Università degli Studi di Bologna
DEIS

Logic Based Semantics for an Agent Communication Language

Marco Alberti, Anna Ciampolini
Marco Gavanelli, Evelina Lamma
Paola Mello, Paolo Torroni

January 17, 2003

DEIS Technical Report no. DEIS-LIA-03-001 *LIA Series no. 62*

Logic Based Semantics for an Agent Communication Language

Marco Alberti [1] Anna Ciampolini [2] Marco Gavanelli [1]
Evelina Lamma [1] Paola Mello [2] Paolo Torroni [2]

[1] *Dip. di Ingegneria, Università di Ferrara*
V. Saragat, 1
44100 Ferrara (Italy)
{malberti|mgavanelli|elamma}@doc.ic.ac.uk
[2] *DEIS, Università di Bologna*
V.le Risorgimento 2
40136 Bologna, Italy
{aciampolini|pmello|ptorroni}@deis.unibo.it

January 17, 2003

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) 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.

Keywords: *Multi-Agent Systems, Computational Logic, Constraints, Set Based Unification, Operational Semantics, Constraint Handling Rules*

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. Three main approaches to the definition of ACLs can be found in literature: *mentalistic*, *conversational* and *social*.

Mentalistic approaches [3, 4] define ACL semantics in terms of agents mental states. For instance, FIPA ACL [4] 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). The mentalistic approach to ACL definition has been mainly criticized because its assumptions regarding agents' internal structure are not realistic in open societies of autonomous and heterogeneous agents. As Singh stated [10], emphasizing mental agency leads to the assumption that agents should be primarily understood in terms of mental concepts, such as beliefs and intentions: this approach supposes, in essence, that agents can read (or at least model) each other's minds. Whenever agents' mental states are not accessible, which is reasonably the case if agents operate in open and heterogeneous environments, it is impossible to verify semantic compliance of communicative acts.

The conversational approach relies on Conversation Policies (CP) [7], which express meaning and composition of speech acts for agents by means of static structures defining the sequences of utterances making a coherent conversation. The main advantage of this approach is that in this perspective a communicative act does not have an absolute meaning, but its semantics rather depends on the conversation (i.e., the context in which it is uttered), thus overcoming one of the main drawbacks of the mentalistic approach. However, the conversational approach has been criticized for its lack of flexibility, i.e., the lack of compositional rules governing how protocols are extended or merged (see [8]).

The social approach [11] moves from criticism to the mentalistic approach, and defines ACL semantics in terms of the effects of the communicative acts on the society. Following this approach, even if the agents' mental state cannot be accessed, it is possible to verify whether communicating agents in a society comply to the social laws that regulate the interactions. Significant social proposals are based on the concept of *commitments* [5, 13], 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 present a logic-based approach to the definition of an ACL semantics which follows the principles of the social approach. It defines the meaning of communicative acts not as their effect on the mental state of single agents (upon which no *a priori* assumptions are made) but as their (verifiable) social effect. Our semantics is based on the concept of *obligation/prohibition* (which, as explained in Section 6, is broader than that of commitment), linked to events by *deontic integrity constraints*, which provide a declarative specification of the proper behavior of interaction in a society of agents. This specification can be modeled in a computational logic framework. A suitable tool for implementing deontic constraints in a computational logic setting is the *Constraint Handling Rules* (*CHR*) language.

The paper is structured as follows. In Section 2 we describe the model of agent society in which we set our work; in Section 3 we introduce the concept of deontic constraint, and in Section 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 *CHR* language is presented in Section 5; discussion and directions for future work follow.

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 [12].

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 possible actual implementation of some of these features in Section 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 Figure 1).

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

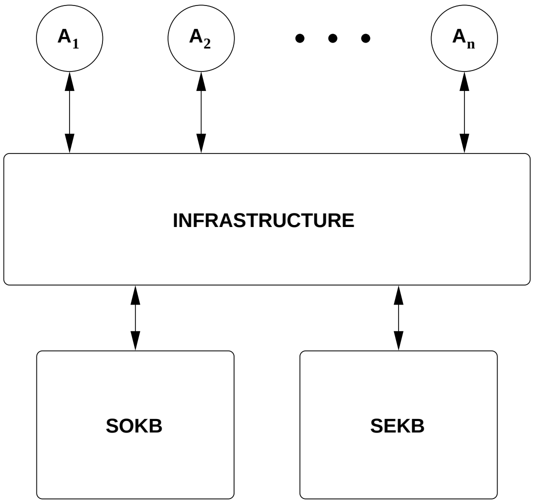

Figure 1. *Functional model of agent society*

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 obligations and prohibitions 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 in the following) link events and obligations/prohibitions.

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

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

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.

Definition 1 (Deontic entailment) is indicated with the symbol $\models_D$ and is recursively defined as follows:

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

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 $\mathcal{O}$ of obligations and prohibitions such that:

$$\mathbf{SOKB} \cup \mathcal{H} \cup \mathcal{IC}_\mathcal{D} \cup \mathcal{O} \not\models \bot \tag{2}$$

and

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

It should be noticed that, in Eq. (2), we separate the set $\mathcal{IC}_\mathcal{D}$ of deontic constraints out of **SOKB** even though it is a part of **SOKB**, to emphasize its importance.

Intuitively, there can be more than one $\mathcal{O}$ satisfying Eq. (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 not a deontic violation, but it is an inappropriate modeling of the society: we call this situation a *failure*.

Definition 3 (

)A society *fails* if $\nexists \mathcal{O}$ satisfying Eq. 2.

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 (A) protocol is *violated* with respect to a history $\mathcal{H}$ iff there exists a formula $\mathcal{O}$ such that Eq. 2 holds, and for all $\mathcal{O}$ satisfying Eq. 2,

$$\mathcal{H} \nvDash_{\mathcal{D}} \mathcal{O}$$

When a failure arises, it means that the society has been ill modeled since a constraint representing social laws is violated. In this case, 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**.

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. Intuitively, obligations and prohibitions 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 obligations and prohibitions in the head ϕ should be raised. Obligations and prohibitions 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 Section 5.

4 Constraint-based ACL semantics

In the following, we show how social semantics of an ACL can be defined by means of deontic constraints. We consider communicative acts from a social perspective, i.e., we do not constrain the internal communicative model of agents (by which communicative acts are *motivated*), but instead link communicative acts to the obligations/prohibitions they generate. Social semantics of communicative acts is thus verifiable regardless of the internal structure of agents.

Throughout this section, we show how social semantics of communicative acts can be defined by means of an example where we consider a set of 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).

In the remainder, we describe a communicative act by an atom of this structure:

$$act\_name(Sender, Receiver, Content, Dialog, Time) \tag{4}$$

A *request* does not generate any obligations.

If an agent *accept*s a commitment, 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 obligations (or prohibitions), in the sense that they should be fulfilled by some event in the future.

Moreover, an *accept* should only be issued after a corresponding *request* has been issued. This can be expressed by a *backward* obligation, i.e., an obligation regarding an event that *should have happened* before current time. Backward obligation allow us to ensure a correct sequentiality of communicative acts.

This can be expressed in out formalism by the following two deontic constraints:

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

$$\begin{aligned} &\mathbf{H}(request(B, A, P, D, T_r)) \\ \wedge &\mathbf{H}(accept(A, B, P, D, T_a)) \\ \wedge &T_r < T_a \\ \rightarrow &\mathbf{O}(do(A, B, P, D, T_d)) : T_d \leq T_a + \tau \end{aligned} \tag{6}$$

It is worthwhile noticing that the obligation of constraint (6) is raised as the effect of two sequential communicative acts (namely *request* from B to A and *accept* from A back to B), and not of only one. It should also be noticed that the $<$ in $T_r < T_a$ is an atom, whereas the $\leq$ in $T_d \leq T_a + \tau$ is a constraint.

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*.

$$\begin{aligned} &\mathbf{H}(reject(A, B, P, D, T_2)) \\ \rightarrow &\mathbf{O}(request(B, A, P, D, T_1)) : T_1 < T_2 \end{aligned} \tag{7}$$

do is a physical action, which fulfills the obligation of deontic constraint (6) 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 Section 5) automatically matches an obligation with a corresponding event.

If no proper *do* action has been performed at the deadline, a violation should be generated.

For instance, let us consider the effect of this 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 $\mathcal{O}$ of obligations[1] is depicted in Figure 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 obligation 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) is raised, and, because of deontic constraint (6), the two happened events, whose contents match and whose time condition is fulfilled, raise a forward obligation 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.

5 A prototypical implementation with Constraint Handling Rules

Constraint Handling Rules [6] (*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. The implementation is written for the *CHR* library of SICStus Prolog [9].

[1] We assume $\mathcal{O}$ is built incrementally, as explained in Section 3.

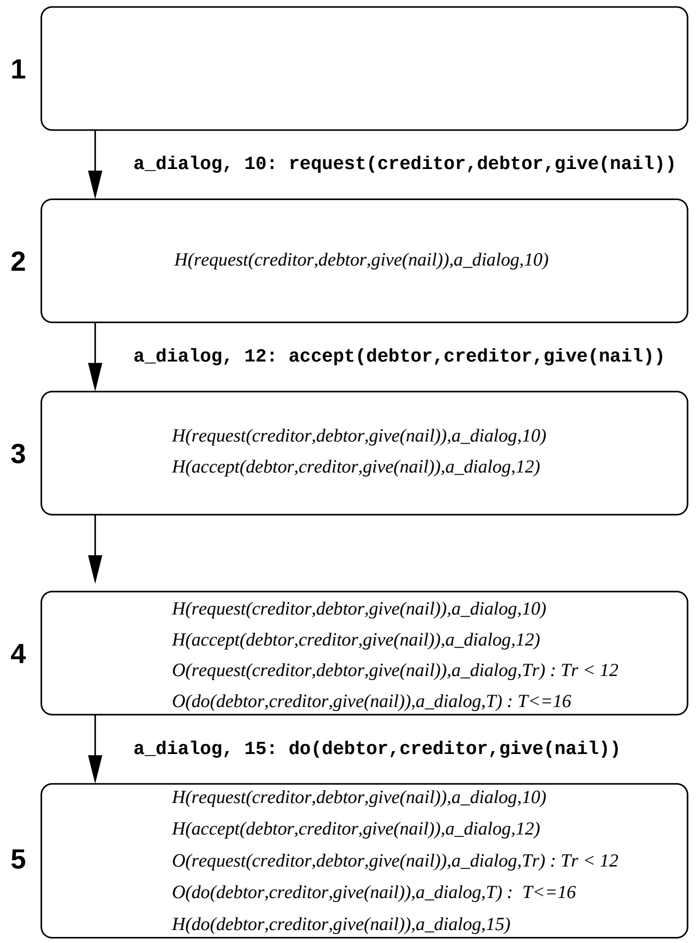

Figure 2. *Simple example of a dialog*

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[2] `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.

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[3]. 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)])`,

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

[3] In the current implementation, obligations and prohibitions cannot share constrained variables.

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 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:

- 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:

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

Whenever an event matching[4] the head is added to the store, this rule will add the corresponding obligations and/or prohibitions to the store.

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

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 obligation/prohibition unify in a Prolog sense and the constraints in the obligation/prohibition 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)|
oldought(OEvent,Constraints).
```

It is worthwhile noticing that a fulfilled obligation is stored as an `oldought/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 an `oldought/2` constraint.

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

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 obligations/prohibitions need to be detected without waiting for some event to happen. In order to handle this case, 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[5].

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, 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.

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

6 Discussion and related works

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 [2]) 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).

For instance, a commitment can have several states, while obligations do not have states. We can indeed map to our framework the states of *empty* (no obligation), *active* (a pending obligation), *fulfilled* (an obligation which has been fulfilled) and *violated* (an obligation which has been violated); since we cannot map the states of *pre-commitment*, *canceled* and *conditional*, our approach might, at a first look, seem less expressive.

However, if we think of the social infrastructure/institution as a black box with agents' actions as inputs and fulfillment or violation of expectations/commitments as outputs, it is possible to obtain an equivalent behavior with our approach to one specified by the approach of [5].

For instance, according to [5], the sequence of a *request*, an *accept* and the instantiation to *true* of the condition would result in an a *active commitment* for the *debtor*, which, depending on its actions, will then fulfill it or violate it.

The same effect can obtained by the following deontic constraint:

$$
\begin{aligned}
&\mathbf{H}(\mathit{conditionalRequest}(A, B, (P, Q), D, T_r))\\
&\wedge\mathbf{H}(\mathit{conditionalAccept}(B, A, (P, Q), D, T_a))\\
&\wedge\mathbf{H}(\mathit{true}(Q, T_Q))\\
&\wedge T_r < T_a\\
&\rightarrow\mathbf{O}(\mathit{do}(B, A, P), D, T_d) : T_d \leq T_Q + \tau
\end{aligned}
\tag{14}
$$

where *conditionalAccept* and *conditionalRequest* are the conditional versions of the communicative acts described in Section 4, i.e., they raise an obligation only when an external condition (Q) becomes true, and with $\mathbf{H}(\mathit{true}(Q, T_Q))$, we mean that, from T_Q on, Q is true.

In general, it is sufficient to specify the sequences of communicative actions that should raise an obligation. This has an advantage in terms of expressiveness in that a communicative act can be given different semantics depending on the context.

Furthermore, our notion of *obligation* 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 [12], 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 [14]. However, in [14] 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 FSM). 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* obligations and prohibitions 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.

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 [1].

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] A. Ciampolini, E. Lamma, P. Mello, and P. Torroni. An operational semantic for the safe execution of tasks in a constrained multi-agent setting. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *LNCS*, pages 14–26. Springer Verlag, September 2002.

[2] M. Colombetti, 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.

[3] T. Finin, Y. Labrou, and J. Mayfield. *Software Agents*, chapter KQML as an agent communication language. MIT Press, Cambridge, 1997.

[4] FIPA. FIPA Communicative Act Library Specification, aug 2001. Published on August 10th, 2001, available for download from the FIPA website: `http://www.fipa.org`.

[5] N. Fornara 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, Bologna, Italy*, pages 535–542. ACM, 2002.

[6] Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.

[7] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy? In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 118–131. Springer-Verlag: Heidelberg, Germany, 2000.

[8] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols – new trends in agent communication languages. *The Knowledge Engineering Review*, 2003. to appear.

[9] SICS. SICStus Prolog manual. `http://www.sics.se/sicstus`.

[10] M. Singh. Agent communication language: rethinking the principles. *IEEE Computer*, pages 40–47, December 1998.

[11] M. P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag, Heidelberg, Germany, 2000.

[12] 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.

[13] P. Yolum and M. Singh. Commitment machines. In *Intelligent Agents VIII, 8th International Workshop, ATAL 2001 Seattle, WA, USA, August 1-3, 2001 Revised Papers*, volume 2333 of *LNAI*, pages 235–247. Springer Verlag, 2002.

[14] 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.