Logic Based Semantics for Agent Communication Languages

Marco Alberti[1], Anna Ciampolini[2], 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] Dipartimento di Elettronica, Informatica e Sistemistica,
Università degli Studi di Bologna
Viale Risorgimento, 2 - 40136 Bologna (Italy)
{aciampolini|pmello|ptorroni}@deis.unibo.it

Abstract. Agent communication is one of the key issues in multi-agent systems. Traditional interprocess communication formalisms are usually considered insufficient for this purpose because of their lack of expressiveness; thus, in most proposals for multi-agent architectures, an Agent Communication Language (ACL) is designed to provide for agent communication. However, a universally accepted standard for ACLs is still missing. Agent communication in open societies of heterogeneous agents poses requirements on ACLs semantics (formal syntax and semantics, declarativeness, verifiability, meaningfulness) which a *social* approach seems to meet best.
In this paper we propose a logic-based social approach for ACL semantics. We give a functional abstract model of societies and agents. Then we propose a formalism (deontic constraints, grounded on a computational logic framework) to express interaction protocols and give a social semantics to the behavior of agents, focusing on communicative acts. Finally, we sketch a prototypical implementation of deontic constraints exploiting the Constraint Handling Rules language.

1 Introduction

In multi-agent systems, agents interact and exchange pieces of knowledge. If agents are viewed as rational entities, traditional low-level interprocess communication mechanisms are not sufficient for the purpose of knowledge sharing, and agents should be provided with a formalism that abstracts away from implementation issues and is expressive enough for exchanging knowledge at a high level. This is why most proposals for multi-agent systems are equipped with an Agent Communication Language (ACL hereafter). However, there is still no agreed upon standard for ACLs.

Mentalistic approaches [1, 2] define ACL semantics from an internal perspective, i.e., in terms of agents mental states. For instance, FIPA ACL [2] 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 [3]. In particular, whenever agents' mental states are not accessible, as in the case of open societies of heterogeneous agents, it is impossible to verify whether agents are acting according to the given semantics [4].

The social approach [4] defines semantics of communicative acts in terms of their public (social) effects, rather than of their internal (mental) motivations. Following this approach, compliance to a given semantics can be verified of agents' *observable* acts, even in cases where the internal state of agents is not accessible, or does not have a proper mental structure. Significant social proposals are based on the concept of *commitments* [5, 6], 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.

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

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 communication. Our semantics is based on the concept of *expectation* (which, as explained in Sect. 6, is broader than that of commitment), linked to events by *deontic integrity constraints*.

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, that corresponds to the verification of compliance by observation proposed by Guerin and Pitt [7].

The paper is structured as follows. In Sect. 2 we describe the model of agent society in which we set our work; in Sect. 3 we introduce the concept of deontic constraint, and in Sect. 4 we show how it can be exploited to specify the social semantics of communicative acts. A sketch of a prototypical implementation of the framework by means of the *Constraint Handling Rules* language is presented in Sect. 5; discussion and directions for future work follow.

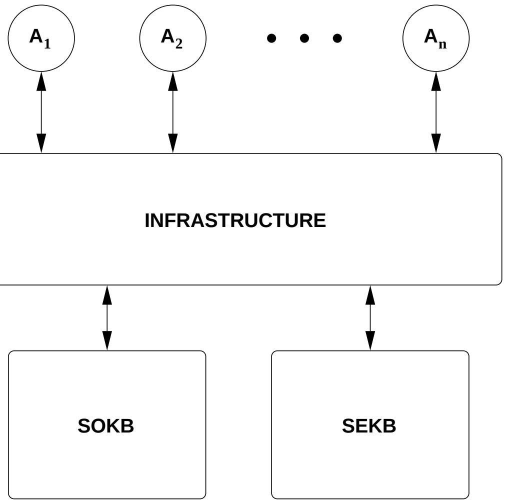

Fig. 1. Functional model of agent society

2 Model of agent society

Agent society modeling is a major issue in the field of multi-agent systems. In this section, we briefly describe our model of agent society; a more detailed discussion is presented in [8].

Since we are interested in modeling an open society of heterogeneous agents, we will rely on a social perspective: the model does not constrain the internal structure of single agents. Rather, it provides formalisms for constraining their social behavior, i.e., their observable actions which have an effect upon other agents or the environment. Our model is meant to be a functional abstraction: a society is not necessarily a computational entity, but a set of features that some computational entities are supposed to provide. We describe a prototypical implementation of some of these features in Sect. 5.

Functionally, a society can be viewed as composed of a *social infrastructure* (its operational part) and two knowledge bases: *Social Organization Knowledge Base* (**SOKB**) and *Social Environment Knowledge Base* (**SEKB**). The infrastructure interacts with individual agents, while querying and updating **SOKB** and **SEKB** as needed (see Fig. 1).

SOKB defines structure and properties of the society, namely: roles, norms, topology (communication channels, groups, etc.), protocols (including deontic constraints, explained in Sect. 3), common knowledge and capabilities.

SEKB defines:

- Members of the society (agents);
- The history $\mathcal{H}$, composed of *observable* and *relevant* events for the society (*happened* events, indicated by the functor **H**), including performed communicative and non-communicative acts.

– *Expectations*: events that should (but might not) happen (*obligations*, indicated by the functor **O**), and events that should not (but might indeed) happen (*prohibitions*, indicated by the functor **ON**).

The social infrastructure

– records events (both agents' socially significant acts and changes in the environment) in **SEKB**;
– depending on events, updates the expectations in **SEKB**;
– monitors **SEKB**, especially in order to start appropriate recovery procedures in case of violations of expectations.

3 Deontic constraints

Integrity deontic constraints (also *deontic constraints* or ic_Ds for brevity hereafter) link events and expectations.

We give here the syntax of deontic constraints, in BNF.

$$
\begin{aligned}
ic_D &::= \chi \rightarrow \phi \\
\chi &::= \mathit{ExtendedLiteral}\ [\ \wedge \mathit{ExtendedLiteral}\ ]^\star \\
\phi &::= \mathit{OughtList}[: \mathit{ConstraintList}] \\
\mathit{ExtendedLiteral} &::= \mathit{Expectation} \mid \mathit{Event} \mid \mathit{Literal} \\
\mathit{Expectation} &::= \mathbf{O}(\mathit{Atom}) \mid \mathbf{ON}(\mathit{Atom}) \\
\mathit{Event} &::= \mathbf{H}(\mathit{Atom}) \\
\mathit{Literal} &::= \mathit{Atom} \mid \neg \mathit{Atom} \\
\mathit{OughtList} &::= \mathit{Expectation}\ [\ \wedge \mathit{Expectation}\ ]^\star
\end{aligned}
\tag{1}
$$

ConstraintList is a conjunction of constraints.

In order to give a declarative semantics, we first define the concept of entailment in presence of deontic constraints. In the following, we comply to the conventions in [9]; in particular, the $\models$ symbol represents the usual relationship of logical consequence as intended in logic programming.

Definition 1. Deontic entailment *is indicated with the symbol* $\models_\mathcal{D}$ *and is recursively defined as follows:*

– *For every atom* e, $\mathcal{H} \models_\mathcal{D} \mathbf{O}(e) \leftrightarrow \mathcal{H} \models \mathbf{H}(e)$.
– *For every atom* e, $\mathcal{H} \models_\mathcal{D} \mathbf{ON}(e) \leftrightarrow \mathcal{H} \not\models \mathbf{H}(e)$.
– *For every event atom* α , $\mathcal{H} \models_\mathcal{D} \alpha \leftrightarrow \mathcal{H} \models \alpha$.
– $\mathcal{H} \models_\mathcal{D} (\alpha \wedge \beta) \leftrightarrow (\mathcal{H} \models_\mathcal{D} \alpha) \wedge (\mathcal{H} \models_\mathcal{D} \beta)$

This means, for example, that if some event E has happened, the obligation $\mathbf{O}(E)$ is fulfilled, whereas the prohibition $\mathbf{ON}(E)$ is violated.

Declaratively, compliance of a set of events to the specification of a society can be *successful*, *failed* or *violated*. We give the following definitions.

Definition 2. *A set of events $\mathcal{H}$ fulfills the specification of a society if there exists a conjunction $\mathcal{E}$ of expectations such that:*

$$\mathbf{SOKB} \cup \mathcal{H} \cup \mathcal{E} \not\models \perp \tag{2}$$

and

$$\mathcal{H} \models_{\mathcal{D}} \mathcal{E} \tag{3}$$

It should be noticed that, in (2), **SOKB** also contains the set $\mathcal{IC}_{\mathcal{D}}$ of deontic constraints.

Intuitively, there can be more than one $\mathcal{E}$ satisfying (2). We are interested in finding if, among these, there exists one such that the happened events cover all the events that ought to happen, and none of the events that should not happen. This is the case of successful compliance.

Such a formula might not exist because of two reasons. The first is that the knowledge base of the society cannot properly tackle the history $\mathcal{H}$. This is consequence of an inappropriate modeling of the society: we call this situation a *failure.*

Definition 3. *A society* fails *if $\nexists\mathcal{E}$ satisfying (2).*

When a failure arises, it could be necessary to modify the society's **SOKB** by means of a theory revision process. In this work, we do not deal with the evolution of the **SOKB** of societies, but only with the evolution of their **SEKB**.

The second reason is the violation of the deontic specification of the behavior of the society: some agent did not behave as expected.

Definition 4. *The deontic specification of a society is* violated *with respect to a history $\mathcal{H}$ iff there exists a formula $\mathcal{E}$ such that (2) holds, and for all $\mathcal{E}$ satisfying (2), (3) does not hold, i.e.:*

$$\mathcal{H} \not\models_{\mathcal{D}} \mathcal{E}$$

Operationally, deontic compliance of the event history to the specification of a society must be ensured by an appropriate proof procedure. Obviously, such proof procedure should operate incrementally during the evolution of the **SEKB**, so to make it possible to detect violations as soon as they occur and possibly start recovery procedures; in particular, constraint propagation techniques should be used to detect that an expectation will never be fulfilled as early as possible (see Sect. 5.2). Intuitively, expectations should be generated so to satisfy deontic constraints: when the **SEKB** contains all the events in the body χ of a deontic constraint $\chi \rightarrow \phi$, 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. We describe implementation issues for this proof procedure in greater detail in Sect. 5.

4 Constraint-based ACL semantics

By means of deontic constraints, we specify a social semantics for an ACL that is:

- *formal*: the declarative semantics of communicative acts is defined in a computational logic framework (see Sect. 3);
- *verifiable*: compliance to deontic constraint of an event history to a declarative specification can be checked by an appropriate proof procedure (we sketch a prototypical implementation of it in Sect. 5);
- *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 these features by means of a simple example where, for the sake of synthesis, we consider a set of only three communicative acts (namely, *request*, *accept* and *reject*). We also consider a physical action (*do*) to show how physical actions can interact with obligations and prohibitions generated by communicative acts (for instance, by fulfilling or violating them).

$$\begin{array}{l} \quad\mathbf{H}(\mathit{request}(B,A,P,D,T_r)) \\ \wedge\mathbf{H}(\mathit{accept}(A,B,P,D,T_a)) \\ \wedge T_r < T_a \\ \rightarrow\mathbf{O}(\mathit{do}(A,B,P,D,T_d)) : T_d \leq T_a + \tau \end{array} \tag{4}$$

Deontic constraint (4) means that if an agent *accept*s 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.

The functor *do* represents a physical action, which fulfills the obligation of deontic constraint (4) 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 the social infrastructure (as explained in Sect. 5) automatically matches an obligation with a corresponding event.

Obviously, it is possible to design agents so that they will always (try to) conform to this specification. However, relying only on agents' internal structure for compliance is not satisfactory for two reasons:

- in open societies of autonomous and heterogeneous agents, internal structure of individual agents, is, in general, not known, and agents could be malicious, or, more simply, erroneously designed;
- even when agents' internals were known to be compliant, an action might be impossible to perform.

Constraints like (4) provide a compliance check mechanism that is independent from individual agents design in that it guarantees that, if the deadline is not met, declaratively Equation (3) will not hold and operationally a violation will be detected.

Moreover, we may want to specify that an *accept* should only be issued after a corresponding *request* has been issued. Such a requirement (as all those regarding proper sequentiality of communicative acts) is usually specified at a different level than that of ACL semantics, e.g. by means of a protocol. Our framework, however, can capture this case by means of a *backward* obligation, i.e., an obligation regarding an event that *should have happened* before current time.

$$\begin{aligned} &\mathbf{H}(\mathit{accept}(A, B, P, D, T_a)), \\ \rightarrow &\mathbf{O}(\mathit{request}(B, A, P, D, T_r)) : T_r < T_a \end{aligned} \tag{5}$$

If an agent issues an *accept* and a corresponding *request* has not been issued before, constraint (5) allows to detect a violation.

An agent that *reject*s a *request* is obviously not committed to its content. Thus, the only obligation associated to a *reject* is backward, and it is needed to ensure that a *reject* is only performed in reply to a corresponding *request* (6).

$$\begin{aligned} &\mathbf{H}(\mathit{reject}(A, B, P, D, T_2)) \\ \rightarrow &\mathbf{O}(\mathit{request}(B, A, P, D, T_1)) : T_1 < T_2 \end{aligned} \tag{6}$$

For example, let us consider the following dialog (each line has the format: *Dialog*, *Time* : *Act*, and we assume $\tau = 4$):

```
a_dialog, 10: request(creditor,debtor,give(nail))
a_dialog, 12: accept(debtor,creditor,give(nail))
a_dialog, 15: do(debtor,creditor,give(nail))
```

The expected effect of events over the sets $\mathcal{H}$ of events and $\mathcal{E}$ of expectations[3] is depicted in Fig. 2.

At step 1 (before time 10), we suppose that the sets are empty. At time 10, when the *request* takes place, the event is recorded in the store as happened, but no expectation is raised. At step 3 (time 12), the *accept* event happens and is stored; now, because of deontic constraint (5), a backward obligation (which is already fulfilled[4]) is raised, and, because of deontic constraint (4), the two happened events, whose contents match and whose time condition is fulfilled, raise a forward obligation (step 4). At step 5 (time 15), the *do* action is stored as happened; it matches with the obligation and has been performed by the deadline, thus the forward obligation is fulfilled.

The above specification can be easily extended to a more comprehensive set of communicative acts. For instance, constraint (7) specifies how the sequence

[3] We assume $\mathcal{E}$ is built incrementally, as explained in Sect. 3.

[4] If the *request* act had not been issued, the backward obligation in constraint (5) would have been violated.

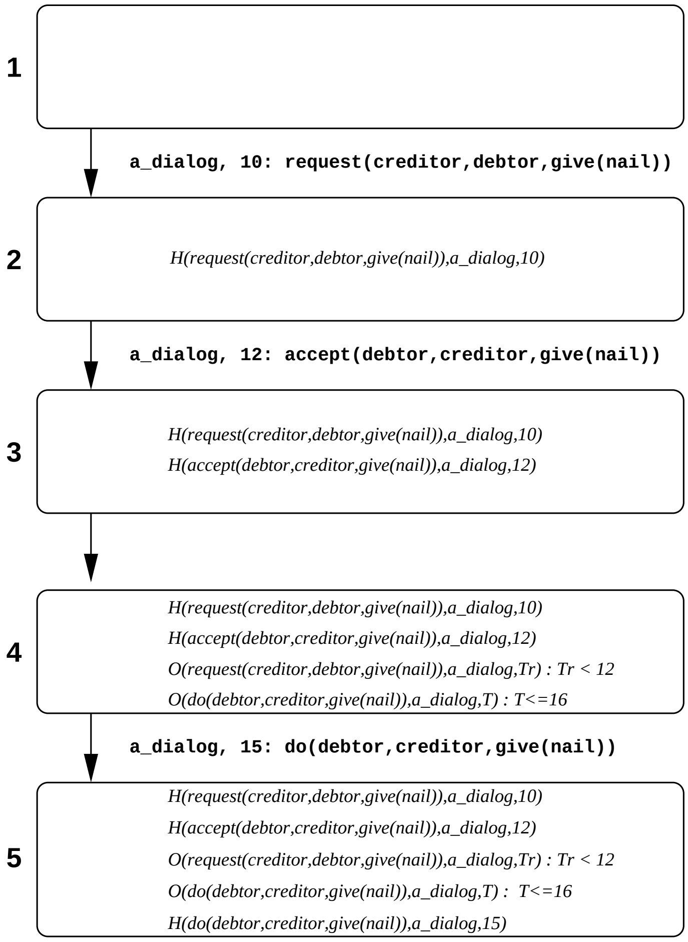

Fig. 2. Simple example of a dialog

of a *propose* and an *acceptProposal* causes, for the two agents involved in the dialog, respective obligations to accomplish the actions object of the proposal.

$$\begin{aligned}
&\mathbf{H}(propose(B, A, (P_A, P_B), D, T_p)) \\
&\wedge \mathbf{H}(acceptProposal(A, B, (P_A, P_B), D, T_a)) \\
&\wedge T_p < T_a \\
&\rightarrow \mathbf{O}(do(A, B, P_A, D, T_{d_A})) \\
&\wedge \mathbf{O}(do(B, A, P_B, D, T_{d_B})) \\
&: T_{d_A} \leq T_a + \tau_A, T_{d_B} \leq T_a + \tau_B
\end{aligned} \tag{7}$$

5 A prototypical implementation with Constraint Handling Rules

Constraint Handling Rules [10] (*CHR* for brevity hereafter) are essentially 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 for *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 social infrastructure and the associated proof procedures. In the following, we sketch a prototype implementation of deontic constraints, which will be extended and refined in future work. It is worth noticing that what follows describes an operational counterpart of the declarative framework described in Sect. 3, and an ACL semantics as that described in Sect. 4 is a parameter to the implementation. This allows for easy implementation of extensions of the specification.

5.1 Representation of entities

Basically, the social infrastructure, along the whole life of the society that it is managing, repeats the following two operations in a cycle:

1. record an event;
2. 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[5] `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

$$H_1, \ldots, H_i \Longleftrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{8}$$

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.

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

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.

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))`.

Obligations. Obligations are represented by `ought/2` constraints. The first argument is the event associated with the obligation, the second is a list of constraints over the variables contained in the event[6]. For instance, the obligation for $agent_1$ to *give* a *nail* to $agent_2$ by time 20 in the context of dialog d would be represented as follows:
`ought(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 Sect. 5.2).

Prohibitions. Prohibitions are represented by `oughtnot/2` constraints, having the same structure of `ought/2` constraints.

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

$$H_1, \ldots, H_i \Longrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{9}$$

where the symbols have the same meaning and constraints of those in the simplification rules (8).

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 obligations and prohibitions into the store as prescribed by deontic constraints as follows:

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

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

For instance, deontic constraint (5) can be represented by the following propagation rule[7]:

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

Whenever an event matching[8] the head is added to the store, this rule will add the corresponding expectations to the store.

5.2 Procedures

Fulfillment (resp. violation) of obligations (resp. prohibitions). Operationally, an event fulfills (resp. violates) an obligation (resp. a prohibition) if the contents of the event and the expectation unify in a Prolog sense and the constraints in the expectation are true. This is achieved by a *simpagation CHR*.

In general, simpagation rules have the form

$$H_1, \ldots, H_l \backslash H_{l+1}, \ldots, H_i \Longleftrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{10}$$

where $l > 0$ and the other symbols have the same meaning and constraints of those of simplification *CHR*s (8). Operationally, when the constraints in the head are in the store and the guard is true, $H_1, \ldots, H_l$ remain in the store, and $H_{l+1}, \ldots, H_i$ are substituted by $B_1, \ldots, B_k$.

A simplified version of the rule handling fulfillment of obligation is the following:

```
happened(HEvent) \ ought(OEvent,Constraints)
<=>
HEvent=OEvent,
verify_constraints(Constraints)|
fulfilled_ought(OEvent,Constraints).
```

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

[8] As we will explain in Sect. 5.2, by "matching" we mean "unifying with", à la Prolog.

It is worthwhile noticing that a fulfilled obligation is stored as a `fulfilled_ought/2` constraint, and will not be considered anymore by the procedure.

`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 obligation is in the store:

`ought(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 obligation and the event succeeds with the binding `T=15`. Since `tle(15,20)` is true, the obligation is fulfilled and, operationally, can be stored as a `fulfilled_ought/2` constraint.

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

Let us suppose that an obligation with an associated deadline is in the store. If the social infrastructure simply waits for a matching event to happen, 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 obligation or prohibition 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). We call this kind of obligations and prohibitions *active*, as opposed to *passive* ones, which are checked for fulfillment or violation only when a corresponding event happens.

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[9]. Thus, if $T_{current}$ represents the current time, we can infer the following inequality:

$$T_{current} \leq T \tag{11}$$

which has the intuitive meaning that, if an event has not happened yet, it may only happen in the future. Inequality (11), combined with a constraint

$$T \leq T_{deadline} \tag{12}$$

implies

$$T_{current} \leq T_{deadline} \tag{13}$$

Thus, if the current time goes beyond the deadline while T is unbound, the associated obligation cannot be fulfilled, and a violation can be detected (or,

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

dually, the associated prohibition cannot be violated, and its fulfillment can be detected).

In order to implement this behavior in our framework, whenever an obligation (resp. prohibition) 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 obligation (resp. prohibition). If the `tcurrent_le/1` constraint is violated (which is detected by means of a *CHR*), a violation (resp. fulfillment) of the original obligation (resp. prohibition) is detected; if the obligation (resp. prohibition) is fulfilled (resp. violated) before the deadline, the associated `tcurrent_le/1` constraint is removed.

6 Discussion and related work

In defining the ACL semantics, we have followed a social approach which, better than the mentalistic, seems to fulfill the requirements of communication in open societies of autonomous and heterogeneous agents.

In part, we draw inspiration from [5], 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 there presented is similar to ours in that it is *social-based*: it makes no assumptions on the nature of agents, specifies semantics of actions as their social effects and presupposes a social framework (which is called *institution* in [11]) for assigning agents with roles, verifying agents' social behavior and, possibly, 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 deontic 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 obligation, whereas it is not obvious how to express this in [5]'s framework.

The deontic framework can express with the same formalism both protocols and social semantics of communicative acts. In [8], we have exploited deontic 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 [12]. However, in [12] 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 obligations is certainly an interesting option.

Another useful property of the deontic framework is *compositionality*: new deontic constraints can simply be added to an existing specification in order to make it stricter, if so wanted.

Future work will regard both the declarative specification of a society of agents and its operational counterpart.

From a declarative point of view, we intend to extend the framework by adding the possibility of *alternative* expectation which can be fulfilled or violated by different events.

Operationally, we plan to extend the implementation accordingly, and to improve and refine it as needed to operate in an open environment. We will also address the non-trivial issue of recovery from violations.

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 as done, for instance, in [13].

Artikis, Pitt and Sergot [14] propose a framework for the specification of open societies of agents grounded on event calculus. We are considering the possibility to map our concept of expectation as hypotheses in an abductive framework, extended to map the concepts of confirmation (fulfillment) and refutation (violation) of hypotheses. We intend to compare the two frameworks in terms of expressiveness, verifiable properties and computational complexity.

Acknowledgments

This work has been supported by the SOCS project, funded by CEC, contract IST-2001-32530. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of the data appearing in this publication.

References

1. Finin, T., Labrou, Y., Mayfield, J.: KQML as an agent communication language. In: Software Agents. MIT Press, Cambridge (1997)

2. FIPA: FIPA Communicative Act Library Specification (2001) Published on August 10th, 2001, available for download from the FIPA website: `http://www.fipa.org`.
3. Singh, M.: Agent communication language: rethinking the principles. IEEE Computer (1998) 40–47
4. Singh, M.P.: A social semantics for agent communication languages. In Dignum, F., Greaves, M., eds.: Issues in Agent Communication. Springer-Verlag, Heidelberg, Germany (2000) 31–45
5. Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002). Part II, Bologna, Italy, ACM (2002) 535–542
6. Yolum, P., Singh, M.: Commitment machines. In: Intelligent Agents VIII, 8th International Workshop, ATAL 2001 Seattle, WA, USA, August 1-3, 2001 Revised Papers. Volume 2333 of LNAI., Springer Verlag (2002) 235–247
7. Guerin, F., Pitt, J.: Guaranteeing properties for e-commerce systems. Technical Report TRS020015, Imperial College of Science, Technology and Medicine, London, UK. (2002)
8. Torroni, P., Mello, P., Maudet, N., Alberti, M., Ciampolini, A., Lamma, E., Sadri, F., Toni, F.: A logic-based approach to modeling interaction among computees (preliminary report). In: UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK. (2002)
9. Lloyd, J.W.: Foundations of logic programming; (2nd extended ed.). Springer-Verlag New York, Inc. (1987)
10. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming, Special Issue on Constraint Logic Programming **37** (1998) 95–138
11. Colombetti, M., Fornara, N., Verdicchio, M.: 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)
12. Yolum, P., Singh, M.: Flexible protocol specification and execution: applying event calculus planning using commitments. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy, ACM (2002) 527–534
13. Ciampolini, A., Lamma, E., Mello, P., Torroni, P.: An operational semantic for the safe execution of tasks in a constrained multi-agent setting. In Greco, S., Leone, N., eds.: Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA). Volume 2424 of LNCS., Springer Verlag (2002) 14–26
14. Artikis, A., Pitt, J., Sergot, M.: Animated specifications of computational societies. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy, ACM (2002) 1053–1061