Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Applied Logic ●●● (●●●●) ●●●–●●●
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Specifying and verifying interaction protocols in a temporal action logic

Laura Giordano [a,1], Alberto Martelli [b,*,1], Camilla Schwind [c]

[a] *Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy*
[b] *Dipartimento di Informatica, Università di Torino, Italy*
[c] *MAP, CNRS, Marseille, France*

Abstract

In this paper we develop a logical framework for specifying and verifying systems of communicating agents and interaction protocols. The framework is based on Dynamic Linear Time Temporal Logic (DLTL), which extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. The framework 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, where 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 verifying compliance of agents with 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. In particular, we show that the verification of the compliance of an agent with the protocol requires to move to the logic DLTL$^{\otimes}$, the product version of DLTL.

1. Introduction

One of the central issues in the field of multi-agent systems concerns the specification of conversation policies, which govern the communication between software agents in an agent communication language (ACL). Conversation policies (or interaction protocols) define stereotypical interactions in which ACL messages are used to achieve communicative goals. They define patterns of communication which “can actually simplify the computational complexity of ACL message selection” [27], by providing a context in which ACL messages are interpreted.

Many ACL designers have included conversation policies as part of the ACL definition. The specification of interaction protocols has been traditionally done by making use of finite state machines and transition nets, but these approaches have been recognized as being too rigid to allow for the flexibility needed in agent communication [17,27].

[*] Corresponding author.
E-mail addresses: laura@mfn.unipmn.it (L. Giordano), mrt@di.unito.it (A. Martelli), camilla.schwind@map.archi.fr (C. Schwind).
[1] This research has been partially supported by the project PRIN 2003 “Logic-based development and verification of multi-agent systems”.

1570-8683/$ – see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.jal.2005.12.011

For these reasons, several proposals have been put forward to address the problem of specifying (and verifying) agent protocols in a flexible way. One of the most promising approaches to agent communication, first proposed by Singh [32], is the social approach [1,8,18,27,33]. In the social approach, communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the permissions and the commitments of the agents. The dynamics of the system emerges from the interactions of the agents, which must respect these permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and does not require the rigid specification of the allowed action sequences. It is well suited for dealing with "open" multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

In this paper we develop a logical framework for reasoning about communicating agents. More precisely, we deal with the specification and verification of agent interaction protocols in Dynamic Linear Time Temporal Logic (DLTL) [21], which extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic.

Temporal logics are widely used in the specification and verification of distributed systems and they have recently gained attention in the area of reasoning about actions and planning [2,6,12,16,30], as well as in the specification and verification of systems of communicating agents. The last topic will be dealt with in Section 7, where various approaches will be presented.

Our proposal for the specification and verification of interaction protocols in DLTL is based on the theory for reasoning about actions developed in [12] which allows reasoning about action effects and preconditions, reasoning with incomplete initial states, and dealing with postdiction, ramifications as well as with nondeterministic actions. We make use of temporal logic to provide a simple formalization of communicative actions in terms of their effects and preconditions, to specify interaction protocols, to constrain behavior of autonomous agents and formulate the properties of these agents. As proposed in [17], in our approach conversation protocols are modelled as a set of constraints on the sequences of semantically coherent ACL messages.

The verification of the compliance of an agent with an interaction protocol, the verification of protocol properties, and the verification that an agent is (is not) respecting its social facts (commitments and permissions) at runtime are all examples of tasks which can be formalized either as validity or as satisfiability problems in DLTL. Such verification tasks can be automated by making use of Büchi automata. In particular, we make use of the tableau-based algorithm presented in [10] for constructing a Büchi automaton from a DLTL formula. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automaton is, in the worst case, exponential in the size of the input formula.

In [13] we have presented a proposal for reasoning about communicating agents with the Product Version of DLTL, called DLTL$^{\otimes}$, which allows to describe the behavior of a network of sequential agents which coordinate their activities by performing common actions together. In the first part of this paper we focus on the non-product version of DLTL, which appears to be a simpler choice and also a more reasonable choice when a social approach is adopted, since the "social state" of the system is inherently global and shared by all of the agents. In Section 2 we describe the logic DLTL, while in Section 3 we show how this logic can be used to specify protocols, in particular by describing the structure of the action theory which is used to model communicative actions. In Section 4 we present some kinds of verification tasks which can be conveniently represented in DLTL. In the next section we show that some verification problems, such as the verification of the compliance of an agent with the protocol, require the use of the logic DLTL$^{\otimes}$, which is presented there. In Section 6 we give some hints on how proofs can be carried out in DLTL, and we conclude with a survey of related work and some conclusions.

2. Dynamic linear time temporal logic

In this section we briefly define the syntax and semantics of DLTL as introduced in [21]. In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL).

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, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leqslant$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by $\mathit{prf}(u)$ 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 as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1; \pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]]$ and $\tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup [[\pi^i]]$, where
    - $[[\pi^0]] = \{\varepsilon\}$,
    - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]]$ and $\tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$.

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

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

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 follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha\mathcal{U}^\pi\beta$ 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 $\varepsilon \leqslant \tau'' < \tau'$,[2] $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha\mathcal{U}^\pi\beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

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 $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \cdots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [21], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

3. Protocol specification

In the social approach [8,18,33,38] an interaction protocol is specified 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 proposal the meaning of communicative actions is fixed by the *protocol* which describes the effects of each action on the social state of the system. In this sense, our work is part of that "...thread of research which takes conversational sequences themselves to be semantically primitive so that the meaning of individual messages depends on the conversation and may slightly vary in the context of different agent conversations" [27]. 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*. The notion of *commitment* has a prominent role in the social approach and in the following we introduce two

[2] We define $\tau \leqslant \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leqslant \tau'$ and $\tau \neq \tau'$.

different kinds of commitments. Commitment policies, which rule the dynamic of commitments, can be described by *causal laws* which establish causal dependencies among fluents.

In the specification of the protocol, the social state of the system is viewed as a global one. However, as we will see when dealing with the problem of verifying the conformance of each agent to the protocol (given the program the agent executes), the single agents may only have a partial "local" view of the social state. In particular, each agent can only see the effects on the social state of the actions to which it participates (as sender or receiver of the message). 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 to the protocol. As we will see, the task of verifying the conformance of an agent to a protocol, will in fact require, in the general case, to move to the product version of the logic, DLTL$^{\otimes}$ [20], which allows modelling the behavior of a network of sequential agents that coordinate their activities by performing common actions together.

The specification of a protocol can be further constrained through the addition of suitable *temporal formulas*, and also the agents' programs can be modelled, by making use of complex actions (regular programs).

Below we recall those aspects of the temporal action theory developed in [12] that we use in the specification of interaction protocols. In particular, we define what we mean by action laws, precondition laws and causal laws. Then, we introduce the notion of commitments and conditional commitments together with the laws ruling their interplay and, finally, we treat, as a running example, the Contract Net protocol.

3.1. Action theories

The social state of the protocol, which describes the stage of execution of the protocol, is described by a set of atomic properties (*fluents*) which may hold or not in a state and may change value with the execution of communicative actions.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluent names*. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by *Lit* the set of all fluent literals.

A *domain description* D is defined as a tuple $(\Pi, \mathit{Frame}, \mathcal{C})$, where Π is a set of *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*. *Frame* provides a classification of fluents as frame fluents and nonframe fluents as we will define below.

The *action laws* in Π have the form:

$$\Box(\alpha \to [a]l),$$

with $a \in \Sigma$ and α an arbitrary non-temporal formula and l a fluent literal. The meaning is that executing action a in a state where precondition α holds causes the effect l to hold.

The *causal laws* in Π have the form:

$$\Box((\alpha \wedge \bigcirc\beta) \to \bigcirc l),$$

with $a \in \Sigma$, α, β arbitrary non-temporal formulas and l a fluent literal. The meaning is that if α holds in a state and β holds in the next state, then l also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents.

The *constraints* in $\mathcal{C}$ are, in general, arbitrary temporal formulas of DLTL. Constraints put restrictions on the possible correct behaviors of a protocol. The kind of constraints we will use in the specification of a protocol include the observations on the value of fluents in the *initial state* and the precondition laws. The initial state *Init* is a (possibly incomplete) set of fluent literals.

The *precondition laws* have the form:

$$\Box(\alpha \to [a]\bot),$$

with $a \in \Sigma$ and α an arbitrary non-temporal formula. The meaning is that the execution of an action a is not possible if α holds (i.e. there is no resulting state following the execution of a if α holds). Observe that, when there is no precondition law for an action, the action is executable in all states.

Frame is a set of pairs (f, a), where $f \in \mathcal{P}$ is a fluent and $a \in \Sigma$ is an action, meaning that f is a frame fluent for action a, that is, f is a fluent to which persistency applies when action a is executed. On the other hand, those fluents which are not frame with respect to a do non-persist and may change value in a nondeterministic way, when executing a.

As in [24,26] we call *frame* fluents those fluents to which the law of inertia applies. However, as in [11], we consider frame fluents as being dependent on the actions. Action laws and causal laws, which describe the immediate and indirect effects of actions, have a special role in action theories, as frame fluents only change values according to the immediate and indirect effects of actions described by the action laws and causal laws. All the frame fluents whose values are not changed by such actions are assumed to persist unaltered to the next state. When an action is executed, all the fluents which are non-frame with respect to that action may change value nondeterministically, as they are not subject to persistency.

The action language also contains *test actions*, which allow the choice among different behaviors to be controlled. As DLTL does not include test actions, we introduce them in the language as atomic actions in the same way as done in [12]. More precisely, we introduce an atomic action $\phi?$ for each proposition ϕ we want to test. The test action $\phi?$ is executable in any state in which ϕ holds and it has no effect on the state. Therefore, we introduce the following laws which rule the modality $[\phi?]$:

$$\Box(\neg\phi \to [\phi?]\bot)$$
$$\Box(\langle\phi?\rangle\top \to (l \leftrightarrow [\phi?]l)), \quad \text{for all fluent literals } l.$$

The first law is a precondition law, saying that action $\phi?$ is only executable in a state in which ϕ holds. The second law describes the effects of the action on the state: the execution of the action $\phi?$ leaves the state unchanged. We assume that, for all test actions occurring in a domain description, the corresponding action laws are implicitly added.

Test actions are specific actions and belong therefore to a particular action language (and the above laws belong to a particular action theory). They do not belong to the language of regular programs of the logic DLTL.

The action theory will be used for modelling communicative actions and the social behavior in multi-agent systems. In this framework for modelling protocols, we will define two special actions

$$\mathit{begin\_protocol}(s, \mathit{all}) \quad \text{and} \quad \mathit{end\_protocol}(s, \mathit{all})$$

which are supposed to start and to finish a *run* of the protocol. The first message is sent by the initiator s of the protocol to all other participating agents, while the second message is sent by any of the agents which may close the protocol to all the participants. For each protocol, we introduce a special fluent *Pn* (where *Pn* is the "protocol name") which is true during the whole execution of the protocol: *Pn* is made true by the action $\mathit{begin\_protocol}(s, r)$ and it is made false by the action $\mathit{end\_protocol}(s, r)$.

Note that protocol "runs" are always finite, while the logic DLTL is characterized by infinite models. Therefore the formulation of the protocol would have no model in DLTL. To take this into account, we assume that each domain description of a protocol will be suitably extended with an action *noop* which does nothing and which can be executed only after termination of the protocol, so as to allow a computation to go on forever after termination of the protocol. So every model will consist of a finite sequence of actions corresponding to a "run" of the protocol, followed by an infinite sequence of *noop* actions.

The action theory described above relies on a solution to the *frame problem* similar to the one described in [12]. In [12], 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 [31] in the context of the situation calculus. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. By completing the action laws and causal laws we formalize the fact that action laws and causal laws are the only laws which may change the value of frame fluents: the value of a frame fluent persists (stays unaltered) unless the change is caused by the execution of an action as an immediate effect (effect of the action laws) or an indirect effect (effect of the causal laws). We call $\mathit{Comp}(\Pi)$ the completion of a set of laws Π and, in the following subsection, we describe the details on the completion construction.

3.1.1. Completion of a set of action laws and causal laws

Let Π be a set of action laws and causal laws. In this paper, we have assumed that the consequents of all action laws and causal rules in Π are fluent literals rather than general formulas. Hence, Π will contain formulas of the form:

$$\Box(\alpha_i \to [a]f) \qquad \Box(\beta_j \to [a]\neg f),$$

as well as causal laws of the form

$$\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc l),$$

with $a \in \Sigma$, $\alpha, \beta, \alpha_i, \beta_j$ arbitrary (non-temporal) formulas and l a fluent literal.

Observe that, given the definition of the next operator $\bigcirc$ (namely, $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$), the causal law above can be written as follows:

$$\Box\left(\left(\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle\beta\right) \rightarrow \bigvee_{a\in\Sigma}\langle a\rangle l\right).$$

Observe also that, when a given action a is executed in a state (i.e. in a world of a model), this is the only action executed in it, since models of DLTL are linear (and each models describes a single run on the protocol). Hence, from the formula above it follows:

$$(*)\quad \Box((\alpha \wedge \langle a\rangle\beta) \rightarrow \langle a\rangle l).$$

Moreover, as the axioms $\langle a\rangle l \rightarrow [a]l$ and $\langle a\rangle\top \wedge [a]l \rightarrow \langle a\rangle l$ hold in DLTL (see [21]), from $(*)$ we can get:

$$(**)\quad \Box(\langle a\rangle\top \rightarrow ((\alpha \wedge [a]\beta) \rightarrow [a]l)).$$

This formula has a structure very similar to action laws. We call these formulas *normalized causal laws*.

We can now define our completion construction starting from the action laws in Π and from the normalized causal laws in Π. Both kinds of laws have the general form:

$$\Box(\langle a\rangle\top \rightarrow (\alpha_i \wedge [a]\gamma_i \rightarrow [a]f)) \qquad \Box(\langle a\rangle\top \rightarrow (\beta_j \wedge [a]\delta_j \rightarrow [a]\neg f)),$$

where $\alpha_i, \beta_j, \gamma_i, \delta_j$ are arbitrary (non-temporal) formulas and some of the conjuncts in the antecedents may be missing.

We define the completion of Π as the set of formulas $Comp(\Pi)$ containing, for all actions a and fluents f such that $(f, a) \in Frame$, the following axioms:

$$\Box\left(\langle a\rangle\top \rightarrow \left([a]f \leftrightarrow \bigvee_i(\alpha_i \wedge [a]\gamma_i) \vee (f \wedge \neg[a]\neg f)\right)\right), \tag{1}$$

$$\Box\left(\langle a\rangle\top \rightarrow \left([a]\neg f \leftrightarrow \bigvee_j(\beta_j \wedge [a]\delta_j) \vee (\neg f \wedge \neg[a]f)\right)\right). \tag{2}$$

Notice that, for each action a and fluent f which is nonframe with respect to a, i.e. $(f, a) \notin Frame$, axioms (1) and (2) above are not added in $Comp(\Pi)$. As in [31], these laws express that a fluent f (or its negation $\neg f$) holds either as a consequence of some action a or some causal law, or by persistency, since f (or $\neg f$) held in the state before the occurrence of a and $\neg f$ (or f) is not a result of a.

From the two axioms above we can derive the following axioms, which are similar, in their structure, to Reiter's successor state axioms [31]:

$$\Box\left(\langle a\rangle\top \rightarrow \left([a]f \leftrightarrow \left(\bigvee_i(\alpha_i \wedge [a]\gamma_i)\right) \vee \left(f \wedge \bigwedge_j(\neg\beta_j \vee \neg[a]\delta_j)\right)\right)\right),$$

$$\Box\left(\langle a\rangle\top \rightarrow \left([a]\neg f \leftrightarrow \left(\bigvee_j(\beta_j \wedge [a]\delta_j)\right) \vee \left(\neg f \wedge \bigwedge_i(\neg\alpha_i \vee \neg[a]\gamma_i)\right)\right)\right).$$

The construction above is similar to the one that we have introduced in [12], though there are some differences for the fact that here we have adopted a different formalization of causal laws by using the next operator. Also, here causal laws are more general than in [12], as they refer (in their antecedent) to the values of fluents in the current state as well as in the next state.

3.2. Commitments and permissions

As we have said, from the standpoint of the specification of the protocol, the social state of the system can be regarded as a "global" set of properties which describe the execution stage of the protocol. In our proposal, the social state contains domain specific fluents and special fluents representing commitments. The *domain specific fluents* describe observable facts concerning the execution of the protocol, that is, facts that could be observed by an external observer, who sees the history of the messages exchanged by the communicating agents (without knowing their internal behavior).

The use of social commitments has long been recognized as a "key notion" to allow coordination and communication in multi-agent systems [25]. Nevertheless the first attempts to use these notions to ground the communicative theories are more recent—and essentially motivated by requirements of verifiability [27]. Among the most significant proposals to use commitments in the specification of protocols (or more generally, in agent communication) are Singh [33], Guerin and Pitt [18], Colombetti [8]. An alternative notion to commitment, which has been proposed in [1] for the specification of agent protocols, is the notion of *expectation*.

In order to handle commitments and their behavior during runs of a protocol Pn, we introduce two special fluents. One represents *base-level commitments* and has the form $C(Pn, i, j, \alpha)$ meaning that agent i is committed to agent j to bring about α, where α is an arbitrary propositional formula not containing commitment fluents. The second commitment fluent models *conditional commitments* and has the form $CC(Pn, i, j, \beta, \alpha)$ meaning that in protocol Pn the agent i is committed to agent j to bring about α, if the condition β is brought about (where α is an arbitrary propositional formula not containing commitment fluents). As already mentioned above, Pn is a fluent which has to be verified during the whole execution of the protocol. The two kinds of base-level and conditional commitments we use here are essentially those introduced in [38]. Our present choice is different from the one in [18] and in [13], where agents are committed to execute an action rather than to achieve a condition. Let us point out that this present choice is more general than the one in our previous work [13]. If the agent is committed to perform an action (instead of being committed to produce some result) he can only do one thing: perform the action. When he is committed to bring about a condition (or result) α, he could achieve that in principle by more than one action or he could get some help by some other agent who could perform actions to produce α. On the other hand, the solution in [13] may be considered being more precise in case the commitment is considered to be fulfilled only in case agent i itself has brought about α and not if some other agent has brought about it.

To give an example, in the specification of the Contract Net protocol (named CN) we introduce the following commitments:

$$C(CN, M, P, acc\_rej),$$

meaning that the manager is committed to the participant to accept or to reject a proposal. As an example of a conditional commitment we have:

$$CC(CN, M, P, proposal, acc\_rej),$$

meaning that the manager is committed to the participant to accept or to reject a proposal, whenever the proposal has been made.

The idea is that each commitment has a life time which is included in each "run" of a protocol. A run is a finite sequence of actions consistent with the protocol (for the formal Definition 4 see in next Section 3.3). Commitments are created as effects of the execution of communicative actions in the protocol and they are "discharged" when they have been fulfilled.

A commitment $C(Pn, i, j, \alpha)$, created at a given world of a run, is regarded to be fulfilled in a run if there is a later world in the run in which α holds. As soon as commitment is fulfilled in a run, it is considered to be satisfied and no longer active: it can be discharged.

We introduce the following *causal laws* for automatically discharging fulfilled commitments:

$$\begin{aligned}
&\text{(i)}\quad \Box(\bigcirc\alpha \to \bigcirc\neg C(Pn, i, j, \alpha)),\\
&\text{(ii)}\quad \Box((CC(Pn, i, j, \beta, \alpha) \wedge \bigcirc\beta) \to \bigcirc C(Pn, i, j, \alpha)),\\
&\text{(iii)}\quad \Box((CC(Pn, i, j, \beta, \alpha) \wedge \bigcirc\beta) \to \bigcirc\neg CC(Pn, i, j, \beta, \alpha)).
\end{aligned}$$

A commitment to bring about α is considered fulfilled and is discharged (i) as soon as α holds. A conditional commitment $CC(Pn, i, j, \beta, \alpha)$ becomes a base-level commitment $C(Pn, i, j, \alpha)$ when β has been brought about (ii) and the conditional commitment is discharged (iii).

Observe that, it might not always be reasonable to discharge conditional commitments. A commitment $CC(Pn, i, j, \beta, \alpha)$ might be interpreted as meaning that agent i is committed to agent j to bring about α, any time the condition β is brought about. In this case the conditional commitment should not be discharged and should persist until the end of the protocol. For simplicity, in this paper we assume that also conditional commitments are discharged after their condition β is made true and the base-level commitment is created.

In our formalization we have not introduced explicit *create* and *discharge* operations on commitments, nor we have introduced operations for manipulating commitments like *cancel*, *release*, *delegate*, etc. (see [35]). This choice has been adopted for its simplicity (a commitment either is active or it is not), though it limits substantially the flexibility of the commitment based approach. The definition of these operations on commitments in the formalism would, however, be possible by introducing explicit (*create, discharge, cancel, release, delegate*, etc.) actions and by describing the effects of such actions on commitments by means of action laws. In such a case, the communicative actions in the protocol should have the effect of "calling" these primitive operations rather than directly creating the commitments.

We can express the condition that a commitment $C(Pn, i, j, \alpha)$ has to be fulfilled before the "run" of the protocol is finished by the following *fulfillment constraint*:

$$\Box(C(Pn, i, j, \alpha) \rightarrow Pn\mathcal{U}\alpha).$$

We will call Com_i the set of constraints of this kind for all commitments of agent i. Com_i states that agent i will fulfill all the commitments of which it is the debtor.

At each stage of the protocol only some of the messages can be sent by the participants, depending on the social state of the conversation. *Permissions* allow to determine which messages are allowed at a certain stage of the protocol. The permissions to execute communicative actions in each state are determined by social facts. We represent them by precondition laws. Preconditions on the execution of action a can be expressed as:

$$\Box(\alpha \rightarrow [a]\bot)$$

meaning that action a cannot be executed in a state if α holds in that state. If α holds in a state the execution of action a does not lead to a resulting state (there is no state where $\bot$ can be true).

We call $Perm_i$ (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent i is the sender.

3.3. Protocols and their runs

A protocol is specified by defining a domain description. Each communicative action is specified by defining its effects on the social state (by means of action laws) and by defining its executability conditions (by precondition laws). The domain description also includes the causal laws defining the commitment rules, the initial state of the protocol, as well as the fulfillment constraints for each commitment in the social state.

Definition 1. A protocol is specified by defining a domain description $D = (\Pi, Frame, \mathcal{C})$ as follows:

- Π is the set of the action and causal laws containing:
  - the laws describing the effects of each communicative actions on the social state;
  - the causal laws (i), (ii) and (iii) above defining the commitment rules.
- $\mathcal{C} = Init \wedge \bigwedge_i (Perm_i \wedge Com_i)$ is the conjunction of the constraints on the initial state of the protocol and the permissions $Perm_i$ and the commitments Com_i of all the agents i.
- $Frame = \{(f, a)\colon\ a \in \Sigma,\ f \in \mathcal{P}\}$.

In the following we assume that all fluents are frame and, in particular, we assume that all the commitment fluents are frame with respect to all actions. This condition will be removed in Section 5 when we will address the problem of verifying the conformance of an agent to the protocol in DLTL, by analyzing the behavior of agent i in isolation.

This requires to assume that the behavior of part of the system is unpredictable, which can be modelled by making use of nondeterministic fluents.

It is clear that, while the social state, as a whole, is global to all agents, each single agent can only be aware of the communicative actions to which it participates and of the changes produced by those actions. For this reason, we stipulate that each agent participating in the protocol has only a partial visibility of the social state, and in particular an agent can only see the commitments of which he is the debtor or the creditor as well as those fluents of the social state which are involved in the description of the actions to which he participates as a sender or receiver.

Definition 2. We define the set P_i of the *fluents visible to agent* i as follows:

- The commitments $C(Pn, i, j, \alpha)$, $CC(Pn, i, j, \beta, \alpha)$, $C(Pn, j, i, \alpha)$, $CC(Pn, j, i, \beta, \alpha)$ belong to P_i and all the fluents occurring within α and β belong to P_i.
- For each communicative action a of which i is the sender or the receiver, and for each action law
$$\Box(\alpha \to [a(i,j)]l) \quad (\text{or } \Box(\alpha \to [a(j,i)]l))$$
$|l|$ and the fluents in α belong to P_i.
- For each communicative action a of which i is the sender, for each precondition law
$$\Box(\alpha \to [a(i,j)]\bot)$$
the fluents in α belong to P_i.

Observe that the definition above also applies to communicative actions which are broadcast from one agent (sender) to a set of agents (receivers).

A protocol specification is *well-defined* when each fluent visible to agent i cannot be modified by other agents without i being aware of the modification.

Definition 3. We say that a domain description D specifying a protocol is *well defined* if, for all agents i and for all fluents $f \in P_i$, f does not occur as a positive or negative effect of any communicative action $a(k,l)$ of which i is neither the sender nor the receiver.

It is easy to see that, due to the kinds of causal laws we have introduced (namely (i), (ii) and (iii)), Definition 3 guarantees that, if the specification D of a protocol is well-defined, the value of a fluent $f \in P_i$ can only be changed as an immediate or indirect effect of the communicative actions of which agent i is sender or receiver.

In particular, by the fact that a commitment $C(Pn, i, j, \ldots) \in P_i \cap P_j$ the specification of a protocol is not well-defined in the case the value of the commitment fluent is changed by a communicative action which does not involves *both* i and j.

Observe that the condition of well-definedness above excludes that a commitment $C(l, k, \ldots)$ may occur within another commitment $C(i, j, \beta, \alpha)$ involving different agents, if the protocol is well-defined. In fact, a commitment $C(Pn, i, j, C(Pn, k, l, \alpha_1), \alpha_2)$ where $l \neq i$, $l \neq j$, $k \neq i$, $k \neq j$ is not admitted in a well-defined domain description. By the first item in Definition 2, we get that $C(k, l, \alpha_1) \in P_k \cap P_l$. But, as the commitment occurs within the commitment $C(Pn, i, j, \beta, \alpha_2)$ in β, then $C(Pn, k, l, \alpha_1) \in P_i \cap P_j$. Assume the value of the commitment $C(k, l, \alpha_1)$ is changed by executing a communication action $a(k,l)$ between k and l, then, as $C(Pn, k, l, \alpha_1) \in P_i$, the domain description is not well-defined. Similarly, if $C(k, l, \alpha_1)$ is changed by executing a communication action $a(i, j)$: the domain description is not well-defined as $C(Pn, k, l, \alpha_1) \in P_k$.

Given a domain description D, we denote by $Comp(D)$, the completed domain description, the set of formulas:
$$\left(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)\right) \wedge \langle begin\_protocol(s, all)\rangle\top$$
where agent s is the initiator of the protocol.

Definition 4. Given the specification of a protocol by a domain description D, the runs of the system according the protocol are exactly the models of $Comp(D)$.

The last conjunct in the definition of $Comp(D)$ is introduced to force each model $M = (\sigma, V)$ satisfying $Comp(D)$ to have action $begin\_protocol(s, all)$ as the first action in σ.

Given the definition above, in all protocol runs all permissions and commitments of all agents are fulfilled. This is needed as we want include among the runs of the protocol, only those models in which all agent respect their permissions and commitments. However, it is clear that, if Com_j were not included in $Comp(D)$ for some agent j, the models satisfying $Comp(D)$ might contain commitments which have not been fulfilled by j. They are runs in which agent j may not be compliant with the protocol.

3.4. The Contract Net protocol

As a running example we will use the Contract Net protocol [7].

Example 5. The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task.

We assume there are $N + 1$ agents: the manager M and N participants $1, \ldots, N$. A communicative action *act* is represented by the notation $act(s, r)$, where s is the sender and r is the receiver. With $act(M, all)$ we mean that the communicative action *act* is broadcast by the manager to all participant.

The communicative actions are the following ones (where $i = 1, \ldots, N$ ranges over the participants $1, \ldots, N$): $begin\_protocol(M, all)$ (the manager announces to all participants the beginning of the protocol), $cfp(M, all)$ (the manager issues—broadcasts—a task announcement), $accept(M, i)$ and $reject(M, i)$ whose sender is the manager, $refuse(i, M)$ and $propose(i, M)$ whose sender is the participant i, $inform\_done(i, M)$ by which agent i informs the manager that the task has been executed, and $end\_protocol(M, all)$ by which the manager broadcasts the completion of the protocol.

The social state contains the following domain specific fluents: *CN* (which is true during the execution of the protocol), *task* (whose value is true after the task has been announced), $replied(i)$ (the participant i has replied), $proposal(i)$ (the participant i has sent a proposal), $acc\_rej(i)$ (the manager has sent an accept or reject message to the participant i) $accepted(i)$ (the manager has accepted the proposal of participant i) and $done(i)$ (participant i has performed the task). Such fluents describe observable facts concerning the execution of the protocol.

Among the fluents, we introduce the following *conditional commitments* which have to be satisfied during the Contract Net protocol CN:

$CC(CN, i, M, task, replied(i))$ if a task has been announced, the participant i has to reply
$CC(CN, M, i, proposal(i), acc\_rej(i))$ if i has made a proposal the manager has to accept or to reject it
$CC(CN, i, M, accepted(i), done(i))$ if the participant's i proposal has been accepted he is committed to execute is (to make true $done(i)$)

and the corresponding *base-level commitments*

$C(CN, i, M, replied(i))$ participant i must reply to a proposal
$C(CN, M, i, acc\_rej(i))$ the manager has to bring about a response (accept or reject)
$C(CN, i, M, done(i))$ the participant i is committed to the manager to perform the task (to produce $done(i)$).

The effects of communicative actions are described by the following *action laws* (where $i = 1, \ldots, N$ ranges over the participants $1, \ldots, N$):

(A1) $\Box[begin\_protocol(M, all)]CN$
(A2) $\Box[cfp(M, all)]task$
(A3) $\Box[cfp(M, all)]CC(CN, M, i, proposal(i), acc\_rej(i))$
(A4) $\Box[accept(M, i)]acc\_rej(i)$

(A5) $\Box[accept(M,i)]accepted(i)$
(A6) $\Box[reject(M,i)]acc\_rej(i)$
(A7) $\Box[refuse(i,M)]replied(i)$
(A8) $\Box[propose(i,M)]replied(i)$
(A9) $\Box[propose(i,M)]proposal(i)$
(A10) $\Box[propose(i,M)]CC(CN,i,M,accepted(i),done(i))$
(A11) $\Box[inform\_done(i,M)]done(i)$
(A12) $\Box[end\_protocol(M,all)]\neg CN.$

Action *begin_protocol* initiates the protocol by putting *CN* to true, while *end_protocol* terminates it by putting *CN* to false. Since there is no other action making *CN* false, *CN* remains true, i.e. *CN* persists during the whole run of this protocol according to the frame laws in the completion, as described in Section 3.1 (see rule 1). Moreover, *CN* remains false after the end of the protocol. The laws for action *cfp* add to the social state the information that a call for proposal has been done for a task, and that, if the manager receives a proposal, it is committed to accept or reject it. The laws (A4) and (A6) say that when the manager accepts (rejects) the proposal by participant i, this is recorded in the social state by making the fluent $acc\_rej(i)$ true. Furthermore, if the proposal is accepted, the fluent $accepted(i)$ becomes true due to law (A5). The laws (A7) and (A8) say that when the participant i sends a refusal or a proposal to the manager, this is recorded in the social state by making the fluent $replied(i)$ true. If the reply is a proposal, the fluent $proposal(i)$ becomes true, and the agent is committed to perform the task if the manager accepts the proposal (laws (A9) and (A10)). By action $inform\_done(i,M)$ agent i informs the manager M that the task has been performed.

The permissions to execute communicative actions in each state can be defined by the following precondition laws:

(A13) $\Box(CN \to [begin\_protocol(M,all)]\bot)$
(A14) $\Box(\neg CN \vee task \to [cfp(M,all)]\bot)$
(A15) $\Box(\neg CN \vee \neg proposal(i) \vee acc\_rej(i) \to [accept(M,i)]\bot)$
(A16) $\Box(accepted(i) \to [accept(M,j)]\bot)$ for all $i \neq j$
(A17) $\Box(\neg CN \vee \neg proposal(i) \vee acc\_rej(i) \to [reject(M,i)]\bot)$
(A18) $\Box(\neg CN \vee \neg task \vee replied(i) \to [refuse(i,M)]\bot)$
(A19) $\Box(\neg CN \vee \neg task \vee replied(i) \to [propose(i,M)]\bot)$
(A20) $\Box(\neg CN \vee \neg accepted(i) \vee done(i) \to [inform\_done(i,M)]\bot)$
(A21) $\Box\neg CN \vee \neg task \to [end\_protocol(M,all)]\bot.$

All actions, except for $begin\_protocol(M,all)$, can be executed only after the protocol has started (A13). Action $begin\_protocol(M,all)$ can be executed if *CN* is false, i.e. the protocol has not yet started. The law for *cfp* says that the manager cannot issue a call for proposal if a task has already been announced (A14). The precondition laws for actions $accept(M,i)$ and $reject(M,i)$ ((A15) and (A17)) say that action *accept* can be executed only if a proposal has been issued and the manager has not already replied. Moreover, we stipulate that the manager cannot accept more that one proposal (A16). The precondition laws for actions $propose(i,M)$ and $refuse(i,M)$ ((A19) and (A18)) say that a proposal can only be done if a task has already been announced and the participant has not already replied. The action $inform\_done(i,M)$ can be executed by agent i only after his proposal has been accepted (law (A20)). Action $end\_protocol(M,all)$ can be executed by the manager at any moment during the execution of the protocol after the task has been announced. Of course in all runs satisfying the protocol, all commitments have to be fulfilled before the action *end_protocol* is executed by the manager.

To express that we want the manager to accept or reject a proposal only after all participants have replied, we can add the following precondition laws for $accept(M,i)$ and $reject(M,i)$:

(A22) $\Box(\bigvee_{j=1,N} \neg replied(j) \to [accept(M,i)]\bot)$
(A23) $\Box(\bigvee_{j=1,N} \neg replied(j) \to [reject(M,i)]\bot).$

Assume now that we want all participants to be committed to reply to the task announcement. We can express this by putting the following conditional commitment in the initial state $Init_{CN}$ of the Contract Net of the protocol:

$CC(CN, i, M, task, replied(i))$, for each i. All domain specific fluents and all other commitments will be false in the initial state.

The domain description $D_{CN} = (\Pi_{CN}, Frame_{CN}, \mathcal{C}_{CN})$ for the Contract Net protocol (with N participants) can therefore be defined as follows:

- Π_{CN} is the set of the action laws (A1)–(A12) and all instances of the causal laws (i), (ii) and (iii) above (Section 3.2);
- $\mathcal{C}_{CN} = Init_{CN} \wedge \bigwedge_i (Perm_i \wedge Com_i)$;
- $Frame_{CN} = \{(f, a)\colon\ a \in \Sigma,\ f \in \mathcal{P}\}$,

where $\bigwedge_i Perm_i$ is the set of precondition laws (A13)–(A23) and $\bigwedge_i Com_i$ contains the instances of the *fulfillment constraint* for each of the commitments introduced in the domain description.

It is easy to see that the specification of the Contract Net protocol given above is well-defined.

All the possible runs of the protocol can be obtained as finite substrings of linear models of $Comp(D_{CN})$. In these protocol runs all permissions are satisfied and all commitments are fulfilled.

4. Protocol verification in DLTL

Given the DLTL specification of a protocol by a domain description as defined above, we can now describe the different kinds of verification problems which can be addressed. In particular, in this section we consider verification problems which do not require the availability of an agent program, namely, the verification of agents compliance at runtime and the verification of properties of the protocol.

4.1. *Verifying agents compliance at runtime*

Given an execution history describing the interactions of the agents, we want to verify the compliance of that execution to the protocol. This verification is carried out at runtime.

This kind of verification does not require to be aware of the internal behavior of the communicating agents. We only know the history of the communications among the agents (that is the sequence of communicative messages they have exchanged) and we have to check the conformance of this execution with the protocol, that is, we have to check that history is a prefix of a run of the protocol.

We are given a history $\tau = a_1, \ldots, a_n$ of the communicative actions executed by the agents, and we want to verify that the history τ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the protocol. This problem can be formalized by requiring that the formula

$$\Bigl(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)\Bigr) \wedge \langle a_1; a_2; \ldots; a_n\rangle \top$$

(where i ranges on all the agents involved in the protocol) is satisfiable. In fact, the above formula is satisfiable if it is possible to find a run of the protocol starting with the action sequence $a_1, \ldots, a_n$. On the one hand, this means that when the actions $a_1, \ldots, a_n$ in the sequence are executed their preconditions hold. On the other hand, this does not mean that in the stretch of behavior $a_1 \ldots a_n$ all the created commitments have already been fulfilled, but only that it is possible to continue the conversation so that they will eventually be fulfilled.

4.2. *Verifying protocol properties*

A second problem is that of proving a property φ of a protocol. This can be formulated as the validity of the formula

$$\Bigl(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)\Bigr) \to \varphi, \tag{3}$$

according to which all the runs of the protocol satisfy the (temporal) property ϕ. As an example of property we want to check, we consider the property of termination of the protocol. After the manager has announced a task, the protocol will eventually arrive to completion. This property can be formalized by the temporal formula:

$$\varphi = \Box[cfp(M, all)]\Diamond\neg CN$$

meaning that, always, after a call for proposal has been issued by the manager, the protocol will eventually reach a state in which the proposition *CN* is false, i.e. the protocol is finished, for all possible runs of the protocol.

As a further example, let us consider a version of the Contract Net protocol with a single participant P. In such a case, given that the protocol is quite rigid, the correct behaviors of the protocol could be described by the following regular program π:

$$\begin{aligned}
&start\_protocol(M, P); cfp(M, P);\\
&\qquad\qquad (refuse(P, M) +\\
&\qquad\qquad\ (propose(P, M);\ (reject(M, P) +\\
&\qquad\qquad\qquad\qquad\qquad\qquad accept(M, P);\ inform\_done(P, M))));\\
&end\_protocol(M, P).
\end{aligned}$$

We could wonder whether the specification of the protocol given in Section 3.4 in terms of commitments is equivalent (when restricted to a single participant P) to the rigid specification π above.

We can verify that each runs of the Contract Net protocol as defined in Section 3.4 is an execution of the program π by proving the validity of the formula:

$$\left(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)\right) \rightarrow \langle\pi\rangle\top. \tag{4}$$

Moreover, we can verify that all the behaviors admitted by π correspond to runs of the protocol (as defined in Section 3.4), that is, they satisfy the permissions and commitments:

$$\left(Comp(\Pi) \wedge Init \wedge \langle\pi\rangle\top\right) \rightarrow \left(\bigwedge_i (Perm_i \wedge Com_i)\right). \tag{5}$$

5. Verifying the compliance of an agent with the protocol at compile-time

When the program executed by an agent is given (or, at least, its logical specification is given), we are faced with the problem of verifying if the agent is compliant with the protocol, that is, to verify if the agent's program respects the protocol. The logical specification of the protocol is given as described in the previous sections. Solving this problem requires: first to provide an *abstract* specification of the behavior (program) of the agent; and, second, to check that all the executions of the agent program satisfy the specification of the protocol, assuming that the other agents are compliant with the protocol. This requires comparison of the executions of the protocol, which contain the communicative actions executed by all the agents, and the executions of the single agent we want to check for conformance, which contain its communicative actions as well as its internal actions.

In [14] we have shown that this verification problems can be represented, in some cases, in DLTL without requiring the product version. This is true in particular for protocols involving two agents, where all fluents and all actions of the social state are shared by both agents. However, in the general case, addressing this problem requires to move to the Product Version of DLTL, which allows description of the behavior of a network of sequential agents that coordinate their activities by performing common actions together. Such logic allows the actions of the different agents to be interleaved and the state of the system described as partitioned into local states, representing the local views of the social state that are visible to each single agent. In the product version, we can focus separately on those actions of a given run pertaining to a single agent, while abstracting away from all other actions in the run. So, for instance, while the protocol specification puts requirements on the communicative actions in the run, the specification of the agent to be verified only imposes requirements on the sequences of his communicative and internal actions.

In the following we first introduce the Product Version of DLTL, then we provide an abstract specification of the agent program and, finally, we describe the problem of verifying the conformance of the agent to the protocol.

5.1. The product version of DLTL

Let us now recall the definition of DLTL$^{\otimes}$ from [20]. 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

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

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 $\varepsilon \leqslant \tau'' \uparrow i < \tau' \uparrow i$, 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.

In the product version of DLTL 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.

5.2. Describing the program of an agent

For the purpose of verifying if behavior of agent i is compliant with the protocol, we need to introduce two locations, one location i for the agent i to be verified and one location P for the protocol.[3] Hence, we have $Loc = \{P, i\}$.

For the location i, the agent to be verified, we need to specify both the internal actions of the agent (which are private) and the communicative actions of the agent (which are shared with the protocol P).

[3] As a difference with [13] here we do not introduce a distinct location for each different agent participating in the protocol. This simplifies substantially the specification of the protocol, as we do not need to project the protocol on the different agents, and to repeat the specification of the communicative actions for each of the participating agent.

In our formalism we can specify the (abstract) behavior of an agent by making use of complex actions (regular programs). Consider for instance the following program π_i for a participant i:

$$\begin{aligned}
&[\neg end_i?; ((cfp(M, all); eval\_task; (\neg ok_i?; refuse(i, M) + ok_i?; propose(i, M))) + \\
&\qquad (reject(M, i)) + \\
&\qquad (accept(M, i); \mathrm{do}\_task; inform\_done(i, M)) + \\
&\qquad (end\_protocol(M, all); exit))]^*; end_i?
\end{aligned}$$

The participant cycles and reacts to the messages that he received by the manager: for instance, if the manager has issued a call for proposal, the participant can either refuse it or make a proposal according to its evaluation of the task; if the manager has rejected the proposal, the participant does nothing; if the manager has accepted the proposal, the participant performs the task; if the manager concludes the protocol by executing the action *end_protocol*, the agent i executes the *exit* action, which concludes the execution of the cycle.

The state of the participant i contains the following local fluents: end_i, which is initially false and is made true by action *exit*, and ok_i which says if the agent must make a bid or not. The local actions are *eval_task*, which evaluates the task and sets the fluent ok_i to true or false, *do_task* and *exit*. Furthermore, $end_i?$ and $ok_i?$ are test actions.

Agent i has a local view of the social state. It can see the social fluents which are used and modified by the communicative actions to which it participates. Therefore, the state of participant i contains, in addition to the local fluents mentioned above, also the social fluents: CN_i, $task_i$, $replied_i(i)$, $proposal_i(i)$, $acc\_rej_i(i)$, $accepted_i(i)$ and $done_i(i)$, together with all the commitments involving agent i, which have been introduced in Section 3.4. We have added the index i to each fluent to make it clear that these are the local views of the fluents at location i.

We define the set Σ_i of the actions at location i as the set containing the local actions mentioned above, and the following communicative actions (of which i is sender or receiver): $begin\_protocol(M, all)$, $cfp(M, all)$, $accept(M, i)$, $reject(M, i)$, $refuse(i, M)$, $propose(i, M)$, $inform\_done(i, M)$ and $end\_protocol(M, all)$.

The program of the participant i can then be specified by a domain description $Prog_i = (\Pi_i, \mathcal{C}_i, Frame_i)$, where Π_i is a set of action laws describing the effects of the private and communicative actions of the participant i. For instance, the action *exit* sets the proposition end_i to true:

$$\Box[exit]_i end_i.$$

The action *eval_task* has the nondeterministic effect of assigning a value true or false to the fluent ok_i. In the action theory, this is modeled by stating that the fluent ok_i is non-frame with respect to action *eval_task*, that is:

$$Frame_i = \{(f, a)\colon a \in \Sigma, f \in \mathcal{P}\} \setminus \{(eval\_task, ok_i)\}.$$

Π_i also contains the action laws (A1)–(A12) for the communicative actions (where the parameter i occurring in the action laws is the same as the location i). For instance, (A4) becomes

$$\Box[accept(M, i)]_i acc\_rej_i(i).$$

The set of constraints $\mathcal{C}_i$ contains $Init_i$ which provides the initial values for the local fluents ($\neg done$, $\neg ok$) and for the social fluents of agent i (which must be the same as in the protocol initial state *Init*). Moreover, $\mathcal{C}_i$ contains the formula $\langle \pi_i \rangle \top$ stating that the program of the participant is executable in the initial state.

Concerning the location P of the protocol, the local propositions at P are all the social fluent used in the definition of the protocol. Moreover, Σ_P is the set of actions containing all the communicative actions in the protocol specification (as defined in Section 3). We can therefore associate to the location P the domain description D which has been introduced in Section 3.4 for specifying the Contract Net protocol. Strictly speaking, we would need to index all the modalities occurring in the domain description D with the label P of the protocol location and the same for the fluents at location P. However, we will omit the index P with no risk of confusion, as there are just two locations.

5.3. Verifying the compliance of an agent with the protocol

We want now to prove that the participant is compliant with the protocol, i.e. that all executions of program π_i satisfy the specification of the protocol. This property cannot be proved by considering only the program π_i. In fact, it is easy to see that the correctness of the property depends on the behavior of the manager. For instance, if the manager begins with an *accept* action, the participant will execute the sequence of actions *accept*; *do_task*; *exit* and stop, which is not a correct execution of the protocol. Thus we have to take into account also the behavior of the manager. Since we do not know its internal behavior, we will assume that the manager respects its public behavior, i.e. that it respects its permissions and commitments in the protocol specification.

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C}, Frame)$ be the domain description describing the protocol, as defined in Section 3.4, and let $Prog_i = (\Pi_i, \mathcal{C}_i, Frame_i)$ be the domain description for the behavior of the participant i. The formula

$$Comp(\Pi) \wedge Init \wedge \bigwedge_{j \neq i} (Perm_j \wedge Com_j) \wedge Comp(\Pi_i) \wedge Init_i \wedge \langle \pi_i \rangle_i \top \rightarrow (Perm_i \wedge Com_i)$$

is valid if in all the behaviors of the system, in which the participant executes its program π_i and all other agents (whose internal program is unknown) respect the protocol specification (in particular, their permissions and commitments), the permissions and commitment of the participant are also satisfied.

The left-hand side of the formula puts constraint on the run: the effects of the communicative actions in the run are defined by the action laws and causal law in $(Comp(\Pi)$; the initial values of the state of the protocol are given in *Init*; all agents except from i are assumed to satisfy their permissions and commitments $(\bigwedge_{j \neq i}(Perm_j \wedge Com_j))$; the effects of the communicative and internal actions of agent i are defined by the action laws and causal laws in $(Comp(\Pi_i)$; the initial values of its local fluents are given by $Init_i$; and, finally, the formula $\langle \pi_i \rangle \top$ requires the run to contain a finite execution of the program π_i of agent i.

Observe that the last point means that, given a run σ, there is a prefix of the run τ such that its projection $\tau \uparrow i$ on the actions of agent i is an execution of the program π_i. Other actions of the protocol can be contained in τ, but we ignore them when we verify $\langle \pi_i \rangle_i \top$.

6. Proofs and model checking in DLTL

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. We recall that a *Büchi automaton* has the same structure as a traditional finite state automaton, with the difference that it accepts infinite words. More precisely a Büchi automaton over an alphabet Σ is a tuple $\mathcal{B} = (Q, \rightarrow, Q_{in}, F)$ where:

- Q is a finite non-empty set of states;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation;
- $Q_{in} \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^\omega$. Then a run of $\mathcal{B}$ over σ is a map $\rho : prf(\sigma) \rightarrow Q$ such that:

- $\rho(\varepsilon) \in Q_{in}$;
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in prf(\sigma)$.

The run ρ is *accepting* iff $inf(\rho) \cap F \neq \emptyset$, where $inf(\rho) \subseteq Q$ is given by $q \in inf(\rho)$ iff $\rho(\tau) = q$ for infinitely many $\tau \in prf(\sigma)$.

As described in [21], the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in \text{DLTL}(\Sigma)$ 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. Actually a stronger property holds, since there is a one to one correspondence between models of the formula and infinite words accepted by $\mathcal{B}_\alpha$. The size of the automaton can be exponential in the size of α, while emptiness can be detected in a time linear in the size of the automaton.

The validity of a formula α can be verified by constructing the Büchi automaton $\mathcal{B}_{\neg\alpha}$ for $\neg\alpha$: if the language accepted by $\mathcal{B}_{\neg\alpha}$ is empty, then α is valid, whereas any infinite word accepted by $\mathcal{B}_{\neg\alpha}$ provides a counterexample to the validity of α.

For instance, let CN be the completed domain description of the Contract Net protocol. Then every infinite word accepted by $\mathcal{B}_{CN}$ corresponds to a possible run of the protocol. To prove a property φ of the protocol, we can build the automaton $\mathcal{B}_{\neg\varphi}$ and check that the language accepted by the product of $\mathcal{B}_{CN}$ and $\mathcal{B}_{\neg\varphi}$ is empty.

The construction given in [21] is highly inefficient since it requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state. A more efficient approach for constructing a Büchi automaton from a DLTL formula making use of a tableau-based algorithm has been proposed in [10]. The construction of the states of the automaton is similar to the standard construction for LTL [9], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. Given a formula φ, the algorithm builds a graph $\mathcal{G}(\varphi)$ whose nodes are labelled by sets of formulas. States and transitions of the Büchi automaton correspond to nodes and arcs of the graph. The algorithm makes use of an auxiliary tableau-based function which expands the set of formulas at each node. As for LTL, the number of states of the automaton is, in the worst case, exponential in the size if the input formula, but in practice it is much smaller.

LTL is widely used to prove properties of (possibly concurrent) programs by means of *model checking* techniques. The property is represented as an LTL formula φ, whereas the program generates a Kripke structure (the model), which directly corresponds to a Büchi automaton where all the states are accepting, and which describes all possible computations of the program. The property can be proved as before by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

Standard model checking techniques cannot be immediately applied to our approach, because protocols are formulated as sets of properties rather than as programs. Furthermore, in principle, with DLTL we do not need to use model checking, because programs and domain descriptions can be represented in the logic itself, as we have shown in the previous section. However representing everything as a logical formula can be rather inefficient from a computational point of view. In particular all formulas of the domain description are universally quantified, and this means that our algorithm will have to propagate them from each state to the next one, and to expand them with the tableau procedure at each step.

Therefore we have adapted model checking to the proof of the formulas given in the previous section, by deriving the model from the domain theory in such a way that the model describes all possible runs allowed by the domain theory. In particular, we can obtain from the domain description a function $\mathit{next\_state}_a(S)$, for each action a, for transforming a state in the next one, and then build the model (an automaton) by repeatedly applying these functions starting from the initial state. We can then proceed as usual to prove a property φ by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

Note that, although this automaton has an exponential number of states, we can build it step-by-step by following the construction of the algorithm on-the-fly. The state of the product automaton will consist of two parts $\langle S_1, S_2\rangle$, where S_1 is a set of fluents representing a state of the model, and S_2 is a set of formulas. The initial state will be $\langle I, \neg\varphi\rangle$, where I is the initial set of fluents. A successor state through a transition a will be obtained as $\langle \mathit{next\_state}_a(S_1), S_2'\rangle$ where S_2' is derived from S_2 by the on-the-fly algorithm. If the two parts of a state are inconsistent, the state is discarded.

The incremental nature of the algorithm is especially helpful in the verification of agent compliance at runtime. In fact, an external observer will obtain step by step the sequence of communicative actions executed by the agents, and use it to constrain the incremental construction of the automaton so that at each step the automaton contains only partial runs corresponding to the observed sequence.

This construction developed for DLTL can be easily extended to deal with DLTL$^{\otimes}$.

An alternative way for applying this approach in practice, is to make use of existing model checking tools. In particular, by translating DLTL formulas into LTL formulas, it would be possible to use LTL-based model checkers such as for instance SPIN [22]. Although in general DLTL is more expressive than LTL, many protocol properties, such as for instance fulfillment of commitments, can be easily expressed in LTL.

We have done some experiments with the model checker SPIN on proving properties of protocols expressed according to the approach presented in this paper. The model is obtained as suggested above by formulating the domain description as a PROMELA program, which describes all possible runs allowed by the domain theory. Properties and constraints are expressed as LTL formulas. In the case of verification of compliance of an agent implementation with the protocol, we have used different PROMELA processes for representing the agent and the protocol. The representation of the agent is derived from its regular program.

7. Related work

The issue of developing semantics for agent communication languages has been examined in [36], by considering the problem of giving a *verifiable* semantics, i.e. a semantics *grounded* on the computational models. The author gives an abstract formal framework, in which he defines what it means for an agent program, in sending a message while in some particular state, to be respecting the semantics of the communicative action. The author also points out the difficulties of carrying out the verification of this property when the semantics are given in terms of mental states, since we do not understand how such states can be systematically attributed to programs. The paper deals only with single communicative actions, and does not consider communication protocols.

Guerin and Pitt [18,19] define an agent communication framework which gives agent communication a grounded declarative semantics. The framework allows to accommodate communication languages based on agents' mental states as well as those based on social states. Several different types of verification are possible depending on the information available and whether the verification is done at design time or at run time. In particular they point out the following types of verification which are useful in an open system:

- verify that an agent always satisfies its social facts;
- prove a property of a protocol;
- determine if an agent is not respecting its social facts at run time.

The framework introduces different languages: a language for agent programming, a language for specifying agent communication and social facts, and a language for expressing temporal properties. Our approach instead provides a unified framework for describing multi-agent systems using DLTL. 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 temporal operators.

In [32] Singh advocates the need to define the semantics of ACLs in terms of social notions. In [33] he proposed a social semantics for ACLs, which uses a branching time logic. An approach for testing whether the behavior of an agent complies with a commitment protocol is presented in [35], where the protocol is specified in the temporal logic CTL as a set of metacommitments (where the condition committed to is a temporal formula possibly involving base-level commitments and commitment operations) as well as a mapping between messages and commitments. Based on this specification together with the notion of "potential causality" among the messages occurring in the multi-agent system, a concrete model is built describing all the possible behaviors of the system which are compliant with the protocol. A model checking procedure can then be used at runtime for verifying if a given execution respects the protocol. The paper also shows how the verification can be done based on local information (concerning a single agent or a subset of agents). The paper does not address the problem of statically verifying the compliance of an agent implementation with the protocol.

Fornara and Colombetti in [8] propose a method for the definition of interaction protocols starting from a semantics of communicative actions based on social commitments. Commitments are represented as objects, and their content and conditions may consist of temporal expressions. Protocols are defined by means of interaction diagrams whose nodes represent states, and whose edges correspond to the execution of communicative actions. The authors define a number of soundness conditions which must be satisfied by the protocol. In particular they require that all communicative actions must have their preconditions satisfied and that all commitments must be cancelled, fulfilled or violated in the last state of the protocol. The soundness conditions can be used to verify whether a protocol is "reasonable", although no formal framework for carrying out the verification is provided in the paper.

Yolum and Singh [38] developed a social approach to protocol specification and execution, where the content of the communicative actions is captured through agents' commitments to one another. Commitments are formalized in a variant of event calculus, and the evolution of commitments is described through the agents' actions. Using these rules enables agents to reason about their actions. By using an event calculus planner it is possible to determine execution paths that respect the protocol specification, i.e. where there are no pending base-level commitments. In this paper we adopt a similar approach to the specification of protocols, although we formalize it in the logic DLTL. In our approach a planning problem can be formulated as satisfiability of the completed domain description of the protocol. In fact, by Definition 4, a model of the completed domain description corresponds to a correct run of the protocol. However our approach based on DLTL is more general, since, as we have shown, we can deal with other kinds of protocol verification, including the verification of the compliance of an agent to the protocol.

Alberti et al. [1] address the problem of verifying agents' compliance with a protocol at runtime. Protocols are specified in a logic-based formalism based on Social Integrity Constraints, which constrain the agents' observable behavior. Their approach is based on the concepts of event (to represent an agent's actual behavior), and expectations (to express the desired behavior). Social Integrity Constraints express which expectations are generated as consequence of events. The paper present a system that, during the evolution of a society of agents, verifies the compliance of the agents' behavior to the protocol, by checking fulfillment or violation of expectations.

As we have pointed out in the previous section, the verification of protocols expressed in DLTL can be carried out by means of model checking techniques. Various authors have proposed the use of model checking for the verification of multi-agent systems, but, while in this paper we follow a social approach to the specification and verification of systems of communicating agents, most of them have adopted a mentalistic approach. The goal of [3] 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. The model checking algorithm keeps the two aspects separated: when considering the temporal evolution of an agent, BDI atoms are considered as atomic proposition; if an agent a_1 has a BDI attitude about another agent a_2, this is modeled as the fact that a_1 has access to a representation of a_2 as a process, in which it will verify the truth value of BDI atoms about a_2.

In [23,37] 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. Instead [4,5] deals with programs written in AgentSpeak(F), a variation of the BDI logic programming language AgentSpeak(L). Properties of MABLE or AgentSpeak programs can be verified by means of the SPIN model checker, by translating BDI formulas into the LTL form used by SPIN. In the case of AgentSpeak, BDI properties are extracted from the data structures used to implement AgentSpeak in SPIN.

A different framework for verifying temporal and epistemic properties of multi-agent systems by means of model checking techniques is presented by Penczek and Lomuscio [29]. Here multi-agent systems are formulated in the logic language CTLK, which adds to the temporal logic CTL an epistemic operator to model knowledge, using *interpreted systems* as underlying semantics.

8. Conclusions

In this paper we have proposed an approach for the specification and verification of interaction protocols in a temporal logic. We have shown that DLTL and its product version DLTL$^\otimes$ are a suitable formalisms for the specification of a system of communicating agents. Our approach provides a unified framework for describing different aspects of multi-agent systems. Programs are expressed as regular expressions, (communicative) actions can be specified by means of action and precondition laws, social facts can be specified by means of commitments whose dynamics is ruled by causal laws, and temporal properties can be expressed by means of the *until* operator. We have addressed several kinds of verification problems, including the verification of agents compliance at runtime, the verification of protocol properties and the verification that and agent (whose specification is given) is compliant with a protocol.

Such verification problems can be formalized as satisfiability and validity problems in DLTL or DLTL$^\otimes$ and they can be solved by developing automata-based model checking techniques.

A preliminary implementation of a model checker based on the algorithm in [10] is being tested in the verification of the properties of various protocols. This implementation has been useful to prove properties of small protocols, but larger protocols might require the adoption of optimized techniques, similar to the ones adopted by the state of the art model checkers for LTL.

Our proposal is based on a social approach to agent communication, which allows a high level specification of the protocol and does not require a rigid specification of the correct action sequences. For this reason the approach appears to be well suited for protocol composition. The problem of protocol composition is strongly related with that of service composition, where the objective is "to describe, simulate, compose and verify compositions of Web services". Recently, technologies have been proposed which use some form of semantic markup of Web services in order to automatically compose Web services to perform a desired task [28,34]. As a first step in this direction, in [15] we have addressed the problem of combining two protocols to define a new more specialized protocol and present a notion of protocol specialization which is based on the well-known notion of stuttering equivalence between runs.

Acknowledgements

We would like to thank the anonymous referees for their comments which helped to improve this paper.

References

[1] M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, P. Mello, Specification and verification of agent interaction protocols in a logic-based system, in: SAC'04, 2004, pp. 72–78.

[2] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. AI 22 (1998) 5–27.

[3] M. Benerecetti, F. Giunchiglia, L. Serafini, Model checking multiagent systems, J. Logic Comput. 8 (3) (1998) 401–423, special issue on Computational Aspects of Multi-Agent Systems.

[4] R. Bordini, M. Fisher, C. Pardavila, M. Wooldridge, Model checking AgentSpeak, in: AAMAS 2003, 2003, pp. 409–416.

[5] R. Bordini, M. Fisher, W. Visser, M. Wooldridge, State-space reduction techniques in agent verification, in: AAMAS 2004, 2004, pp. 894–901.

[6] D. Calvanese, G. De Giacomo, M.Y. Vardi, Reasoning about actions and planning in LTL action theories, in: Proc. KR'02, 2002, pp. 593–602.

[7] FIPA contract net interaction protocol specification, 2002, available at http://www.fipa.org.

[8] N. Fornara, M. Colombetti, Defining interaction protocols using a commitment-based agent communication language, in: Proc. AAMAS'03, Melbourne, 2003, pp. 520–527.

[9] R. Gerth, D. Peled, M.Y. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: Proc. 15th Work. Protocol Specification, Testing and Verification, Warsaw, North Holland, Amsterdam, 1995.

[10] L. Giordano, A. Martelli, On-the-fly automata construction for dynamic linear time temporal logic, in: Proc. TIME 04, 2004, pp. 133–139.

[11] L. Giordano, A. Martelli, C. Schwind, Ramification and causality in a modal action logic, J. Logic Comput. 10 (5) (2000) 625–662.

[12] L. Giordano, A. Martelli, 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, Logic J. IGPL 9 (2) (2001) 289–303.

[13] L. Giordano, A. Martelli, C. Schwind, Specifying and verifying systems of communicating agents in a temporal action logic, in: Proc. AI*IA'03, Pisa, in: Lecture Notes in Artificial Intelligence, vol. 2829, Springer, Berlin, 2003, pp. 262–274.

[14] L. Giordano, A. Martelli, C. Schwind, Verifying communicating agents by model checking in a temporal action logic, in: Proc. Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, in: Lecture Notes in Artificial Intelligence, vol. 3229, Springer, Berlin, 2004, pp. 57–69.

[15] L. Giordano, A. Martelli, C. Schwind, Specialization of interaction protocols in a temporal action logic, in: LCMAS05, 3rd Int. Workshop on Logic and Communication in Multi-Agent Systems, Edinburgh, 2005.

[16] F. Giunchiglia, P. Traverso, Planning as model checking, in: Proc. the 5th European Conf. on Planning (ECP'99), Durham, UK, 1999, pp. 1–20.

[17] M. Greaves, H. Holmback, J. Bradshaw, What is a conversation policy?, in: Issues in Agent Communication, in: Lecture Notes in Computer Science, vol. 1916, Springer, Berlin, 2000, pp. 118–131.

[18] F. Guerin, Specifying agent communication languages, PhD Thesis, Imperial College, London, April 2002.

[19] F. Guerin, J. Pitt, Verification and compliance testing, in: Communications in Multiagent Systems, in: Lecture Notes in Artificial Intelligence, vol. 2650, Springer, Berlin, 2003, pp. 98–112.

[20] J.G. Henriksen, P.S. Thiagarajan, A product version of dynamic linear time temporal logic, in: Proc. CONCUR'97, in: Lecture Notes in Computer Science, vol. 1243, Springer, Berlin, 1997, pp. 45–58.

[21] J.G. Henriksen, P.S. Thiagarajan, Dynamic linear time temporal logic, Ann. Pure Appl. Logic 96 (1–3) (1999) 187–207.

[22] G.J. Holzmann, The SPIN Model Checker. Primer and Reference Manual, Addison-Wesley, Reading, MA, 2003.

[23] M.P. Huget, M. Wooldridge, Model checking for ACL compliance verification, in: Proc. ACL 2003, in: Lecture Notes in Computer Science, vol. 2922, Springer, Berlin, 2003, pp. 75–90.

[24] G.N. Kartha, V. Lifschitz, Actions with indirect effects (preliminary report), in: Proc. KR'94, 1994, pp. 341–350.

[25] N.R. Jennings, Commitments and conventions: the foundation of coordination in multi-agent systems, Knowledge Engrg. Rev. 8 (3) (1993) 233–250.

[26] V. Lifschitz, Frames in the space of situations, Artificial Intelligence 46 (1990) 365–376.

[27] N. Maudet, B. Chaib-draa, Commitment-based and dialogue-game based protocols: new trends in agent communication languages, Knowledge Engrg. Rev. 17 (2) (2002) 157–179.

[28] S. Narayanan, S. McIlraith, Simulation, verification and automated composition of web services, in: Proceedings of the Eleventh International World Wide Web Conference (WWW-11), 2002, pp. 77–88.

[29] W. Penczek, A. Lomuscio, Verifying epistemic properties of multi-agent systems via bounded model checking, Fundamenta Informaticae 55 (2) (2003) 167–185.

[30] M. Pistore, P. Traverso, Planning as model checking for extended goals in non-deterministic domains, in: Proc. IJCAI'01, Seattle, 2001, pp. 479–484.

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

[32] M.P. Singh, Agent communication languages: rethinking the principles, IEEE Computer 31 (12) (1998) 40–47.

[33] M.P. Singh, A social semantics for agent communication languages, in: Issues in Agent Communication 2000, in: Lecture Notes in Computer Science, vol. 1916, Springer, Berlin, 2000, pp. 31–45.

[34] P. Traverso, M. Pistore, Automated composition of semantic web services into executable processes, in: Proc. Third International Semantic Web Conference (ISWC2004), 2004, pp. 380–394.

[35] M. Venkatraman, M.P. Singh, Verifying compliance with commitment protocols, Autonomous Agents and Multi-Agent Systems 2 (3) (1999) 217–236.

[36] M. Wooldridge, Semantic issues in the verification of agent communication languages, Autonomous Agents and Multi-Agent Systems 3 (2000) 9–31.

[37] M. Wooldridge, M. Fisher, M.P. Huget, S. Parsons, Model checking multi-agent systems with MABLE, in: Proc. AAMAS'02, Bologna, Italy, 2002, pp. 952–959.

[38] P. Yolum, M.P. Singh, Flexible protocol specification and execution: applying event calculus planning using commitments, in: Proc. AAMAS'02, Bologna, Italy, 2002, pp. 527–534.