A logic based approach to interaction design in open multi-agent systems

Marco Alberti   Marco Gavanelli
Evelina Lamma
DI, Università di Ferrara
Via Saragat, 1
44100 Ferrara, Italy
{malberti, mgavanelli, elamma}@ing.unife.it

Federico Chesani   Paola Mello
Paolo Torroni
DEIS, Università di Bologna
V.le Risorgimento, 2
40136 Bologna, Italy
{fchesani,pmello,ptorroni}@deis.unibo.it

Abstract

An important challenge posed by the design of open information systems concerns the choice of suitable methods to harness their complexity and to guarantee the correctness of their behaviour. In recent times, logic programming has been proposed as a powerful technology, formal and declarative, for the specification and verification of agent based and open systems. In this work, we focus on the interaction design. We base our approach on a logic-based formalism, which can be used to define the semantics of agent communication languages and interaction protocols. We advocate its use within a more general framework, drawing a design methodology which encompasses the specification of the interaction space and of its desired properties, and their verification.

1. Introduction

The multi-agent computational paradigm is often used to model open information systems as complex and dynamic structures of autonomous entities. One challenge posed by the design of this kind of systems concerns the choice of suitable methods to harness their complexity and to guarantee the correctness of their behaviour. At the level of modelling and specification, complex solutions required to tackle complex application domains will benefit from a declarative approach, especially in terms of knowledge representation and software management. On the other hand, if we think for instance of dependable infrastructures, networked enterprises, and electronic health care as possible application domains, the correctness of models and implemented systems is a requirement that must be met by a design methodology encompassing theoretical and practical aspects.

In recent times, logic programming has been proposed as a powerful technology, formal and declarative, for the specification and verification of agent based and open systems. Most interestingly, logic programming provides a direct link between specification and implementation, which opens the possibility to easily derive an implementation from a synthetic and readable system specification, and to formally prove properties about the behaviour of the implemented system. Recent advances in logic programming frameworks, such as those based on constraint satisfaction technology, improved the efficiency of tools that are now used in many commercial applications such as those involving planning and scheduling.

In this work, we focus on the interaction design in open multi-agent systems. As discussed by Omicini and Ossowski in [15], following Gelernter and Carrero [11], the agent interaction space could be designed using a subjective perspective, derived from the agent specifications, or an objective coordination model, independently of the agents which will populate the system. In this work, unlike other agent oriented software methodologies [21, 14, 7], we take an objective approach to the interaction space design. The main purpose of our approach is in fact to give the abstractions required to be able to prove global properties of a system of interacting agents, and to give the formal machinery needed to perform a verification on their externally observable behaviour.

Agent Communication Languages (ACLs) semantics and Interaction Protocols (IPs) definition, intended to be part of the interaction design, are well known for being a well suited domain for formal approaches. We ground our approach on a logic-based formalism, called Social Integrity Constraints (IC_S), introduced in [4], for the specification and verification of agent interaction. In particular, IC_S can be used to define the semantics of ACLs and IPs, and a proof-procedure based on Abductive Logic Programming (called *SC*-IFF) can be used to verify the compliance of agent interaction to such semantics and protocols. We then advocate the use of IC_S within a more general framework in the overall design of the interaction space in an open multi-agent system. We draw a design methodology composed of a number of steps, including the specification of interaction and of desired properties, and their verification. The advantage of such an approach is given by the uniform formalism used to specify ACLs, IPs, and properties, and by the existence of a proof-procedure, proven correct with respect to the declarative semantics of the IC_S framework, and for which an implementation is available, based on constraint technology.

This paper is structured as follows. In Section 2 we introduce an example that we use throughout the paper. Section 3 briefly explains the IC_S framework. In Section 4 we describe our proposed methodology. Section 5 concludes the paper.

2. An auction example

In this Section, we introduce a running example which we will refer to throughout the paper. The example is a first price sealed bid auction, where agents bid to buy goods. Highest bid wins. The actions involved in the auctioning are both communicative (auction announcement, bid, notification) and physical (delivery, payment). For the sake of simplicity, we model all of them as communicative actions, and we assume for example that a message stating a delivery can be taken as a proof of the delivery itself. In a concrete application, this assumption will rely for instance on a trusted third party, which we do not model here. Also, we assume that the auction participants are known to the bidder. This can be achieved in a concrete implementation by a registering service such as those used in most Internet auction sites.

The auction protocol is composed of the following steps:

1. *Announcement.* The auctioneer broadcasts an *opauc* (*op*en *auc*tion) message to all potential bidders. The *opauc* message will contain information about item(s) on sale and deadlines.

2. *Bid.* The interested agents make their *bid*.

3. *Notification.* The auctioneer notifies the bidders with *win* or *lose*.

While the protocol specifies the sequence of actions ruling a certain interaction, the communicative actions define the atomic steps involved in the interaction itself. We can express their intuitive meaning based on a social notion of commitment [19, 6]. The (communicative) actions involved in the auction protocol are as follows:

- *opauc*, specifying items on sale and deadlines. The meaning of *opauc* is a “commitment” for the auctioneer to answer to bidders by notifying their winning or losing the auction, given some deadlines;

- *bid*, specifying the price or quote that an agent is ready to pay for some item(s). The meaning of *bid* is a “commitment” for the bidder to *pay* the price in case he turns out to be the winner;

- *answ*, used in the notification phase of the protocol (we use *answ*(*win*) to notify a winner, *answ*(*lose*) to notify a loser). The meaning if of *answ*(*win*) is a “commitment” for the auctioneer to deliver the good to the winning agent, while the *answ*(*lose*) bears no commitment;

- *pay*, used to notify the payment of the goods by a winning agent. It bears no social commitment;

- *deliver*, used to notify the delivery of goods by the auctioneer to the winning agents. It bears no social commitment.

3. Social Integrity Constraints

Social Integrity Constraints are an abductive [13] logic-based formalism, that can be used to specify the *social* semantics of ACL/IP in a uniform way. By social we mean that the semantics of interaction is not given in terms of specific agent architectures (such as mental states), but in terms of externally observable agent behaviour.

Agent interaction is represented by means of *events* (the *actual* agent behaviour) and *expectations*. The idea of social expectations is related to that of commitments, with the difference that they do not necessarily represent commitments, but more in general what is expected, given a certain history of events and a specification of ACL semantics and IPs. Expectations represent the *desired* agent behaviour, i.e., the possible courses of events that comply with the given IPs. They can be positive (events that are expected to happen) or negative (events that are expected not to happen), and it is possible to generate sets of alternative expectations, to model possible alternative “desired” courses of events.

In such an abductive framework, IC_S specify the link between events and expectations, modelled as abductive hypotheses. IC_S can be seen as forward rules, stating that if a conjunction of events has (not) happened, or is (not) expected (not to) happen, then one among several alternative situations is (not) expected (not to) occur.

Such situations are expressed as conjunctions of expected events, possibly containing variables whose domains can be related to each other by some constraints. CLP [12] constraints over variables allow for a fine-grained specification of expectations: in particular, they are often used to express time deadlines.

Once we have a formal specification of ACL semantics and IPs, it is possible to formally verify whether the behaviour of a group of agents is compliant with such a specification. The verification procedure is the abductive proof procedure *SC*-IFF [5, 2] inspired by Fung and Kowalski's IFF [10], augmented with transitions for constraint propagation and reasoning about events or expectations (in particular, to check if the events fulfill or violate the expectations). Given the specification of ACL semantics and IPs, and a history of events as input, the proof procedure yields as output a set of possible alternative sets of expectations, and an answer of *fulfillment* (which indicates that the interaction protocols have been satisfied by the interacting agents) or *violation* (which is caused by agents violating the IPs or behaving in contrast with the ACL semantic specifications).

As an example of ACL semantic specification, we express the semantics of *opauc*, a communicative action introduced in Section 2. Events are identified by the functor $\mathbf{H}$, while expectations are identified by the functor $\mathbf{E}$ (meaning positive expectations) or $\mathbf{EN}$ (meaning negative expectation).

$$
\begin{array}{l}
[IC_{open}]: \\
\mathbf{H}(tell(A, B, \textit{opauc}(\textit{Item}, T_{End}, T_{Dead}), A_{ID}), T_O), \\
\mathbf{H}(tell(B, A, \textit{bid}(\textit{Item}, Q), A_{ID}), T_{Bid}) : T_{Bid} < T_{End} \quad \Rightarrow \\
\quad \mathbf{E}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}) : \\
\quad T_{Win} < T_{Dead}, T_{End} < T_{Win} \quad \vee \\
\quad \mathbf{E}(tell(A, B, \textit{answ}(\textit{lose}, \textit{Item}, Q), A_{ID}), T_{Lose}) : \\
\quad T_{Lose} < T_{Dead}, T_{End} < T_{Lose}
\end{array}
$$

$[IC_{open}]$ formally states that, if an *opauc* event is issued by an agent A at a time T_O, announcing an auction henceforth identified by A_{ID}, where an *Item* is on sale, and where bids are accepted until time T_{End} and notification is given by time T_{Dead}, then, if a recipient B makes a *bid* by T_{End}, proposing a *Q*uote, A is expected to notify B by T_{Dead} whether B is the winner or not. This IC_S defining the production of expectations from a set of facts defines the social semantics of the communication action *opauc*. In other words, an agent A issuing an *opauc* action may modify, under certain circumstances, the state of social expectations, by introducing one among two new possible expectations: Saying this is enough to define the semantics of the act of A issuing *opauc*.

The *SC*-IFF operates based on a set of integrity constraints and on a history of events, and it generates a disjunction $\mathbf{EXP}$ of sets of events expected (not) to happen.

Given a set of IC_S and a history of events, at every step the *SC*-IFF generates a proof tree in which the leaves contain the state of the interaction, in terms of "history" of messages (called $\mathbf{HAP}$), and pending, fulfilled or violated expectations. It may be the case that, given a certain history $\mathbf{HAP}$, all the branches present a semantic inconsistency. The inconsistency may stems from the semantics given to the communicative actions or from the protocol definitions (for instance, the same event being both expected to happen, and expected *not* to happen), or it may be due to the wrong behaviour of an agent (for instance, an event expected not to happen which instead happens).

The possibility to detect such situations and to distinguish among various cases of inconsistency, which is a feature of the *SC*-IFF, is our basis for the interaction design methodology, which we will present in the next section. In particular, inconsistent situations can suggest modifications and refinements of the language and protocol specification, or of the properties that we want to hold in the system.

The *SC*-IFF has been implemented in SICStus Prolog [18], exploiting the Constraint Handling Rules (CHR) [9] and CLP(FD) libraries. The proof tree is explored with a depth-first strategy, thus enabling the implementation to exploit the Prolog stack directly. Most of the data structures representing each node of the computation tree are implemented as CHR constraints, so to exploit the CHR computational model for the implementation of *SC*-IFF transitions, which define its operational semantics [5]. The *SC*-IFF is integrated in the *SOCS-SI* tool, described in [3]. A demo example of interaction verification using *SOCS-SI* can be seen from [1].

4. A methodology for designing the agent interaction space using *SOCS-SI*

In the past, many methodologies have been proposed for designing and engineering multi-agent systems. We have cited the Gaia methodology [21], the KGR approach [14], and the Agentis approach [7]. A common characteristic of such methodologies is to be found in their spirit of helping the construction of a full-fledged agent system. Differently from them, in our methodology we do not aim at considering all aspects of multi-agent system design. We focus instead on agent interaction. Our goal is to help the design of ACLs, IPs, and the definition and verification of properties. Differently from the approach followed in [21], protocols are not given (agents could be given, instead, since our aim is not agent design). Also, we do not directly refer to the concept of role, although useful, but we aim at designing the interaction space independently of the social structure.

Before we describe the methodology, let us briefly identify the components coming into play in the interaction space design process and their relationships. The main actors are indeed the *agents*. They could be either seen as black-boxes exhibiting a behaviour to the outside, or as transparent components of a systems, or as partially known/observable entities. In the case of open systems, the first approach is often the one which is adopted, whereas the latter (*grey-box* model in [20]) is more suitable for the design of an agent system which needs to interact with other agent systems, as in the most general case.

Protocol and language definitions represent another component. In particular, we refer to the definition of the syntax, semantics, and pragmatics of ACLs, and to the definition of protocols, as it could be done, e.g., by means of AUML protocol diagrams.

A third component is the *observable behaviour* of agents, i.e., the output of their activity. It could be for instance the sequence of their communicative acts, or the physical delivery of goods.

Finally, we have the *properties*, which we would like to achieve by defining the interaction space. Such properties could be regarding the interaction itself, or its outcome, or both. We are mostly interested in those properties which can be formally defined, as it will soon become clear.

In the abstract framework that we propose, the design of the interaction space can be described as an iterated process consisting of the following phases:

1. definition/refinement of the *environment* (agents systems and interaction media): this can be done using a functional and data-flow representation;

2. definition/refinement of the *interaction space* (in particular, ACL semantics and protocol specification);

3. definition/refinement of formal *properties* that we would like the system to exhibit, and their *verification*;

4. if properties are disproved, back to phase 1.

Once a model is done which satisfies the properties that we have defined, it can be implemented into a concrete agent system. We will not discuss in this paper the ways to ensure that the model specification and its implementation are coherent with each other, although we stress that it is a very important point, and we indeed believe that logic programming can help in this. We will instead now analyze the steps above in more detail.

4.1. Environment definition

The *environment* is composed by the agents themselves, the communication media, and by the contextual entities that are relevant to the operation of the agent system.

In the auction example, the environment will include at least two agents having one of two possible roles (n bidders, $n \geq 1$, and one auctioneer), and a communication medium that permits bidirectional communication between auctioneer and each bidder.

The environment definition is part of our methodology, but we do not propose any new formalism or tool for it: it could well be done by following any of the aforementioned approaches and possibly achieving a first definition of the entities agents, and a concrete realisation of the multi-agent system. When we proceed to the subsequent phases, we might find out that the current definition of the environment does not allow for modelling a system which exhibits the properties that we are interested in. In that case, this step should be iterated starting from different assumptions. Indeed, in some cases part of the environment could be given as a specification.

4.2. Interaction space definition

The *interaction space* is defined in terms of ACLs and IPs. Although many ways are possible to give such definitions, since our main objective is to help the design of agent systems which exhibit some formally defined properties, we ought to consider formal approaches. In particular, will choose to use *Social Integrity Constraints* as a uniform means to specify both ACL semantics and IPs.

In order to put things more concretely, let us give the specification of the auction example. We have already defined the semantics of *opauc* by $[IC_{open}]$. Let us now define the semantics of *bid* and *win*:

$[IC_{bid}]$:
$\mathbf{H}(tell(B, A, \textit{bid}(\textit{Item}, Q), A_{ID}), T_{Bid}) : T_{Bid} < T_{End},$
$\mathbf{H}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}),$
$\mathbf{H}(tell(A, B, \textit{deliver}(\textit{Item}), A_{ID}), T_{Del}) \;\; \Rightarrow$
$\mathbf{E}(tell(B, A, \textit{pay}(Q, \textit{Item}), A_{ID}), T_{Pay}) :$
$T_{Pay} < T_{Del} + T_{Pay\_Deadline}$

$[IC_{win}]$:
$\mathbf{H}(tell(B, A, \textit{bid}(\textit{Item}, Q), A_{ID}), T_{Bid}) : T_{Bid} < T_{End},$
$\mathbf{H}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}), \;\; \Rightarrow$
$\mathbf{E}(tell(A, B, \textit{deliver}(\textit{Item}), A_{ID}), T_{Del})$
$T_{Del} < T_{Win} + T_{Deliver\_Deadline}$

Communicative acts, such as *bid* and *answ*(*win*), can be defined in a general enough way, such that we can use the same acts in different protocols (for instance, in other auction protocols). IPs can then be seen, in this perspective, as additional sets of constraints, defining relations among communicative actions, which are to be added to those already defining the ACL, and which have to be consistent with them. The specification of the IP in our example will be as follows:

$[IC_{open-if-bid}]$:
$\mathbf{H}(tell(B, A, \textit{bid}(\textit{Item}, Q), A_{ID}), T_{Bid}) \;\; \Rightarrow$
$\mathbf{E}(tell(A, B, \textit{opauc}(\textit{Item}, T_{End}, T_{Dead}), A_{ID}), T_O) :$
$T_O < T_{Bid}, T_{Bid} \leq T_{End}$

$[IC_{auc-win-no-lose}]$:
$\mathbf{H}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}), \;\; \Rightarrow$
$\mathbf{EN}(tell(A, B, \textit{answ}(\textit{lose}, \textit{Item}, Q), A_{ID}), T_{Lose}) :$
$T_{Lose} < T_{Win}$

$[IC_{auc-lose-no-win}]$:
$\mathbf{H}(tell(A, B, \textit{answ}(\textit{lose}, \textit{Item}, Q), A_{ID}), T_{Lose}), \;\; \Rightarrow$
$\mathbf{EN}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}) :$
$T_{Win} < T_{Lose}$

The first IC_S states that in the protocol a *bid* must be preceded by an *opauc*. The second and third IC_S state that an agent (auctioneer) may not tell both *answ*(*win*) and *answ*(*lose*) to the same agent, within the same auction.

This protocol specification is one choice among several options. The next phases will serve to decide whether we need to define the IP differently instead.

4.3. Properties definition and verification

An interaction space is "properly" designed if it exhibits some formally defined *properties*. For instance, in the design of an auction protocol, we would like to ensure that the agent who utters the highest bid will be assigned the goods at a certain price, and that it will pay for the price specified in its *bid*.

Following Pitt and Guerin [16], the definition of properties, again, could follow a declarative and logic-based methodology, and in particular it could be done by means of integrity constraints. Let us consider the following example, adapted from [16].

The property informally stated above can be formally defined as follows: "For all courses of events $\mathbf{HAP}$ such that there exists a consistent set of expectations $\mathbf{EXP}$, such that $\mathbf{EXP}$ is fulfilled by $\mathbf{HAP}$, $[IC_{prop}]$ holds", where $[IC_{prop}]$ is defined above:

$[IC_{prop}]$:
$\mathbf{H}(tell(B, A, \textit{bid}(\textit{Item}, Q), A_{ID}), T_{Bid}),$
$\neg\mathbf{H}(tell(B', A, \textit{bid}(\textit{Item}, Q'), A_{ID}), T'_{Bid})$:
$B' \neq B, Q' > Q \ \ \Rightarrow$
$\quad \mathbf{H}(tell(A, B, \textit{answ}(\textit{win}, \textit{Item}, Q), A_{ID}), T_{Win}),$
$\quad \mathbf{H}(tell(A, B, \textit{deliver}(\textit{Item}), A_{ID}), T_{Del})$:
$\quad T_{Del} < T_{Win} + T_{Deliver\_Deadline},$
$\quad \mathbf{H}(tell(B, A, \textit{pay}(Q, \textit{Item}), A_{ID}), T_{Pay})$:
$\quad T_{Pay} < T_{Del} + T_{Pay\_Deadline}$

Checking this property in this framework in general means considering all possible histories $\mathbf{HAP}$ complying with the ACLs and IPs, and checking whether $[IC_{prop}]$ is *entailed* by $\mathbf{HAP}$.

Verifying this kind of entailment given a specific history instance is not hard, since all history instances are ground sets of facts. Therefore, this method is directly applicable when the set of possible histories of events is countable, or when we are interested in verifying properties only in some particular situations, which could be given a-priori.

We are currently studying how to extend this methodology to tackle the more general case. It could be interesting to explore work done in model checking, to see if we can use some existing results for this purpose.

During the verification of properties, it may turn out that under some circumstances a property is not achieved. In that case, this step or some previous ones should be iterated, and either the interaction space definition or the target properties refined.

Let us assume, for the sake of example, that the interaction space has been defined without $[IC_{bid}]$. This means that there is no IC_S stating that after a *bid* and a notification of kind *answ*(*win*) the winning bidder will pay the price of the item(s) on sale. Therefore, it is possible to find a history of events which is compliant to the specification, but which does not entail $[IC_{prop}]$. Once this turns out, options could be: to modify $[IC_{prop}]$, if we realise that it is not what we want to achieve by the auction protocol, or to modify the semantics of the ACL, by introducing $[IC_{bid}]$.

Another situation that may occur is that some history instance produces only inconsistent sets of expectations, but no violation. This is normally a sign that the interaction space has been ill-defined. Conversely, we also aim to avoid having history instances which we intend to be marked as inconsistent, but for which the *SC*-IFF generates a success node. This normally means that the interaction space is under-constrained, and again, we need to refine its specification.

5. Discussion and Related Work

The contribution of this work is two-fold. On the one hand, we discuss about the problem of defining and reasoning about the agent interaction space, and propose a methodology where the designing process consists of several steps guided by the definition and verification of properties. On the other hand, we give a concrete instantiation of the abstract framework, based on the *SOCS-SI* platform and using a logic-based protocol definition language.

In Section 1 we have mentioned some well known methodologies aimed at designing agent systems. Our approach is rather focussed on the design of agent interaction than to agent systems as a whole. There are other tools which are more focussed on the social aspects of multi-agent systems design. They often define structured hierarchies, roles, and deontic concepts such as norms and obligations as first class entities. Among others, we cite ISLANDER, [8] which can be used for the specification and verification of complex social infrastructures, such as electronic institutions. ISLANDER allows to analyze situations, called scenes, and visualize liveness or safeness properties in some specific settings. Although we focus on social aspects, we do not aim at capturing complex institutional aspects, nor at helping programming single agents, but we rather propose a framework to reason about properties which can be formally defined and verified in the general context of agent interaction.

The framework that we propose focusses on interactions, but it does not ensure that the implemented system will indeed behave in the desired way. In particular, in the case of proprietary systems, one should make sure that the implementation reflects the specification of the system. This is not an easy task, and methodologies such as Gaia and formal approaches such as those based on computational logic have this as their main objective [17]. In the case of open systems instead, where the only available knowledge about the agents comes from the observation of their behaviour, the only thing that one can do is dynamic on-the-fly verification [16]. *SOCS-SI* can be used for this purpose, as it is described in [3].

We would like to conclude by discussing some weak points of our methodology. Firstly, we can easily imagine that system designers used to defining protocols based on other "pictorial" notations, such as AUML interaction diagrams or Coloured Petri Nets, will not feel comfortable with the IC_S formalism and its associated social semantics based on expectations. It could help to define a mapping between IC_S and other formalisms, but this does not seem to be an easy task, mainly because AUML and IC_S differ in spirit, and there could be several ways to translate an AUML interaction diagram into IC_S.

Secondly, the *SC*-IFF can be easily used to dynamically check the compliance of agent interaction to ACL semantics and IP definitions, based on the current history of events. It is not straightforward instead to apply it when a specific history instance to verify cannot be given. Other techniques, such as those based on model checking, seem to be well suited to dealing with this case.

In the future, we would like to investigate the possibility to map and automatically translate protocols defined by AUML diagrams into the IC_S formalism, and to define a library of protocols such as those proposed by the FIPA standardization body. In this way, a designer could test the properties of combinations of off-the-shelf solutions, aiming at producing a multi-agent system which is both compliant to the standards and exhibiting some desired formal properties. Also, we would like to explore the issue of automatic verification of properties, without having to enumerate all the possible history instances.

Acknowledgements

We sincerely appreciate the helpful suggestions of the anonymous reviewers. This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 SOCS project, and by the MIUR COFIN 2003 projects *Sviluppo e verifica di sistemi multiagente basati sulla logica*, and *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici*.

References

[1] A demonstration of SOCS-SI for AAMAS'04. Demo storyboard available from the public area of the SOCS web site: `http://lia.deis.unibo.it/research/SOCS`.

[2] The SCIFF abductive proof procedure. `http://lia.deis.unibo.it/research/sciff/`.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. In *Proc. 17th EMCSR*, pages 570–575, 2004. Austrian Society for Cybernetic Studies.

[4] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Modeling interactions using *Social Integrity Constraints*: a resource sharing case study. In *Declarative Agent Languages and Technologies*. *LNAI* 2990, pages 243–262. Springer, May 2004.

[5] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of interaction protocols: a computational logic approach based on abduction. Tech. Rep. CS-2003-03, Dip. di Ingegneria, Ferrara, Italy, 2003.

[6] M. Colombetti, N. Fornara, and M. Verdicchio. A social approach to communication in multiagent systems. In *Declarative Agent Languages and Technologies*, *LNAI* 2990, pages 193–222. Springer, May 2004.

[7] M. d'Inverno, D. Kinny, and M. Luck. Interaction protocols in Agentis. In *Proc. 3rd ICMAS*, pages 261–268, 1998.

[8] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proc. AAMAS*, pages 1045–1052, 2002. ACM.

[9] T. Frühwirth. Theory and practice of constraint handling rules. *J. of Logic Programming*, 37(1-3):95–138, Oct. 1998.

[10] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *J. of Logic Programming*, 33(2):151–165, Nov. 1997.

[11] D. Gelernter and N. Carriero. Coordination languages and their significance. *CACM*, 35(2):97–107, Feb. 1992.

[12] J. Jaffar and M. Maher. Constraint logic programming: a survey. *J. of Logic Programming*, 19/20:503–582, 1994.

[13] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *J. of Logic and Computation*, 2(6):719–770, 1993.

[14] D. Kinny, M. Georgeff, and A. S. Rao. A methodology and modelling technique for systems of BDI agents. In *Agents Breaking Away*, *LNCS* 1038, pages 42–55. Springer, 1996.

[15] A. Omicini and S. Ossowski. Objective versus subjective coordination in the engineering of agent systems. In *Intelligent Information Agents: An AgentLink Perspective*, *LNAI: State-of-the-Art Survey* 2586, pages 179–202. Springer, 2003.

[16] J. Pitt and F. Guerin. Guaranteeing properties for e-commerce systems. Tech. Rep. TRS020015, Dept. of Electrical and Electronic Engineering, Imperial College London, UK, 2002.

[17] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Intelligent Agents VIII*, *LNAI* 2333, pages 405–421. Springer, 2002.

[18] SICStus prolog user manual, release 3.11.0, Oct. 2003. `http://www.sics.se/isl/sicstus/`.

[19] M. P. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, *LNCS* 1916, pages 31–45. Springer, 2000.

[20] M. Viroli and A. Omicini. Specifying agent observable behaviour. In *Proc. AAMAS*, pages 712–720, 2002. ACM.

[21] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, Sept. 2000.