

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.unibo.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.

$$\begin{aligned}
& \mathbf{H}(\text{tell}(U, S, \text{keyboard_event}(\text{print}), \text{DialogId}), T_{\text{Print}}) \\
\rightarrow & \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Close}), \text{DialogId}), T_{\text{Close}}) \\
& \wedge T_{\text{Close}} > T_{\text{Print}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Exit}), \text{DialogId}), T_{\text{Exit}}) \\
& \wedge T_{\text{Exit}} > T_{\text{Print}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{quit}), \text{DialogId}), T_{\text{Exit}}) \\
& \wedge T_{\text{Exit}} > T_{\text{Print}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{alt} + f), \text{DialogId}), T_{\text{File}}) \\
& \wedge \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Close}), \text{DialogId}), T_{\text{Close}}) \\
& \wedge T_{\text{Print}} < T_{\text{File}} \wedge T_{\text{File}} < T_{\text{Close}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{keyboard_event}(\text{alt} + f), \text{DialogId}), T_{\text{File}}) \\
& \wedge \mathbf{E}(\text{tell}(U, S, \text{mouse_event}(\text{menu_File_Exit}), \text{DialogId}), T_{\text{Exit}}) \\
& \wedge T_{\text{Print}} < T_{\text{File}} \wedge T_{\text{File}} < T_{\text{Exit}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{close_document}, \text{DialogId}), T_{\text{Close}}) \\
& \wedge T_{\text{Close}} > T_{\text{Print}} \\
\vee & \mathbf{E}(\text{tell}(U, S, \text{close_of_fice}, \text{DialogId}), T_{\text{Close}}) \\
& \wedge T_{\text{Close}} > T_{\text{Print}}
\end{aligned} \tag{8}$$

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 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 treated 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.

Concurrency 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 treated 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.

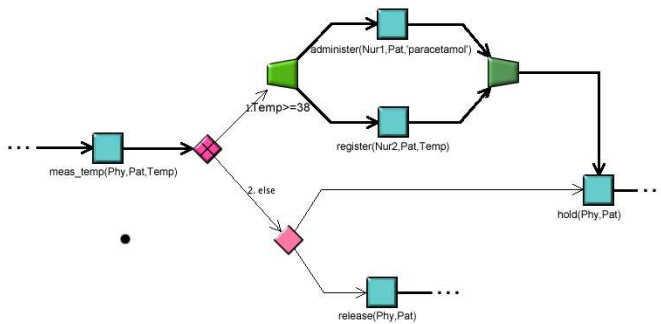


Fig. 1. Example of GOSPEL language: a fragment of a hypothetical clinical guideline

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.

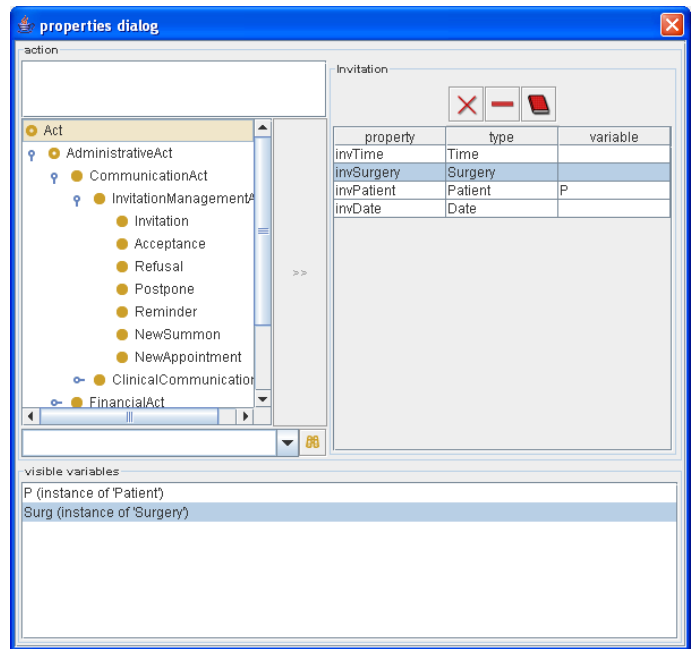


Fig. 2. Action specification in the GOSPEL editor

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_5 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

$$\begin{aligned}
& \mathbf{H}(temp_meas(Phy, Pat, Temp), T_m) \\
& \quad \wedge Temp \geq 38 \\
\rightarrow & \mathbf{E}(register(Nur1, Pat, Temp), T_r) \\
& \quad \wedge T_r > T_m \\
& \wedge \mathbf{E}(administer(Nur2, Pat, paracetamol), T_a) \\
& \quad \wedge T_a > T_m
\end{aligned} \tag{9}$$

$$\begin{aligned}
& \mathbf{H}(temp_meas(Phy, Pat, Temp), T_m) \\
& \quad \wedge Temp < 38 \\
\rightarrow & \mathbf{E}(hold(Phy, Pat), T_h) \\
& \quad \wedge T_h > T_m \\
& \quad \wedge \mathbf{EN}(release(Phy, Pat), T_r) \\
& \quad \wedge T_r > T_h \\
\vee & \mathbf{E}(release(Phy, Pat), T_r) \\
& \quad \wedge T_r > T_m \\
& \quad \wedge \mathbf{EN}(hold(Phy, Pat), T_h) \\
& \quad \wedge T_h > T_r
\end{aligned} \tag{10}$$

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

$$\begin{aligned}
& \mathbf{H}(register(Nur1, Pat, Temp), T_r) \\
& \quad \wedge \mathbf{H}(administer(Nur2, Pat, paracetamol), T_a) \\
\rightarrow & \mathbf{E}(hold(Phy, Pat), T_h) \\
& \quad \wedge T_h > T_r \\
& \quad \wedge T_h > T_a
\end{aligned} \tag{11}$$

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.

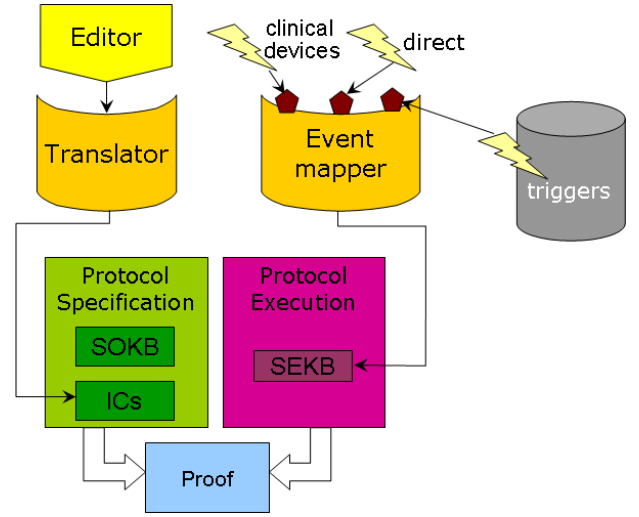


Fig. 3. Overview of the integration between GOSPEL and SOCS

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/}`
- [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. Huget, "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. Torchio, 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.