Specifying and Verifying Systems of Communicating Agents in a Temporal Action Logic

Laura Giordano[1], Alberto Martelli[2], and Camilla Schwind[3]

[1] Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria
[2] Dipartimento di Informatica, Università di Torino, Torino
[3] MAP, CNRS, Marseille, France

Abstract. In this paper we develop a logical framework for specifying and verifying systems of communicating agents. The framework is based on a Dynamic Linear Time Temporal Logic (DLTL). It provides a simple formalization of the communicative actions in terms of their effects and preconditions and the specification of an interaction protocol by means of temporal constraints. We adopt a social approach to agent communication (as proposed by Singh): communication can be described in terms of changes in the social relations between participants, and protocols in terms of creation, manipulation and satisfaction of commitments among agents. The description of the interaction protocol and of communicative actions is given in a temporal action theory, and agent programs, when known, can be specified as complex actions (regular programs in DLTL). The paper addresses several kinds of verification problems (including the problem of compliance of agents to the protocol), which can be formalized either as validity or as satisfiability problems in the temporal logic and can be solved by model checking techniques.

1 Introduction

In this paper we develop a logical framework for specifying and verifying systems of communicating agents. The framework relies on the theory for reasoning about action developed in [8] which is based on Dynamic Linear Time Temporal Logic (DLTL). It allows a simple formalization of the communicative actions in terms of their effect and preconditions as well as the specification of an interaction protocol to constrain the behaviours of autonomous agents.

The paper adopts a social approach to agent communication [17, 11]. This approach can be contrasted with the mentalistic approach, which describes the semantics of communication on the base of mental attitudes of the communicating agents, namely, their beliefs, desires and intentions [4] which are represented by modal operators[1]. The mental approach is not well suited for the verification of an "open" multiagent system, where the history of communications is

[1] A closely related approach was proposed by Labrou and Finin [15] to define a semantics for KQML.

observable, but the internal states of the single agents may not be observable. In contrast, in the social approach communicative actions affect the "social state" of the system, rather than the internal states of the agents. The social state records the social facts, like the permissions and the commitments of the agents, which are created and modified in the interactions among them. The dynamics of the system emerges from the interactions of the agents, which must respect these permissions and the commitments (if they are compliant with the protocol). The social approach provides a high level specification of the protocol, and does not require the rigid specification of all the allowed action sequences by means of finite state diagrams, as in a behavioral approach.

Dynamic and temporal logics have recently gained renewed attention in the areas of reasoning about actions and planning [1, 10, 8, 3], fertilizing them with the well known results and techniques developed for such logics. In particular, linear time temporal logic has become a well established tool for specifying the behaviour of distributed systems, for which a rich theory has been developed and the verification task can be automated. In this paper we present a theory for reasoning about communicative actions in a multiagent system which is based on the Product Version of Dynamic Linear Time Temporal Logic (denoted $DLTL^{\otimes}$) [13], an extension of LTL (the propositional linear time temporal logic). It allows to describe the behaviour of a network of sequential agents which coordinate their activities by performing common actions together. $DLTL^{\otimes}$ extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. It is, essentially, a dynamic logic equipped with a linear time semantics. It allows the formulas of the logic to be decorated with the names of sequential agents, and it provides a simple way of constraining the (possibly infinite) behaviours of the agents by making use of regular programs. As we will see, regular programs are well suited to model both the agent behaviours and the protocol.

In [13] $DLTL^{\otimes}$ has been shown to be expressively equivalent to the regular product languages and to admit an exponential time decision procedure. In particular, the satisfiability and model checking problems for $DLTL^{\otimes}$ can be solved by product Büchi automata.

The action theory we introduce to model agent communication is an extension of the theory presented in [8], which is based on the logic DLTL, and allows reasoning with incomplete initial states and dealing with postdiction, ramifications as well as with nondeterministic actions. This theory, when extended to the multiagent case [9], allows reasoning about a fixed number of finite state sequential agents, that coordinate their activities by performing their actions together[2].

[2] $DLTL^{\otimes}$ does not allow to describe global properties of a system of agents, since the truth of a formula is evaluated at a state local to an agent and the temporal modalities define causal relationships among local states. The specification of the dynamic of the system is given through the separate specification of the different agents.

In this paper we adapt the theory to the specification and verification of systems of communicating agents, whose behaviour is ruled by an interaction protocol. Following [17, 11] a protocol is defined by describing the effects of communicative actions on the social state, and by specifying the permissions and the commitments that arise as a result of the current conversation state. In our action theory the effects of communicative actions will be modeled by *action laws* which describe the immediate effects produced by an action on the social facts. Permissions, which determine when an action can be taken by an agent, can be modeled by *precondition laws*. Commitment policies, which rule the dynamic of commitments, can be described by *causal laws* which establish the causal dependencies among fluents. The specification of a protocol can be further constrained through the addition of suitable *temporal formulas*, and also the agents' programs can be modeled, by making use of complex actions (regular programs).

The verification that an agent respects commitments and permissions and is compliant with respect to a given protocol, can be formalized as a validity problem in the temporal logic and can be solved by means of temporal model checking. Similarly, the problem of proving protocol properties can be formalized as a validity problem. On the other hand, the problem of determining if an agent is not respecting its social facts at runtime can be modeled as a satisfaction problem (and it can be regarded as an instance of the well known "temporal projection problem").

2 The Logic *DLTL* and Its Product Version

In this section first we recall the syntax and semantics of DLTL as introduced in [14], and then recall its product version. *DLTL* is an extension of LTL in which the next state modality is labelled by actions and the until operator is indexed by programs in Propositional Dynamic Logic (PDL) [12].

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, by $prf(u)$ the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows: $Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1;\pi_2 \mid \pi^*$, where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined in the standard way.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) is defined as follows:

$$\mathrm{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha\mathcal{U}^\pi\beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ) and π ranges over $Prg(\Sigma)$.

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as usual for the classical connectives. Moreover:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \alpha \mathcal{U}^{\pi} \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$.

The formula $\alpha \mathcal{U}^{\pi} \beta$ is true at τ if "α until β" is true on a finite stretch of behaviour which is in the linear time behaviour of the program π. A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The derived modalities $\langle \pi \rangle$ and $[\pi]$ can be defined as follows: $\langle \pi \rangle \alpha \equiv \top \mathcal{U}^{\pi} \alpha$ and $[\pi]\alpha \equiv \neg \langle \pi \rangle \neg \alpha$. Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, then $\bigcirc$ (next), $\Diamond$ and $\Box$ of LTL can be defined as follows: $\bigcirc \alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$, $\Diamond \alpha \equiv \top \mathcal{U}^{\Sigma^*} \alpha$, $\Box \equiv \neg \Diamond \neg \alpha$.

Let us now recall the definition of $DLTL^{\otimes}$ from [13]. Let $Loc = \{1, \ldots, K\}$ be a set of *locations*, the names of the agents synchronizing on common actions. A *distributed alphabet* $\tilde{\Sigma} = \{\Sigma_i\}_{i=1}^{K}$ is a family of (possibly non-disjoint) alphabets, with each Σ_i a non-empty, finite set of actions (Σ_i is the set of actions which require the participation of agent i). Let $\Sigma = \bigcup_{i=1}^{K} \Sigma_i$. For $\sigma \in \Sigma^{\infty}$, we denote by $\sigma \uparrow i$ the projection of σ down to Σ_i.

Atomic propositions are introduced in a local fashion, by introducing a non-empty set of atomic propositions $\mathcal{P}$. For each proposition $p \in \mathcal{P}$ and agent $i \in Loc$, p_i represents the "local" view of the proposition p at i, and is evaluated in the local state of agent i.

Let us define the set of formulas of $DLTL^{\otimes}(\tilde{\Sigma})$ and their locations (if α is a formula, then $loc(\alpha)$, which is a subset of Loc, denotes its location): (a) $\top$ is a formula and $loc(\top) = \emptyset$; (b) if $p \in \mathcal{P}$ and $i \in Loc$, p_i is a formula and $loc(p_i) = \{i\}$; (c) if α and β are formulas, then $\neg\alpha$ and $\alpha \vee \beta$ are formulas and $loc(\neg\alpha) = loc(\alpha)$ and $loc(\alpha \vee \beta) = loc(\alpha) \cup loc(\beta)$; (d) if α and β are formulas and $loc(\alpha), loc(\beta) \subseteq \{i\}$ and $\pi \in Prg(\Sigma_i)$, then $\alpha \mathcal{U}_i^{\pi} \beta$ is a formula and $loc(\alpha \mathcal{U}_i^{\pi} \beta) = \{i\}$. Notice that no nesting of modalities $\mathcal{U}_i$ and $\mathcal{U}_j$ (for $i \neq j$) is allowed, and the formulas in $DLTL^{\otimes}(\tilde{\Sigma})$ are boolean combinations of formulas from the set $\bigcup_i DLTL_i^{\otimes}(\tilde{\Sigma})$, where

$$DLTL_i^{\otimes}(\tilde{\Sigma}) = \{\alpha \mid \alpha \in DLTL^{\otimes}(\tilde{\Sigma}) \; and \; loc(\alpha) \subseteq \{i\} \}.$$

A model of $DLTL^{\otimes}(\tilde{\Sigma})$ is a pair $M = (\sigma, V)$, Where $\sigma \in \Sigma^{\infty}$ and $V = \{V_i\}_{i=1}^{K}$ is a family of functions V_i, where $V_i : prf(\sigma \uparrow i) \rightarrow 2^{\mathcal{P}}$ is the valuation function for location i.

The satisfiability of formulas in a model is defined as above, except that propositions are evaluated locally. In particular, for all $\tau \in prf(\sigma)$:

- $M, \tau \models p_i$ iff $p \in V_i(\tau \uparrow i)$;
- $M, \tau \models \alpha \mathcal{U}_i^{\pi} \beta$ iff there exists a τ' such that $\tau' \uparrow i \in [[\pi]]$, $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for all $\tau'' \in prf(\tau')$, if $\varepsilon \leq \tau'' \uparrow i < \tau' \uparrow i$, then $M, \tau\tau'' \models \alpha$.

Satisfiability in $DLTL^{\otimes}$ is defined as above. Moreover, the derived modalities $\langle\pi\rangle_i$, $[\pi]_i$, $\bigcirc_i$, $\Diamond_i$ and $\Box_i$ are defined as above, but only considering the actions in Σ_i.

3 Action Theories and Systems of Communicating Agents

In this section we introduce the main features of the action theory developed in [8] and (for the multiagent case) in [9]. The behavior of the global system will emerge as the product of the behaviors of the single agents which interact by synchronizing on communicative actions. Each agent participating in the action execution has its own local description of the action determining the action effects on its local state.

The global state of the system can be regarded as a set of local states, one for each agent i. The action laws and causal laws of agent i describe how the local state of i changes when an action $a \in \Sigma_i$ is executed. The underlying model of communication is the synchronous one: the communication action $send_{i,j}(m)$ (message m is sent by agent i to agent j) is shared by agent i (the sender) and agent j (the receiver) and executed synchronously by them. Their local states are updated separately, according to their action specification. Though, for simplicity, we adopt the synchronous model, an asynchronous model can be easily obtained by explicitly modelling the communication channels among the agents as distinct locations.

The meaning of communicative actions is fixed by the *protocol* which describes the effects of each action on the social state of the system. These effects, including the creation of new commitments, can be expressed by means of *action laws*. Moreover, the protocol establishes a set of preconditions on the executability of actions, which can be expressed by means of *precondition laws*. Each agent has a local view of the social state and the execution of a communicative action can in general affect both the state of the sender and the state of the receiver. In particular, all agents can see the effects on the social state of the actions to which they participate[3]. We assume that the executability of actions is only determined by preconditions on the state of the sender and that a communicative action is always executable for the receiver. Causal and precondition laws are as in [8, 9][4].

As a running example we will use the NetBill protocol, developed for buying and selling goods on the Internet [20].

[3] Observe that, while in the case of a two agents system the history of all communications is known to both agents (as they participate in all communicative actions) and they have the same local view of the social state, this is not true for more than two participants.

[4] Given an action a with precondition α, a precondition law has the form $\neg\alpha \rightarrow [a]\bot$, which expresses that the execution of an action a is not possible (i.e. there is no resulting state following the execution of a) if α does not hold.

Example 1. The protocol begins with a customer requesting a quote for some desired goods, followed by the merchant sending the quote. If the customer accepts the quote, then the merchant delivers the goods and waits for an electronic payment order. After receiving the payment, the merchant forwards the receipt to the customer, who can then successfully decrypt and use the goods. The agents have the freedom to start an exchange at any point in this sequence. For instance, the merchant may begin by sending the quote without waiting for a request; the customer may send an 'accept' message without prior conversation on the price of the goods.

To model this example, we introduce two locations, corresponding to the two agents mr, the merchant, and ct, the customer, that is $Loc = \{mr, ct\}$. We call *fluents* the atomic propositions in $\mathcal{P}$ indexed by an agent name i. Namely, for each $p \in \mathcal{P}$ and $i \in Loc$, p_i is a fluent local to i. However, we will omit the index i in p_i when it is clear from the context, for instance, when p_i is in the scope of a modality labeled with i (and we will write $[a]_i p$ for $[a]_i p_i$).

The (domain specific) fluents for this protocol are the following ones: *goods*, *request*, *receipt*, *paid*, *accept*. There are also special fluents to represent commitments. They can be base-level commitments, of the form $C(ag_1, ag_2, action)$ (agent ag_1 is committed to agent ag_2 to execute the $action$), or they can be conditional commitments of the form $CC(ag_1, ag_2, p, action)$ (agent ag_1 is committed to agent ag_2 to execute $action$, if the condition p is brought out)[5]. In this example, we will use the following fluents to represent commitments: $C(mr, ct, sendGoods)$, $C(mr, ct, sendReceipt)$, $CC(mr, ct, accept, sendGoods)$ and $CC(mr, ct, paid, sendReceipt)$.

In this example, the two agents share exactly the same communicative actions, which are the following ones: *sendQuote*, *sendGoods*, *sendReceipt* for which the sender is the merchant and the receiver is the customer; *sendRequest*, *sendAccept* and *sendPayment*, for which the sender is the customer and the receiver is the merchant. These actions are executed synchronously by the sending and receiving agents.

Let us consider the protocol from the point of view of the merchant. The effects of actions on the *merchant's view of the social state* are described by the following action laws:

Actions of the merchant:
(1) $\Box_{mr}([sendQuote]_{mr}(CC(mr, ct, accept, sendGoods) \wedge$
$CC(mr, ct, paid, sendReceipt)))$
(2) $\Box_{mr}(request \rightarrow [sendQuote]_{mr} \neg request)$
(3) $\Box_{mr}([sendGoods]_{mr}(goods \wedge CC(mr, ct, paid, sendReceipt)))$
(4) $\Box_{mr}([sendReceipt]_{mr} receipt)$

[5] The two kinds of base-level and conditional commitments we allow are essentially those introduced in [20]. For simplicity, we do not consider the more general higher level commitments in [17]. Moreover, as in [11] and differently from [20], agents are committed to execute an action rather than to achieve a condition.

Actions of the customer:
(5) $\Box_{mr}([sendRequest]_{mr} request)$
(6) $\Box_{mr}([sendAccept]_{mr} accept)$
(7) $\Box_{mr}([sendPayment]_{mr} paid)$

For instance, the first law says that, when the merchant sends the quote for the good then he commits to send the goods if the customer accepts the request, and he also commits to send the receipt if the customer pays the agreed amount. It is an example of action, whose effect is to create new commitments. The second law says that when the merchant sends the quote for the good when there is a request, then the request is cancelled.

The preconditions on the executability of actions are social facts. We assume they are preconditions on the state of the sender[6], while we assume that it is always possible for an agent to receive a message. The only *precondition law* for the merchant is the following one:

(8) $\Box_{mr}(\neg paid \rightarrow [sendReceipt]_{mr}\bot)$

meaning that when the payment has not been done by the customer, the action *sendReceipt* cannot be executed. All other actions are always executable by the merchant. We will denote by $Perm_i$ the set of precondition laws of agent i.

The action laws describing the effects of communicative actions on the state of the customer are exactly the same as for the merchant (as the two agents share all their communicative actions), apart from the fact that all modalities and propositions are indexed with *ct*. We number them (1')–(7'). The only condition on the executability of actions for the customer is the following precondition law:

(9) $\Box_{ct}(\neg goods \rightarrow [sendPayment]_{ct}\bot)$

meaning that the customer may send a payment for the goods only if he has received the goods (all other actions are always executable for the customer).

Let us now define the rules for reasoning about the commitments. In the general case, the actions in the protocol can be regarded as operations on commitments, not only creation operations, but also cancellation, fulfilling and manipulation operations [17]. While it is clear that in this logical formalization it would be simple to define operations to manipulate commitments, in this paper we will assume that all (base-level) commitments of an agent have to be fulfilled for the agent to be compliant with the protocol. Some reasoning rules have to be defined, for cancelling commitments when they have been fulfilled and for dealing with conditional commitments. We introduce the following laws (where $k = i, j$):

(10) $\Box_k([a]_k \neg C(i,j,a))$
(11) $\Box_k([a]_k \neg CC(i,j,p,a))$
(12) $\Box_k((CC(i,j,p,a) \wedge \bigcirc_k p) \rightarrow \bigcirc_k(C(i,j,a) \wedge \neg CC(i,j,p,a)))$

[6] These precondition laws provide a formalization of what are called the PERMIT constraints in [11]. Such permissions do not force the agent to do anything, but determine which actions are executable in a state.

where we assume that the action a is shared by the agents i (debtor) and j (creditor) (if not an explicit *discharge* action should be introduced to make the creditor aware that the commitment has been fulfilled by the debtor). A commitment (or a conditional commitment) to execute an action a is cancelled when the action a has been executed by the debtor (action laws (10) and (11)). A conditional commitment $CC(i, j, p, a)$ becomes a base-level commitment $C(i, j, a)$ when the condition p has been brought out (law (12)). This last law is a *causal law*. Causal laws describe the causal dependencies among fluents and are needed to model the ramification effects of actions [8].

The specification of a system of communicating agents is hence given by defining the protocol for each agent i through a domain description D_i. D_i is a tuple $(\Pi_i, \mathcal{C}_i, Perm_i)$ where: Π_i is a set of *action laws* and *causal laws*; $\mathcal{C}_i$ is a set of *constraints* and $Perm_i$ is a set of *precondition laws*. The set of constraints $\mathcal{C}_i$ contains all the temporal formulas which might be needed to constrain the behaviour of agent i. It includes the *observations* about the value of fluents in different i-states, which have the form $[a_1; \ldots; a_j]_i \alpha$, meaning that α holds in the i-state obtained after executing the action sequence $a_1, \ldots, a_j$[7]. Though also the precondition laws in $Perm_i$ are constraints, we have kept them separate from $\mathcal{C}_i$, as they have a special role in the verification of the agent system. A *(distributed) domain description* D is a family of domain descriptions D_i, one for each agent i.

Observe that action laws and causal laws only describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state. They are called *frame* fluents. To cope with the frame problem, the laws in Π_i, describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in $\mathcal{C}_i$ and given a special treatment. In [8], to deal with the frame problem, a completion construction is defined which, given a domain description, introduces frame axioms for all the frame fluents in the style of the successor state axioms introduced by Reiter [16] in the context of the situation calculus. The completion construction is applied only to the action laws and causal laws in Π_i and not to the constraints. In the following we call $Comp(\Pi_i)$ the completion of a set of laws Π_i (the action laws and causal rules of each agent i have to be completed separately) and we refer to [8] for the details on the completion construction.

Given a distributed domain description D, let the completed domain description $Comp(D)$ be the set of formulas $\bigwedge_j (Comp(\Pi_j) \wedge \mathcal{C}_j \wedge Perm_j)$. The runs of the system according the protocol are the linear models of $Comp(D)$: in a model (σ, V), the behaviour of each agent i is a subsequence σ_i of the system behaviour σ. Observe that these protocol runs may still contain pending commitments, that is commitments which have not been fulfilled.

In the next section we will address, among others, the problem of verifying the compliance of an agent with respect to a protocol. In our formalism the program

[7] For instance, in the example above we can assume that all fluents are false in the initial state, i.e. $\mathcal{C}_i = \{\neg f_k,\ \text{for all fluents } f_k\}$

of an agent can be specified by making use of complex actions. Consider the following program π_{mr} for the merchant:

$$[\neg done?; ((sendRequest; sendQuote) + (sendAccept; sendGoods) + (sendPayment; sendReceipt; exit))]^*; done?$$

The program above describes the behaviour of a reactive agent which sends the quote when receiving a request, sends the goods when receiving the "accept", sends a receipt after receiving the payment and terminates after sending the receipt. We assume that the fluent *done* is initially false and that the action *exit* has the only effect of making it true (i.e. $\Box_{mr}[exit]_{mr} done$). In general, the *program of an agent i* is specified by a domain description $Prog_i = (\Pi_i^p, \mathcal{C}_i^p)$. In the example above, we have: $\Pi_{mr}^p = \{\Box_{mr}[exit]_{mr} done\}$ and $\mathcal{C}_{mr}^p = \{\langle \pi_{mr} \rangle_i \top, \neg done\}$. Observe that Π_{mr}^p contains the specification of those actions which are local to the agent *mr* (in this case only the action *exit*). The constraints in $\mathcal{C}_{mr}^p$ say that in the initial state the program π_{mr} for the merchant is executable and the fluent *done* is is false.

The action *done*? is a test action, in the style of dynamic logic. Test actions allow the choice among different behaviours to be controlled. As $DLTL^{\otimes}$ does not include test actions, we introduce them in the language as atomic actions in the same way as done in [8]. To allow agent i to test the value of a proposition ϕ in its local state, we introduce the modality $[\phi?]_i$ (regarded as an atomic action in Σ_i). We assume that, for all test actions occurring in a domain description, the corresponding action laws are implicitly added.

4 Verification of Communicating Agents

In this section we consider some different types of verification problems that can be solved in this framework. As in [11] we consider the following four types of verification for an open system:

1. Verify that an agent will always satisfy its social facts;
2. Verify the outcome of a system, assuming unknown agents are compliant;
3. Prove a property of a protocol;
4. Determine if an agent is not respecting the social facts at runtime.

While the fourth property is a run-time property, the first three can be verified at design time.

For the first verification problem, we assume that we have access to agent i program code, $Prog_i$. Then at design time we can verify that this agent respects its permissions and its commitments. We say that an agent i *satisfies its permissions* if, in all runs of the system, when a communicative action a is executed, then its execution is possible according to the precondition laws in $Perm_i$, that is, all precondition laws $\Box(\alpha \rightarrow [a]\bot) \in Perm_i$ are satisfied. An agent i *satisfies its commitments* when, in all the runs of the system, for all the possible commitments $C(i, j, a)$ in which agent i is the debtor, the formula

$$\Box_i(C(i, j, a) \rightarrow \Diamond_i\langle a \rangle_i \top)$$

holds. Such a formula says that, when an agent is committed to execute action a, then it must eventually execute a. We denote with Com_i the set of all the formulas, as the one above, describing the satisfaction of the commitments of agent i.

To formalize the first verification problem assume that the specification of the program of agent i is given by the domain description $Prog_i = (\Pi_i^p, \mathcal{C}_i^p)$, and the protocol is given by the distributed domain description D, where the domain description for each agent j in Loc is $D_j = (\Pi_j, \mathcal{C}_j, Perm_j)$. The verification problem 1 can be formalized as a validity check. The formula $(\bigwedge_j(Comp(\Pi_j) \wedge \mathcal{C}_j) \wedge \bigwedge_{j \neq i}(Perm_j \wedge Com_j) \wedge Comp(\Pi_i^p) \wedge \mathcal{C}_i^p) \rightarrow (Perm_i \wedge Com_i)$ (where j ranges on all agents) is valid if in all the behaviours of the system, in which agent i respects its specification $Prog_i$ and all other agents j (whose internal program is unknown) respect the protocol specification (including their permissions and commitments), the permissions and commitment of agent i are also satisfied. This means that agent i is compliant (respects its "social facts") under the assumption that all other agents in the protocol are compliant[8].

The verification problem 2 can be formalized similarly. Again we only have information about the behaviour of agent i and we want to show that the outcome α of the system will eventually be achieved. The kind of properties that can be captured in this framework have to refer to the local states of the different agents separately (there is no global state). For instance, we can require that agents h and i will eventually reach the states α_h and α_i. To check this, we can check the validity of the formula: $(\bigwedge_j(Comp(\Pi_j) \wedge \mathcal{C}_j) \wedge \bigwedge_{j \neq i}(Perm_j \wedge Com_j) \wedge Comp(\Pi_i^p) \wedge \mathcal{C}_i^p) \rightarrow (\Diamond_h \alpha_h \wedge \Diamond_i \alpha_i)$. Observe that here we have not made any assumption that agent i is compliant, we just assume that it behaves according to its program. The other agents, instead, are assumed to be compliant.

In the verification problem 3, the internals of all agents are not accessible. We only know that the agents are compliant with the protocol and we want to prove some property p of the protocol. Again we can formalize it as a validity check of the formula: $(\bigwedge_j(Comp(\Pi_j) \wedge \mathcal{C}_j) \wedge \bigwedge_j(Perm_j \wedge Com_j)) \rightarrow p$.

The verification problem 4 is performed at run-time. Nothing is known about the internals of the agents. An external observer observes the history of the communications among the agents up to one state (that is the sequence of communicative actions that have been executed). Given such a history we want to check if there is evidence that some agent is not compliant. In essence, we want to verify if the history is a possible (prefix of) an allowed behaviour of the system, that is a behaviour in which all agents are compliant with the protocol (if not, the history gives evidence that some agent violates the protocol). This can be formalized as a satisfiability problem. Let $\tau = a_1, \dots, a_n$ be the observed actions sequence. Let $\tau \uparrow 1, \dots, \tau \uparrow K$ be the projections of the action sequence τ on the actions of the agents $\{1, \dots, K\}$, and let $\tau \uparrow i = a_1^i, \dots, a_{n_i}^i$. The formula:

[8] Observe that while the behaviours of agent i are constrained both by the protocol and by its own specification, other agents may perform any action sequence which is compatible with the constraints posed by the protocol on their behaviour and the fact that they have to fulfill their commitments.

$\bigwedge_j(Comp(\Pi_j)\wedge\mathcal{C}_j)\wedge\bigwedge_j(Perm_j\wedge Com_j)\wedge(\langle a_1^1;\ldots;a_{n_1}^1\rangle\top\wedge\ldots\wedge\langle a_1^K;\ldots;a_{n_K}^K\rangle\top)$ will be satisfied if there is a behaviour σ of the system (a model (σ, V)) in which, for each $i = 1 \ldots N$, the behaviour of agent i starts with the action sequence $\tau \uparrow i = a_1^i, \ldots, a_{n_i}^i$.

Example 2. We provide an example of the verification problem 1 to verify that the program of the merchant, specified by the domain description $Prog_{mr} = (\Pi_{mr}^p, \mathcal{C}_{mr}^p)$ above, satisfies the social facts. Let

$\Pi_{mr} = \{(1), \ldots, (7), (10), (11), (12)\}$, $\mathcal{C}_{mr} = \{\neg f_{mr} : \forall\text{fluents } f_{mr}\}$,
$Perm_{mr} = \{(8)\}$
$\Pi_{ct} = \{(1'), \ldots, (7'), (10), (11), (12)\}$, $\mathcal{C}_{ct} = \{\neg f_{ct} : \forall\text{fluents } f_{ct}\}$,
$Perm_{ct} = \{(9)\}$

Com_{mr} contains the formulas

$$\Box_{mr}(C(mr, ct, sendGoods) \rightarrow \Diamond_{mr}\langle sendGoods\rangle_{mr}\top)$$
$$\Box_{mr}(C(mr, ct, sendReceipt) \rightarrow \Diamond_{mr}\langle sendReceipt\rangle_{mr}\top)$$

and $Com_{ct} = \emptyset$. The formula:

$$Comp(\Pi_{mr}) \wedge \mathcal{C}_{mr} \wedge Comp(\Pi_{ct}) \wedge \mathcal{C}_{ct} \wedge Perm_{ct} \wedge Com_{ct}\wedge$$
$$\wedge Comp(\Pi_{mr}^p) \wedge \mathcal{C}_{mr}^p \rightarrow (Perm_{mr} \wedge Com_{mr})$$

is valid if in all the behaviours of the system in which the merchant respects its specification $Prog_{mr}$ and the customer (whose internal program is unknown) respects its permissions and commitments, the permissions and commitments of the merchant are also satisfied.

The above verification and satisfiability problems can be solved by extending the standard approach for verification and model-checking of Linear Time Temporal Logic, based on the use of Büchi automata. As described in [14], the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in DLTL(\Sigma)$ a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable. This result has been extended in [13] to $DLTL^{\otimes}$. The verification of a formula $\alpha \rightarrow \beta$ can be carried out by first constructing the two Büchi automata for α and the *negation* of β. If the two automata have a common execution sequence, this sequence provides a counterexample for $\alpha \rightarrow \beta$. Thus $\alpha \rightarrow \beta$ is valid if the language accepted by the product of the two automata is empty.

The similarities between the verification approach for LTL and that for $DLTL^{\otimes}$ suggest the possibility of using techniques and tools which have been developed for LTL. For instance, it is possible to extend to $DLTL^{\otimes}$ the efficient tableau-based algorithm of [6] for constructing the automaton on the fly.

5 Conclusions

We have shown that $DLTL^{\otimes}$ is a suitable formalism for specifying and verifying a system of communicating agents. The issue of developing semantics for agent communication languages has been examined in [18], by considering in particular the problem of giving a *verifiable* semantics, i.e. a semantics *grounded* on the computational models.

Guerin's thesis [11] defines an agent communication framework which gives agent communication a grounded declarative semantics. In particular it has different languages for agent programming, for specifying agent communication and social facts, and for expressing temporal properties. Our approach provides a unified framework for describing different aspects of multi-agent systems using $DLTL^{\otimes}$. Programs are expressed as regular expressions, (communicative) actions can be specified by means of action and precondition laws, properties of social facts can be specified by means of causal laws and constraints, and temporal properties can be expressed by means of the *until* operator.

While in this paper we follow a social approach to the specification and verification systems of communicating agents, [2] and [19] have adopted a mentalistic approach. The goal of [2] is to extend model checking to make it applicable to multi-agent systems, where agents have BDI attitudes. This is achieved by using a new logic which is the composition of two logics, one formalizing temporal evolution and the other formalizing BDI attitudes. In [19] agents are written in MABLE, an imperative programming language, and have a mental state. MABLE systems may be augmented by the addition of formal claims about the system, expressed using a quantified, linear time temporal BDI logic. MABLE makes use of the SPIN model checker to automatically verify the claims. The paper does not deal with the problem of proving properties of protocols. The same can be said for the paper by Yolum and Singh [20], which presents a social approach based on event calculus to protocol specification and execution.

A related approach is that of ConGolog [5], an extended version of the language Golog, that incorporates a rich account of concurrency, in which complex actions (plans) can be formalized as Algol-like programs in the situation calculus. A substantial difference with ConGolog, apart from the different logical foundation, is that here we model agents with their own local states, while in Congolog the agents share a common global environment and all the properties are referred to a global state.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998. 263

[2] M. Benerecetti, F. Giunchiglia and L. Serafini. Model Checking Multiagent Systems. *Journal of Logic and Computation*. Special Issue on Computational Aspects of Multi-Agent Systems, 8(3):401-423. 1998. 273

[3] D. Calvanese, G. De Giacomo and M. Y.Vardi. Reasoning about Actions and Planning in LTL Action Theories. In Proc. *KR'02*, 2002. 263

[4] P. Cohen and H. Levesque, Communicative actions for artificial agents. *Int. Conf. on Multi Agent Systems*, pp.65-72, MIT Press,1995. 262

[5] G. De Giacomo, Y. Lespérance, H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(2000), pp.109-169. 273

[6] R. Gerth and D. Peled and M. Y.Vardi and P. Wolper. Simple On-the-fly Automatic verification of Linear Temporal Logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, Warsaw, June 1995, North Holland. 272

[7] L. Giordano, A. Martelli, and C. Schwind. Ramification and causality in a modal action logic. In *Journal of Logic and Computation*, 10(5):625-662, 2000.

[8] L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In FAPR'00 - Int. Conf. on Pure and Applied Practical Reasoning, London, September 2000. Also in *The Logic Journal of the IGPL*, Vol. 9, No. 2, pp. 289-303, March 2001. 262, 263, 266, 269, 270

[9] L. Giordano, A. Martelli, and C. Schwind. Reasoning about Actions in a Multiagent Domain. In *Proc. AI*IA'01*, Bari, pp. 237–248, September 2001. 263, 266

[10] F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. The 5th European Conf. on Planning (ECP'99)*, pp.1–20, Durham (UK), 1999. 263

[11] F. Guerin. Specifying Agent Communication Languages. PhD Thesis, Imperial College, London, April 2002. 262, 264, 267, 268, 270, 273

[12] D. Harel. First order dynamic logic in *Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984. 264

[13] J. G. Henriksen and P. S. Thiagarajan A product Version of Dynamic Linear Time Temporal Logic. in *CONCUR'97*, 1997. 263, 265, 272

[14] J. G. Henriksen and P. S. Thiagarajan Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999 264, 272

[15] Y. Labrou and T. Finin, A semantic approach for KQML - a general purpose communication language for software agents. In *3rd Int Conf. on Information and Knowledge Management, CIKM'94*, pp.447-455, 1994. 262

[16] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed.,pages 359–380, Academic Press, 1991. 269

[17] M. P. Singh. A social semantics for Agent Communication Languages. In *IJCAI-98 Workshop on Agent Communication Languages*, Springer, Berlin, 2000. 262, 264, 267, 268

[18] M. Wooldridge. Semantic Issues in the Verification of Agent Communication Languages. *Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 9-31, 2000. 273

[19] M. Wooldridge, M. Fisher, M. P. Huget and S. Parsons. Model Checking Multi-Agent Systems with MABLE. In *AAMAS'02*, pp. 952–959, Bologna, Italy, 2002. 273

[20] P. Yolum and M. P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, pp. 527–534, Bologna, Italy, 2002. 266, 267, 273