Electronic Notes in Theoretical Computer Science 85 No. 2 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume85.html 23 pages

Specification and Verification of Agent Interaction using Social Integrity Constraints

Marco Alberti [a] Marco Gavanelli [a] Evelina Lamma [a]
Paola Mello [b] Paolo Torroni [b]

[a] *Dipartimento di Ingegneria*
Università degli Studi di Ferrara
Via Saragat, 1 - 44100 Ferrara (Italy)

[b] *Dipartimento di Elettronica, Informatica e Sistemistica*
Università degli Studi di Bologna
Via del Rinascimento, 2 - 40126 Bologna (Italy)

Abstract

In this paper we propose a logic-based social approach to the specification and verification of agent interaction. We firstly introduce integrity constraints about social acts (called *Social Integrity Constraints*) as a formalism to express interaction protocols and to give a social semantics to the behavior of agents, focusing on communicative acts. Then, we discuss several possible kinds of verification of agent interaction, and we show how social integrity constraints can be used to verify some properties in this respect. We focus our attention on static verification of compliance of agent specifications to interaction protocols, and on run-time verification, based on agents' observable behavior. We adopt as a running example the NetBill security transaction protocol for the selling and delivery of information goods.

1 Introduction

Specification and verification are two important steps in the design and deployment of computer systems. In particular, they play a key role in the development of open systems, composed of autonomous entities that we would like to be to some extent predictable, or at least compliant to some interaction rules.

This is often the case in multi-agent systems. A possible approach to this problem is to rely on a formal computational framework, that can be used to give formal specifications to agent interaction, and to automatically check their "correct" behaviour. In doing this, two viewpoints are possible: one, which we could define *individual*, aims at designing agents so that they will "spontaneously" comply to the specifications; the other one, which we could

call *social*, aims at verifying the compliance of agents' *observable* behavior, regardless of their internal structure. The difference between the two approaches becomes evident in the definition of the semantics of Agent Communication Languages (ACLs).

Mentalistic approaches [6,7] define ACL semantics from an internal perspective, i.e., in terms of agents mental states. For instance, FIPA ACL [7] assumes a BDI (Belief, Desire, Intention) model for the agents, and relies upon it for defining the semantics of communicative acts in terms of Feasibility Preconditions (i.e., the conditions that have to be satisfied for the communicative act to be planned) and Rational Effects (i.e., the expected effect of the communicative act).

FIPA ACL offers a comprehensive set of communication primitives and its semantics appears adequate to account for *motivation* of communicative acts from an individual agent's viewpoint. However, its semantics, as that of other mentalistic ACLs, does not seem to fully meet the requirements of agent communication in open environments where heterogeneous agents interact [16]. In particular, whenever the agents' mental states are not accessible, as in the case of open societies, it is impossible to verify whether agents are acting according to the given semantics [17].

The social approach [17] defines the semantics of communicative acts in terms of their public (social) effects, rather than in terms of their internal (mental) motivations. Following this approach, compliance to a given semantics can be verified based on the agents' *observable* acts, even in cases where the internal state of agents is not accessible, or does not have an architecture described in terms of mental categories. Significant social proposals are based on the concept of *commitments* [8,19], computational objects which keep track of obligations between agents and can evolve depending on agents' (observable) communicative and non-communicative acts, and eventually be resolved to fulfillments or violations.

In this paper, we propose a logic-based approach to the specification and verification of agent interaction. In doing this, we adopt a social perspective, abstracting away from the agent internals.

The contribution of this paper is twofold. In the first part of the paper, we introduce a logic-based formalism (*Social Integrity Constraints*) for the specification of the expected behavior of agents.

We make no *a priori* assumptions on the internal structure of individual agents (in particular, we neither assume nor reject a BDI-like internal structure for individual agents), rather focusing on a public viewpoint on interaction. Our semantics is based on the concept of *expectation*, linked to events by social integrity constraints.

The aim of this first part of the paper is not to propose a new ACL, as a set of communication primitives, or a particular interaction protocol; rather, we propose a logic-based approach to the definition of social semantics for a given ACL, or an interaction protocol.

This specification is grounded on a computational logic framework, so making it possible to provide both a declarative specification of the desired behavior of interaction in a multi-agent system, and its automatic verification by means of a formal proof procedure.

In order to explain our ideas, we adopt the NetBill protocol as a running example. NetBill is a security and transaction protocol for the sale and delivery of low-priced information goods, such as software or journal articles. We chose NetBill because we believe that it is a good representative for the class of protocols related to information exchange in multi-agent systems, and it lends itself well to discussing the problems that can arise in this respect.

The study on verification of agent interaction is the subject of the second part of this paper. We refer to a classification of the different kinds of verification introduced by Guerin and Pitt in [14], which distinguishes among static verification, verification by observation, and verification of protocol properties. We focus on the first two kinds of verification. As for the first kind, we propose a method to verify the compliance of agent specifications to a set of protocols. As for the second kind, we sketch a prototypical implementation of the framework by means of *Constraint Handling Rules* [9], and show how this can be used to verify compliance by observation.

The paper is structured as follows. In Section 2 we give the necessary background by introducing the concept of social integrity constraint. In Section 3 we show how it can be exploited to specify the social semantics of communicative acts and the agent interaction protocols. Section 4 is devoted to studying the different kinds of verification. Discussion and directions for future work conclude the paper.

2 Social integrity constraints

In this section, we briefly describe a model of agent society, which is a simplified version of that presented in a companion paper [1].

In our model, a society is composed of a *social infrastructure* (its operational part) and a *knowledge base*, containing information about structure and properties of the society. In particular, the social knowledge base contains information about protocols and regulations for entrance, exit and role assignment. Such information is represented by in an (extended) logic program which we call *Social Organization Knowledge Base* (**SOKB**), and by *Social Integrity Constraints* (*SIC*) [1] . In addition, we assume that the society records in a "history" file **HAP** the observable and relevant events for the society (*happened* events, denoted by the functor **H**), including communicative and non-communicative acts exchanged or performed within the society.

[1] In this paper, for the sake of simplicity, we assume that **SOKB** is composed of ground facts. In [1], in order to capture more structured knowledge and goal-directed societies, we define **SOKB** and *SIC* as an abductive logic program.

For instance, $\mathbf{H}(tell(thomas, yves, start))$ denotes a social event by which an agent, *thomas*, tells another agent, *yves*, "*start*". It is a communicative act. $\mathbf{H}(do(thomas, yves, buy(ticket)))$ denotes a non-communicative act: an action made by *thomas* of buying a ticket from *yves*.

From a logic programming viewpoint, **HAP** can be understood as a set of ground facts defining a predicate indicated by the functor **H**.

A course of events **HAP** might give rise to social *expectations* about the future behaviour of its members. Expectations are collected in a set **EXP**. It will contain, in particular, events which are expected to happen (denoted by the functor **E**), and events which are expected not to happen (denoted by the functor **NE**).

For instance, $\mathbf{E}(do(thomas, yves, buy(ticket)))$ denotes an expectation about a non-communicative act: *thomas* is expected to buy a ticket from *yves*. $\mathbf{NE}(tell(yves, thomas, reject(ticket)))$ denotes a negative expectation about a communicative act: *yves* is expected not to *reject thomas*' request.

The social infrastructure records events; depending on events, updates the expectations in **EXP**. Moreover, monitors the agent interaction, for instance, in order to start appropriate recovery procedures in case of violations of expectations.

Protocols are formalized in terms of *Social Integrity Constraints* (*SIC*). *SIC* describe the evolution of the expectations in the society, based on the current history. The syntax of *SIC* is the following:

$$
\begin{aligned}
SIC &::= [\ ic_S\ ]^\star \\
ic_S &::= \chi \rightarrow \phi \\
\chi &::= \mathit{ExtendedLiteral}\ [\ \wedge \mathit{ExtendedLiteral}\ ]^\star \\
\phi &::= \mathit{ExpectList}[\text{: } \mathit{ConstraintList}] \\
\mathit{ExtendedLiteral} &::= \mathit{Expectation} \mid \mathit{Event} \mid \mathit{Atom} \\
\mathit{Expectation} &::= \mathbf{E}(\mathit{Term}) \mid \mathbf{NE}(\mathit{Term}) \\
\mathit{Event} &::= \mathbf{H}(\mathit{Term}) \\
\mathit{ExpectList} &::= \mathit{Expectation}\ [\ \wedge \mathit{Expectation}\ ]^\star
\end{aligned}
\tag{1}
$$

*Atom*s are defined (by ground facts) in **SOKB**.

ConstraintList is a conjunction of CLP [12] constraints such as — for instance — the $\leq$ relation that could be imposed between two variables of the integrity constraint.

All variables in the body χ of a social integrity constraint are universally quantified, with scope the entire integrity constraint, except those occurring only in an **NE** *Expectation* of χ, which are universally quantified with scope the **NE** *Expectation* in which they occur.

All other variables are existentially quantified, with scope the head ϕ of the integrity constraint in which they occur, except those occurring in an **NE** *Expectation* of ϕ, which are universally quantified, with scope ϕ.

A simple example of social integrity constraint is the following:

$$SIC = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y))\}$$

Its intuitive meaning is: if agent X told agent Y "*start*", then we expect a social event denoted by $pass(Y)$.

A more elaborate example, including CLP constraints, is the following social integrity constraint taken from an example that we will discuss in the next section.

$$\begin{aligned} &\mathbf{H}(request(B, A, P, D, T_r)) \\ \wedge&\mathbf{H}(accept(A, B, P, D, T_a)) \\ \wedge&T_r < T_a \\ \rightarrow&\mathbf{E}(do(A, B, P, D, T_d)) : T_d \leq T_a + \tau \end{aligned} \tag{2}$$

Intuitively, it means: if agent B sent a *request* P to agent A at time T_r, in the context of a dialogue D, and A sent an *accept* to B's *request* at a later time T_a, then A is expected to *do* P before a deadline $T_a + \tau$.

Based on SIC, we can define a semantics to the social interaction of agents. In the example above, (2) could be interpreted as a definition of the semantics of $request(B, A, P, D, T_r)$; $\mathbf{H}(accept(A, B, P, D, T_a)) \wedge T_r < T_a$ can be seen as a condition, and $\mathbf{E}(do(A, B, P, D, T_d)) : T_d \leq T_a + \tau$ as its social effect.

We first introduce the concept of *admissible set of social expectations*. Intuitively, given a society, and a set **HAP** of events, an admissible set **EXP** of social expectations consists of a set of expectations about social events that are compatible with the set **HAP**, with the society's **SOKB** and with the social integrity constraints.

More formally, we introduce the following definition.

Definition 2.1 Given a society's **SOKB** and SIC and a set of events **HAP**, an *admissible set* of social expectations **EXP** is a set of expectations such that:

$$\mathbf{SOKB} \cup \mathbf{HAP} \cup \mathbf{EXP} \models SIC \tag{3}$$

We interpret $P \models C$ as expressing that C is true in all the *intended* models of P. If we interpret social expectations as abducible predicates (see [13]) we can rely upon a three-valued model-theoretic semantics as intended meaning, as done, for instance, in a different context, by [10,5]. An abductive based semantics is discussed more extensively in [1].

We would like to stress that, up to now, we do not assume that expected events actually happen. This is in accordance with an *open* view for society where social expectations are just a suggestion for what should be done (or not done). A further refined semantics is then given by checking, for a history

HAP, whether the admissible set **EXP** of expectations is *fulfilled* by **H**. This reflects the *ideal* behaviour of a society.

Definition 2.2 Given a set of events **HAP**, an admissible set of social expectations **EXP** is *fulfilled* if and only if:

$$\mathbf{HAP} \cup \mathbf{EXP} \vDash \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \tag{4}$$

Example 2.3 Let us consider the following situation:

- $\mathbf{HAP}_1 = \{\mathbf{H}(tell(thomas, yves, start)), \mathbf{H}(pass(yves))\}$
- $SIC = \{\mathbf{H}(tell(X, Y, start)) \rightarrow \mathbf{E}(pass(Y))\}$

$\mathbf{EXP}_2 = \{\mathbf{E}(pass(yves))\}$ is the admissible and fulfilled set of expectations, w.r.t $\mathbf{HAP}_1$ and SIC.

But if we consider a different history:

$$\mathbf{HAP}_2 = \{\mathbf{H}(tell(thomas, yves, start))\},$$

then, $\mathbf{EXP}_2$ is not a fulfilled set of expectations w.r.t. $\mathbf{HAP}_2$ and SIC (still, it is admissible).

It may be impossible to fulfill an admissible set of expectations, due to its possible intrinsic incoherence. This is captured by the following definition.

Definition 2.4 A set of social expectations **EXP** is *coherent* if and only if :

$$\nexists p : (\{\mathbf{E}(p), \mathbf{NE}(p)\} \subseteq \mathbf{EXP})$$

If both $\mathbf{E}(p)$ and $\mathbf{NE}(p)$ belong to a set **EXP** of expectations, it cannot be fulfilled: if $\mathbf{H}(p)$ belongs to the history, $\mathbf{E}(p)$ is not fulfilled; otherwise, $\mathbf{NE}(p)$ is not fulfilled.

Even if an admissible set of expectations is coherent, it may not be fulfilled as well. The reason is the *violation* of the protocol: some agent did not behave as expected.

Definition 2.5 Given a society and a set **HAP** of events, if there exists at least an admissible and coherent set of expectations, and all admissible and coherent sets of expectations are not fulfilled, then we say that **HAP** produces a *violation* in the society.

In [1] we define the full syntax of social integrity constraints, the scope of variables, quantification, and we give a formal semantic characterization of our framework in terms of abduction.

3 Constraint-based specification of agent interaction

In this section, we introduce the concept of constraint-based specification of agent interaction at the level of ACL.

Agent communication, from a social viewpoint, can be constrained in two ways:

(i) expectations of the society on the behavior of agents as consequence of communicative acts, in a similar way to that of commitment-based ACL semantics [17,8];

(ii) allowed sequences of communicative acts (which is commonly indicated as interaction protocols).

In literature, these two types of specification are most commonly expressed with distinct formalisms, and compliance of agents to those specifications is supposed to be verified at different levels.

Social integrity constraints allow for the specification and, thanks to their grounding on computational logics, for the verification of both the aspects. The resulting specification is

- *formal*: the declarative semantics of communicative acts is defined in a computational logic framework (see Section 2);
- *verifiable*: compliance to social integrity constraints of an event history to a declarative specification can be checked by an appropriate proof procedure;
- *flexible*: the semantics of a given set of communicative acts, or the set itself, can be easily adapted to the application.

In this section we demonstrate, by simple examples, these features with respect to social ACL semantics (Section 3.1) and interaction protocols (Section 3.2). Verification will be covered in Section 4.

In the examples, we will represent events in the form

$$EventName(Performer, Addressee, Content, Dialog, Time)$$

where *EventName* is the event type, *Performer* is the agent performing the associated action, *Addressee* is the receiver of the action, *Content* is the content, *Dialog* is an identifier of the interaction where the event takes place,[2] and *Time* is the time at which the event happens.

3.1 Social ACL semantics

Social integrity constraints can be used to express that expectations are raised on the behavior of agents as consequence of their communicative acts. For the sake of synthesis, we show a simple example where only three communication

[2] The *Dialog* parameter is used to separate the contexts of several dialogues that an agent can handle. For instance, it can be used to distinguish communicative acts with the same content: two requests to buy a book with the same title can be considered a repeated request if the *Dialog* parameter is the same, and distinct requests to buy two copies of the book if the *Dialog* parameters are different.

primitives are considered (namely *request*,*accept* and *reject*).[3] We also consider a non-communicative action (*do*) to show how these can interact with expectations raised by communicative actions (i.e., by fulfilling or violating them).

Constraint (5) means that if an agent does *accept* a *request*, it is obliged to fulfill it by some deadline (for instance, by τ time ticks, where τ is some constant). We call *forward* this kind of expectations, in the sense that they should be fulfilled by some event in the future.

$$\begin{aligned} &\mathbf{H}(request(B, A, P, D, T_r)) \\ \wedge&\mathbf{H}(accept(A, B, P, D, T_a)) \\ \wedge&T_r < T_a \\ \rightarrow&\mathbf{E}(do(A, B, P, D, T_d)) : T_d \leq T_a + \tau \end{aligned} \tag{5}$$

It should be noticed that a *request* does not, by itself, cause any expectation to be raised.

do is a physical action, which fulfills the expectation of constraint (5) if it matches with its content, provided that it is performed before a certain amount of time τ has passed since the request has been *accept*ed. It is not necessary to impose any constraint to express the semantics of *do*, because this is captured by the semantics of our social framework. As it will be explained in Section 4, matching an expectation with a corresponding event can be performed automatically.

No constraint is necessary to express the semantics of a *reject* act because a *reject*ed *request* generates no expectation.

In the following paragraph, we show a more complex example.

Expectations in the NetBill protocol.

NetBill [4] is a security and transaction protocol for the sale and delivery of low-priced information goods, such as software or journal articles. The protocol rules transactions between two actors: *merchant* and *customer*. Accounts for merchants and customers, linked to traditional financial accounts (like credit cards), are maintained by a NetBill server.

The protocol prescribes a price negotiation phase, in which the customer presents evidence of his/her identity, and requests the quote for a good, and the merchant answers with the price. Then, the customer accepts or declines the offer; acceptance constitutes an order for delivery of the goods. The merchant then delivers the goods under encryption (remind they are information goods), witholding the encryption key. To fulfill payment, the customer constructs and digitally signs an Electronic Payment Order (EPO) and sends it back to the merchant, who appends the encryption key to it, digitally sign it and

[3] We do not use here any standard syntax for agent communicative acts, but the primitives that we use could be easily mapped into - say - a FIPA ACL communication action.

forward it to the NetBill server. The NetBill server takes care of the actual money transfer and returns a digitally signed receipt, including the key, to the merchant. The merchant forwards receipt and key to the customer, who will thus be able to decrypt the good.

In the following, we suppose that the roles of consumer and merchant are played by agents. We also suppose that primitives, with intuitive meaning, are available for all actions involved in the protocol. We consider a simplified version of the protocol, where we do not specify anything about the negotiation phase (also, this phase could remain unconstrained, at a social level). Finally, the protocol does not deal with deadlines, so neither will we: no constraints are imposed on time variables in expectations.

A possible specification of the protocol by means of social integrity constraints follows (constraints (6) to (8)):

$$\begin{aligned} &\mathbf{H}(\mathit{present}(M, C, (G, Q), D, T_p)) \\ \wedge&\mathbf{H}(\mathit{accept}(C, M, (G, Q), D, T_a)), \\ &T_p < T_a \\ \rightarrow&\mathbf{E}(\mathit{deliver}(M, C, (G, Q), D, T_d)) \end{aligned} \tag{6}$$

An expectation for merchant M to *deliver* the good is generated by social integrity constraint (6) only if *both* a *present* and an *accept* event have occurred. If consumer C *accept*s a quote which merchant M has not *present*ed, no expectation is raised.

$$\begin{aligned} &\mathbf{H}(\mathit{deliver}(M, C, (G, Q), D, T_d)) \\ \wedge&\mathbf{E}(\mathit{deliver}(M, C, (G, Q), D, T_d)) \\ \rightarrow&\mathbf{E}(\mathit{epo}(C, M, (G, Q), D, T_e)) \end{aligned} \tag{7}$$

Constraint (7) is effective (i.e. generates an *epo* expectation) only if the *deliver* event corresponds to an existent *deliver* expectation. Since a *deliver* expectation can only be raised by constraint (6), this implies that corresponding *present* and *accept* events have occurred before; in other words, this prevents consumer C from being obliged to pay for unrequested goods that have been delivered to it. The same mechanism is used in constraint (8).

$$\begin{aligned} &\mathbf{H}(\mathit{epo}(C, M, (G, Q), D, T_e)) \\ \wedge&\mathbf{E}(\mathit{epo}(C, M, (G, Q), D, T_e)) \\ \rightarrow&\mathbf{E}(\mathit{receipt}(M, C, (G, Q), D, T_r)) \end{aligned} \tag{8}$$

3.2 Interaction protocols

Our framework can be used to express allowed sequences of communicative acts, as well as their social semantics. This is achieved by means of *backward* expectations, which regard events that *should have happened* when the expectation is raised. In this way, we can express by constraints (9) and (10) that

*accept*s and *reject*s, respectively, should only be issued after corresponding *request*s.

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(A,B,P,D,T_a)),\\ \rightarrow&\mathbf{E}(\mathit{request}(B,A,P,D,T_r)) : T_r < T_a\end{aligned} \tag{9}$$

$$\begin{aligned}&\mathbf{H}(\mathit{reject}(A,B,P,D,T_2))\\ \rightarrow&\mathbf{E}(\mathit{request}(B,A,P,D,T_1)) : T_1 < T_2\end{aligned} \tag{10}$$

If an *accept* (or a *request*) is issued and a corresponding *request* has not happened before, the expectation in constraint (9) (or (10)) will be violated.

Backward expectations can be used in the same way to constrain communicative acts in the NetBill protocol to be issued in the desired sequence.

In line with the philosophy of Yolum and Singh [20], in the protocol specification we tend to constrain agents' interaction as little as possible, only when it is needed. In this example, if an agent makes the same *request* twice in the same dialogue, for instance because it is not sure whether the intended recipient received the first one, we do not consider this as a violation of the protocol. [4] The same holds for a repeated communication of *accept* or *reject*. [5] Instead, we do want to prevent an agent from contradicting a previous *accept* by subsequently sending a *reject*, or vice versa. To this purpose, we introduce the two following constraints:

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(B,A,P,D,T_a))\\ \rightarrow&\mathbf{NE}(\mathit{reject}(B,A,P,D,T_r)) : T_r > T_a\end{aligned} \tag{12}$$

$$\begin{aligned}&\mathbf{H}(\mathit{reject}(B,A,P,D,T_r))\\ \rightarrow&\mathbf{NE}(\mathit{accept}(B,A,P,D,T_a)) : T_a > T_r\end{aligned} \tag{13}$$

It is worthwhile noticing that, in our framework, we use the same proof for concepts that are semantically different. In fact, in the specification of agent interaction we can distinguish between regulative rules, like for instance: "accepted requests are expected to be fulfilled in the future", constitutive rules, like: "acceptance is expected to correspond to a past request", and analytic constraints, like: "to messages cannot be sent at the same time by the same agent". Analytic constraints predicate about the correct functioning of the

[4] Instead, if an agent wants to request two instances of the same good, it will then initiate two different dialogues.

[5] However, if we wish to prevent agents from repeating a communicative act, e.g. *accept*, twice in the same dialogue, we can write the following constraint:

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(B,A,P,D,T_{a1}))\\ \rightarrow&\mathbf{NE}(\mathit{accept}(B,A,P,D,T_{a2})) : T_{a2} > T_{a1}\end{aligned} \tag{11}$$

In the sequel, we assume that an agent cannot send two communicative actions at the same time. This could be physically enforced by a suitable social infrastructure, which "officially" time-stamps their messages using a fine-grained time tick.

system with respect to the model, in terms of "physical" rules that must never be violated. They do not generate expectations, and we did not include them in this paper, but they can indeed be considered in the framework and modelled as integrity constraints. The distinction between regulative/normative rules and constitutive rules instead would become particularly explicit in the case of sanctions, and countermeasures to be taken when a violation occurs, but we are not dealing with sanctions yet.

4 Automatic verification of agent interaction using social integrity constraints

In this section, we discuss two approaches to the verification of compliance of agent interaction to a specification given by means of social integrity constraints.

In [11,14], F. Guerin and J. Pitt propose a classification of properties that are relevant for e-commerce systems, in particular with respect to properties of protocols and interaction. They propose a formal framework for verification of properties of "low level computing theories required to implement a mechanism for agents" in an *open* environment, where by "open" the authors mean that in general it is not possible to observe or constrain the agents' mental state.

Verification of properties is classified into three types, depending on the information available and whether the verification is done at design time or at run time:

Type 1: verify that an agent will always comply;

Type 2: verify compliance by observation;

Type 3: verify protocol properties.

As for *Type 1* verification, the authors propose using a model checking algorithm for agents implemented by a finite state program. As for *Type 2* verification, the authors refer to work done by Singh [17], where "agents can be tested for compliance on the basis of their communications", and suggest policing the society as a way to enforce a correct behaviour of its inhabitants. As for verification of *Type 3*, the authors show how it is possible to prove properties of protocols by using only the ACL specification. They construct a fair transition system representing all possible observable sequences of states and prove by hand that the desired properties hold over all computations of the multi-agent system. This type of verification is demonstrated by an auction example.

The formal framework that we introduced, based on social integrity constraints, lends itself well to verification of Type 2 and 3, which do not rely on a representation of the agents' internals. Moreover, if we are able to specify the internal policies of agents in a logic programming based formalism, similar to the one that we use for social integrity constraints, then it becomes possible

to tackle verification of Type 1 by relating social and individual aspects, as we will see in the following. By proposing our framework, we aim at uniformly and automatically dealing with all the above three types of verification, in a computational logic setting.

In general, verification that an agent will always comply cannot be done by externally monitoring its behaviour. For this kind of verification, we need to have access to at least a specification to the agents' internals. Section 4.1 presents a mechanism to automatically obtain such a proof (or its failure) for a restricted set of programs and protocols.

For verification of Type 2 we need to be able to observe the agents' social actions, i.e., the communicative acts that they exchange. As in [14], we can assume that this can be achieved by policing the society. In particular, "police" agents will "snoop" the communicative acts exchanged by the society members and check at run time if they comply with the protocol specifications (social integrity constraints). Section 4.2 is devoted to the presentation of a mechanism for run-time verification, based on Constraint Handling Rules.

Verification of Type 3 is about protocol properties. In order to prove them we do not need to access the agents' internals, nor to know anything about the communication acts of the system, because such a verification is statically done at design time. We will not present any result on this respect in this paper.

4.1 Static verification of agent compliance to protocols

In a very general understanding, verification that an agent "will always comply" poses obvious decidability problems. For this reason, we will not approach this kind of verification by looking at the agent code, but based on some specifications of the social behaviour of the agent of which we want to predict compliance. We will therefore talk about compliance of an agent's specifications to a set of social integrity constraints expressing interaction protocols.

The specifications of agent social behaviour could be given again in terms of integrity constraints. We will call such constraints ic_σ. They have the following syntax:

$$\begin{aligned} ic_\sigma &::= \chi \rightarrow \phi \\ \chi &::= Event\ [\ \wedge Event\ ]^\star \\ \phi &::= Event\ [\ \vee Event\ ]^\star \\ Event &::= \mathbf{H}(Atom) \end{aligned} \tag{14}$$

All variables in the body χ of an integrity constraint ic_σ are universally quantified, with scope ic_σ. All other variables are existentially quantified, with scope the **H** predicate in which they occur.

The integrity constraints ic_σ express the possible reactions of an agent to

an external input (typically, a communicative action). The way to read a constraint of the kind $\chi \rightarrow \phi$ is the following: if χ is the case, then *at some point* the agent will make ϕ true. χ is based on a possible history of the agent interaction, therefore it is a conjunction of events.

In general, we assume that it is possible to give the specifications of the social behaviour of an agent in terms of integrity constraints ic_σ. It would indeed be possible, for instance, if agents implemented an operational model such as that proposed in [15]: in that case, the agent specifications could be automatically derived from the agent program (negotiation policies and dialogue cycle), by a suitable syntactic transformation.

Let us give a concrete example of such constraints. Let us consider a consumer agent, c, and a merchant agent, m, whose task is to buy/sell goods using a NetBill protocol. We can assume that both c and m are members of a society s, which defines the NetBill protocol in terms of social integrity constraints.

c's specifications could be the following:

$$\begin{aligned}&\mathbf{H}(\mathit{present}(M, c, (G, Q), D, T_1))\\ \rightarrow&\mathbf{H}(\mathit{accept}(c, M, (G, Q), D, T_2))\end{aligned} \tag{15}$$

$$\begin{aligned}&\mathbf{H}(\mathit{deliver}(M, c, (G, Q), D, T_1))\\ \rightarrow&\mathbf{H}(\mathit{epo}(c, M, (G, Q), D, T_2))\end{aligned} \tag{16}$$

m's specifications could be the following:

$$\begin{aligned}&\mathbf{H}(\mathit{request}(C, m, (G, Q), D, T_1))\\ \rightarrow&\mathbf{H}(\mathit{present}(m, C, (G, Q), D, T_2))\end{aligned} \tag{17}$$

$$\begin{aligned}&\mathbf{H}(\mathit{accept}(C, m, (G, Q), D, T_1))\\ \rightarrow&\mathbf{H}(\mathit{deliver}(m, C, (G, Q), D, T_2))\end{aligned} \tag{18}$$

$$\begin{aligned}&\mathbf{H}(\mathit{epo}(C, m, (G, Q), D, T_1))\\ \rightarrow&\mathbf{H}(\mathit{receipt}(m, C, (G, Q), D, T_2))\end{aligned} \tag{19}$$

In this simplified example, we considered only deterministic agent specifications. In general, those specifications will be non-deterministic, and they will need to be expressed as constraints that may have disjunctions in the head. The NetBill protocol as defined in [4], for instance, allows for reiterating the negotiation process if needed, for abandoning the interaction after an unsuccessful negotiation process, for not sending the electronic payment order in case the delivered good is corrupted, etc. This makes the protocol non deterministic and allows for compliant non deterministic agent specifications. An example of such specifications could be the following:

- c could replace (15) with the following:

$$\begin{aligned} &\mathbf{H}(\mathit{present}(M, c, (G, Q), D, T_1)) \\ \rightarrow &\mathbf{H}(\mathit{accept}(c, M, (G, Q), D, T_2)) \\ \vee &\mathbf{H}(\mathit{refuse\_quote}(c, M, (G, Q), D, T_2)) \end{aligned} \tag{20}$$

- m could replace (17) with the following:

$$\begin{aligned} &\mathbf{H}(\mathit{request}(C, m, (G, Q), D, T_1)) \\ \rightarrow &\mathbf{H}(\mathit{present}(m, C, (G, Q), D, T_2)) \\ \vee &\mathbf{H}(\mathit{refuse\_request}(C, m, (G, Q), D, T_2)) \end{aligned} \tag{21}$$

In the sequel, we will give a notion of compliance of agents/specifications based on a restricted class of protocols and specifications. This simplification lets us provide some initial results about static verification of agents/specifications.

Let us restrict ourselves to the case of SIC expressing protocols in the form $\chi \rightarrow \phi$, where $\phi ::= \mathit{Expectation}$ (no constraints are imposed on variables in expectations). Let us consider a subset $SIC(a)$ of SIC only containing constraints whose head is an expectation about the behaviour of agent a (e.g., a message sent by a). We want to check for compliance of a to $\mathcal{IC}(a)$. [6]

Let σ be a's specification, given in terms of constraints, and let σ be deterministic in the sense explained above. Let $ic_S^i \in SIC(a)$ be the i-th social integrity constraint in $SIC(a)$, $ic_S^i = \chi_i \rightarrow \mathbf{E}(\alpha_i)$ or $ic_S^i = \chi_i \rightarrow \mathbf{NE}(\alpha_i)$. There could be different cases:

(i) $\forall ic_S^i = \chi_i \rightarrow \mathbf{E}(\alpha_i)$, $\sigma \cup \chi_i \vDash \mathbf{H}(\alpha_i)$ (positive compliance);

(ii) $\forall ic_S^i = \chi_i \rightarrow \mathbf{NE}(\alpha_i)$, $\sigma \cup \chi_i \nvDash \mathbf{H}(\alpha_i)$ (negative compliance);

(iii) both (i) and (ii) hold (strong compliance);

(iv) neither (i) nor (ii) holds.

If the agent implementation respects the specifications given in σ, then, depending on which of the four classes of compliance defined above it falls in, we know what we can expect by its interaction w.r.t. the protocols defined in $SIC(a)$. (i) can be seen as a liveness property, while (ii) as a safeness property. In particular, if the agent is positive compliant (i), it will always produce a move which is expected according to $SIC(a)$ (but we do not know whether it will also produce some other moves which violate some protocol). If the agent is negative compliant (ii), it will never produce a move which is expected not to be produced (but we do not know whether it will fulfill all expectations). If the agent is strong compliant (iii), it will always comply with the protocol, by fulfilling all expectations and not violating any social

[6] It is up to the user to select the set of constraints that are relevant to determine if a is compliant or not, although this process could be easily made automatic, for instance by introducing a notion of "role" (agent compliant "as a merchant", "as a customer", as both, etc.

integrity constraint. Finally, if the agent's specifications fall in the last class (iv), we are not able to say a priori how the agent will behave.

These results hold for a restricted class of programs and protocols. We gave a declarative characterization of the concept of compliance. Since such a characterization is based on a concept of entailment, we believe that its operational counterpart would not be difficult to obtain as a suitable extension of classical deductive inference engines.

In the NetBill example, we can see that the deterministic version of c is strongly compliant to $SIC(c) = \{(7)\}$, and the deterministic version of m is strongly compliant to $SIC(m) = \{(6), (8)\}$.

4.2 Dynamic verification using Constraint Handling Rules

Verification of Type 2 is aimed at checking the agents' *observable* behavior for compliance to a given specification. We have given the declarative definition of this kind of compliance in Sect. 2.

Operationally, compliance of the event history to the specification of a society must be verified by an appropriate proof procedure. Obviously, such proof procedure should operate incrementally as agents interact, so to make it possible to detect violations as soon as they occur and possibly start recovery procedures. In particular, constraint propagation techniques can be used to detect as early as possible whether an expectation will never be fulfilled (see Section 4.2.2). Intuitively, expectations should be generated so to satisfy social integrity constraints: when all the events and expectations in the body χ of a social integrity constraint $\chi \rightarrow \phi$ have been inserted into the **HAP** and **EXP** sets by the society, the expectations in the head ϕ should be raised. Expectations will remain pending until they are either fulfilled or violated, and the social infrastructure will be watching them to check whether happened events match and fulfill them or violate them.

Essentially, *Constraint Handling Rules* [9] (*CHR* for brevity hereafter) represent a committed-choice language consisting of guarded rules that rewrite constraints in a store into simpler ones until they are solved. *CHR* define both *simplification* (replacing constraints by simpler constraints while preserving logical equivalence) and *propagation* (adding new, logically redundant but computationally useful, constraints) over user-defined constraints.

The main intended use of *CHR* is to write constraint solvers, or to extend existing ones. However, although ours is not a classic constraint programming setting, the computational model of *CHR* presents features that make it a useful tool for the implementation of the proof procedures we are aiming at. In the following, we sketch a prototype implementation, by means of *CHR*, of a verification procedure of compliance to social integrity constraints, which will be extended and refined in future work. It is worthwhile noticing that what follows describes an operational counterpart of the declarative framework described in Section 2, and a specification as that described in Section 3.1 is

a parameter to the implementation. This allows for easy implementation of extensions of the specification.

4.2.1 Representation of entities

We assume that the proof procedure for Type 2 verification is part of a social infrastructure there agents live. Along the whole life of the society, the social infrastructure repeats the following two operations in a cycle:

(i) record an event;

(ii) handle the event according to the specification of the society.

In the prototype implementation described in this work, events are simulated by the programmer. In an open implementation, this task could be accomplished by a suitable network communication mechanism, such as a socket-listening procedure.

Time.

In the current implementation, time is represented by an integer number, incremented by one at each clock tick.

The current time is represented by the *CHR* constraint [7] `current_time/1`, whose argument is the number of clock ticks since the beginning of the life of the society. The current time is updated by means of a *simplification CHR*. Simplification *CHR*s are of the form

$$H_1, \ldots, H_i \iff G_1, \ldots, G_j | B_1, \ldots, B_k \tag{22}$$

with $i > 0$, $j \geq 0$, $k \geq 0$ and where the multi-head $H_1, \ldots, H_i$ is a nonempty sequence of *CHR* constraints, the guard $G_1, \ldots, G_j$ is a sequence of built-in constraints, and the body $B_1, \ldots, B_k$ is a sequence of built-in and *CHR* constraints.

Declaratively, a simplification rule is a logical equivalence, provided that the guard is true. Operationally, when constraints $H_1, \ldots, H_i$ are in the head are in the store and guard $G_1, \ldots, G_j$ is true, they are substituted by constraints $B_1, \ldots, B_k$ in the body.

This operational behavior can be exploited to update the current time at every clock tick by means of the following simplification rule:

```
clock,
current_time(OldTime)
<=>
NewTime is OldTime + 1|
current_time(NewTime).
```

where constraint `clock/0` is imposed when a clock tick is wanted.

[7] i.e., a constraint which is defined by means of *CHR*s and is not built-in.

Events.

Happened events are represented by a *CHR* constraint, `happened/1`, whose argument is a ground term describing the event. For instance, if, during the dialog d and at time 10, $agent_1$ has *accept*ed to *give* a *nail* to $agent_2$, the following constraint will be stored:

`happened(accept(agent1,agent2,give(nail),d,10))`.

Expectations.

Expectations are represented by `expect/2` constraints. The first argument is the event associated with the expectation, the second is a list of constraints over the variables contained in the event [8]. For instance, the expectation for $agent_1$ to *give* a *nail* to $agent_2$ by time 20 in the context of dialog d would be represented as follows:

`expect(do(agent1,agent2,give(nail),d,T),[tle(T,20)])`,

where `tle` is the binary constraint of lesser-or-equal between time of events (which needs to be distinguished from an ordinary $\leq$ constraint between numeric variables as will be explained in Section 4.2.2).

Negative expectations.

Negative expectations are represented by `expectnot/2` constraints, having the same structure of `expect/2` constraints.

Social integrity constraints.

Social integrity constraints are represented by *propagation CHR*s. In general, propagation rules have the form

$$H_1, \ldots, H_i \Longrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{23}$$

where the symbols have the same meaning and constraints of those in the simplification rules (22).

Declaratively, a propagation rule is an implication, provided that the guard is true. Operationally, when the constraints in the head are in the store, and the guard is true, the constraints in the body are added to the store. This mechanism can be exploited to post expectations into the store as prescribed by social integrity constraints as follows:

- events and expectations in the body of the constraint are represented by constraints in the head of the *CHR*;
- atoms in the body of the constraint are represented by built-in constraints in the guard of the *CHR*;
- expectations (with the associated constraints) in the head of the constraint are represented by constraints in the body of the *CHR*.

[8] In the current implementation, expectations cannot share constrained variables.

For instance, constraint (9) can be represented by the following propagation rule[9]:

```
happened(accept(Agent2,Agent1,Content,Dialog,AcceptTime))
==>
expect(request(Agent1,Agent2,Content,Dialog,RequestTime),
       [tlt(RequestTime,AcceptTime)]).
```

Whenever an event matching[10] the head is added to the store, this rule will add the corresponding expectations to the store.

4.2.2 Procedures

Fulfillment and violation of expectations.

Operationally, an event fulfills an expectation if the contents of the event and the expectation unify in a Prolog sense and the constraints in the expectation are true. In case of negative expectations, indeed we do not have a fulfillment but a violation. This is achieved by the following *propagation CHR*:

```
happened(HEvent),
expect(EEvent,Constraints)
==>
HEvent=EEvent,
verify_constraints(Constraints)|
fulfilled_expect(EEvent,Constraints).
```

A fulfilled expectation is stored as a `fulfilled_expect/2` constraint.

`verify_constraints/1` is a user-defined predicate which checks for the truth of a constraint with ground arguments.

For instance, let us suppose the following expectation is in the store:

`expect(do(agent1,agent2,give(nail),d,T),[tle(T,20)])`,

and that the following event happens at time 15:

`happened(do(agent1,agent2,give(nail),d,15))`.

Unification of the content of the expectation and the event succeeds with the binding `T=15`. Since `tle(15,20)` is true, the expectation is fulfilled, and, operationally, can be stored as a `fulfilled_expect/2` constraint.

Exploiting constraint propagation for early detection of violations.

The mechanism described in the previous paragraph is not adequate for all kinds of expectations.

Let us suppose that an expectation with an associated deadline is in the store. If the social infrastructure only waits for a matching event to happen,

[9] As can be noticed, the mapping of a social integrity constraint into a *CHR* is extremely straightforward, thus making it easy to implement extension to a given specification.

[10] As it is explained in the next Section, by "matching" we mean "unifying with", à la Prolog.

and the event never happens, the violation of the deadline will never be detected. Obviously, this is not the desired behavior: we want the violation to be detected as soon as time is over.

More generally, violations of certain expectations need to be detected without waiting for some event to happen. In order to handle this case, we exploit constraint propagation: when the variables of an expectation are constrained by certain constraints (such as `tle/2`) new *CHR* constraints are inserted in the store, meant to interact with other constraints representing conditions which may change (such as the current time).

For example, let us consider deadlines again. If T is a variable representing the time at which an event with an associated deadline should happen, and T is unbound, then the event has not happened yet, because otherwise the event would have been recorded in the store and T would have been instantiated to a specific value[11] . Thus, if $T_{current}$ represents the current time, we can infer the following inequality:

$$T_{current} \leq T \tag{24}$$

which has the intuitive meaning that, if an event has not happened yet, it may only happen in the future. Inequality (24), combined with a constraint

$$T \leq T_{deadline} \tag{25}$$

implies

$$T_{current} \leq T_{deadline} \tag{26}$$

Thus, if the current time goes beyond the deadline while T is unbound, the associated expectation cannot be fulfilled, and a violation can be detected (or, dually, the associated negative expectation cannot be violated, and its fulfillment can be detected).

In order to implement this behavior in our framework, whenever an expectation is imposed which contains a constraint expressing a deadline (such as `tle(T,20)`), the social infrastructure infers a `tcurrent_le/1` constraint (which means that the current time must not exceed the argument, unified with the deadline) and associates it to the original expectation. If the `tcurrent_le/1` constraint is violated (which is detected by means of a *CHR*), a violation/fulfillment of the original expectation is detected; if the expectation is fulfilled/violated) before the deadline, the associated `tcurrent_le/1` constraint is removed.

5 Discussion and future work

In this paper, we put together two aspects: that of specification of agent interaction and that of formal verification. We have approached both problems from a social viewpoint which, in our opinion, seems to better fulfill

[11] We always assume that the social infrastructure is aware of all socially significant events.

the requirements of open societies of autonomous and heterogeneous agents, compare to a mentalistic approach.

In part, we draw inspiration from [8], where an operational specification of an ACL is given in an object-oriented framework by means of the *commitment* class. A commitment represents an obligation for its *debtor* towards its *creditor*. A commitment is described by a finite state automaton, whose states (which can take the values of *empty*, *pre-commitment*, *canceled*, *conditional*, *active*, *fulfilled* and *violated*) can change by application of methods of the commitment class, or of rules triggered by external conditions. Semantics of communicative acts is specified in terms of methods to be applied to a commitment when a communicative act is issued.

The approach presented in [8] is similar to ours in that it is social-based: it makes no assumptions on the nature of agents, it specifies semantics of actions with respect to their social effects, and it presupposes a social framework (which is called *institution* in [3]) for assigning roles to agents, for verifying the agent social behavior and, possibly, for recovering from violation conditions.

There are, however, some significant differences, mainly originating from the different paradigm we have chosen to express semantics (logic-based instead of object-oriented). Furthermore, our notion of expectation is more general than that of commitment: it represents the necessity of a (past or future) event, and is not bound to have a debtor or a creditor, or to be brought about by an agent. For instance, if we want to express the constraint that an agent is only *allowed* to perform an act when a previous event has occurred (not necessarily for an agent's action), we can simply impose it as a backward expectation, whereas it is not obvious how to express this in Fornara and Colombetti's framework [8].

Our framework can express with the same formalism both protocols and social semantics of communicative acts. In [18], we have exploited social integrity constraints for expressing protocols for an agent society rather than for the semantics of communication.

Another interesting way of linking social semantics of communicative acts and protocol specification is described in [20]. However, in [20] it is the single agent which, by exploiting its reasoning/planning capabilities, must find a communication path leaving no pending commitments (the alternative, to be applied when agents lack reasoning capabilities, is to compile a protocol specification to a Finite State Machine). Our approach ensures protocol compliance regardless of agents' reasoning capabilities, since it lets us explicitly express constraints between communicative acts, if so desired; however, equipping the communication model of single agents with sufficient knowledge to reason about expectations is certainly an interesting option.

Finally, our concept of expectations is indeed related to that of obligations and permissions. To this regard we cite [2], where Artikis et al. propose a framework for the specification of open societies of agents grounded on event calculus. Such a framework allows to reason upon deontic categories such

as permissions, obligations, fulfillment, and violations, and comes together with a graphic tool to visualize the state of a society of agents with respect to such categories. Despite the very related context and domain, our work differentiates itself from [2], as well as from most work done about deontic formalisms for social interactions, in both aims and method.

We are currently extending the proof, allowing for a more comprehensive syntax of SIC and we are developing a prototype of the social infrastructure. In the future, we would like to investigate the issue of sanctions and recovery from violation states. In addition, we intend to investigate more deeply the formal properties of the proof procedures involved in the framework, such as soundness and completeness w.r.t. the declarative specification.

Acknowledgments

This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 project.

We thank the anonymous reviewers for their useful comments.

References

[1] Alberti, M., M. Gavanelli, E. Lamma, P. Mello and P. Torroni, *An abductive interpretation for open societies*, in: A. Cappelli and F. Turini, editors, *AI*IA 2003: Advances in Artificial Intelligence*, number 2829 in Lecture Notes in Artificial Intelligence (2003), pp. 287–299.

[2] Artikis, A., J. Pitt and M. Sergot, *Animated specifications of computational societies*, in: C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III* (2002), pp. 1053–1061.

[3] Colombetti, M., N. Fornara and M. Verdicchio, *The role of institutions in multiagent systems*, in: *Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002, Siena, Italy*, 2002.

[4] Cox, B., J. Tygar and M. Sirbu, *Netbill security and transaction protocol*, in: *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, 1995.

[5] Denecker, M. and D. D. Schreye, *SLDNFA: an abductive procedure for abductive logic programs*, Journal of Logic Programming **34** (1998), pp. 111–167.

[6] Finin, T., Y. Labrou and J. Mayfield, *KQML as an agent communication language*, in: J. Bradshaw, editor, *Software Agents*, MIT Press, Cambridge, 1997 .

[7] *FIPA Communicative Act Library Specification* (2001), published on August 10th, 2001, available for download from the FIPA website.
URL `http://www.fipa.org`

[8] Fornara, N. and M. Colombetti, *Operational specification of a commitment-based agent communication language*, in: C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II* (2002), pp. 535–542.

[9] Frühwirth, T., *Theory and practice of constraint handling rules*, Journal of Logic Programming **37** (1998), pp. 95–138.

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

[11] Guerin, F. and J. Pitt, *Proving properties of open agent systems*, in: C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II* (2002), pp. 557–558.

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

[13] Mello, P., P. Torroni, M. Gavanelli, M. Alberti, A. Ciampolini, M. Milano, A. Roli, E. Lamma, F. Riguzzi and N. Maudet, *A logic-based approach to model interaction amongst computees*, Technical report, SOCS Consortium (2003), deliverable D5.

[14] Pitt, J. and F. Guerin, *Guaranteeing properties for e-commerce systems*, Technical Report TRS020015, Department of Electrical and Electronic Engineering, Imperial College, London, UK (2002).
URL `http://www.iis.ee.ic.ac.uk/reports`

[15] Sadri, F., F. Toni and P. Torroni, *An abductive logic programming architecture for negotiating agents*, in: S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, Lecture Notes in Computer Science **2424** (2002), pp. 419–431.

[16] Singh, M., *Agent communication language: rethinking the principles*, IEEE Computer (1998), pp. 40–47.

[17] Singh, M. P., *A social semantics for agent communication languages*, in: F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, Springer-Verlag, 2000 pp. 31–45.

[18] Torroni, P., P. Mello, N. Maudet, M. Alberti, A. Ciampolini, E. Lamma, F. Sadri and F. Toni, *A logic-based approach to modeling interaction among computees (preliminary report)*, in: *UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK*, 2002.

[19] Yolum, P. and M. Singh, *Commitment machines*, Lecture Notes in Artificial Intelligence **2333** (2002), pp. 235–247.

[20] Yolum, P. and M. Singh, *Flexible protocol specification and execution: applying event calculus planning using commitments*, in: C. Castelfranchi and W. Lewis

Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II* (2002), pp. 527–534.