Modeling interactions using *social integrity constraints*: a resource sharing case study*

Marco Alberti[1], Marco Gavanelli[1], Evelina Lamma[1],
Paola Mello[2], and Paolo Torroni[2]

[1] Dipartimento di Ingegneria, Università degli Studi di Ferrara
Via Saragat, 1 - 44100 Ferrara (Italy)
{malberti|mgavanelli|elamma}@ing.unife.it
[2] DEIS, Università degli Studi di Bologna
Viale Risorgimento, 2 - 40136 Bologna (Italy)
{ptorroni|pmello}@deis.unibo.it

Abstract. *Computees* are abstractions of the entities that populate global and open computing environments. The *societies* that they populate give an institutional meaning to their interactions and define the allowed interaction protocols. *Social integrity constraints* represent a powerful though simple formalism to express such protocols. Using social integrity constraints, it is possible to give a formal definition of concepts such as violation, fulfillment, and social expectation. This allows for the automatic verification of the social behaviour of computees. The aim of this paper is to show by a concrete example how the theoretical framework can be used in practical situations where computees can operate. The example that we choose is a resource exchange scenario.

1 Introduction

Global Computing [12] is a European Union initiative which aims at obtaining technologies to harness the flexibility and power of rapidly evolving interacting systems composed of autonomous computational entities, where activity is not centrally controlled, where the computational entities can be mobile, the configuration may vary over time, and the systems operate with incomplete information about the environment.

The problems that such an ambitious objective poses can be tackled from different perspectives: along with aspects such as efficiency and scalability, there are also other ones like formal soundness, possibility to prove properties and to reason on the behaviour of the system, predictability, verifiability, semantics. Declarative methods and formalisms can be used to face the problem from this second perspective. Logic programming in particular is a suitable tool to model the reasoning capabilities of autonomous entities, to give semantics to their interaction, and to throw a bridge between formal specification and operational model.

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

Computees [25] are abstractions of the entities that populate global and open computing environments. Computees have a declarative representation of knowledge, capabilities, resources, objectives and rules of behaviour. Each computee typically has only a partial, incomplete and possibly inaccurate view of the society and of the environment and of the other computees, and it might have inadequate resources or capabilities to achieve its objectives. Computees are characterized by exhibiting reasoning abilities, based on a declarative representation of knowledge and on computational logic-based functionalities (e.g. abduction, learning, planning, and so forth). These entities can form complex organizations, which we call *societies of computees*.

Using a computational logic based approach it is possible to specify the set of allowed interaction patterns among computees in the society. In general, such patterns are expressed as protocols, which we model by means of *social integrity constraints*.

Using social integrity constraints, it is possible to give a formal definition of concepts such as violation, fulfillment, and social expectation. The use of social integrity constraints is twofold. In fact, through them the society can automatically verify the compliance of its members to the protocols, and can actively suggest to its members what are possible conforming behaviours, thus guiding them in their social life.

The idea is to exploit abduction for checking the compliance of the computation at a social level. Abduction captures relevant events (or hypotheses about future events, that we call *expectations*), and a suitably extended abductive proof procedure can be used for integrity constraint checking. Expectations are therefore mapped into abducible predicates, and the social infrastructure is based on an extended abductive framework where socially relevant events are dynamically taken into consideration, as they happen.

In this work, we explain our ideas with a focus on protocols: we provide an intuitive understanding of the semantics of social integrity constraints, and we show how they can be used to verify protocol compliance. We adopt as a running example a resource sharing problem, inspired by the work done by Sadri & al. [23], where the authors propose a multi-agent solution to the problem, based on a logic programming framework. In such a framework, agents initiate negotiation dialogues to share resources along time, and can follow one among several protocols, depending on how much information they wish to disclose in order to achieve some missing resources.

Drawing inspiration from [23], in our example the society of computees defines as interaction protocols those for resource sharing. As the interactions proceed, we show the evolution of social expectations and their possible fulfillment, or the raising of a violation if some expectations are disregarded.

The paper is structured as follows. In Section 2 we introduce the idea of social integrity constraints. In Section 3 we explain the resource exchange scenario. In Section 4 we give an example of social integrity constraints and we show how social expectations are generated as the computees interact among each other. In Section 5 we show how the formalism that we propose can be used to verify

the conformance of computees to social interaction protocols. In Section 6 we relate our work with other proposals of literature and with our past and current work within the SOCS project. Conclusions follow.

2 Social integrity constraints

We envisage a model of society, tolerant to partial information, which continues to operate despite the incompleteness of the available knowledge. In such a model, the society is time by time aware of social events that dynamically happen in the social environment. Moreover, the society can reason upon the happened events and the protocols that must be followed by the computees, and therefore define what are the *expected social events*. These are events which are not yet available (to the society), but which are expected if we want computees to exhibit a *proper* behaviour, i.e., conforming to protocols.

Such expectations can be used by the society to behave pro-actively: suitable social policies could make them public, in order to try and influence the behaviour of the computees towards an ideal behaviour.

Indeed, the set of expectations of the society are adjusted when it acquires new knowledge from the environment on social events that was not available while planning such expectations. In this perspective, the society should be able to deal with unexpected social events from the environment, which violate the previous expectations.

This can be the case in an open environment where "regimentation" (see [7]) cannot be assumed. In an open society, where computees are autonomous, unexpected events can bring to a state of *violation*, from which it could be necessary to recover by taking appropriate measures (e.g., sanctions), in order to bring the society back to a consistent state.

The knowledge in a society is composed of three parts: organizational knowledge, environmental (including an events record), and Social Integrity Constraints to express the allowed interaction protocols. In this document, we will focus more on this last part of the society knowledge, which expresses what is expected to happen or not to happen, given some event record. For example, a social integrity constraint could state that the manager of a resource should give an answer to whomever has made a request for that resource.

Protocols are specified by means of Social Integrity Constraints (IC_S). IC_S relate socially significant happened events and expected events. Intuitively, IC_S are forward implications used to produce *expectations* about the behavior of computees. They are used to check if a computee inside the society behaves in a permissible way with respect to its "social" behavior.

IC_S are forward implications

$$\chi \rightarrow \phi$$

which contain in χ a conjunction of social events or expectations, and in ϕ a disjunction of conjunctions of expectations. Expectations can be of two kinds: positive (**E**) and negative (**NE**), and their variables can be constrained.

Happened events are denoted by **H**. Intuitively, an **H** atom represents a socially significant event that happened in the society, i.e., social events are mapped into **H** predicates. Events that happen, such as dialogue moves (social events), are part of the environmental knowledge of the society.

Being the focus of this paper on the motivation of the use of social integrity constraints in a declarative agent programming setting, we will not give here more detail about IC_S. In a companion paper [3] we define the full syntax of social integrity constraints, the scope of variables, quantification, and we give some results about the conditions for a proper behaviour of the framework. Also, we give a formal semantic characterization of concepts such as coherence and consistency of sets of expectations and their fulfillment. Instead, in [3] we do not discuss how the framework can be used to prove properties of interactions. The aim of this paper is to show by a concrete example how the theoretical framework can be used in practical situations where computees can operate. We will therefore give below a flavour of the operational behaviour of the framework.

In our approach, computees autonomously perform some form of reasoning, and the society infrastructure is devoted to ensure that in performing their tasks they do not violate the established rules and protocols.

H events, **E**/**NE** expectations, and society knowledge and protocols can be smoothly recovered into an abductive framework, so to exploit well-assessed proof-theoretic techniques in order to check the compliance of the overall computation with respect to the expected social behavior.

Here, we adopt an approach where abduction is used to record expectations. The dynamic knowledge available at social level grows up during the computees' own reasoning, through knowledge acquisition.

We represent the knowledge available at the social level as an Abductive Logic Program (ALP) [8], since we want to deal with incomplete knowledge. In particular, in order to model interactions, the incompleteness of their knowledge includes ignorance about communicative acts that still have to be made. The idea of modelling communicative acts by abduction is derived from [16], where the abducibles are produced within an agent cycle, and represent actions in the external world.

In our proposal, social events are recorded as **H** events. A second class of events is represented by raised expectations (about the happening, **E**, and not happening, **NE**, of events: we will call **E** and **NE** respectively positive and negative expectations), represented as abducible atoms in our case. Finally, we have negated expectations ($\neg$**E** and $\neg$**NE**), also represented as abducible atoms, in accordance with the usual way abduction can be used to deal with negation [8]. The set **EXP** can be seen as a set of hypotheses (possibly, as it will be exemplified in Section 4, a set of disjunctions of atomic hypotheses [21,11]).

At the society level, knowledge can be represented as an abductive logic program, i.e., the triple: $\langle KB, \mathcal{E}, IC \rangle$ where:

- KB is the knowledge base of the society. It contains the organizational and environmental knowledge, including happened events (we denote by **HAP** the event record: $KB \supseteq$ **HAP**);

– $\mathcal{E}$ is a set of *abducible predicates*, standing for positive and negative expectations, and their negation;
– IC is the set of social integrity constraints, IC_S.

The idea is to exploit abduction for checking the compliance of the computation at a social level. Abduction captures relevant events (or hypotheses about future events), and a suitably extended abductive proof procedure can be used for integrity constraint checking. **EXP** is a coherent and consistent set of abducibles if and only if

$$KB \cup \mathbf{EXP} \cup IC_S \nvdash \bot \tag{1}$$

and conformance to rules and protocols is guaranteed by:

$$\mathbf{HAP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{NE}(p) \rightarrow \neg\mathbf{H}(p)\} \cup \mathbf{EXP} \nvdash \bot \tag{2}$$

In this last condition, expectations are put into relationship with the events, which gives a notion of fulfillments. If (2) is not verified, then a *violation* occurs.

In [3] we provide a declarative semantics. A suitable proof procedure still has to be defined in order to efficiently deal with such a semantics for this framework. In particular, we envisage an incremental fulfillment check, in order to detect violations as soon as possible.

3 Negotiation for resource sharing

In this section we briefly recall the resource exchange scenario defined in [23]. Let us consider a system where computees have *goals* to achieve, and in order to achieve them they use plans. Plans are partially ordered sequences of activities. Activities have a duration and a time window in which they have been scheduled by the computee. In order to execute the activities, computees may need some resources during the scheduled time window.

An activity that requires a resource r is said to be *infeasible* if r is not available to the computee that intends to execute it. Similarly, infeasible is also a *plan* that contains an activity which is infeasible, and so is the *intention* of a computee, containing such plan.[1] The resources that computees *need* in order to perform an action in a plan but that they do not possess are called *missing* resources.

In fact, what we mean when we say *resource* is only an abstract entity, identified by its *name*, which possibly symbolizes a physical resource, such as a bike or a scooter. We do not explicitly model the actual delivery of physical resources either.

The *resource exchange problem* is the problem of answering to the following *question*: Does there exist a time τ during the negotiation process when the resource distribution is such that each computee has the resources it requires for

[1] In [23], *plans* are modeled as part of the computee *intentions*.

time periods that would allow it to perform the activities in its intention, within their specified time windows?

In [23], the authors propose a framework for resource exchange, where the computees can interact following different protocols. Without loss of generality, resources are assumed to be non consumable. The protocols are ordered into stages, each characterised by an increased chance of a mutually agreeable deal but at the price of disclosing more and more information. In the sequence of stages, the computees may agree to move on to the next stage if the previous stage fails to produce a deal amongst them.

In particular, the authors define two different stages of negotiation, each characterized by the degree of flexibility of the computees and the amount of information disclosed and used by them:

Stage 1: Request/flexible schedule
Stage 2: Blind deal

The first stage implements a two-step request/accept/refuse protocol. The policy that a computee may adopt upon receival of a request is to try and give the resource in question. This may require a change in its own activity schedule. The second stage implements a more elaborate protocol, where the computee who answers to a request can either accept or refuse the request, as in Stage 1, but can also propose a series of deals.

In this paper, we choose the multi-stage negotiation architecture not only because it is based on abductive logic programming, which makes it easy to define the negotiation protocols into social integrity constraints, but also because it presents different protocols, which suggests interesting observations about society engineering and the need to have a formal definition of interaction protocols, and an operational framework to reason upon.

The policies adopted by the negotiating peers at Stage 1 and Stage 2 implement a very collaborative behaviour. The set of problems that can be solved at the various stages can be formally defined (see [23] for details). But in an open society, where computees are free to choose their own policy, the only thing that we can look at, from a social verification perspective, are the exchanged messages, and the interaction protocols that they follow.

In the next section, we will define the protocols that are followed when negotiating at Stage 1 and at Stage 2, and show how expectations evolve as the negotiation proceeds.

4 Social expectations for a resource sharing scenario

The protocol followed at Stage 1 is shown in Figure 1 for two generic computees X and Y: once X makes a *request* to Y for R, Y can either *accept* or *refuse* the request. Note that there is no mention of the policies adopted by the computees (for instance, a computee could well refuse all requests and not be collaborative at all, whatever the request and its actual resource allocation: it would still be compiant to the protocol).

$$(IC_{S1})\ \ \mathbf{H}(tell(X, Y, \mathbf{request}(\mathbf{give}(R, (Ts, Te))), D, T)) \\ \rightarrow \mathbf{E}(tell(Y, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R, (Ts, Te)))), D, T')) : T < T' \\ \vee \mathbf{E}(tell(Y, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R, (Ts, Te)))), D, T')) : T < T'$$

Fig. 1. *Stage 1* protocol

H and **E** enclose *tell* atoms, which represent communicative acts, and whose parameters are: sender, receiver, subject, dialogue identifier and time of the communicative act.

Social integrity constraints can encode extra knowledge about this protocol: in particular, in Figure 1 there is nothing saying how a protocol starts/ends. In Figure 2, we see how by IC_S we can express the fact that some moves are "final", in the sense that they mean to put an end to a conversation, while some others are "initial", i.e., they are never preceded by other moves in the same conversation. We refer to the language $\mathcal{L}_{1-TW}$ defined in [23], where *request*(...) is the only allowed initial move, and *accept*(*request*(...)), *refuse*(*request*(...)) and *refuse*(*promise*(...)) are final moves. We see here another use of integrity constraints, different from that in Figure 1: IC_S are used in fact to generate *negative* expectations, which tell what should not happen, given a certain course of events.

$$(IC_{S2})\ \ \mathbf{H}(tell(\_, \_, \mathbf{request}(Req), D, T)) \\ \rightarrow \mathbf{NE}(tell(\_, \_, \_, D, T')) : T' < T$$

$$(IC_{S3})\ \ \mathbf{H}(tell(\_, \_, \mathbf{accept}(Req), D, T)) \\ \rightarrow \mathbf{NE}(tell(\_, \_, \_, D, T')) : T' > T$$

$$(IC_{S4})\ \ \mathbf{H}(tell(\_, \_, \mathbf{refuse}(Req), D, T)) \\ \rightarrow \mathbf{NE}(tell(\_, \_, \_, D, T')) : T' > T$$

Fig. 2. Negative expectations following from the semantics of the negotiation language

In Figure 2, *Req* can be any kind of request or proposed deal (*promise*).

In Figure 3 we define the protocol for Stage 2 negotiation. By IC_{S5}, after a *request* the possible moves are those of Stage 1, plus *promise*. By IC_{S6}, after *promise* one expects either an *accept* or a *change*(*promise*(...)). By IC_{S7}, after a *change*(*promise*(...)) one expects either another *promise* or a *refuse* of the original request, which terminates the dialogue. Finally, by IC_{S8}, one does not expect the same *promise* to be made twice.

$$
\begin{aligned}
(IC_{S5})\ \ & \mathbf{H}(\mathit{tell}(X, Y, \mathbf{request}(\mathbf{give}(R, (Ts, Te))), D, T)) \\
& \quad \rightarrow \mathbf{E}(\mathit{tell}(Y, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R, (Ts, Te)))), D, T')) : T < T' \\
& \quad \vee \mathbf{E}(\mathit{tell}(Y, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R, (Ts, Te)))), D, T')) : T < T' \\
& \quad \vee \mathbf{E}(\mathit{tell}(Y, X, \mathbf{promise}(R, (Ts', Te'), (Ts, Te)), D, T')) : T < T' \\
\\
(IC_{S6})\ \ & \mathbf{H}(\mathit{tell}(X, Y, \mathbf{promise}(R, (Ts, Te)), D, T)) \\
& \quad \rightarrow \ \mathbf{E}(\mathit{tell}(Y, X, \mathbf{change}(\mathbf{promise}(R, (Ts', Ts'), (Ts, Te)), D, T'))) \ : \\
& T < T' \\
& \quad \vee \mathbf{E}(\mathit{tell}(Y, X, \mathbf{accept}(\mathbf{promise}(R, (Ts, Te))), D, T')) : T < T' \\
\\
(IC_{S7})\ \ & \mathbf{H}(\mathit{tell}(X, Y, \mathbf{change}(\mathbf{promise}(R, (Ts', Te'), (Ts, Te))), D, T)) \\
& \quad \rightarrow \mathbf{E}(\mathit{tell}(Y, X, \mathbf{promise}(R, (Ts", Te"), (Ts, Te)), D, T')) : T < T' \\
& \quad \vee \mathbf{E}(\mathit{tell}(Y, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R, (Ts, Te)))), D, T')) : T < T' \\
\\
(IC_{S8})\ \ & \mathbf{H}(\mathit{tell}(X, Y, \mathbf{promise}(R, (Ts', Te'), (Ts, Te)), D, T)) \\
& \quad \rightarrow \mathbf{NE}(\mathit{tell}(X, Y, \mathbf{promise}(R, (Ts', Te'), (Ts, Te)), D, T')) : T < T'
\end{aligned}
$$

Fig. 3. *Stage 2* protocol

If we look at both protocols, for Stage 1 and Stage 2, we understand that $IC_{S1} \Rightarrow IC_{S5}$, in the sense that IC_{S5} is more general than IC_{S1}. In a more elaborate solution, we could explicitly add a "stage identifier" in the communication acts, and keep a *request* made at Stage 1 different from a *request* made at Stage 2. Here, for the sake of brevity, we will simply override Stage 2 with Stage 1. The social integrity constraints that we consider, in particular, are only those of Figure 2 and 3.

From this example it is possible to see a first achievement of the use of social integrity constraints to formally define protocols: we are able to formally reason on the protocols and adopt a modular approach to society engineering. For instance, it would be interesting to know that if we put Stage 1 constraints and Stage 2 constraints together as they are (without the stage identifier), then the resulting protocol is not the union of the two stages, but a more restrictive protocol than that. The study of tools to automatically prove some relationships among integrity constraints or protocols is subject for current investigation.

We would now like to show how social expectations are created and evolve as computees exchange requests and reply to each other. For the sake of the example, we consider a system composed of three computees: *david*, *yves*, and *thomas*. The resource that they share is a *scooter*. *david* is normally entitled to have the scooter in the afternoon, while *yves* in the morning.

Let us assume that at time 1 *david* makes a request to *yves*, because he needs the scooter from 10 to 11 in the morning.[2] This request is recorded by the society and put into **HAP**:

$$\mathbf{H}(tell(david, yves, request(give(scooter(10, 11))), d, 1))$$

By looking at the history, the society modifies its set of expectations. Such a set could be initially empty. After the event at time 1, due to IC_{S5}, the society grows with expectations. Since IC_{S5} has a disjunction as a consequence of an **H** atom, the expectations will also be disjunctions of atoms. In general, we will have an expression **EXP** which could be put in the form of a disjunction of conjunction of expectations.

At time 2, we have

$$\begin{gathered}\mathbf{EXP}_1 = \{ \\ (\ (\ \mathbf{E}(tell(yves, david, accept(request(give(scooter, (10, 11)))), d, T) : T > 1)\ ) \\ \vee(\ \mathbf{E}(tell(yves, david, refuse(request(give(scooter, (10, 11)))), d, T) : T > 1)\ ) \\ \vee(\ \mathbf{E}(tell(yves, david, promise(scooter, (10, 11), (Ts, Te)), d, T) : T > 1)\ )\ ) \\ \wedge(\ \mathbf{NE}(tell(\_, \_, \_, d, T') : T' < 1)\ ) \\ \}\end{gathered}$$

Let us assume that at time 3 *yves* proposes a deal to *david*:

$$\mathbf{H}(tell(yves, david, promise(scooter, (10, 11), (20, 23)), d, 3))$$

EXP changes. It becomes:

$$\begin{gathered}\mathbf{EXP}_2 = \{ \\ (\ (\mathbf{E}(tell(yves, david, accept(request(give(scooter, (10, 11)))), d, T) : T > 1)) \\ \vee(\mathbf{E}(tell(yves, david, refuse(request(give(scooter, (10, 11)))), d, T) : T > 1)) \\ \vee(\mathbf{E}(tell(yves, david, promise(scooter, (10, 11), (Ts, Te)), d, T) : T > 1))\ ) \\ \wedge\ (\ (\mathbf{E}(tell(david, yves, accept(promise(scooter, (10, 11), (20, 23))), d, T) : T > 3)) \\ \vee(\mathbf{E}(tell(david, yves, change(promise(scooter, (10, 11), (20, 23))), d, T) : T > 3))\ ) \\ \wedge\ \mathbf{NE}(tell(yves, david, promise(scooter, (10, 11), (20, 23)), d, T') : T' > 3) \\ \}\end{gathered}$$

The disjunction **EXP** will keep evolving along with the social events. In [3] the authors give a declarative semantics to expectations and social integrity constraints. They define a notion of fulfillment of expectations. For instance, in our example, we can see that an expectation which is present in $\mathbf{EXP}_1$ at time 2 is then fulfilled by *yves*' message to *david* at time 3. Similarly, we have a violation if at a later time something happens which is expected not to happen, e.g., *yves* repeats the same promise for a second time.

The resource exchange scenario allows us to exemplify some advantages of our formalism. Social integrity constraints gave use the ability to:

[2] We make the simplifying assumption that the time of communication acts is centrally assigned, e.g. by the "social infrastructure", and that all computees are able to cope with this. We can then consider the time of the society as a transaction time.

– express allowed communication patterns inside a society (e.g., the protocols for Stage 1 and Stage 2). This is done independently of the computees' internal policies and implementation;
– use the protocol definitions and the history of socially relevant events to generate at run-time the possible combinations of future events that represent a "proper" behaviour of the computees (e.g., $\mathbf{EXP}_1$ and $\mathbf{EXP}_2$);
– formally reason on the composition of protocols, and adopt a modular approach to society engineering (e.g. the composition of Stage 1 and Stage 2 defines a new, more restrictive protocol);
– verify at run-time the correct behaviour of some computees, with respect to the protocols, without having access to their internals (e.g., if *yves* repeats the same promise for a second time we enter a violation state).

As future extensions, we intend to investigate the issue of protocol composition, and the notion of violation, particularly about how to recover from a state of violation, and possibly — given such a state — how to identify one or more "culprits" and generate appropriate sanctions. Having all these elements in a unified declarative framework would be an important achievement.

5 Social integrity constraints for verification

In [13,20], 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 interactions. In this setting, 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 author mean that the internals of agents are not public.

Verification 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 [24], 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 propose for modelling interactions in an open society of computees lends itself very well to all these kinds of verification. In

fact, both (public) protocols and (internal) policies are expressed in the same formalism, which makes it possible to relate social aspects with individual aspects in a static verification. We believe that all the above three types of verification can be *automatically* done in a computational logic setting.

Verification that a computee will always comply cannot be done by externally monitoring its behaviour. As Hume says [15], we are in a natural state of ignorance with regard to the powers and influence of all objects when we consider them a priori (IV.ii.32). For this kind of verification, we need to have access to the computees' internals, or to its specifications. We would be advantaged in this task if we could express the computee program and policies by means of an abductive logic programming based formalism, as it is shown in [22]: in that case, specification and implementation coincide.

The proof that a given computee c will always comply with a set $\mathcal{IC}_S$ of constraints representing a protocol, based on the agents' specifications, could be the following. We show that for all the constraints in $\mathcal{IC}_S$, if in the head of a constraint there is a social event which is expected from c, then, the history of events that from a social viewpoint leads to such expectation, leads by that computee's viewpoint to producing an event that fulfills it (or prevents the computee from producing any event that violates it, in the case of negative expectations).

In the scooter example, if *yves*'s specifications are such that the constraint:

$$\mathbf{H}(tell(C, yves, request(give(scooter, (10, 11))), d, T))$$
$$\rightarrow \mathbf{H}(tell(yves, C, accept(request(give(scooter, (10, 11)))), d, T') : T' > T)$$

is never violated, then *yves* is compliant with the protocol, because the event that raises the expectation

$$\mathbf{E}(tell(yves, C, accept(request(give(scooter, (10, 11)))), d, T))$$

in the society also makes *yves* generate an event which fulfills that expectation.

The study of a mechanism to automatically obtain such a proof (or its failure) is subject for current investigation.

For verification of Type 2 we need to be able to observe the computees' social actions, i.e., the communicative acts that they exchange. As in [20], we can assume that this can be achieved by policing the society. In particular, "police" computees will "snoop" the communicative acts exchanged by the society members and check at run time if they comply with the protocols specifications (social integrity constraints).

This kind of run-time verification is theoretically already built in our framework and we do not need to provide additional formal tools to achieve it. In [1] Alberti & al. show an implementation of social integrity constraints based on the CHR language [10].

Verification of Type 3 is about protocol properties. In order to prove them we do not need to access the computees' internals, nor to know anything about the communication acts of the system, because it is a verification which is statically done at design time. As we already mentioned in Section 4, we are working on

the design of a logic-based formalism to automatically detect inconsistencies of combinations of protocols. In general, one of the main motivations behind our formal approach is to be able to prove all properties that can be formally defined, e.g. by means of an invariant or an implication, as they are defined in [20].

6 Discussion

Computees are abstractions of the entities that populate global and open computing environments. The intuition behind a computee is very similar to that behind an agent. The reason why we adopt a different name is because we want to refer to a particular class of agents, which can rely on a declarative representation of knowledge, and on reasoning methods grounded on computational logic. In this way, a declarative representation of the society knowledge and of social categories such as expectations, as we defined them in this work, can be smoothly integrated with the knowledge of its inhabitants, and similar techniques can be used by them to reason upon either knowledge base or upon a combination of them. The main motivation of our approach is in its declarative nature, which has the potential to aiding a user's understanding of a system's specification, and in its solid formal basis, which puts together, under the same formalism, specification, implementation, and verification for multi-agent systems.

In [9], Esteva & al. give a formal specification of agents societies, focussing on the social level of electronic institutions. In that model, each agents in a society plays one or more *roles*, and must conform to the pattern of behavior attached to its roles. The *dialogic framework* of a society defines the common ground (ontology, communication language, knowledge representation) that allows heterogeneous agents to communicate. Interactions between agents take place in group meetings called *scenes*, each regulated by a communication protocol; connections between scenes (for instance, an agent may have to choose between different scenes, or its participation in a scene may be causally dependent on its participation in another) are captured by the *performative structure*. *Normative rules* specify how agents' actions affect their subsequent possible behavior, by raising obligations on them.

In our framework, agents are only required to perform their communicative acts by a given computee communication language; they are not required to share an ontology (which, strictly speaking, need not even be specified). Possible interactions are not organized in separate (although inter-connected) scenes; agents' interactions are supposed to take place in one shared interaction space. For this reason, we do not distinguish, as in [9], between intra-scene (possible interaction paths *inside* inside a scene) and inter-scene (possible paths of an agent *through* scenes) normative rules. Without this distinction, normative rules are strictly related to our social integrity constraints, in that both constrain agents' future behavior as a consequence of their past actions. We would like to stress that Esteva & al., based on the analysis presented in [6] by Dellarocas and Klein, start from the same requisites as we, considering as major issues heterogeneity, trust and accountability, exception handling and societal changes,

and provide in their work a basis for a verified design of electronic marketplaces, by giving a very detailed formal specification of all the aspects of their electronic institutions, and graphical tools to make such a specification easy to understand. However, the focus of their work does not seem to be on a direct and automatic specification-verified implementation relationship, as in ours.

Our framework does not specifically cater for roles (very little we said about he structure of the knowledge related to the society). However, our aim is to propose a declarative framework where roles can indeed be expressed, but do not represent a first class entity in the definition of agent interactions in general. For instance, in a semi-open society [5] roles could be assigned to incoming computees by "custom officer" computees, or they could be statically defined in the society knowledge base, or else dynamically acquired along the time. It is not difficult to imagine social integrity constraints that define the protocols for role management.

In [27], Vasconcelos presents a very neat formalization based on first order logics and set theory to represent an expressive class of electronic institutions. As in our work, the use of variables and quantification over finite sets allow to express significant protocols patterns. The definition of allowed interaction patterns is based on the concept of scene, similarly to what is done in [9]. How expressive is the formalism proposed in [27] with respect to ours, and what are the differences in terms of verification, are issues that we would like to address in the future.

In [19] Moses and Tennenholtz focus on the problem of helping interacting agents achieve their individual goals, by guaranteeing not only mere achievability of the goals, but also computational feasibility of planning courses of actions to realize them. Since an agent's behavior can affect the achievability of other agents' goals, the agents' possible actions are restricted by means of *social laws*, in order to prevent agents from performing actions that would be detrimental to other agents.

In our work, we aim more at verifying sound social interaction, rather than at guaranteeing properties of individual computees. Moreover, computing appropriate social laws requires a (correct) modelling of individual agents in terms of states, state-actions relations and plans, thus assuming reliable knowledge about agents' internals, which we do not require.

Gaia is a methodology in software engineering developed by Wooldridge & al. for agent-oriented analysis and design [28]. The developer of a MAS has to define *roles* that agents will then embody. Each role is defined by its *responsibilities*, *permissions*, *activities* and *protocols*. In particular, responsibilities explain how an agent embodying the corresponding role should behave in terms of *liveness* (good things that should happen) and *safety* (bad things that should not happen) expressions.[3] These expressions can be seen as abstractions of our expectations, or, in a sense, our Social Integrity Constraints could be seen as a further refinement and a formalization of the concepts of responsibilities. More-

[3] This distinction is due to Lamport [17].

over, we use Social Integrity Constraints also to model *protocols*, i.e., interaction with other agents/roles.

Considerable work has been done about the formal definition and verification of properties of multi-agent systems, but to the best of our knowledge there is not yet a reference paper with a formal classification and definition of properties that are considered interesting in a general setting. In Section 5, we referred to the work by Guerin and Pitt because it neatly pins down the main requisites and characteristics of a multi-agent system verification process (run-time verification vs. static-verification, and visibility of agents's internals). Although it is specially oriented to electronic commerce applications, we believe that its main ideas can be extended to a more general case of agent interactions.

Among other work that proposes list of properties of agent systems, more or less formally defined and related to particular domains, we cite work done by Mazouzi & al. [18], Yolum & Singh [29], Hewitt [14], Artikis & al. [4], and Davidsson [5].

We conclude this section by putting this work in relationship with our past and current activity within the SOCS project. In [26] a layered architecture for societies of computees has been proposed, where at the bottom level a platform is used to implement the system and give support to computees' communication, and a communication language layer defines syntax and semantics of communicative acts, while society and protocols are in a higher layer. The purpose the higher layers is to determine the set of allowed interaction patterns among computees in the society. A social semantics for communicative acts has been presented in [2], along with a discussion about the advantages and motivation of a social semantics of communication with respect to other approaches, and in [1] an implementation of a restricted class of Social Integrity Constraints is proposed, based on the CHR language [10]. [3] defines the full syntax of social integrity constraints, the scope of variables, quantification, and gives some results about the conditions for a proper behaviour of the framework, along with a formal semantic characterization of concepts such as coherence and consistency of sets of expectations and their fulfillment.

With respect to the work that we have been doing and that we have briefly reviewed above, this is the first paper which aims at showing the practical use of our theoretical framework, by means of a simple though realistic case study. The protocols that we used for our example are taken from the literature, and could be used to solve resource sharing problems among agents. We also contributed in showing how the framework can be the basis for the automatic proof of properties of interactions and protocols.

7 Conclusion

In this work, we have shown how a logic programming based framework is a suitable tool to give semantics to the interactions of autonomous entities populating a global computing environment. We illustrated the framework by means of a resource sharing example. The main idea is that of social integrity constraints,

and of a correspondence with abductive frameworks of logic programming to provide a semantic to such constraints.

This paper wants to give a motivation to a formal approach by showing a concrete example of the expected operation of the framework.

The use of social integrity constraints could be twofold. In fact, through them the society can automatically verify the compliance of its members to the protocols, and ideally it could actively suggest to its members what are possible conforming behaviours, thus guiding them in their social life. The paper does not cover the last issue, which is subject for future work. In both aspects, our work represents a novel contribution to the research in the area.

References

1. M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Logic Based Semantics for an Agent Communication Language. In *Proceedings of the International Workshop on Formal Approaches to Multi-Agent Systems (FAMAS)*, Warsaw, Poland, April 12 2003. To appear.
2. M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In V. Marik, J. Müller, and M. Pechoucek, editors, *Proceedings of the 3rd International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS)*, 2003.
3. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An Abductive Computational Model for Open Societies. 2003. Under review.
4. A. Artikis, 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, Bologna, Italy*, pages 1053–1061. ACM, 2002.
5. P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents World II*, volume 2203 of *LNAI*, pages 1–9. Springer-Verlag, December 2001. 2nd International Workshop (ESAW'01), Prague, Czech Republic, 7 July 2001, Revised Papers.
6. C. Dellarocas and M. Klein. Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In *Agent Mediated Electronic Commerce (IJCAI Workshop)*, pages 24–39, 1999.
7. T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, March 1999.
8. K. Eshgi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255. MIT Press, 1989.
9. M. Esteva, J. A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent-mediated Electronic Commerce (The European AgentLink Perspective)*, number 1991 in LNAI, pages 126–147. Springer Verlag, 2001.
10. T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, October 1998.
11. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.
12. Global Computing: Co-operation of Autonomous and Mobile Entities in Dynamic Environments. `http://www.cordis.lu/ist/fetgc.htm`.

13. F. Guerin 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, Bologna, Italy*, pages 557–558. ACM, 2002.
14. C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47(1-3):79–106, 1991.
15. D. Hume. *An Enquiry Concerning Human Understanding*. 1748.
16. R. A. Kowalski and F. Sadri. From logic programming to multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 1999.
17. L. Lamport. What Good Is Temporal Logic? In R. E. A. Mason, editor, *Information Processing*, volume 83, pages 657–668. Elsevier Sciene Publishers, 1983.
18. H. Mazouzi, A. El Fallah Seghrouchni, and S. Haddad. Open protocol design fo complex interactions in multi-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 I, Bologna, Italy*, pages 402–409. ACM, 2002.
19. Y. Moses and M. Tennenholtz. Artificial social systems. *Computers and AI*, 14(6):533–562, 1995.
20. J. Pitt and F. Guerin. Guaranteeing properties for e-commerce systems. Technical Report TRS020015, Department of Electrical and Electronic Engineering, Imperial College, London, UK, 2002.
21. D. L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.
22. F. Sadri, 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)*, volume 2424 of *LNCS*, pages 419–431. Springer Verlag, September 2002.
23. F. Sadri, F. Toni, and P. Torroni. Minimally intrusive negotiating agents for resource sharing. In G. Gottlob, editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. AAAI Press, August 2003. To appear.
24. M. P. Singh. A social semantics for agent communication languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag, Heidelberg, Germany, 2000.
25. SOCS: Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. `http://lia.deis.unibo.it/Research/SOCS/`.
26. P. Torroni, 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*, December 2002.
27. W. W. Vasconcelos. Logic-based electronic institutions. In this volume, pages 65–80, 2003.
28. 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, September 2000.
29. P. Yolum and M.P. 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, Bologna, Italy*, pages 527–534. ACM, 2002.