriented Software Development (AOSD)*. ACM Press, 2004, pp. 90–100.
- [16] M. Papazoglou and B. Kramer, “A database model for object dynamics,” *The VLDB Journal*, vol. 6(2), pp. 73–96, 1997.
- [17] N. Scharli, S. Ducasse, O. Nierstrasz, and A. Black, “Traits: Composable units of behavior,” in *LNCS, vol. 2743: Procs. of ECOOP’03*, S. Verlag, Ed., Berlin, 2003, pp. 248–274.
- [18] F. Steimann, “On the representation of roles in object-oriented and conceptual modelling,” *Data and Knowledge Engineering*, vol. 35, pp. 83–848, 2000.
- [19] —, “A radical revision of UML’s role concept,” in *Procs. of UML2000*, 2000, pp. 194–209.
- [20] F. Steimann and P. Mayer, “Patterns of interface-based programming,” *Journal of Object Technology*, 2005.
- [21] F. Steimann, W. Siberski, and T. Kühne, “Towards the systematic use of interface in java programming,” in *Proc. of 2nd Int. Conf. on the Principle and Practice of Programming in Java*, 2003, pp. 13–17.
- [22] R. Wong, H. Chau, and F. Lochovsky, “A data model and semantics of objects with dynamic roles,” in *Procs. of IEEE Data Engineering Conference*, 1997, pp. 402–411.

A temporal approach to the specification and verification of Interaction Protocols

L. Giordano^{*}, A. Martelli[†], P. Terenziani^{*}, A. Bottrighi^{*} and S. Montani^{*}

^{*} Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy

[†] Dipartimento di Informatica, Università di Torino, Torino, Italy

Abstract—The paper presents a proposal for the specification and verification of systems of communicating agents in a temporal logic. The proposal is based on a social approach to agent communication, where communication is described in terms of changes to the social state, and interaction protocols are defined by a set of temporal constraints, which specify the effects and preconditions of the communicative actions on the social state. The paper addresses the problem of combining protocols to define new more specialized protocols and exploits this idea in the specification of clinical guidelines.

I. INTRODUCTION

Agent technology has been rapidly developing in the last decade to answer the needs for new conceptual tools for modelling and developing complex software systems and it has given rise to a large amount of literature [6]. Autonomous agents can communicate, cooperate and negotiate using commonly agreed communication languages (ACLs) and protocols. The issue of interoperability has lead to the development of standardized agent communication languages, including KQML [21] and FIPA-ACL [4]. One of the central issues in the field concerns the specification of conversation policies, which govern the communication between software agents in an agent communication language (ACL). Conversation policies (or interaction protocols) define stereotypical interactions in which ACL messages are used to achieve communicative goals.

The specification of interaction protocols has been traditionally done by making use of finite state machines, but the transition net approach has been soon recognized to be too rigid to allow for the flexibility needed in agent communication [24], [16]. For these reasons, several proposals have been put forward to address the problem of specifying (and verifying) agent protocols in a flexible way. One of the most promising approaches to agent communication, first proposed by Singh [28], is the social approach [1], [8], [18], [24]. In the social approach, communicative actions affect the “social state” of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the permissions and the commitments of the agents.

In this paper we present a temporal approach to the specification and verification of interaction protocols among agents. Temporal logics are extensively used in the area of reasoning about actions and planning [2], [14], [11], [26], [3], and, in particular, they have been used in the specification and in the verification of systems of communicating agents. In [34], [22]

agents are written in MABLE, an imperative programming language, and the formal claims about the system are expressed using a quantified linear time temporal BDI logic and can be automatically verified by making use of the SPIN model checker. Guerin in [17] defines an agent communication framework which gives agent communication a grounded declarative semantics. In such a framework, temporal logic is used for formalizing temporal properties of the system. Our theory for reasoning about communicative actions is based on the Dynamic Linear Time Temporal Logic (DLTL) [19], which extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. As a difference with [34] we adopt a social approach to agent communication. The dynamics of the system emerges from the interactions of the agents, which must respect permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and it is well suited for dealing with “open” multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

The paper provides an overview of the approach developed in [12], [13], and describes the different kinds of verification problems which can be addressed, which can be formalized either as validity or as satisfiability problems in DLTL. These verification tasks can be automated by making use of Büchi automata. In particular, we can make use of the tableau-based algorithm presented in [10] for constructing a Büchi automaton from a DLTL formula. The construction of the automata can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automata is, in the worst case, exponential in the size of the input formula. We discuss the applicability of this approach to the specification of clinical guidelines.

II. DYNAMIC LINEAR TIME TEMPORAL LOGIC

In this section we shortly define the syntax and semantics of DLTL as introduced in [19]. In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the *until* operator is indexed by programs in Propositional Dynamic Logic (PDL).

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ , where $\omega = \{0, 1, 2, \dots\}$ and let ϵ denote the empty word. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^* . Moreover, we denote by

\leq the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by $prf(u)$ the set of finite prefixes of u .

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[\cdot]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$, which is defined as usual.

Let $\mathcal{P} = \{p_1, p_2, \dots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) is defined as follows:

$$DLTL(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ) and π ranges over $Prg(\Sigma)$.

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α , the satisfiability of a formula α at τ in M , written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'^1$, $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at τ if “ α until β ” is true on a finite stretch of behavior which is in the linear time behavior of the program π .

The derived modalities $\langle \pi \rangle$ and $[\pi]$ can be defined as follows: $\langle \pi \rangle \alpha \equiv \top \mathcal{U}^\pi \alpha$ and $[\pi] \alpha \equiv \neg \langle \pi \rangle \neg \alpha$.

Furthermore, if we let $\Sigma = \{a_1, \dots, a_n\}$, the \mathcal{U} , \bigcirc (next), \diamond and \square operators of LTL can be defined as follows: $\bigcirc \alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$, $\alpha \mathcal{U} \beta \equiv \alpha \mathcal{U}^{\Sigma^*} \beta$, $\diamond \alpha \equiv \top \mathcal{U} \alpha$, $\square \alpha \equiv \neg \diamond \neg \alpha$, where, in \mathcal{U}^{Σ^*} , Σ is taken to be a shorthand for the program $a_1 + \dots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [19], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω -sequences.

III. PROTOCOL SPECIFICATION

In the social approach an interaction protocol is specified by describing the effects of communicative actions on the social state, and by specifying the permissions and the commitments that arise as a result of the current conversation state.

Let us shortly recall the action theory developed in [11] that we use for the specification of interaction protocols.

Let \mathcal{P} be a set of atomic propositions, the *fluents*. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l , such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by Lit the set of all fluent literals.

¹We define $\tau \leq \tau'$ iff $\exists \tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

A *domain description* D is defined as a tuple (Π, \mathcal{C}) , where Π is a set of *action laws* and *causal laws*, and \mathcal{C} is a set of *constraints*.

Action laws in Π have the form: $\Box(\alpha \rightarrow [a]\beta)$, with $a \in \Sigma$ and α, β arbitrary formulas, meaning that executing action a in a state where precondition α holds causes the effect β to hold.

Causal laws in Π have the form: $\Box((\alpha \wedge \bigcirc \beta) \rightarrow \bigcirc \gamma)$, meaning that if α holds in a state and β holds in the next state, then γ also holds in the next state. Such laws are intended to express “causal” dependencies among fluents.

Constraints in \mathcal{C} are arbitrary temporal formulas of DLTL. In particular, the set of constraints includes *precondition laws* of the form: $\Box(\alpha \rightarrow [a]\perp)$, meaning that the execution of an action a is not possible if α holds. (i.e. there is no resulting state following the execution of a if α holds). Observe that, when there is no precondition law for an action, the action is executable in all states.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state. To cope with the *frame problem*, the laws in Π , describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in \mathcal{C} and given a special treatment. In [11], we defined a completion construction which, given a domain description, introduces frame axioms in the style of the successor state axioms introduced by Reiter [27]. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. In the following we call $Comp(\Pi)$ the completion of a set of laws Π .

Let us now provide the specification of the Contract Net protocol [4].

Example 1: The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task.

Let us consider first the simplest case where we have only two agents: the manager (M) and the participant (P). The two agents share all the communicative actions, which are: *cfp* (the manager issues a call for proposals for task T), *accept* and *reject* whose sender is the manager, *refuse* and *propose* whose sender is the participant, *inform_done* by which the participant informs the manager that the task has been executed and *end_protocol* by which the manager declares the completion of the protocol.

The social state contains the following domain specific fluents: *CN* (which is true during the execution of the protocol), *task* (whose value is true after the task has been announced), *replied* (the participant has replied), *proposal* (the participant has sent a proposal), *acc_rej* (the manager has sent an accept or reject message to the participant) *accepted* (the manager has accepted the proposal of participant) and *done* (the participant

has performed the task). Such fluents describe observable facts concerning the execution of the protocol.

We also introduce special fluents to represent *base-level commitments* of the form $C(i, j, \alpha)$, meaning that agent i is committed to agent j to bring about α , where α is an arbitrary formula, or they can be *conditional commitments* of the form $CC(i, j, \beta, \alpha)$ (agent i is committed to agent j to bring about α , if the condition β is brought about). The two kinds of base-level and conditional commitments we allow are essentially those introduced in [35]. For modelling the Contract Net example we introduce the following commitments

$$\begin{aligned} &C(P, M, \text{replied}) \quad C(M, P, \text{acc_rej}) \\ &C(i, M, \text{done}) \quad C(M, P, \text{task}) \end{aligned}$$

and conditional commitments

$$\begin{aligned} &CC(P, M, \text{task}, \text{replied}) \\ &CC(M, P, \text{proposal}, \text{acc_rej}) \\ &CC(i, M, \text{accepted}, \text{done}). \end{aligned}$$

Some reasoning rules have to be defined for cancelling commitments when they have been fulfilled and for dealing with conditional commitments. We introduce the following *causal laws*:

$$\begin{aligned} &\Box(\Box\alpha \rightarrow \Box\neg C(i, j, \alpha)) \\ &\Box(\Box\alpha \rightarrow \Box\neg CC(i, j, \beta, \alpha)) \\ &\Box((CC(i, j, \beta, \alpha) \wedge \Box\beta) \rightarrow \\ &\quad \Box(C(i, j, \alpha) \wedge \neg CC(i, j, \beta, \alpha))) \end{aligned}$$

A commitment (or a conditional commitment) to bring about α is cancelled when α holds, and a conditional commitment $CC(i, j, \beta, \alpha)$ becomes a base-level commitment $C(i, j, \alpha)$ when β has been brought about.

Let us now describe the effects of communicative actions by the following *action laws*:

$$\begin{aligned} &\Box[\text{cfp}](\text{task} \wedge \text{CN} \wedge CC(M, P, \text{proposal}, \text{acc_rej})) \\ &\Box[\text{accept}]\text{acc_rej} \\ &\Box[\text{reject}]\text{acc_rej} \\ &\Box[\text{refuse}]\text{replied} \\ &\Box[\text{propose}](\text{replied} \wedge \text{proposal} \wedge \\ &\quad CC(P, M, \text{accepted}, \text{done})) \\ &\Box[\text{inform_done}]\text{done} \\ &\Box[\text{end_protocol}(\text{CN})]\neg \text{CN} \end{aligned}$$

The laws for action *cfp* add to the social state the information that a call for proposal has been done for the *task*, and that, if the manager receives a proposal, it is committed to accept or reject it.

The permissions to execute communicative actions in each state are determined by social facts. We represent them by precondition laws. Preconditions on the execution of action *accept* can be expressed as:

$$\Box(\neg \text{CN} \vee \neg \text{proposal} \vee \text{acc_rej} \rightarrow [\text{accept}]\perp)$$

meaning that action *accept* cannot be executed outside the protocol, or if a proposal has not been done, or if the manager has already replied. Similarly we can give the *precondition laws* for the other actions:

$$\begin{aligned} &\Box(\neg \text{CN} \vee \text{task} \rightarrow [\text{cfp}]\perp) \\ &\Box(\neg \text{CN} \vee \neg \text{proposal} \vee \text{acc_rej} \rightarrow [\text{reject}]\perp) \end{aligned}$$

$$\begin{aligned} &\Box(\neg \text{CN} \vee \neg \text{task} \vee \text{replied} \rightarrow [\text{refuse}]\perp) \\ &\Box(\neg \text{CN} \vee \neg \text{task} \vee \text{replied} \rightarrow [\text{propose}]\perp) \\ &\Box(\neg \text{CN} \vee \neg \text{accepted} \vee \text{done} \rightarrow [\text{inform_done}]\perp) \\ &\Box(\neg \text{CN} \vee \neg \text{task} \rightarrow [\text{end_protocol}(\text{CN})]\perp) \end{aligned}$$

The precondition law for action *propose* (*refuse*) says that a proposal can only be done if a task has already been announced and the participant has not already replied. The last law says that the manager cannot issue a new call for proposal if a task has already been announced.

In the following we will denote Perm_i (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent i is the sender.

Assume now that we want the participant to be committed to reply to the task announcement. We can express it by adding the following conditional commitment to the initial state of the protocol: $CC(P, M, \text{task}, \text{replied})$. Furthermore the manager is committed initially to issue a call for proposal for a task. We can define the *initial state* Init of the protocol as follows:

$$\begin{aligned} &\{\neg \text{CN}, \neg \text{task}, \neg \text{replied}, \neg \text{proposal}, \neg \text{done}, \\ &CC(P, M, \text{task}, \text{replied}), C(M, P, \text{task})\} \end{aligned}$$

In the following we will be interested in those execution of the protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i, j, \alpha)$ will be fulfilled by the following constraint:

$$\Box(C(i, j, \alpha) \rightarrow \text{CN} \mathcal{U} \alpha)$$

We will call Com_i the set of constraints of this kind for all commitments of agent i . Com_i states that agent i will fulfill all the commitments of which he is the debtor.

Given the above rules, the domain description $D = (\Pi, \mathcal{C})$ of a protocol is defined as follows: Π is the set of the action and causal laws given above, and $\mathcal{C} = \text{Init} \wedge \bigwedge_i (\text{Perm}_i \wedge \text{Com}_i)$ is the set containing the constraints on the initial state, the permissions Perm_i and the commitments Com_i of all the agents (the agents P and M , in this example).

Given a domain description D , let the completed domain description $\text{Comp}(D)$ be the set of formulas $(\text{Comp}(\Pi) \wedge \text{Init} \wedge \bigwedge_i (\text{Perm}_i \wedge \text{Com}_i))$. The runs of the system according to the protocol are the linear models of $\text{Comp}(D)$. Observe that in these protocol runs all permissions and commitments are fulfilled. However, if Com_j is not included for some agent j , the runs may contain commitments which have not been fulfilled by j .

IV. PROTOCOL VERIFICATION

Different kinds of verification problems can be addressed, given the specification of a protocol by a domain description.

A. Verifying agents compliance at runtime

We are given a history $\tau = a_1, \dots, a_n$ of the communicative actions executed by the agents, and we want to check the compliance of that execution with the protocol. Namely, we want to verify that the history τ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the

protocol. This problem can be formalized as a satisfiability problem. The formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge \langle a_1; a_2; \dots; a_n \rangle \top$$

(where i ranges on all the agents involved in the protocol) is satisfiable if it is possible to find a run of the protocol starting with the action sequence a_1, \dots, a_n .

B. Verifying protocol properties

Proving that the protocol satisfies a given (temporal) property φ can be formalized as a validity check. The formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \rightarrow \varphi. \quad (1)$$

is valid if all the runs of the protocol satisfy φ . Observe that, all the agents are assumed to be compliant with the protocol. As an example of property to be checked, we consider the property of termination of the protocol. After the manager has announced a task, the protocol will eventually arrive to completion. This property can be formalized by the temporal formula:

$$\varphi = \Box[cfp] \Diamond \neg CN$$

meaning that, always, after a call for proposal has been issued by the manager, the protocol will eventually reach a state in which the proposition CN is false, i.e. the protocol is finished, for all possible runs of the protocol.

C. Verifying the compliance of an agent with the protocol at compile-time

When the program executed by an agent is given (or, at least, its logical specification is given), we are faced with the problem of verifying if the agent is compliant with the protocol, that is, to verify if the agent's program respects the protocol. Solving this problem requires: first to provide an *abstract* specification of the behavior (program) of the agent; and, second, to check that all the executions of the agent program satisfy the specification of the protocol, assuming that the other agents are compliant with the protocol.

In the general case, addressing this problem requires to move to the Product Version of DTL [13]. However, for protocols involving two agents, where all fluents and all actions of the social state are shared by both agents, this verification problems can be represented in DTL.

In DTL the behavior of an agent can be specified by making use of complex actions (regular programs). Consider for instance the following program π_P for the participant:

$$\begin{aligned} &[\neg end?; ((cfp; eval_task; (\neg ok?; refuse + \\ &\quad ok?; propose)) + \\ &\quad reject + \\ &\quad (accept; do_task; inform_done) + \\ &\quad (end_protocol(CN); exit))]^*; end? \end{aligned}$$

The participant cycles and reacts to the messages received by the manager: for instance, if the manager has issued a call for proposal, the participant can either refuse or make a

proposal according to his evaluation of the task; if the manager has accepted the proposal, the participant performs the task; and so on.

The state of the agent is obtained by adding to the fluents of the protocol the following local fluents: *end*, which is initially false and is made true by action *exit*, and *ok* which says if the agent must make a bid or not. The local actions are *eval_task*, which evaluates the task and sets the fluent *ok* to true or false, *do_task* and *exit*. Furthermore, *end?* and *ok?* are test actions.

The program of the participant can be specified by a domain description $Prog_P = (\Pi_P, \mathcal{C}_P)$, where Π_P is a set of action laws describing the effects of the private actions of the participant. For instance, the action *exit* sets the proposition *end* _{i} to true:

$$\Box[exit]end$$

The set of constraints \mathcal{C}_P contains *Init_P* which provides the initial values for the local fluents ($\neg end$, $\neg ok$) of the participant as well as the formula $\langle \pi_P \rangle \top$ stating that the program of the participant is executable in the initial state.

To prove that the participant is compliant with the protocol, i.e. that all executions of program π_P satisfy the specification of the protocol, we cannot consider the program π_P alone. In fact, it is easy to see that the correctness of the behavior of the participant depends on the behavior of the manager. Since we don't know its internal behavior, we will assume that the manager respects its public behavior, i.e. that it respects its permissions and commitments in the protocol specification.

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C})$ be the domain description describing the protocol, as defined above. The formula

$$(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \mathcal{C}_P) \rightarrow (Perm_P \wedge Com_P)$$

is valid if in all the behaviors of the system, in which the participant executes its program π_P and the manager (whose internal program is unknown) respects the protocol specification (in particular, its permissions and commitments), the permissions and commitment of the participant are also satisfied.

D. Proofs and model checking in DTL

The above verification and satisfiability problems can be solved by extending the standard approach for verification and model-checking of Linear Time Temporal Logic, based on the use of Büchi automata. An approach for constructing a Büchi automaton from a DTL formula making use of a tableau-based algorithm has been proposed in [10]. The construction of the states of the automaton is similar to the standard construction for LTL [9], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automaton is, in the

worst case, exponential in the size of the input formula, but in practice it is much smaller.

Standard model checking techniques [5] cannot be immediately applied to our approach, because protocols are formulated as sets of properties rather than as programs. Furthermore, in principle, with DTL we do not need to use model checking, because programs and domain descriptions can be represented in the logic itself, as we have shown in the previous section. However representing everything as a logical formula can be rather inefficient from a computational point of view. In particular all formulas of the domain description are universally quantified, and this means that our algorithm will have to propagate them from each state to the next one, and to expand them with the tableau procedure at each step.

Therefore we have adapted model checking to the proof of the formulas given in the previous section, by deriving the model from the domain theory in such a way that the model describes all possible runs allowed by the domain theory. In particular, we can obtain from the domain description a function $next_state_a(S)$, for each action a , for transforming a state in the next one, and then build the model (an automaton) by repeatedly applying these functions starting from the initial state. We can then proceed as usual to prove a property φ by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

An alternative way for applying this approach in practice, is to make use of existing model checking tools. In particular, by translating DTL formulas into LTL formulas, it would be possible to use LTL-based model checkers such as for instance SPIN [20]. Although in general DTL is more expressive than LTL, many protocol properties, such as for instance fulfillment of commitments, can be easily expressed in LTL.

We have done some experiments with the model checker SPIN on proving properties of protocols expressed according to the approach presented in this paper. The model is obtained as suggested above by formulating the domain description as a PROMELA program, which describes all possible runs allowed by the domain theory. Properties and constraints are expressed as LTL formulas. In the case of verification of compliance of an agent implementation with the protocol, we have used different PROMELA processes for representing the agent and the protocol. The representation of the agent is derived from its regular program.

V. AN APPLICATION TO CLINICAL GUIDELINES

Clinical guidelines can be roughly defined as frameworks for specifying the "best" clinical procedures and for standardizing them. Clinical guidelines play different roles in the clinical process: for example, they can be used to support physicians in the treatment of diseases, or for critiquing, for evaluation, and for education purposes. Many different systems and projects have been developed in recent years in order to realize computer-assisted management of clinical guidelines (see e.g., [15], [7]). GLARE (Guidelines Acquisition, Representation and Execution) [29], [31] is one of such

domain-independent systems. GLARE is being developed by a group of computer scientists from Università del Piemonte Orientale and Università di Torino, in collaboration with Azienda Ospedaliera S. Giovanni Battista in Torino, one of the largest hospitals in Italy. Despite the system is basically a research product, whose features are continuously refined and updated, the facilities it embeds have been formally tested or at least carefully examined by physicians. Some of the peculiar features of GLARE (with respect to the other computer-based approaches to clinical guidelines in the literature) are its decision-making facilities, which also involve advanced decision theory features [32], and its treatment of temporal constraints [30]. Despite the fact that several specialized "reasoning" facilities are provided by GLARE (see, e.g., [30], [32]), extensive logical reasoning capabilities such as the one which can be provided by theorem proving and/or model checking techniques can provide critical advances (see also [23]). We thus started to analyze (i) how clinical guidelines (such as the one represented by the GLARE system) can be modeled in our framework (ii) how the reasoning facilities provided by the model checker can be exploited within the clinical application environment.

As regards modeling, clinical guidelines are a hierarchical description of clinical procedures. At the lower level, they are basically composed by sequences of elementary actions (corresponding to actions to be executed on the specific patient) and decision actions needed to choose among alternative paths. All the elementary actions in a chosen path must be necessarily executed, unless their preconditions are not satisfied by the patient's data. This can be easily modeled by making use of precondition laws and obligations. Decisions are the core elements in clinical guidelines and are preceded by a data acquisition phase, which can be modeled as an interaction between the physician executing the guideline, the clinical record containing the patient data and, possibly, laboratories, which can be modeled as follows. The physician sends a data request to the database containing clinical records, which is committed to send back the requested data (if available) together with a timestamp stating their time of validity. If the data are not available or not up to date, the physician asks for them to the proper laboratories and waits for the answers. When all the up to date data are available, the decision process can start. In the GLARE approach, decision is modeled as an interaction between the system and the physician. On the basis of the decision criteria embodied in the guideline, and of the patient's data, the system proposes to the physician the subset of alternative paths suggested for the given patient. The physician can commit to one of the suggested alternatives or even to a non suggested one. In the latter case, however, the system sends a warning to the physician.

As regards reasoning, model checking can be used in order to instantiate a guideline on a specific patient, for instance, by checking, on the basis of the patient data, whether there are executable paths. Analogously, guidelines can be contextualized to specific hospitals, considering locally available laboratories and resources. Moreover, model checking capabilities can be

used to look for executable paths which satisfy a given set of requirements (concerning e.g. costs, execution times, goals and intention).

VI. CONCLUSIONS

In the paper we have presented an approach to the specification and verification of interaction protocols in a multiagent system that has been developed in the context of the national project PRIN 2003 “Logic-based development and verification of multi-agent systems”. We are currently investigating the applicability of the approach, on the one hand to the specification and verification of clinical guidelines and, on the other hand, to the specification and verification of Web Services, with a particular regard to the problem of service composition.

REFERENCES

- [1] M. Alberti, D. Daolio and P. Torroni. Specification and Verification of Agent Interaction Protocols in a Logic-based System. *SAC'04*, March 2004.
- [2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998.
- [3] D. Calvanese, G. De Giacomo and M.Y.Vardi. Reasoning about Actions and Planning in LTL Action Theories. In *Proc. KR'02*, 2002.
- [4] FIPA Contract Net Interaction Protocol Specification, 2002. Available at <http://www.fipa.org>.
- [5] E.M.Clarke, O.Grumberg, and D. Peled, *Model Checking*, MIT Press, 2000.
- [6] F.Dignum and M.Greaves. Issues in Agent Communication: An Introduction”. In F.Dignum and M.Greaves (Eds.), *Issues in Agent Communication*, LNAI 1916, pp. 1-16, 1999.
- [7] Special Issue on Workflow Management and Clinical Guidelines, D.B. Fridsma (Guest ed.), *JAMIA*, 22(1), 1-80, (2001).
- [8] N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *Proc. AAMAS'03*, Melbourne, pp. 520–527, 2003.
- [9] R. Gerth, D. Peled, M.Y.Vardi and P. Wolper. Simple On-the-fly Automatic verification of Linear Temporal Logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, Warsaw, June 1995, North Holland.
- [10] L. Giordano and A. Martelli. On-the-fly Automata Construction for Dynamic Linear Time Temporal Logic. *TIME 04*, June 2004.
- [11] L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In *The Logic Journal of the IGPL*, Vol. 9, No. 2, pp. 289-303, March 2001.
- [12] L. Giordano, A. Martelli, and C. Schwind. Verifying Communicating Agents by Model Checking in a Temporal Action Logic. *JELIA 2004*, Lisbon, Portugal, September 27-30, 2004, pp. 57-69.
- [13] L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, Accepted for publication, 2005.
- [14] F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. The 5th European Conf. in Planning (ECP'99)*, pp.1–20, Durham (UK), 1999.
- [15] C. Gordon and J.P. Christensen, *Health Telematics for Clinical Guidelines and Protocols*. IOS Press, Amsterdam, 1995.
- [16] M. Greaves, H. Holmback and J. Bradshaw. What Is a Conversation Policy?. *Issues in Agent Communication*, LNCS 1916 Springer, pp. 118-131, 2000.
- [17] F. Guerin. Specifying Agent Communication Languages. PhD Thesis, Imperial College, London, April 2002.
- [18] F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems*, Springer LNAI 2650, pp. 98–112, 2003.
- [19] J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999
- [20] G.J. Holzmann *The SPIN Model Checker. Primer and Reference Manual*. Addison-Wesley, 2003
- [21] Y. Labrou and T. Finin. A semantic approach for KQML - a general purpose communication language for software agents. In *3rd Int Conf. on Information and Knowledge Management, CIKM'94*, pp.447-455, 1994.
- [22] M.P. Huget and M. Wooldridge. Model Checking for ACL Compliance Verification. *ACL 2003*, Springer LNCS 2922, pp. 75–90, 2003.
- [23] M. Marcos, M. Balser, A. ten Teije, F. van Harmelen, C. Duelli. Experiences in the formalisation and verification of medical protocols, *AIME'03*.
- [24] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols: new trends in agent communication languages. In *The Knowledge Engineering Review*, 17(2):157-179, June 2002.
- [25] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-11)*, May, 2002.
- [26] M.Pistore and P.Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. *Proc. IJCAI'01*, Seattle, pp.479-484, 2001.
- [27] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed., pages 359–380, Academic Press, 1991.
- [28] M. P. Singh. A social semantics for Agent Communication Languages. In *IJCAI-98 Workshop on Agent Communication Languages*, Springer, Berlin, 2000.
- [29] P. Terenziani, G. Molino, and M. Torchio. A Modular Approach for Representing and Executing Clinical Guidelines. *Artificial Intelligence in Medicine* 23, 249-276, 2001.
- [30] P. Terenziani, C. Carlini, S. Montani. Towards a Comprehensive Treatment of Temporal Constraints in Clinical Guidelines. *Proc. TIME'02*, Manchester, UK, IEEE Press, 20-27, 2002.
- [31] Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G., Correndo, G. A context-adaptable approach to clinical guidelines. *Proc. MEDINFO'04*, M. Fieschi et al. (eds), Amsterdam, IOS Press, (2004), 169-173.
- [32] P. Terenziani, S. Montani, A. Bottrighi. Exploiting Decision Theory for Supporting Therapy Selection in Computerized Guidelines. *Proc. Int'l Conf. Artificial Intelligence in Medicine Europe*, LNCS, Springer Verlag, 2005.
- [33] P.Traverso and M.Pistore. Automated Composition of Semantic Web Services into Executable Processes. *Proc. Third International Semantic Web Conference (ISWC2004)*, November 9-11, 2004, Hiroshima, Japan.
- [34] M. Wooldridge, M. Fisher, M.P. Huget and S. Parsons. Model Checking Multi-Agent Systems with MABLE. In *AAMAS'02*, pp. 952–959, Bologna, Italy, 2002.
- [35] P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, pp. 527–534, Bologna, Italy, 2002.

Personalization, verification and conformance for logic-based communicating agents

Matteo Baldoni, Cristina Baroglio, Alberto Martelli,
Viviana Patti, Claudio Schifanella, Laura Torasso
Dipartimento di Informatica
Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
Email: {baldoni,baroglio,mrt,patti,schi,ltorasso}@di.unito.it

Viviana Mascardi
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova
Via Dodecaneso, 35 — I-16146 Genova (Italy)
Email: mascardi@disi.unige.it

Abstract— This paper is an overview of the work that we have carried on in the last two years in the context of the MASSiVE project. The main research lines have concerned personalization of the interaction with web services, personalization of courseware, web services interoperability, and integrated environments for agent oriented software engineering. All of them can be seen as applications of different reasoning techniques to a declarative specification of interaction. A declarative specification makes the study of properties easy and allows a fast prototyping of applications. In particular, we applied reasoning about actions and change to the personalized selection and composition of web services and to the construction of courseware that satisfies the user's needs and goals. This kind of reasoning has also been integrated in the DCaseLP MAS prototyping environment. Declarative specifications have also been helpful to face the problem of proving policy conformance in a way that guarantees web service interoperability. Finally, the adoption of process languages for web services for expressing the procedural behavior of adaptive BDI-style agents have been explored.

I. INTRODUCTION

Computational logics and declarative languages are being rediscovered as a tool for some of the most innovative application areas: the Semantic Web and Web Services. By definition, the Semantic Web comprises a machine-shareable representation of knowledge, and it both requires the development of languages for expressing information in a machine-processable form, and the use of inferencing mechanisms that allow a content-aware navigation. The desired result is an overall behavior that is closer to the user's intuition and desire and the possible applications are really various, depending on the kind of resource that is described and on the tasks to be performed. On the other hand, it is getting more and more common describing and realizing applications as sets of cooperative services. This is the case, for example, for manufacturing processes, on-line markets, distributed network management. The traditional approach is based on a functional view, in which the different components require some specific input, and produce some specific output. The system's architecture is based on the principle of static-functional decomposition, where the interactions among the different components are given by their dependencies. Other approaches are, however, being studied which involve describing at a high-level the behavior of the services. The aim is to enable the adoption

of automated reasoning mechanisms for retrieving, composing, invoking services. One of these approaches is the Multi-Agent paradigm, in which the different components dynamically communicate and coordinate with each other, by means of declarative languages, to reach some common (or their own) goal. Among the social aspects, specifically relevant is the ability of expressing behavioral rules, aiming at controlling the organization of the system; communication protocols are the most significant example of such rules. Protocols are used to rule the agents' interaction, therefore, they can be used to check if a given agent can, or cannot, take part into the system, or to check whether the system is behaving as expected. In general, based on this abstraction, open systems can be realized, in which new components can dynamically join the system. The insertion of a new component in an execution context is determined according to some form of reasoning about its behaviour: it will be added provided that it satisfies the body of the rules within the system, intended as a society.

The researches that we have carried on the last two years tackle different aspects related to the Semantic Web and Web Services, in the setting of Multi-Agent Systems. In particular, we have extended the DyLOG language [13], which is the common tool used in all the branches of the research that we have carried on. The extension [9] mainly concerns the introduction of a communication kit aimed at tackling communication in a way that is fully integrated with the representation and reasoning mechanisms of the language. Each of the next sections describes one of the lines of research that we have been pursued and the work carried on in that context. Section II reports about work in the context of personalization of courseware; Section III discusses personalization in the service selection and composition processes; Section IV reports results concerning the proof of interoperability and conformance of services to a global description of their interaction; Section V describes the adoption of process languages for expressing the procedural behavior of adaptive BDI-style agents; Section VI describes an integrated environment for AOSE.

II. PERSONALIZATION OF COURSEWARE

Personalized information systems aim at giving the individual user optimal support in accessing, retrieving, and

storing information. The individual requirements of the user are to be taken into account in such different dimensions like the current task, the goal of the user, the context in which the user is requesting the information, the previous information requests or interactions, the working process s/he is involved in, the level of expertise, the device s/he is using to display the information, the bandwidth and availability of the communication channel, the abilities (disabilities or handicaps) of the user, his/her time constraints, and many, many more. Different research disciplines have contributed to explore personalization techniques and to evaluate their usefulness within various application areas: adaptive hypertext systems, collaborative filtering, recommender systems, artificial intelligence, uncertainty management, and so forth. In this section we will focus on an e-learning scenario and see how reasoning can help personalization in this context, beginning with the annotation of the learning resources. exploit a new level of knowledge thus allowing a better personalization.

A *learning object* can profitably be used if the learner has a given set of prerequisite competences; by using it, the learner will acquire a new set of competences. It is, therefore, appropriate to interpret learning objects as *actions*. The idea that we have proposed is to introduce at the level of the learning objects, some additional annotation for describing both their *pre-requisites* and their *effects* and to do this by exploiting standard representation languages, like LOM, and *ontologies*, for using terms with a clear and sharable meaning.

The proposed annotation expresses a set of *learning dependencies* between *ontological terms*, dependencies which can be expressed in a declarative formalism, and can be used by a reasoning system. So, given a set of learning objects, each annotated in this way, it is possible to use the standard planners, developed by the Artificial Intelligence community (for instance, the well-known Graphplan [16]), for building the reading sequences.

General-purpose planners search a sequence of interest in the whole space of possible solutions and allow the construction of learning objects on the basis of any learning goal. This is not always adequate in an educational application framework, where the set of learning goals of interest is fairly limited and the experience of the teachers in structuring the courses and the learning materials is important. This kind of constraint cannot be exploited by a general-purpose planner, being related to the strategy adopted by the teacher. The ideal solution is to express them as *rules* that specify an overall structure in terms of ontological terms (competences). We will call such rules *learning strategies*.

Given a set of learning strategies, it is possible to build a learning object by refining a general rule according to specific requirements and, in particular, by choosing those components that best fit the user. An emblematic example is preparing the material for a basic computer science course: the course may, in fact, have different contents depending on the kind of student to whom it will be offered (e.g. a Biology student, rather than a Communication Sciences student, rather than a Computer Science student). In particular, having a learning

strategy and a set of annotated learning objects, it is possible to apply *procedural planning* for assembling a reading path that is a sequence of learning resources that are annotated as required by the strategy. Opposite to general-purpose planners, procedural planning searches for a solution in the set of the possible executions of a learning strategy.

Since the strategy is based on competences, rather than on specific resources, the system might need to select between different courses, annotated with the same desired competence, which could equally be selected in building the actual learning path. This choice can be done based on external information, such as a user model, or it may be derive from a further interaction with the user. Decoupling the strategies from the learning objects results in a greater flexibility of the overall system, and simplifies the reuse of the learning objects. As well as learning objects, also learning strategies could be made public and shared across different systems. Results of these researches in the context of Massive are reported in [14], [6].

III. PERSONALIZATION OF THE INTERACTION WITH WEB SERVICES

In the last years distributed applications over the World-Wide Web have obtained wide popularity and uniform mechanisms have been developed for handling computing problems which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments have begun to coalesce around the *web service* paradigm, where a service can be seen as a component available over the web. Each service has an interface that is accessible through standard protocols and that describes its *interaction capabilities*, and it can be *combined* and *integrated* with others to develop new applications over the web.

In this scenario, one of the needs that have inspired recent research [15] is the study of declarative descriptions of web services, aimed at allowing forms of automated interoperation that include, on the one hand, the automation of tasks like matchmaking and execution, on the other, the automation of service *selection and composition*, in a way that is customized w.r.t. the *user's goals and needs*, a task that can be considered as a form of *personalization* [6]. Indeed, selection and composition not always are to be performed on the sole basis of general properties of the services themselves and of their interactive behavior, such as their category or their functional compositionality, but they should also take into account the user's intentions (and purposes) which both motivate and constrain the search or the composition. As a quick example, consider a service that allows buying products, alternatively paying cash or by credit card: a user might have preferences on the form of payment to enact. In order to decide whether or not buying at this shop, it is necessary to single out the specific course of interaction that allows buying cash. This form of personalization can be obtained by applying *reasoning techniques* on a description of the service process. Such a description must have a well-defined meaning for all the parties involved. In this issue it is possible to distinguish three necessary components:

- web services capabilities must be represented according to some declarative formalism with a well-defined semantics, as also recently observed by van der Aalst [43];
- automated tools for reasoning about such a description and performing tasks of interest must be developed;
- in order to gain flexibility in fulfilling the user's request, reasoning tools should represent such requests as *abstract goals*.

The approach that we propose in [8] inherits from the experience of the research community that studies MAS and, in particular, logic-based formalizations of interaction aspects. Indeed, communication has intensively been studied in the context of formal theories of agency [24], [23] and a great deal of attention has been devoted to the definition of standard agent communication languages (ACL), e.g. FIPA [29] and KQML [28]. Recently, most of the efforts have been devoted to the definition of formal models of interaction among agents, that use *conversation protocols*. The interest for protocols is due to the fact that they improve the interoperability of the various components (often separately developed) and allow the verification of compliance to the desired standards.

The basic idea is to consider a service as a software agent and the problem of composing a set of web services as the problem of making a set of software agents interact and cooperate within a multiagent system (or MAS). This interpretation is, actually, quite natural, and shared in proposals that are closer to the agent research community and more properly set in the *Semantic Web* research field [18], [41]. Among the other proposals, let us recall the OWL-S [37] (formerly DAML-S) experience. In [18] the goal of providing greater expressiveness to service description in a way that can be *reasoned about* has been pursued by exploiting agent technologies based on the *action metaphor*. In particular, at the level of abstraction of the process model, a service is described as atomic, simple or composite in a way inspired by the agent language GOLOG and its extensions [35], [30], [36]; therefore reasoning techniques supported by the language are used to produce composite and customized services.

On this line, we have studied the possible benefits provided by a *declarative description* of their communicative behavior, in terms of personalization of the service selection and composition. Indeed we claim that a better personalization can be achieved by focussing on the abstraction of web services as entities, that communicate by following predefined, public and sharable interaction protocols and by allowing agents to reason about high level descriptions of the *interaction protocols* followed by web services. We model the interaction protocols provided by web services by a set of logic clauses, thus at high (not at network) level. The language we have used for describing conversation protocols, is based on an extension of the agent programming language DyLOG [13], [7].

Having a logic specification of the protocol, it is possible to reason about the effects of engaging specific conversations. In particular, we propose to use techniques for *reasoning about actions* for performing the automatic selection and composition of web services, in a way that is customized w.r.t.

the users's request. Communication can, in fact, be considered as the behavior resulting from the application of a special kind of actions: *speech acts*. The reasoning problem that this proposal faces can intuitively be described as looking for an answer to the question "Is it possible to make a deal with this service respecting the user's goals?". Given a logic-based representation of the service policies and a representation of the customer's needs as abstract goals, expressed by a logic formula, logic programming reasoning techniques are used for understanding if the constraints of the customer fit in with the policy of the service.

Our proposal can be considered as an approach based on the process ontology, a *white box* approach in which part of the behavior of the services is available for a rational inspection. A description of the communicative behavior by policies is definitely richer than the list of input and output, precondition and effect properties usually taken into account for the matchmaking. Actually, the approach can be considered as a *second step* in the matchmaking process, which narrows a set of already selected services and performs a *customization* of the interaction with them.

Moreover the idea of focussing on abstract descriptions of the communicative behavior is, actually, a novelty also with respect to other proposals that are set in the Semantic Web research field. The deductive process on communication policies can exploit more semantic information: in fact, it does not only take into account the pre- and post-conditions, as in OWL-S proposal, it also takes into account the complex communicative behavior of the service.

IV. WEB SERVICE INTEROPERABILITY

According to *Agent-Oriented Software Engineering* [33], a distinction is made between the global and the individual points of view of interaction. The *global* viewpoint is captured by an *abstract protocol*, expressed by formalisms like AUMML, automata or Petri Nets. The *local* viewpoint, instead, regards one of the agents and is captured by its policy; being part of the agent's implementation, the policy is usually written in some executable language. Having these two levels of description it is possible to decide whether an agent can take a role in an interaction. In fact, this problem can be read as the problem of proving if the agent's policy *conforms* to the abstract protocol specification.

A similar need of distinguishing a global and a local view of the interaction is recently emerging also in the area of *Service Oriented Architectures*. In this case a distinction is made between the *choreography* of a set of services, i.e. a global specification of the way in which they should interact, and the concept of *behavioral interface*, seen as the specification of the interaction from the point of view of the individual service. The recent W3C proposal of the choreography language WS-CDL [45], well-characterized and distinguished from languages for business process representation, like BPEL, is emblematic.

Taking this perspective, choreographies and agent communication protocols undoubtedly share a common purpose. In

fact, they both aim at expressing *global interaction protocols*, i.e. rules that define the global behavior of a system of cooperating parties. The respect of these rules guarantees the interoperability of the parties (i.e. the capability of *actually* producing an interaction), and that the interactions will satisfy given requirements.

In this context, one problem that becomes crucial is the development of formal methods for verifying if the behavior of a service respects a choreography. The applications would be various. A choreography could be used *at design time* (a priori) for verifying if the internal processes of a service enable it to participate appropriately in the interaction. At *run-time*, choreographies could be used to verify if everything is proceeding according to the agreements. A choreography could also be used unilaterally to detect exceptions (e.g. a message was expected but not received) or help a participant in sending messages in the right order and at the right time.

In the last years the agent community already started to face the two above mentioned kinds of conformance w.r.t. MASs [31] (e.g. see [25], [26], [11], [10] for *a priori* conformance, and [2] for *run-time* conformance). In the web service community the problem of conformance is arising only recently [21] because so far the focus has been posed on the specification of single services and on standards for their remote invocation. The new interest is emerging due to the growing need of making services, that are heterogeneous (in kind of platform or in language implementation), to interoperate. Therefore, there is a need of giving more abstract representations of the interactions that allow to perform reasoning in order to select and compose services disregarding the specific implementation details. Given our experience in the area of MASs, where the heterogeneity of the components is a fundamental characteristic, we agree with the observation by van der Aalst [43] that there is a need for a more declarative representation of the behaviour of services.

In this line, the work in [11], [10] about conformance of agent implementations w.r.t. protocol specifications has been adapted to the case of web services in [12]. In particular, in [12] we focus on testing *a priori conformance* and develop a framework based on the use of formal languages. In this framework a global interaction protocol (a choreography), is represented as a finite state automaton, whose alphabet is the set of messages exchanged among services. It specifies permitted conversations. Atomic services, that have to be composed according to the choreography, are described as finite state automata as well. Given such a representation we capture a concept of conformance that answers positively to all these questions: *is it possible to verify that a service, playing a role in a given global protocol, produces at least those conversations which guarantee interoperability with other conformant service? Will such a service always follow one of these conversations when interacting with the other parties in the context of the protocol? Will it always be able to conclude the legal conversations it is involved in?* Technically, the conformance test is based on the acceptance of both the service behavior and the global protocol by a special finite

state automaton. Briefly, at every point of a conversation, we expect that a conformant policy never utters speech acts that are not expected, according to the protocol, and we also expect it to be able to handle any message that can possibly be received, once again according to the protocol. However, the policy is not obliged to foresee (at every point of conversation) an outgoing message for every alternative included in the protocol (but it must foresee at least one of them).

The interesting characteristic of this test is that it guarantees the interoperability of services that are proved conformant *individually* and *independently* from one another. By interoperability we mean the capability of an agent of actually producing a conversation when interacting with another. The conformance test has been proved *decidable* when the languages used to represent all the possible conversations w.r.t. the policy and w.r.t. the protocol are *regular*.

The application of our approach is particularly easy in case a logic-based declarative language is used to implement the policies. In logic languages indeed policies are usually expressed by Prolog-like rules, which can be easily converted in a formal language representation. In [10] we show this by means of a concrete example where the language DyLOG [13], based on computational logic, is used for implementing the agents' policies. On the side of the protocol specification languages, currently there is a great interest in using informal, graphical languages (e.g. UML-based) for specifying protocols and in the translation of such languages in formal languages [22], [27]. By this translation it is, in fact, possible to prove properties that the original representation does not allow. In this context, in [11] we have shown an easy algorithm for translating AUML sequence diagrams to finite state automata thus enabling the verification of conformance. Of course, having a declarative representation of the choreographies as well, would help the proof of these properties in the context of the web services.

V. WEB SERVICE PROCESS LANGUAGES FOR BDI-STYLE AGENTS

The adoption of process languages for (semantic) WSs as a means for specifying the behaviour of agents and MASs is envisaged by a growing number of researchers working in the MAS community. For example, in [20] P. Buhler and J. M. Vidal discuss a technique for providing agent software with dynamically configured capabilities described with DAML-S, that can represent atomic or orchestrated WSs. In [19], the same authors advance the idea that BPEL can be used as a specification language for expressing the initial social order of a MAS, which can then intelligently adapt to changing environmental conditions. K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan suggest to extend the OWL-S Model Processing Language by adding to it a new statement called *exec* that takes a process model as input and executes it in order to support a broker agent in both discovery and mediation [42]. More recently, C. Walton [44] proposes to decompose agents into a stub that executes Agent Interaction Protocols and is responsible for communication between agents, and a body

which encapsulates the reasoning processes, and is encoded as a set of decision procedures. Both the stub and the body are implemented as WSs.

Our approach to the specification of the agents' behavioural knowledge by means of process languages for WSs is driven by our previous research on cooperative BDI agents, and thus differs from all the existing proposals discussed so far. In [3], we discuss the idea that BDI-style agents [38] can be extended with a built-in mechanism for retrieving plans from cooperative agents (thus becoming "*CooBDI*" agents), for example when no local plans suitable for achieving a certain desire are available. This feature turns out to be useful in many application fields such as: Personal Digital Assistants (PDAs), whose limited physical resources make dynamic loading and linking of code necessary; Self-repairing agents, namely agents situated in a dynamically changing software environment and able to identify the portions of their code that should be updated to ensure their correct functioning in the evolving environment; Digital butlers, i.e. agents that assist a human user in some task such as managing her/his agenda, filtering incoming e-mail, retrieving interesting information from the web; digital butlers adapt their behaviour to the user's needs by cooperating both with more experienced digital butlers, and with the assisted user.

We have implemented the ideas behind the *CooBDI* theory by means of WS technologies, obtaining what we named "*CooWS*" agents [17]. A *CooWS* agent adopts the following metaphor inspired by *CooBDI*.

- **Beliefs.** The variables local to the BPEL processes that constitute the body of the agent's plans can be considered as a metaphor for the agent's beliefs local to that plan, that are not explicitly represented.
- **Desires.** Desires may be either messages structured according to the FIPA ACL standard (<http://www.fipa.org/>), or unstructured Java strings.
- **Actions.** There are two kinds of actions: those that may appear inside the BPEL specification of the agent's plan body (that, in turn, may be delivery of ordinary events; achievement of new desires; and invocation of existing WSs by means of the BPEL `invoke` statement), and those that must be executed in case of success or failure of the achievement of a desire. Cooperative requests for plans are managed transparently to the agent, and do not belong to the set of actions that can be programmed by the user.
- **Plans.** Plans are defined by a unique plan identifier, a trigger (the desire for which the plan has been defined); a body (a BPEL process); and an access specifier (which may assume one of the three values *OnlyTrusted(Set)* – the plan may be shared only with the agents in the trusted agents set –, *Private* – the plan is private to the agent –, and *Public* – the plan may be shared with any agent).
- **Intentions.** An intention contains a stack of desires, a boolean attribute defining the intention's state (either active or suspended), and the success and failure actions. The set of plans currently available to the agent for man-

aging a given desire, is associated with the corresponding desire on the stack. The set of these "relevant" plans is generated by exploiting the cooperation mechanism (transparent to the user), thus retrieving both local and external plans useful for achieving the desire.

- **Events.** There are three kinds of events: cooperation, ordinary, and achieve events. A cooperation event is either a request, characterised by the desire for which the request has been issued, or a provide event, characterised by the set of plans that are relevant for the desire appearing in the corresponding request. Ordinary events consist of the reception of messages from other agents, while achieve events implement the plan nesting mechanism.

The implementation of the *CooWS* platform, downloadable from the web site <http://coows.altervista.org>, relies entirely on opensource tools that include ActiveBPEL, Apache Tomcat and Axis, jUDDI, UDDI4J, and MySQL. In order to validate the feasibility of our approach, we are currently working on the implementation of digital butlers that query Google (which can be accessed as a web service) to arrange travels and to organise meetings for their principals. The plans available to the digital butlers do not cover all the requests that may arrive from their principals, and the lack of plans for coping with an incoming request fires the collaborative exchange of plans.

In the future, we are willing to explore: 1) the ability to integrate an ontology into the system, so that matching between desires and triggers of plans can become more sophisticated than a simple comparison of strings; 2) the ability to dynamically update the set of trusted partners following reputation mechanisms such those described in [40].

VI. INTEGRATED ENVIRONMENTS FOR AGENT-ORIENTED SOFTWARE ENGINEERING

The correct and efficient engineering of heterogeneous, distributed, open, and dynamic applications is one of the technological challenges faced by Agent-Oriented Software Engineering (AOSE). The lack of mature methodologies, tools, and environments for agent-based system development limits the effectiveness and impact of AOSE [1].

MAS development requires engineering support for a diverse range of non-functional properties, such as understandability of the MAS at various conceptual levels, integrability of heterogeneous agent architectures, usability, re-usability, and testability. Creating one monolithic AOSE approach to support all these properties is not feasible. Rather, we expect different approaches to be suitable for modelling, verifying, or implementing various properties. By providing the MAS developer with an integrated set of languages and tools, and allowing for the choice of the most suitable language/tool to model, verify, or implement each property, we could make a step towards a modular approach to AOSE [34].

DCaseLP [4], [32] provides a prototyping environment where agents specified and implemented in a given set of languages can be seamlessly integrated. It also provides an AOSE methodology to guide the developer during the analysis

of the MAS requirements, its design, and the development of a working MAS prototype.

DCaseLP supports UML and AUML (<http://www.auml.org/>) for the specification of the general structure of the MAS, and Jess (<http://herzberg.ca.sandia.gov/jess/>), Java and tuProlog (<http://lia.deis.unibo.it/research/tuprolog/>) for the implementation of the agents.

As discussed in [4], DCaseLP adopts an existing multi-view, use-case driven and UML-based method in the phase of requirements analysis. Once the requirements of the application have been clearly identified, the developer can use UML and/or AUML to describe the interaction protocols followed by the agents, the general MAS architecture and the agent types and instances. Moreover, the developer can automatically translate the UML/AUML diagrams, describing the agents in the MAS, into Jess rule-based code. The Jess code obtained from the translation of AUML diagrams must be manually completed by the developer with the behavioural knowledge which was not explicitly provided at the specification level. The developer does not need to have a deep insight into rule-based languages in order to complete the Jess code, since he/she is guided by comments included in the automatically generated code.

The agents obtained by means of the manual completion of the Jess code are integrated into the JADE (Java Agent Development Framework, (<http://jade.tilab.com/>)) middleware. By integrating Jess into JADE, we were able to easily monitor and debug the execution of Jess agents thanks to the monitoring facilities that JADE provides. A recent extension of DCaseLP, discussed in [32], has been the integration of a Prolog implementation: tuProlog. The choice of tuProlog was due to two of its features:

- 1) it is implemented in Java, which makes its integration into JADE easier, and
- 2) it is very light, which ensures a certain level of efficiency to the prototype.

By extending DCaseLP with tuProlog we have obtained the possibility to execute agents, whose behavior is completely described by a Prolog-like theory, in the JADE platform. For this purpose, we have developed a library of predicates that allow agents specified in tuProlog to access the communication primitives provided by JADE: asynchronous send, asynchronous receive, and blocking receive (with and without timeout). Finally, a methodological integration of DyLOG into DCaseLP has been proposed in [5]. So far, the integration of DyLOG into DCaseLP is only “methodological” in the sense that it extends the set of languages supported by DCaseLP during the MAS engineering process and augments the verification capabilities of DCaseLP, without requiring any real integration of the DyLOG working interpreter into DCaseLP. Nevertheless, DyLOG can also be used to directly specify agents and execute them inside the DCaseLP environment, in order to exploit the distribution, concurrency, monitoring and debugging facilities that DCaseLP offers.

We have already tested – on a toy application – the ability of Jade, Jess and tuProlog agents to be integrated into the

same MAS and to communicate with each other. Currently, we are developing a much more sophisticated application in the electronic auctions field, whose basic building block are described in [39].

VII. CONCLUSIONS

Mainstream research in Web Services (WS) is looking at two main aspects: first, formally describing interactions among services (possibly over long periods of time and having multiple real-world effects, including legally binding actions); second, finding and combining services (e.g., by extending the simple catalogue contained in UDDI repositories with semantically rich descriptions and using the latter for automated composition via planning and for formal verification). As observed in AgentLink III, 2004, and by M. N. Huhns, 2002, much work made in the intelligent agents area can be applied to these issues.

One of the problems that we have studied is the verification of the a priori conformance of the communication policy of an agent (or web service) w.r.t. a general interaction protocol specification, that rules a system of cooperating parties. The interesting characteristic of the test that we have proposed is that it guarantees the interoperability of services that are proved conformant *individually* and *independently* from one another. It emerged that the application of our approach is particularly easy in case a logic-based declarative language is used to implement the policies.

For what concerns the specification languages, the modelling languages commonly used in the “requirements specification” and “software design” phases proposed in the AOSE community, like AUML, are not declarative and, as such, they do not provide any automatic proof mechanism. In this context it is interesting to study translations between modelling languages and languages with a formalized semantics to enable the use of the automatic proof mechanisms associated to them. For instance in [10] we have proposed the use of finite state automata as formal representation of protocols which supports the proof of conformance and an algorithm for translating a subset of AUML into finite state automata has been proposed in [11]. However this is just a first step and more research should be devoted to the issue of the transformation from semi-formal to formal specification languages.

We have studied how the above approach applies to some concrete domain such as web services and e-learning. In particular web services are an example of a highly dynamic application domain where a challenging problem that we have studied is the development of formal methods for verifying if the behavior of a single service respects a choreography. More specifically the problem consists in deciding if the internal processes of a service enable it to participate appropriately in the interaction encoded by a choreography. Another related problem that it would be interesting to address is the use of choreographies at *run-time* to verify that everything is proceeding according to the agreements. In this context a choreography could also be used unilaterally to detect exceptions (e.g. a message was expected but not received) or

help a participant in sending messages in the right order and at the right time. Also in this case there are logic techniques developed in the agent community that can be adapted to tackle the problem in the web service domain [2].

REFERENCES

- [1] AgentLink III, "Agent technology roadmap: Overview and consultation report," 2004.
- [2] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Specification and verification of agent interactions using social integrity constraints," in *Proc. of the Workshop on Logic and Communication in Multi-Agent Systems, LCMAS 2003*, ser. ENTCS, W. van der Hoek, A. Lomuscio, E. de Vink, and M. Wooldridge, Eds., vol. 85(2). Eindhoven, the Netherlands: Elsevier, 2003.
- [3] D. Ancona and V. Mascardi, "Coo-BDI: Extending the BDI model with cooperativity," in *Post-proc. of DALI'03*, 2004, pp. 109–134.
- [4] E. Astesiano, M. Martelli, V. Mascardi, and G. Reggio, "From Requirement Specification to Prototype Execution: a Combination of a Multiview Use-Case Driven Method and Agent-Oriented Techniques," in *Proc. of SEKE'03*, 2003, pp. 578–585.
- [5] M. Baldoni, C. Baroglio, I. Gungui, A. Martelli, M. Martelli, V. Mascardi, V. Patti, and C. Schifanella, "Reasoning about agents' interaction protocols inside DCasELP," in *Proc. of DALI 2004*, 2004, pp. 112–131.
- [6] M. Baldoni, C. Baroglio, and N. Henze, "Personalization for the Semantic Web," in *Reasoning Web*, ser. LNCS Tutorial, vol. 3564. Springer, 2005, pp. 173–212.
- [7] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about self and others: communicating agents in a modal action logic," in *Proc. of ICTCS'2003*, ser. LNCS, vol. 2841. Springer, 2003, pp. 228–241.
- [8] —, "Reasoning about interaction protocols for web service composition," M. Bravetti and G. Zavattaro, Eds. Elsevier Science Direct, 2004, pp. 21–36, vol. 105 of Electronic Notes in Theoretical Computer Science.
- [9] —, "Reasoning about interaction protocols for customizing web service selection and composition," *The Journal of Logic and Algebraic Programming*, 2005, accepted for publication after major revision.
- [10] —, "Verification of protocol conformance and agent interoperability," in *Pre-proc. of Sixth International Workshop on Computational Logic in Multi-Agent Systems, CLIMA VI*, 2005, pp. 12–27.
- [11] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Verifying protocol conformance for logic-based communicating agents," in *Proc. of 5th Int. Workshop on Computational Logic in Multi-Agent Systems, CLIMA V*, ser. LNCS, no. 3487, 2005, pp. 192–212.
- [12] —, "Verifying the conformance of web services to global interaction protocols: a first step," in *Proc. of 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005*, ser. LNCS, no. 3670, 2005, pp. 257–271.
- [13] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, vol. 41, no. 2–4, pp. 207–257, 2004.
- [14] M. Baldoni, C. Baroglio, V. Patti, and L. Torasso, "Reasoning about learning object metadata for adapting SCORM courseware," in *Int. Workshop on Engineering the Adaptive Web, EAW'04: Methods and Technologies for Personalization and Adaptation in the Semantic Web, Part I*, L. Aroyo and C. Tasso, Eds., Eindhoven, The Netherlands, August 2004, pp. 4–13.
- [15] A. Barros, M. Dumas, and P. Oaks, "A critical overview of the web services choreography description language(ws-cdl)," *Business Process Trends*, 2005, <http://www.bptrends.com>.
- [16] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.
- [17] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta, "CooWS: Adaptive BDI agents meet service-oriented computing," in *Proc. of the Int'l Conference on WWW/Internet*, 2005.
- [18] J. Bryson, D. Martin, S. McIlraith, and L. A. Stein, "Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web," 2002. [Online]. Available: citeseer.nj.nec.com/bryson02agentbased.html
- [19] P. Buhler and J. M. Vidal, "Adaptive workflow = web services + agents," in *Proc. of the Int'l Conference on Web Services*, 2003, pp. 131–137.
- [20] —, "Semantic web services as agent behaviors," in *Proc. of Agentcities: Challenges in Open Agent Environments*, 2003.
- [21] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, "Choreography and Orchestration: a synergic approach for system design," in *Proc. the 3rd Int. Conf. on Service Oriented Computing*, 2005.
- [22] L. Cabac and D. Moldt, "Formal semantics for auml agent interaction protocol diagrams," in *Proc. of AOSE 2004*, 2004, pp. 47–61.
- [23] F. Dignum, Ed., *Advances in agent communication languages*, ser. LNAI, vol. 2922. Springer-Verlag, 2004.
- [24] F. Dignum and M. Greaves, "Issues in agent communication," in *Issues in Agent Communication*, ser. LNCS, vol. 1916. Springer, 2000, pp. 1–16.
- [25] U. Endriss, N. Maudet, F. Sadri, and F. Toni, "Protocol conformance for logic-based agents," in *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, G. Gottlob and T. Walsh, Eds. Morgan Kaufmann Publishers, August 2003, pp. 679–684.
- [26] —, "Logic-based agent communication protocols," in *Advances in agent communication languages*, ser. LNAI, vol. 2922. Springer-Verlag, 2004, pp. 91–107, invited contribution.
- [27] R. Eshuis and R. Wieringa, "Tool support for verifying UML activity diagrams," *IEEE Trans. on Software Eng.*, vol. 7, no. 30, 2004.
- [28] T. Finin, Y. Labrou, and J. Mayfield, "KQML as an Agent Communication Language," in *Software Agents*, J. Bradshaw, Ed. MIT Press, 1995.
- [29] FIPA, "Communicative act library specification," FIPA (Foundation for Intelligent Physical Agents), Tech. Rep., 2002.
- [30] G. D. Giacomo, Y. Lesperance, and H. Levesque, "Congolog, a concurrent programming language based on the situation calculus," *Artificial Intelligence*, vol. 121, pp. 109–169, 2000.
- [31] F. Guerin and J. Pitt, "Verification and Compliance Testing," in *Communication in Multiagent Systems*, ser. LNAI, M. Huget, Ed., vol. 2650. Springer, 2003, pp. 98–112.
- [32] I. Gungui and V. Mascardi, "Integrating tuProlog into DCasELP to engineer heterogeneous agent systems," in *Proc. of CILC 2004*. Available at <http://www.disi.unige.it/person/MascardiV/Download/CILC04a.pdf.gz>.
- [33] M. P. Huget and J. Koning, "Interaction Protocol Engineering," in *Communication in Multiagent Systems*, ser. LNAI, H. Huget, Ed., vol. 2650. Springer, 2003, pp. 179–193.
- [34] T. Juan, M. Martelli, V. Mascardi, and L. Sterling, "Customizing AOSE methodologies by reusing AOSE features," in *Proc. of AAMAS'03*, 2003, pp. 113–120.
- [35] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, "GOLOG: A Logic Programming Language for Dynamic Domains," *J. of Logic Programming*, vol. 31, pp. 59–83, 1997.
- [36] S. McIlraith and T. Son, "Adapting Golog for Programmin the Semantic Web," in *5th Int. Symp. on Logical Formalization of Commonsense Reasoning*, 2001, pp. 195–202.
- [37] OWL-S, "<http://www.daml.org/services/owl-s/1.1/>," 2004.
- [38] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proc. of KR'91*, 1991, pp. 473–484.
- [39] D. Roggero, F. Patrone, and V. Mascardi, "Designing and implementing electronic auctions in a multiagent system environment," 2005, dISI Technical Report.
- [40] J. Sabater, "Trust and reputation for agent societies," *IIIA Monographs*, vol. 20, 2003.
- [41] K. Sycara, "Brokering and matchmaking for coordination of agent societies: A survey," in *Coordination of Internet Agents*, A. O. et al., Ed. Springer, 2001.
- [42] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan, "Dynamic discovery and coordination of agent-based semantic web services agents," *IEEE Internet Computing*, vol. 8, no. 3, pp. 66–73, 2004.
- [43] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed, "Life after BPEL?" in *Proc. of WS-FM'05*, ser. LNCS, vol. 3670. Springer, 2005, pp. 35–50, invited speaker.
- [44] C. Walton, "Uniting agents and web services," *AgentLink News*, vol. 18, pp. 26–28, 2005.
- [45] WS-CDL, "<http://www.w3.org/tr/2004/wd-ws-cdl-10-20041217/>," 2004.

Protocol specification and verification by using computational logic

Federico Chesani,
 Anna Ciampolini,
 Paola Mello,
 Marco Montali,
 Paolo Torroni
 DEIS, University of Bologna
 Viale Risorgimento, 2
 40136 Bologna (Italy)
 Email:

{fchesani|aciampolini|pmello|
 mmontali|ptorroni}
 @deis.unibo.it

Marco Alberti,
 Sergio Storari
 ENDIF, University of Ferrara
 Via Saragat, 1
 44100 Ferrara (Italy)
 Email:
 {malberti|sstorari}
 @ing.unife.it

Abstract—The aim of this paper is to report on some preliminary results obtained in the context of the MASSIVE research project (<http://www.di.unito.it/massive/>) relating the formal specification and verification of protocols in some different application field. A protocol is a way to express the right behavior of entities involved in a (possibly complex and distributed) process. The formalism to be used for protocol description should be as intuitive as possible, but it should be also formally defined, in order to allow formal checks both on the features of the protocol itself (e.g. termination), and also on the execution of it. To this purpose, we will show some results obtained by exploiting the *SOCS – SI* logic-based framework for the specification and the verification of protocols in various applicative fields such as electronic commerce, medicine and e-learning. We will also present a new graphical notation to express medical guidelines, which could be automatically translated into the SOCS formalism.

I. INTRODUCTION AND OBJECTIVES

The advent of distributed systems has focused the attention of the scientific community to interaction protocols between multiple interacting entities. There are many application areas where the concept of protocol has already reached a crucial importance; in some other field this concept has a great potential for improving the design and the execution of application specific processes. Protocols are the way to express the right behavior of entities involved in a (possibly complex and distributed) process; for instance, in a multi-agent setting, an interaction protocol expresses the rules that agents must follow in order to correctly perform the interaction. Even in application areas traditionally far from the computer science area, the *protocol* concept has been imported and used with a more specialized meaning and a different name. For instance, in the clinical field, protocols are named *clinical guidelines* and express the correct ways for treating given classes of clinical cases, possibly involving several *actors*,

each representing a specific medical operator (e.g., a physician, a nurse, a laboratory technician, etc.).

Whatever the considered application field is, once a protocol has been specified it could be very useful (in some cases it is mandatory) to be able to verify that actors executing that protocol are compliant with the behavior rules that the protocol expresses.

To this purpose, the research on protocol verification has greatly benefited from some important contributions achieved in the distributed and concurrent systems research area [1], [2], [3]. Among them, the SOCS european project [4] and the MASSIVE italian project [5] have defined a logic based framework for the specification and the verification of agent interactions within an open and heterogeneous society. This framework allows to specify the rules of each interaction protocol by means of a logic-based formalism based on integrity constraints. This formal language is associated with an operational counterpart implemented by means of an abductive proof procedure, which is able to verify during the execution (*on the fly*) agents compliance with given protocols, and possibly to detect rule violations.

Although the SOCS-SI framework can be used for protocol specification and verification in a wide range of applications, the language provided by SOCS-SI is logic-based and therefore it is not particularly *user-friendly*: it is likely a formalism not suitable to be used by protocol designers in non technological application areas, such as the medicine one. Therefore, in order to support protocol execution and verification in a wider scenario, it is crucial to have a more intuitive way to specify protocols, while the formal rigour in their description.

In the past, the need of formal languages for the definition of interaction protocols had not always been perceived as a fundamental requirement. The case of the TCP protocol (the

Transmission Control Protocol, [6]) is exemplary: the protocol is described through an informal graphical notation, and the semantic of messages is expressed in natural language; a wide part of the protocol (e.g., the timing) is even not specified at all.

The use of a graphical language for protocols definition instead is universally considered a necessary step to the aim of simplifying the job of protocols developers. In the multi-agent system development area, several proposals of graphical languages have been introduced, mostly based on finite state automata. Only recently two languages that follow a different approach (AUML [7], [8] and AML [9]) have been proposed, both extending Interaction Diagrams of standard UML to the aim of modeling agents interactions. However, although these graphical formalism are easy and intuitive, a complete formalization of them still lacks; consequently the support for the formal verification property of the protocol lacks too.

The aim of this paper is to report some preliminary results obtained in the context of the MASSIVE research project [5] relating the formal specification and verification of protocols in some different application field.

The paper is structured as follows. Section 2 briefly sketches on the features of the SOCS framework, with a special focus on verification and specification. Section 3 describes some experiences in protocol specification in the SOCS framework, regarding examples taken from different application areas. Section 4 introduces graphical languages for protocol specifications, and then presents GOSPEL, a new graphical notation which is suitable for the specification of protocols, with particular regard to medical guidelines, and which has been designed to allow automatic translation of protocols into the SOCS framework. Conclusions follow.

II. SOCS-SI: A FRAMEWORK FOR PROTOCOLS FORMALIZATION AND VERIFICATION

In this section we give the necessary background on the formal framework proposed by Alberti et al. [10], [11], [12] for the specification of agent interaction in open¹ societies of agents. The reader is referred to those papers for a complete description. This system was initially aimed at agent interaction protocols; however, in the following section we will show how this framework could be successfully exploited also in other different settings.

The framework assumes the existence of an entity (*Social Compliance Verifier* or SCV, for short) which is external to agents, and is devoted to check their compliance to the specification of agent interaction.

The SCV is aware of the ongoing social agent social behaviour: this is represented by a set of (ground) facts called *events*, and indicated by functor **H**.

For example, $\mathbf{H}(\text{request}(a_i, a_j, \text{give}(10\$), d_1), 7)$ represents the fact that agent a_i requested agent a_j to give 10\$, in the

context of interaction d_1 (dialogue identifier) at time 7.²

In open agent societies, the agent behaviour is unpredictable, because agents are autonomous; however, when interaction protocols are defined, we are able to determine what are the possible expectations about future events. This represents in some sense the “ideal” behaviour of a society. Expectations can be positive (events expected to happen, indicated by the functor **E**) or negative (events expected *not* to happen, functor **EN**). Expectations have the same format as events, but they will, typically, contain variables, to indicate that expected events are not completely specified. CLP [14] constraints can be imposed on variables to restrict their domain.

For instance,

$$\mathbf{E}(\text{accept}(a_k, a_j, \text{give}(M), d_2), T_a) \wedge M \geq 10 \wedge T_a \leq 15$$

represents the expectation for agent a_k to *accept* giving agent a_j an amount M of money, in the context of interaction d_2 (dialogue identifier) at time T_a ; CLP constraints say that M is expected to be greater or equal than 10, and T_a to be less or equal than 15.

The way expectations are generated, given the happened events and the current expectations, is specified by means of *Social Integrity Constraints* (IC_S).

Let us consider an example with two agents involved (although IC_S can be applied to any-party agent interaction):

$$\begin{aligned} &\mathbf{H}(\text{request}(A, B, P, D), T_1) \\ \rightarrow &\mathbf{E}(\text{accept}(B, A, P, D), T_2) \wedge T_2 \leq T_1 + \tau \quad (1) \\ &\vee \mathbf{E}(\text{refuse}(B, A, P, D), T_2) \wedge T_2 \leq T_1 + \tau \end{aligned}$$

states that, if agent A makes a *request* of P to agent B , in the context of interaction D at time T_1 , then agent B is expected to *accept* or *refuse* P by τ time units after the *request*.

The following IC_S :

$$\begin{aligned} &\mathbf{H}(\text{accept}(A, B, P, D), T_1) \\ \rightarrow &\mathbf{EN}(\text{refuse}(A, B, P, D), T_2) \wedge T_2 \geq T_1 \quad (2) \end{aligned}$$

$$\begin{aligned} &\mathbf{H}(\text{refuse}(A, B, P, D), T_1) \\ \rightarrow &\mathbf{EN}(\text{accept}(A, B, P, D), T_2) : T_2 \geq T_1 \quad (3) \end{aligned}$$

express, instead, mutual exclusiveness between *accept* and *refuse*: if an agent performs an *accept*, it is expected *not* to perform a *refuse* with the same content after the *accept*, and vice versa. In this way, we are able to define protocols as sets of forward rules, relating events to expectations.

Abduction [15] is a reasoning paradigm which consists of formulating hypotheses (called *abducibles*) to account for observations; in most abductive frameworks, *integrity constraints* are imposed over possible hypotheses in order to prevent inconsistent explanations. The idea behind our framework is to formalize expectations about agent behaviour as abducibles, and to use Social Integrity Constraints such as (1), (2) or (3) to prevent such agent behaviour that is not compliant with interaction protocols.

¹We intend *openness* in societies of agents as Artikis et al. [13], where agents can be heterogeneous and possibly non-cooperative.

²We make the simplifying assumption about time of events, that the time of sending a message is the same as receiving it, and that such time is assigned by the social framework.

Given the partial history of a society (*i.e.*, the set of already happened events), an abductive proof procedure (SCIFF, [16]) generates expectations about agent behaviour so as to comply with Social Integrity Constraints. SCIFF is inspired by the IFF proof procedure [17], augmented as needed to manage CLP constraints and universal variables in abducibles. The most distinctive feature of SCIFF, however, is its ability to check that the generated expectations are *fulfilled* by the actual agent behaviour (*i.e.*, that events expected (not) to happen have actually (not) happened), which cannot be assumed *a priori* in an open society of autonomous agents.

The SCIFF proof procedure (implemented using SICStus Prolog [18] and Constraint Handling Rules [19]) has been integrated into the Java-based SOCS-SI tool.

III. POTENTIAL FOR REAL EXPLOITATION OF THE SOCS FRAMEWORK

The previous section has shown the main features of the SOCS framework with a special focus on agents interaction protocols. The SOCS proof procedure deals with *events* in general, that in the case of agents interaction are mapped into communicative acts. However the concept of event can be abstracted from multi-agent systems and, dependently on the particular setting, it may represent different actions. In the following we will show how this can be applied to real-life scenarios.

A. Medical guidelines

Medical guidelines [20] are clinical behaviour's recommendations that are used to support physicians in the definition of the most appropriate diagnosis and/or therapy within determinate clinical circumstances.

Unfortunately, guidelines are today described by using several formats, such as flow charts and tables, so that physicians are not properly supported in the detection of possible errors and incompleteness: it is difficult to evaluate who made an error within the protocol's flow and when. As a consequence, guideline's application often loses its benefits.

In the following we show that the logic-based formalism provided by the SOCS framework is general enough to allow us to formally describe medical protocols. The main advantage of using ICs in the context of medical guidelines is the capability to discover some forms of inconsistency and to perform an on-the-fly verification of the protocol's application on a specific patient.

In order to effectively test the potentialities of this approach, we formalized a microbiological guideline [21] which describes how to manage an infectious patient from his arrival at a hospital's emergency room to his recovery and tested this guideline on a set of clinical trials.

The guideline may be structured in seven phases: patient's arrival at the hospital's emergency room; patient examination at the emergency room; possible admission in a specific hospital ward and first therapy prescription made by the ward physician; request of a microbiological test (consisting of many sub-phases, involving both human and artificial actors);

return of the microbiological test report to the ward physician, who must decide the definitive therapy; management of drugs by nurses; evaluation of patient's health and, in case of symptoms persistence, new prescription of microbiological test. In order to formalize the guideline described before, we detected, first of all, all the actors involved (*e.g.* the patient, wards physicians, the microbiological laboratory, etc.) and secondly pointed out all the actions which should be executed (or not, *i.e.* expected or not expected) for an appropriate patient's disease treatment. Each actor has been then mapped into an agent with a specific role, and actors actions (*e.g.* examinations, analysis, etc) has been modeled as SOCS events. For example, the following IC:

$$\begin{aligned} & \mathbf{H}(\text{enter}(\text{Patient}, \text{emergency_ward}), T_{ent}) \\ & \rightarrow \mathbf{E}(\text{examine}(\text{Physician}, \text{Patient}), T_{exam}) \quad (4) \\ & \wedge T_{exam} < T_{ent} + 6 * 60 \end{aligned}$$

expresses that when a patient arrives at the emergency room (at time T_{ent}), we expect that at least one physician would visit him (at time T_{exam}) within the deadline of 6 hours. This deadline is expressed as a CLP constraint, which says that T_{exam} should be lower than T_{ent} plus 6 hours. The complete specification of this protocol consists of about 20 social ICs. It has been tested via the SOCS-SI software, using different set of events, compliant and not. For instance, a non compliant set is the following: a patient (*patientA*) arrives at the hospital's emergency room at time 10, but no physician visits him within 6 hours. The event

$$\text{enter}(\text{patientA}, \text{emergency_ward}), 10$$

matches with the antecedent of (1), generating the expectation in the consequent that a physician should visit *patientA* at time T_{exam} , such that $T_{exam} < 10 + 6 * 60$. No event is afterward registered until this deadline, therefore a violation is raised by the proof procedure.

In this way a simple medical guideline may be mapped into a set of social integrity constraints in the context of SOCS infrastructure, thus enabling an on-the-fly verification about the compliance of the hospital staff to it. We have successfully tested this specification using the SOCS-SI tool with some set of events, compliant and not. Of course, this is only the first step towards an effective tool for defining and verifying guidelines in a clinical environment.

In literature, several formalisms have been proposed for representing medical protocols, like for example GLARE [22] and PROforma [23]. These are complete tool capable to manage both guidelines acquisition and execution, but, to the best of our knowledge, their are not able to verify compliance of actions and interactions of the kind here presented.

B. Electronic Auctions and E-commerce

Auctions have been practically used for centuries in human commerce, and their properties have been studied in detail from economic, social and computer science viewpoints. The raising of electronic commerce has pushed auctions as one of

the favorite dealing protocols in the Internet. Now, the software agent technology seems an attractive paradigm to support auctions [24]: agents acting on behalf of end-users could reduce the effort required to complete auction activities. Agents are intrinsically autonomous and can be easily personalised to embody end-user preferences. In addition, they could be adaptive and capable of learning from both past experience and their environment, in order to cope with changing operating conditions and evolving user requirements [25]. In fact, while in the past bidders were only humans, recent Internet auction servers [26] allow software agents to participate in the auction on behalf of end-users, and some of them even have a built-in support for mobile agents [27].

A first, important issue in e-commerce and, in particular, in electronic auctions, is *trust* [28]. Amongst the various aspects of trust in MASs (often related to credibility levels between agents), we find utterly important that human users trust their representatives: in order for the system to be used at all, each user must trust its representative agent in the auction.

A typical answer to such issues is to model-check the agents with respect to both their specifications and requirements coming from the society. However, this is not always possible in open environments: agents could join the society at all times and their specifications could be unavailable to the society. Thus, the correct behavior of agents can be checked only from the external in an open environment: by monitoring the communicative actions of the agents.

A second, very important issue in e-commerce, is the delivery of the auctioned good: the auctioneer must be guaranteed that he will receive the money, and the winner must be guaranteed that he will get the good.

A possible answer to this problem consists of crafting an interaction protocol for the delivery phase, such that both the seller and the buyer are guaranteed of their rights. An example of such a protocol has been shown in [29], where a third trusted entity (a bank) act as guarantee for the seller and the buyer.

Both the issues presented above show that the verification of the correct behavior of participants to agents plays a fundamental role, since the desired properties are guaranteed only if the agents behave properly w. r. t. the protocols. The SOCS framework provides an answer to this problem, since it is able to determine if an interaction, observed from an external viewpoint, respects a given protocol definition. Some of the integrity constraints ruling a single-item auction protocol are presented in the Specification III.1. In order to cope also with the delivery problem, some rules have been added to the auction protocol; these rules are mainly inspired by the delivery phase presented in the Netbill protocol. The IC_S 5, for example, states that each time a bidding event happens, the auctioneer should have sent an *openauction* event (to all bidders); this is equivalent to assert that no one can place a bid if an auction was not previously declared as “open”. The IC_S 6 implies instead that the auctioneer should answer to each bid, and that the answer should be sent after the auction is closed within the deadline $T_{deadline}$. Finally, the IC_S 7

imposes that if a bid has been declared a winning bid, then the bidder should deliver items involved in the bid.

Specification III.1 The auction protocol expressed using the IC_S language.

$$\begin{aligned} & \mathbf{H}(\text{tell}(B, A, \text{bid}(\text{ItemList}, P), A_{\text{number}}), T_{\text{bid}}) \\ \rightarrow & \mathbf{E}(\text{tell}(A, B, \text{openauction}(\text{Items}, T_{\text{end}}, T_{\text{deadline}}), A_{\text{number}}), T_{\text{open}}), \\ & T_{\text{open}} < T_{\text{bid}} \wedge T_{\text{bid}} \leq T_{\text{end}} \end{aligned} \quad (5)$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(B, A, \text{bid}(\text{ItemList}, P), A_{\text{number}}), T_{\text{bid}}) \wedge \\ & \mathbf{H}(\text{tell}(A, B, \text{openauction}(\text{Items}, T_{\text{end}}, T_{\text{deadline}}), A_{\text{number}}), T_{\text{open}}) \\ \rightarrow & \mathbf{E}(\text{tell}(A, B, \text{answer}(X, S, \text{ItemList}, P), A_{\text{number}}), T_{\text{answer}}), \\ & T_{\text{answer}} \geq T_{\text{end}} \wedge T_{\text{answer}} \leq T_{\text{deadline}}, X :: [\text{win}, \text{lose}] \\ & \dots \end{aligned} \quad (6)$$

$$\begin{aligned} & \mathbf{H}(\text{tell}(B, A, \text{bid}(\text{ItemList}, P), A_{\text{number}}), T_{\text{bid}}) \wedge \\ & \mathbf{H}(\text{tell}(A, B, \text{answer}(\text{win}, B, \text{ItemList}, P), A_{\text{number}}), T_1) \wedge \\ & T_{\text{bid}} < T_1 \\ \rightarrow & \mathbf{E}(\text{tell}(B, A, \text{deliver}(\text{ItemList}, P), A_{\text{number}}), T_3) \wedge \\ & T_3 > T_1 \\ & \dots \end{aligned} \quad (7)$$

C. E-learning by doing

E-learning is a new paradigm for the learning process, based on the growing availability of technology resources such as personal computers and the Internet. The main idea of e-learning consist of distributing the knowledge onto new media support like cd, dvd, or directly through the internet. Around this idea a set of support technologies have been developed, such as content management systems and applications for real-time streaming and interactions. Many advantages are offered by this paradigm: just to mention the more evident, teacher and student are not constrained anymore to be in the same place. Moreover, teacher and student can be decoupled also in the time dimension: it is no longer needed that teacher and student attend the lesson at the same time instant. The learning process can be adapted to each student's needs, taking into account previous knowledge, time availability, and learning capabilities of the student himself.

Several e-learning paradigms have been developed, and amongst them, e-learning “by doing” is one of the most promising in terms of the learning quality. The “by doing” paradigm consists of teaching a topic by letting the student directly practice the argument onto a real system, or a model that simulates the real system. This approach can be applied also to the e-learning processes, and in particular to software applications learning. Of course, the degree of interaction between the student and the teacher, and the possibility to receive help when needed, are of the utmost importance in such process. The student in fact must not be left alone during the learning process, but rather he should be followed interactively, and he should receive help, hints and feedback whenever it is opportune.

To support the e-learning by doing process, it is necessary to tackle several issues: firstly, a mechanism for evaluating the acquired skills is needed, in order to be able to proceed to advanced topics. The evaluation mechanism must provide support for a-posteriori evaluation, as well as run-time evaluation to hint the student. Secondly, it is quite common that the same learning goal can be achieved in more than one way: the tutoring system must be able to evaluate all the options, and should adapt in response to the student choices.

The SOCS framework, and in particular the SOCS-SI application, are general enough to be used also in the context of e-learning by doing. We have tried successfully to adopt our protocol definition language for representing the action expected by the user of a e-learning by doing system (a sort of a protocol where only one peer participate). We have focussed our experiments on the learning process of a writing application within the offices program suites. We developed our prototype on two applications, the MS Word program (part of the Microsoft Office Suite), and the Writer application of the OpenOffice suite. For both applications, a specific filter has been developed, with the purpose of capturing the actions performed by the student. Those actions, after a transformation process, are communicated to the SOCS-SI application, that provide to check the conformance to a special protocol definition. Such definition can be seen in the Specification III.2, where it is defined how the student can achieve the goal of closing the application after printing a file.

Specification III.2 An e-learning goal represented through the IC_S language.

$$\begin{aligned}
 & \mathbf{H}(\text{tell}(U, S, \text{keyboard_event}(\text{print}), \text{DialogId}), T_{\text{Print}}) \\
 \rightarrow & \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Close}), \text{DialogId}), T_{\text{Close}}) \\
 & \wedge T_{\text{Close}} > T_{\text{Print}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Exit}), \text{DialogId}), T_{\text{Exit}}) \\
 & \wedge T_{\text{Exit}} > T_{\text{Print}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{quit}), \text{DialogId}), T_{\text{Exit}}) \\
 & \wedge T_{\text{Exit}} > T_{\text{Print}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{alt} + f), \text{DialogId}), T_{\text{File}}) \\
 & \wedge \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Close}), \text{DialogId}), T_{\text{Close}}) \\
 & \wedge T_{\text{Print}} < T_{\text{File}} \wedge T_{\text{File}} < T_{\text{Close}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{alt} + f), \text{DialogId}), T_{\text{File}}) \\
 & \wedge \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Exit}), \text{DialogId}), T_{\text{Exit}}) \\
 & \wedge T_{\text{Print}} < T_{\text{File}} \wedge T_{\text{File}} < T_{\text{Exit}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{close_document}, \text{DialogId}), T_{\text{Close}}) \\
 & \wedge T_{\text{Close}} > T_{\text{Print}} \\
 \vee & \mathbf{E}(\text{tell}(U, S, \text{close_office}, \text{DialogId}), T_{\text{Close}}) \\
 & \wedge T_{\text{Close}} > T_{\text{Print}}
 \end{aligned} \tag{8}$$

The IC_S 8 shows how it is possible to represent multiple solutions for solving the learning goal. Seven different alternatives are considered, from using the “File” menu and the corresponding voice, to closing directly all the application.

Once the learning goal has been defined through IC_S , the SOCS-SI application can use it in three different ways:

1) the tool can be used as evaluator of the actions of the

student: if at the end of the practicing session, at least one expectation is not satisfied, then the goal has not been achieved;

2) the tool can be used also as an on-the-fly checker: if the student perform an action that will block him for reaching the goal, then it is possible to advice him immediately, rather than waiting for the end of the exercise;

3) the tool can be finally used as a suggesting system: if the student does not know how to achieve the goal, it is possible to hint him the next action by communicating the expectations about his future behavior.

Of course it is up to the teacher (or the e-learning content manager) to decide which modality is more opportune.

IV. TOWARDS A HIGH-LEVEL LOGIC-BASED SPECIFICATION USING GRAPHICAL LANGUAGES

The problem of the specification of protocols involving several entities is becoming a topical subject in many different contexts.

In the multi-agent system development area, several proposals of graphical languages for protocol definition have been introduced, mostly based on finite states automata [30], [31]. Only recently two languages that follow a different approach, AUML [7], [8] and AML [9] have been proposed. AUML proposes an extension of the Interaction Diagram of standard UML to the aim of modeling agents interactions. AUML supports the heterogeneity of interacting entities, since it abstracts from the inner architecture of the agents; it allows to define in an intuitive way which are the actors participants to the interaction, and the messages (specifying both the sender and the addressee) allowed in the protocol. Although the AUML graphical formalism is easy and intuitive, a complete formalization of the language still lacks. The AML language extends the AUML protocols graphical specification language; however it still does not supply a complete language semantics and therefore it does not support any formal verification of properties too.

Although the formalism to be used for protocol description should be as intuitive as possible, it should be also formally defined, in order to allow the execution of automatic checks both on the features of the protocol itself (e.g. termination, etc.), and also on the execution of it. To this purpose, in the following we will present GOSPEL, a new graphical notation to express protocols, with particular regard to medical guidelines, which has been designed to be automatically translated into the SOCS formal language. The automatic translation is ongoing work, and the first experimental results suggest that it feasible.

A. GOSPEL

In literature, several graphical notations have been proposed proposed to represent medical protocols, like for example GLARE [22] and PROforma [23]. These are complete tool capable to manage both guidelines acquisition and execution, but, as for as we are concerned, their are not able to verify

compliance of actions and interactions as those presented in Section III-A.

If we want to effectively bring these advantages in the clinical environment we have to incorporate the SOCS approach in a tool that allows both guidelines acquisition and execution. The first step toward this goal is represented by the Guideline prOcess SPEcification Language (GOSPEL). GOSPEL is a graphical language, inspired by flow charts, for the specification and representation of all the activities that belong to a process and their flow inside it.

The GOSPEL representation of a guideline consists of two different parts: a *flow chart*, which models the process evolution, and an *ontology*, which describes at a fixed level of abstraction the application domain and gives a semantic to the diagram. The GOSPEL flow chart language is described in Section IV-A.1. The GOSPEL ontology management is described in Section IV-A.2. Section IV-A.3 describes how we plan to integrate GOSPEL with the SOCS approach in real world applications.

1) *GOSPEL flow chart language*: The GOSPEL flow chart language describes the process evolution using blocks, which can represent distinct process activities, and connections between blocks.

About the blocks, the ones proposed by GOSPEL are shown in Table I.













LEAF BLOCKS				
action	autom. decision	ex-or	parallel	synch
				
START BLOCKS		MACROBLOCKS		
start	cyclic start	complex action	iteration	while
				
END BLOCKS				
return	end			
				

TABLE I
GOSPEL BLOCKS

These blocks are grouped into four families:

- *Leaf blocks*, blocks which represent atomic process activities at the desired abstraction level;
- *Macroblocks*, blocks that are threatened at their level like simple blocks but that encapsulate a sub-process (that is represented as another GOSPEL guideline);
- *Start blocks*, start points of (sub)processes;
- *End blocks*, end points of (sub)processes.

Among *leaf blocks*, *action blocks* are used to represent single atomic process activities. The other *leaf blocks* are

crucial for modeling complex guidelines as they are used to express workflow's branches and forks, the former related to decision points, the latter to activities parallelization.

GOSPEL supports two different types of decision blocks: the first one, called *ex-or decision leaf block*, is used simply for expressing mutual exclusion between successors; the second one, called *automatic decision leaf block*, permits to automatically decide which path should be followed. In the second case, each outgoing relation is guarded. In order to maintain mutually exclusion, the designer should give a preference about guards evaluation: the i -th guard is evaluated iff the $i-1$ -th previous guards fail.

Concurrence of activities is expressed using *parallel* and *synch leaf blocks*. When the process flow reaches the *parallel leaf block* it is splitted in several subprocesses represented as outgoing relations. When these subprocesses are executed, they are regrouped by the *synch leaf block* in the main process flow.

Thanks to *macroblocks*, GOSPEL allows guideline designer to follow a top-down approach for guideline process description as it is possible to split recursively the process into subprocesses, bringing down the level of abstraction. We said that *Macroblocks* are special blocks threatened like atomic actions at their level, but that incapsulate a new subprocess. Therefore, each *macroblock* is associated to one *start block*, representing the initial point of the (sub)process, and one or more *exit blocks*; when an exit point is encountered, the flow will return to the parent level; entering in and exiting from a *macroblock* follow the same approach of procedure calls. A *macroblock* defines both its inner subprocess and how the flow will walk through it.

GOSPEL proposes three *macroblock* types: *Complex action macroblock*, *Iteration macroblock* and *While macroblock*. The simplest one is the one of *complex action macroblock*, in which we specify directly a new (sub)process. The entire guideline may be viewed as a big *complex action*. Other *macroblocks* show different behavior, because they express workflow cycles. Cycles modeling in GOSPEL is similar to structured programming: the *Iteration macroblock* models a *for* structure, saying how many times the (sub)process should be repeated and the *While macroblock* represents a *do...while* structure, expressing the exit condition with a logic guard. In the case of cyclic *macroblocks*, the modeled (sub)process corresponds to one iteration step. In order to say that the generic step is terminated and that the next should begin (if the cycle condition agrees), we create a connection that goes back into the start point, which is actually a *cyclic start*. The presence of an *exit block* within a cyclic *macroblock* means that the cycle should be prematurely terminated; the same happens in structured programming when we write a *break* command.

About block connections, GOSPEL defines three types of binary relations (connections that involve two blocks): *Order relation*, *Conditional order relation* and *Time relation*.

An *order relation* is an oriented connection used to specify which activity follows a specific one in the process evolution.

A *conditional order relation* may be associated to a logic guard containing the knowledge necessary to automatically choose if the process evolution will walk through this connection. If this knowledge is not modeled, the choice is left to participants.

The *time relation* is used to express a constraint between the execution time of involved activities. As described in III-A, Integrity Constraints can be used to model a protocol in term of observable events. GOSPEL follows the same approach: an *action block* models a relevant and observable activity in the workflow. Since an activity is observable and has a well-defined execution time it possible to made temporal constraints related to the execution time of several of them.

An example of a GOSPEL guideline fragment is shown in Figure 1.

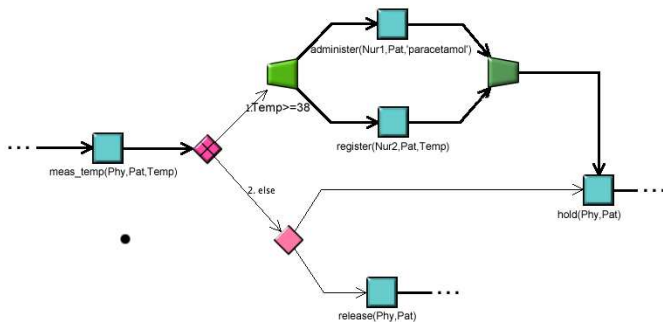


Fig. 1. Example of GOSPEL language: a fragment of a hypothetical clinical guideline

In the example, a ward physician (*Phy*) should measure (*meas_temp(Phy,Pat,Temp)*) the patient's temperature. If the measured value is greater or equal than 38 celsius degrees then two nurses should register that value (*register(Nur2,Pat,Temp)*) and administer paracetamol to the patient (*administer(Nur1,Pat,'paracetamol')*). In this case, the patient is held in the emergency room (*hold(Phy,Pat)*) for further investigations. Otherwise, if the measured value is lower than 38 celsius degrees then the physician should decide if is necessary to hold the patient or to let him go away (*release(Phy,Pat)*).

2) *GOSPEL ontology*: Another crucial component of GOSPEL is the guideline ontology. Since GOSPEL is a general-purpose language, potentially useful in any context that requires to model a workflow, it avoids to fix an ontology *a priori*.

The guideline ontology is used to specify the semantic associated to an activity. It is mainly composed by two hierarchies: a hierarchy of all the activities which belongs to the process domain and a hierarchy of participants, entities that play a role in one or more activities.

This ontology may be created and maintained by using Protege [stanford.protege.org], an open source tool, developed by the Stanford University. The Protege JAVA libraries are used in the graphical guideline editor of GOSPEL to specify

actions and guards. Figure 2 shows an example of action specification in the GOSPEL editor.

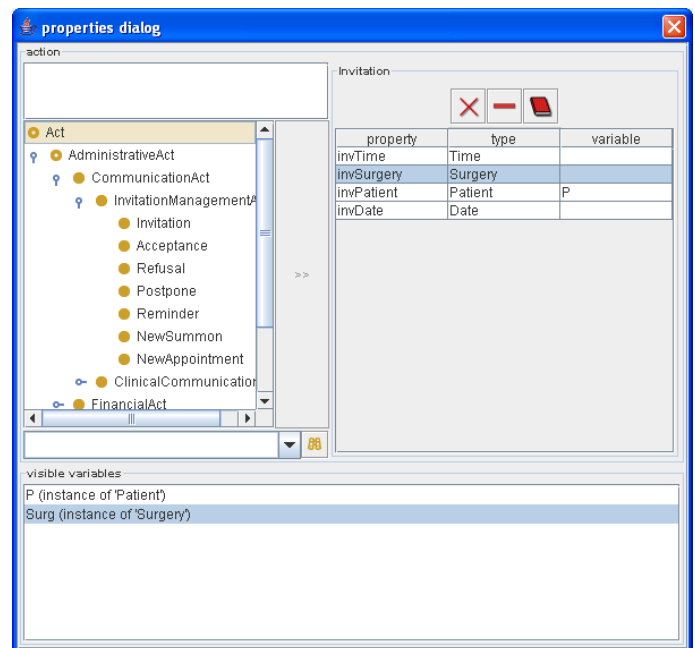


Fig. 2. Action specification in the GOSPEL editor

The complete specification of an action consists of choosing an ontological activity and associating one or more participants to it. Participants are introduced within *macroblocks* giving them a logical name; *macroblocks* realize also precise visibility rules of participants. During an action specification, visible participants can be associated to the selected ontological activity.

3) *Integrating GOSPEL with the SOCS approach*: GOSPEL is part of a complex system that aims to support designers in both the modeling and the execution phases. With respect to the modeling side, we have implemented a GOSPEL editing tool that integrates ontologies developed in Protege. We are working now on a *visitor* application (the *Translator* shown in Figure 3) that is capable to walk through a GOSPEL diagram and translate it into a set of social integrity constraints.

From this perspective, ontological activities become happened events and participants logical variables involved in the events. The visitor makes possible to exploit the benefits of SOCS computational model, providing a framework for the verification of participant behavior compliance to GOSPEL modeled processes. In order to better explain how the visitor works, we take into account, for example, the GOSPEL diagram shown in Figure 1. In this example, the temperature measurement becomes an happened event. The presence of an *automatic decision* with two outcoming relations splits the diagram into two different "worlds", the former associated to a temperature's value greater or equal than 38, the latter to a value lower than 38; therefore, the visitor generates IC_S 9 and 10, mapping directly the *parallel* and the *ex-or decision blocks* into logical AND and exclusive OR.

Specification IV.1 first part of the diagram in Figure 1 translated into IC_S

$$\begin{aligned}
 & \mathbf{H}(\text{temp_meas}(\text{Phy}, \text{Pat}, \text{Temp}), T_m) \\
 & \wedge \text{Temp} \geq 38 \\
 \rightarrow & \mathbf{E}(\text{register}(\text{Nur1}, \text{Pat}, \text{Temp}), T_r) \\
 & \wedge T_r > T_m \\
 \wedge & \mathbf{E}(\text{administer}(\text{Nur2}, \text{Pat}, \text{paracetamol}), T_a) \\
 & \wedge T_a > T_m
 \end{aligned} \tag{9}$$

$$\begin{aligned}
 & \mathbf{H}(\text{temp_meas}(\text{Phy}, \text{Pat}, \text{Temp}), T_m) \\
 & \wedge \text{Temp} < 38 \\
 \rightarrow & \mathbf{E}(\text{hold}(\text{Phy}, \text{Pat}), T_h) \\
 & \wedge T_h > T_m \\
 & \wedge \mathbf{EN}(\text{release}(\text{Phy}, \text{Pat}), T_r) \\
 & \wedge T_r > T_h \\
 \vee & \mathbf{E}(\text{release}(\text{Phy}, \text{Pat}), T_r) \\
 & \wedge T_r > T_m \\
 & \wedge \mathbf{EN}(\text{hold}(\text{Phy}, \text{Pat}), T_h) \\
 & \wedge T_h > T_r
 \end{aligned} \tag{10}$$

Finally, the diagram shows that when the temperature is ≥ 38 and both nurses have finished their tasks, the physician should hold the patient. This behavior translates into a third integrity constraint (IC_S 11).

Specification IV.2 third IC_S generated visiting the diagram example

$$\begin{aligned}
 & \mathbf{H}(\text{register}(\text{Nur1}, \text{Pat}, \text{Temp}), T_r) \\
 & \wedge \mathbf{H}(\text{administer}(\text{Nur2}, \text{Pat}, \text{paracetamol}), T_a) \\
 \rightarrow & \mathbf{E}(\text{hold}(\text{Phy}, \text{Pat}), T_h) \\
 & \wedge T_h > T_r \\
 & \wedge T_h > T_a
 \end{aligned} \tag{11}$$

In a real application, we cannot rely on a manual delivery of relevant events to the proof. We have rather to identify the different types of events sources and try to extract automatically the happened events from them. Many events are recorded in the business database management system, which can be considered as a source of events.

Therefore, at the execution side we are developing an infrastructure that is capable to map ontological activities into a concrete data base management system and interact with it in order to extract at run-time the corresponding events. Two forms of interaction are considered: a *polling* mode and an *interrupt* mode (implemented via triggers).

The complete architecture of the guideline tool integrating both GOSPEL and SOCS is shown in Figure 3.

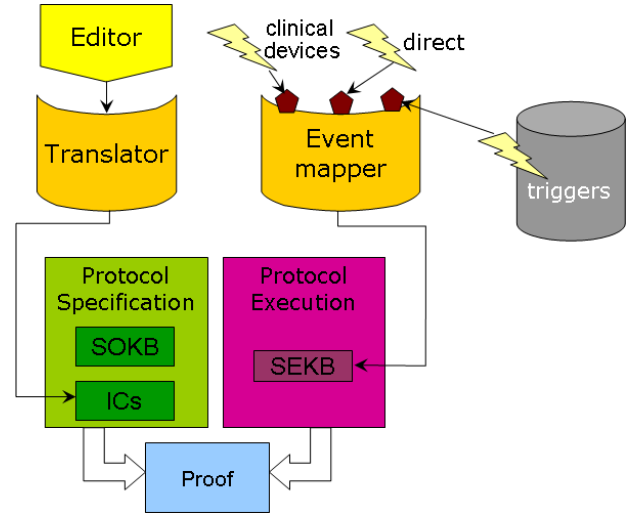


Fig. 3. Overview of the integration between GOSPEL and SOCS

V. CONCLUSIONS

In this paper we have reviewed some of the applications of the logic-based SOCS social framework, and we have introduced the GOSPEL graphical notation for expressing protocols.

The research reported in this paper represents a first step towards a methodology of protocol design meant to exploit the best of two worlds: the ease of use and simplicity of graphical formalisms, and the well-defined declarative and operational semantics of logic-based formalisms.

The next (ongoing) step of our research is the automatic translation of GOSPEL-based protocol specifications to the SOCS framework.

ACKNOWLEDGEMENTS

This work was partially funded by the IST programme of the EU under the IST-2001-32530 SOCS project, by MIUR under the COFIN2004 project "MASSIVE: Sviluppo e verifica di sistemi multiagente basati sulla logica" and under the COFIN2003 project "La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici", by "SPINNER: Servizi per la Promozione dell'INNOvazione e della Ricerca" Project 45/04.

We wish to thank the other partners of the MASSIVE project, Evelina Lamma and Marco Gavanelli for their collaborative participation and for precious comments and suggestions on our work.

REFERENCES

- [1] D. Gollman, "Analysing security protocols," in *Proceedings FASEC*, ser. LNCS, A. Abdallah, P. Ryan, and S. Schneider, Eds., vol. 2629, 2002, pp. 71–80.
- [2] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2000.
- [3] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

- [4] "Societies Of Computees (SOCS), IST-2001-32530, <http://lia.deis.unibo.it/research/socs/>." [Online]. Available: [HomePage: \url{http://lia.deis.unibo.it/Research/SOCS/}](http://lia.deis.unibo.it/Research/SOCS/)
- [5] "Massive - sviluppo e verifica di sistemi multiagente basati sulla logica," <http://www.di.unito.it/massive/>.
- [6] J. Postel, *Transmission Control Protocol*, IETF, September 1981, sTD 7, RFC 793.
- [7] M. P. Huet, "Agent uml notation for multiagent system design," *Internet Computing, IEEE*, vol. 8, no. 4, pp. 63–71, Jul-Aug 2004.
- [8] J. P. M. B. Bauer and J. Odell, "Agent uml: A formalism for specifying multiagent interaction," in *Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge, Eds. Berlin: Springer-Verlag, 2001, pp. 91–103.
- [9] R. Cervenka and I. Trencansky, "Agent modeling language, language specification," Whitestein Technologies, Tech. Rep., 2004, draft proposal v. 0.9.
- [10] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "A social ACL semantics by deontic constraints," in *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, ser. Lecture Notes in Artificial Intelligence, V. Mařík, J. Müller, and M. Pěchouček, Eds., vol. 2691. Prague, Czech Republic: Springer-Verlag, June 16–18 2003, pp. 204–213.
- [11] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Specification and verification of agent interactions using social integrity constraints," *Electronic Notes in Theoretical Computer Science*, vol. 85, no. 2, 2003.
- [12] —, "An Abductive Interpretation for Open Societies," in *AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa*, ser. Lecture Notes in Artificial Intelligence, A. Cappelli and F. Turini, Eds., vol. 2829. Springer-Verlag, Sept. 23–26 2003, pp. 287–299. [Online]. Available: <http://www-aiia2003.di.unipi.it>
- [13] A. Artikis, J. Pitt, and M. Sergot, "Animated specifications of computational societies," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, C. Castelfranchi and W. Lewis Johnson, Eds. Bologna, Italy: ACM Press, July 15–19 2002, pp. 1053–1061. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=545070&type=pdf&dl=GUIDE&dl=ACM&CFID=4415868&CFTOKEN=57395936
- [14] J. Jaffar and M. Maher, "Constraint logic programming: a survey," *Journal of Logic Programming*, vol. 19-20, pp. 503–582, 1994.
- [15] A. C. Kakas, R. A. Kowalski, and F. Toni, "Abductive Logic Programming," *Journal of Logic and Computation*, vol. 2, no. 6, pp. 719–770, 1993.
- [16] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Specification and verification of interaction protocols: a computational logic approach based on abduction," Dipartimento di Ingegneria di Ferrara, Ferrara, Italy, Technical Report CS-2003-03, 2003, available at <http://www.ing.unife.it/informatica/tr/>.
- [17] T. H. Fung and R. A. Kowalski, "The IFF proof procedure for abductive logic programming," *Journal of Logic Programming*, vol. 33, no. 2, pp. 151–165, Nov. 1997.
- [18] "SICSStus prolog user manual, release 3.11.0," Oct. 2003, <http://www.sics.se/isl/sicstus/>.
- [19] T. Frühwirth, "Theory and practice of constraint handling rules," *Journal of Logic Programming*, vol. 37, no. 1-3, pp. 95–138, Oct. 1998.
- [20] C. Gordon, "Practice guidelines and healthcare telematics; towards an alliance," *Health telematics for clinical guidelines and protocols*, pp. 3–15, 1995.
- [21] D. Brock, M. Madigan, J.M. Martinko, and J. Parker, *Microbiologia*. Prentice-Hall International, Milano, 1995.
- [22] P. Terenziani, P. Raviola, O. Bruschi, M. Marzuoli, and G. Molino, "Representing knowledge levels in clinical guidelines," *Proceedings of the Join European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, vol. 1620, pp. 254–258, 1999.
- [23] J. Fox, N. Johns, A. Rahmzadeh, and R. Thomson, "Disseminating medical knowledge: the proforma approach," *Artificial Intelligence in Medicine*, vol. 14, pp. 157–181, 1998.
- [24] A. Chavez and P. Maes, "Kasbah: An agent marketplace for buying and selling goods," in *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, London, Apr. 1996, pp. 75–90.
- [25] R. Guttman, A. Moukas, and P. Maes, "Agent-mediated electronic commerce: A survey," *Knowledge Engineering Review*, vol. 13(2), pp. 143–147, 1998.
- [26] P. Wurman, M. Wellman, and W. Walsh, "The michigan internet auctionbot: A configurable auction server for human and software agents," in *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.
- [27] T. Sandholm, "eMediator: a next generation electronic commerce server," in *Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000)*, 2000.
- [28] S. Marsh, "Trust in distributed artificial intelligence," in *Artificial Social Societies*, ser. Lecture Notes in Artificial Intelligence, C. Castelfranchi and E. Werner, Eds., no. 830. Springer-Verlag, 1994, pp. 94–112.
- [29] B. Cox, J. Tygar, and M. Sirbu, "Netbill security and transaction protocol," in *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, July 1995.
- [30] M. Barbuceanu and M. Fox, "Cool: a language for describing coordination in multi-agent systems," in *Proc. of 1st Intl. Conf. on Multiagent Systems (ICMAS-95)*, 1995, pp. 17–24.
- [31] K. Kuwabara, T. Ishida, and N. Osato, "Agentalk:describing multiagent coordination protocols with inheritance," in *7th Int. Conf. on Tools for Artificial Intelligence (ICTAI-95)*, 1995.