Reasoning about Web Services in a Temporal Action Logic *

Alberto Martelli[1] and Laura Giordano[2]

[1]Dipartimento di Informatica, Università di Torino, Torino
email: mrt@di.unito.it
[2]Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria
email: laura@mfn.unipmn.it

Abstract. The paper presents an approach to reasoning about Web services in a temporal action theory. Web services are described by specifying their interaction protocols in an action theory based on a dynamic, linear-time, temporal logic. The proposed framework is based on a social approach to agent communication, where the effects of communicative actions allow changes in the social state, and interaction protocols are defined in terms of the creation and fulfillment of commitments and permissions among the agents. We show how to introduce epistemic operators in the action theory to deal with incomplete information, and we address the problem of verifying properties of Web services, as well as the problem of reasoning about the composition of Web services.

1 Introduction

Autonomous agents can communicate, cooperate and negotiate using commonly agreed communication languages and protocols. One of the central issues in the field concerns the specification of conversation policies (or interaction protocols), which govern the communication between software agents in an agent communication language [4].

To allow for the flexibility needed in agent communication [10, 14] new approaches have been proposed, which overcome the limitations of the traditional transition net approach, in which the specification of interaction protocols is done by making use of finite state machines. A particularly promising approach to agent communication, first proposed by Singh [21, 22], is the social approach [5, 11, 14]. 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.

In this paper, we adopt a social approach in the specification of the interactions among Web services and, in particular, we address the problem of service verification and that of service composition [15, 18, 23]. In our proposal, Web

* This research has been partially supported by the project MIUR PRIN 2005 "Specification and verification of agent interaction protocols"

services are described by specifying their interaction protocols in an action theory based on a dynamic, linear-time, temporal logic. Such logic has been used in [7, 9] to provide the specification of interaction protocols among agents and to allow the verification of protocol properties as well as the verification of the compliance of a set of services with a protocol. The Web service domain is well suited for this kind of formalization. The proposed 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.

To accommodate the needs of the application domain, in which information is inherently incomplete, in the Section 2, we extend the action theory defined in [9] to deal with incomplete information. More precisely, we introduce epistemic modalities in the language to distinguish what is known about the social state from what is unknown. In this context, the communicative actions by means of which the services interact can be regarded as knowledge-producing actions, and are similar to sensing actions in the context of planning. In order to deal with the frame problem, we introduce a completion construction on the epistemic domain description, which defines suitable successor state axioms.

In Section 3, we show how a Web service can be specified by modeling its interaction protocol in a social approach. We consider, as an example, a service for purchasing a good, whose interaction protocol has the following structure: the customer sends a request to the service, the service replies with an offer or by saying that the service is not available, and finally, if the customer receives the offer, he/she may accept or refuse it. Communicative actions, such as *offer* or *accept*, are modeled in terms of their effects on the social state (action laws). The protocol is specified by putting constraints on the executability of actions (precondition laws) and by temporal constraints specifying the fulfillment of commitments.

In Section 4, we show that several kinds of verification (both runtime and static verification) can be done on the services and the related verification problems can be modeled as satisfiability and validity problems in the logic. We make use of an automata-based approach to solve these problems and, in particular, we work on the Büchi automaton which can be extracted from the logical specification of the protocol.

In Section 5, we then consider the problem of composing Web services, by referring to an example consisting of two services for purchasing and for shipping goods. We define the service composition problem as a planning problem, whose solution requires building a conditional plan and allowing it to interact with the two services. The plan can be obtained from the Büchi automaton derived from the logical specification of the protocol. We address the problem of proving the correctness of a given service implementation with respect to the specification of the component services.

2 The action theory

In this section, we describe the action theory that is used in the specification of the services. We first introduce the temporal logic on which our action theory is based. Then we introduce epistemic modalities and domain descriptions.

2.1 Dynamic Linear-Time Temporal Logic

We briefly define the syntax and semantics of DLTL as introduced in [12]. 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 $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote 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 usual mapping $[[\ ]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$.

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

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

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 $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ 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 (we omit the standard definition for the boolean connectives):

- $M, \tau \models p$ 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 $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

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 $\mathcal{U}$, $\bigcirc$ (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a \in \Sigma}\langle a\rangle\alpha$ (i.e., α holds in the

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

state obtained by executing any action in Σ) , $\alpha \mathcal{U} \beta \equiv \alpha \mathcal{U}^{\Sigma^*} \beta$, $\Diamond \alpha \equiv \top \mathcal{U} \alpha$, $\Box \alpha \equiv \neg \Diamond \neg \alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ).

As shown in [12], DLTL(Σ) is strictly more expressive than LTL(Σ) and the satisfiability and validity problems for DLTL are PSPACE complete problems.

2.2 Epistemic modalities

In the following, we need to describe the effects of communicative actions on the social state of the agents. In particular, we want to represent the fact that each agent can see only part of the social state as it is only aware of some of the communicative actions in the conversation (namely those it is involved in as a sender or as a receiver). For this reason, we introduce knowledge operators to describe the knowledge of each agent as well as the knowledge shared by groups of agents. More precisely, we introduce a modal operator $\mathcal{K}_i$ to represent the knowledge of agent i and the modal operator $\mathcal{K}_A$, where A is a set of agents, to represent the knowledge shared by agents in A. Groups of agents acquire knowledge about social facts when they interact by exchanging communicative actions. The modal operators $\mathcal{K}_i$ and $\mathcal{K}_A$ are both of type KD. They are normal modalities ruled by the axiom schema $\mathcal{K}\varphi \rightarrow \neg\mathcal{K}\neg\varphi$ (seriality). Though the usual modal logic used to represent belief operators is $KD45$, in this formalization we do not add the positive and negative introspection axioms to belief modality $\mathcal{K}$, because, following the solution proposed in [1], we restrict epistemic modalities to be used in front of literals. In particular, epistemic modalities neither can occur nested nor can be applied to a boolean combination of literals.

The relations between the modalities $\mathcal{K}_i$ and $\mathcal{K}_A$ are ruled by the following interaction axiom schema: $\mathcal{K}_A\varphi \rightarrow \mathcal{K}_i\varphi$, where $i \in A$, meaning that what is knowledge of a group of agents is also knowledge of each single agent in the group. As usual, for each modality $\mathcal{K}_i$ (respectively, $\mathcal{K}_A$) we introduce the modality $\mathcal{M}_i$ (resp. $\mathcal{M}_A$), which is defined as the dual of $\mathcal{K}_i$, i.e. $\mathcal{M}_i\varphi$ is $\neg\mathcal{K}_i\neg\varphi$.

2.3 Domain descriptions

The social state of the protocol, which describes the stage of execution of the protocol from the point of view of the different agents, is described by a set of atomic properties called *fluents*, whose epistemic value in a state may change 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 $\neg f$. An *epistemic fluent literal* is a modal atom $\mathcal{K}l$ or its negation $\neg\mathcal{K}l$, where l is a fluent literal and $\mathcal{K}$ is an epistemic operator $\mathcal{K}_i$ or $\mathcal{K}_A$. We will denote by *Lit* the set of all epistemic literals.

An *epistemic state* (or, simply, a state) is defined as a *complete and consistent set of epistemic fluent literals*, and it provides, for each agent i (respectively for each group of agents A) a *three-valued* interpretation in which each literal l is *true* when $\mathcal{K}_i l$ holds, *false* when $\mathcal{K}_i\neg l$ holds, and *undefined* when both $\neg\mathcal{K}_i l$ and $\neg\mathcal{K}_i\neg l$ hold. Observe that, given the property of seriality, consistency guarantees that a state cannot contain both $\mathcal{K}f$ and $\mathcal{K}\neg f$, for some epistemic modality $\mathcal{K}$ and fluent f. In fact, from $\mathcal{K}f$ it follows by seriality that $\neg\mathcal{K}\neg f$, which is inconsistent with $\mathcal{K}\neg f$.

In the following we extend the action theory defined in [9] to accommodate epistemic literals. A *domain description* D is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of (epistemic) *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*.

The *action laws* in Π have the form:

$$\Box(\mathcal{K}l_1 \wedge \ldots \wedge \mathcal{K}l_n \rightarrow [a]\mathcal{K}l) \tag{1}$$

$$\Box(\mathcal{M}l_1 \wedge \ldots \wedge \mathcal{M}l_n \rightarrow [a]\mathcal{M}l) \tag{2}$$

with $a \in \Sigma$, and $\mathcal{K}$ is a knowledge modality. The meaning of (1) is that executing action a in a state where $l_1, \ldots, l_n$ are known (to be true) causes l to become known, i.e. it causes the effect $\mathcal{K}l$ to hold. As an example the law $\Box(\mathcal{K}fragile \rightarrow [drop]\mathcal{K}broken)$ means that, after executing the action of dropping a glass the glass is known to be broken, if the action is executed in a state in which the glass is known to be fragile. (2) is necessary in order to deal with *ignorance* about preconditions of the action a. It means that the execution of a may affect the beliefs about l, when executed in a state in which the preconditions are considered to be possible. When the preconditions of a are unknown, this law allows to conclude that the effects of a are unknown as well. $\Box(\mathcal{M}fragile \rightarrow [drop]\mathcal{M}broken)$ means that, after executing the action of dropping a glass, the glass may be broken, if the action is executed in a state in which the glass may be fragile (i.e. $\mathcal{K}\neg fragile$ does not hold).

The *causal laws* in Π have the form:

$$\Box((\mathcal{K}l_1 \wedge \ldots \wedge \mathcal{K}l_n \wedge \bigcirc(\mathcal{K}l_{n+1} \wedge \ldots \wedge \mathcal{K}l_m) \rightarrow \bigcirc\mathcal{K}l) \tag{3}$$

$$\Box((\mathcal{M}l_1 \wedge \ldots \wedge \mathcal{M}l_n \wedge \bigcirc(\mathcal{M}l_{n+1} \wedge \ldots \wedge \mathcal{M}l_m) \rightarrow \bigcirc\mathcal{M}l) \tag{4}$$

The meaning of (3) is that if $l_1, \ldots, l_n$ are known in a state and $l_{n+1}, \ldots, l_m$ are known in the next state, then l is also known in the next state. Such laws are intended to expresses "causal" dependencies among fluents. The meaning of causal law (4) can be defined accordingly.

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 epistemic fluent literals in the *initial state* and the precondition laws. The initial state $Init$ is a (possibly incomplete) set of epistemic literals, which is made complete by adding $\neg\mathcal{K}l$ to $Init$ when $\mathcal{K}l \notin Init$.

The *precondition laws* have the form:

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

with $a \in \Sigma$ and α an arbitrary non-temporal formula containing a boolean combination of epistemic literals. 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.

In order to deal with the frame problem, we extend the solution proposed in [9] to the epistemic case. We define a completion construction which, given a domain description, introduces frame axioms for all frame fluents in the style of the successor state axioms introduced by Reiter [20] in the situation calculus. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. The value of each epistemic fluent persists from a state to the next one unless its change is caused by the execution of an action as an immediate effect (of an action law) or an indirect effect (of the causal laws). We call $Comp(\Pi)$ the completion of a set of laws Π.

Let Π be a set of action laws and causal laws. Π may contain action laws of the form:

$$\Box(\mathcal{K}\alpha_i \rightarrow [a]\mathcal{K}f) \qquad \Box(\mathcal{K}\beta_j \rightarrow [a]\mathcal{K}\neg f),$$
$$\Box(\mathcal{M}\alpha_i \rightarrow [a]\mathcal{M}f) \qquad \Box(\mathcal{M}\beta_j \rightarrow [a]\mathcal{M}\neg f),$$

as well as causal laws of the form

$$\Box((\mathcal{K}\alpha \wedge \bigcirc\mathcal{K}\beta) \rightarrow \bigcirc\mathcal{K}l),$$
$$\Box((\mathcal{M}\alpha \wedge \bigcirc\mathcal{M}\beta) \rightarrow \bigcirc\mathcal{M}l),$$

where $a \in \Sigma$ and, as a shorthand, $\mathcal{K}\alpha, \mathcal{K}\beta, \mathcal{K}\alpha_i, \mathcal{K}\beta_j$ are conjunctions of epistemic fluents of the form $\mathcal{K}l_1 \wedge \ldots \wedge \mathcal{K}l_n$ and $\mathcal{M}\alpha, \mathcal{M}\beta, \mathcal{M}\alpha_i, \mathcal{M}\beta_j$ are conjunctions of epistemic literals of the form $\mathcal{M}l_1 \wedge \ldots \wedge \mathcal{M}l_n$.

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

$$\Box((\mathcal{K}\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle\mathcal{K}\beta) \rightarrow \bigvee_{a\in\Sigma}\langle a\rangle\mathcal{K}l),$$

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:

$$(*)\ \Box((\mathcal{K}\alpha \wedge \langle a\rangle\mathcal{K}\beta) \rightarrow \langle a\rangle\mathcal{K}l).$$

Moreover, as the axioms $\langle a\rangle\phi \rightarrow [a]\phi$ and $\langle a\rangle\top \wedge [a]\phi \rightarrow \langle a\rangle\phi$ hold in DLTL (see [12]), from (*) we can get:

$$(**)\ \Box(\langle a\rangle\top \rightarrow ((\mathcal{K}\alpha \wedge [a]\mathcal{K}\beta) \rightarrow [a]\mathcal{K}l)).$$

This formula has a structure very similar to action laws. We call these formulas *normalized causal laws*. A similar transformation can be applied to the second causal law, giving: $\Box(\langle a\rangle\top \rightarrow ((\mathcal{M}\alpha \wedge [a]\mathcal{M}\beta) \rightarrow [a]\mathcal{M}l))$.

The action laws and causal laws for a fluent f in Π can then have the following forms:

$$\Box(\langle a\rangle\top \rightarrow (\mathcal{K}\alpha_i \wedge [a]\mathcal{K}\gamma_i \rightarrow [a]\mathcal{K}f)) \qquad \Box(\langle a\rangle\top \rightarrow (\mathcal{K}\beta_j \wedge [a]\mathcal{K}\delta_j \rightarrow [a]\mathcal{K}\neg f))$$
$$\Box(\langle a\rangle\top \rightarrow (\mathcal{M}\alpha_i \wedge [a]\mathcal{M}\gamma_i \rightarrow [a]\mathcal{M}f)) \qquad \Box(\langle a\rangle\top \rightarrow (\mathcal{M}\beta_j \wedge [a]\mathcal{M}\delta_j \rightarrow [a]\mathcal{M}\neg f))$$

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

$$\Box(\langle a\rangle\top \rightarrow ([a]\mathcal{K}f \leftrightarrow (\bigvee_i(\mathcal{K}\alpha_i \wedge [a]\mathcal{K}\gamma_i)) \vee (\mathcal{K}f \wedge \bigwedge_j(\mathcal{K}\neg\beta_j \vee \neg[a]\mathcal{M}\delta_j))))$$
$$\Box(\langle a\rangle\top \rightarrow ([a]\mathcal{K}\neg f \leftrightarrow (\bigvee_j(\mathcal{K}\beta_j \wedge [a]\mathcal{K}\delta_j)) \vee (\mathcal{K}\neg f \wedge \bigwedge_i(\mathcal{K}\neg\alpha_i \vee \neg[a]\mathcal{M}\gamma_i)))).$$

These laws say that a fluent $\mathcal{K}f$ ($\mathcal{K}\neg f$) holds either as (direct or indirect) effect of the execution of some action a, or by persistency, since $\mathcal{K}f$ ($\mathcal{K}\neg f$) held in the state before the occurrence of a and its negation is not a result of a. Observe that the two frame axioms above also determine the values in a state for $[a]\mathcal{M}f$ and for $[a]\mathcal{M}\neg f$.

Observe that, as a difference with [9], in a domain description we do not distinguish between frame and non-frame fluents and in the following we assume that all epistemic fluents are frame, that is, they are fluents to which the law of inertia applies. The kind of non-determinism that we allow here is on the choice of the actions to be executed, which can be represented by the choice construct of regular programs.

3 Web service specification

In this section, we describe how the interface of a Web service can be defined by specifying its interaction protocol. In the social approach [22, 24] 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. These effects, including the creation of new commitments, can be expressed by means of *action laws*.

The action theory introduced above will be used for modeling communicative actions and for describing the social behavior of agents in a multi-agent system. In defining protocols, communicative actions will be denoted by *action_name(s,r)*, where s is the sender and r is the receiver. In particular, two special actions are introduced for each protocol Pn

$$begin\_Pn(s,r) \quad \text{and} \quad end\_Pn(s,r),$$

which are supposed to start and to finish each *run* of the protocol. For each protocol, we introduce a special fluent Pn (where Pn is the "protocol name") which has to be true during the whole execution of the protocol: Pn is made true by the action $begin\_Pn(s,r)$ and it is made false by the action $end\_Pn(s,r)$.

The use of social commitments has long been recognized as a "key notion" to allow coordination and communication in multi-agent systems [13]. Among the most significant proposals to use commitments in the specification of protocols (or more generally, in agent communication) are those by Singh [22], Guerin and Pitt [11], Colombetti [5].

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 in the protocol Pn agent i is committed to agent j to bring about α, where α is an arbitrary non-temporal 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 the protocol Pn the agent i is committed to agent j to bring about α, if the condition β is brought about.

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 state of a run, is regarded to be fulfilled in a run if there is a later state in the run in which α holds.

We introduce the following *causal laws* for automatically discharging fulfilled commitments[2]:

(i) $\Box(\bigcirc\alpha \rightarrow \bigcirc\mathcal{K}_{i,j}(\neg C(Pn, i, j, \alpha)))$
(ii)$\Box((\mathcal{K}_{i,j}(CC(Pn, i, j, \beta, \alpha)) \wedge \bigcirc\beta) \rightarrow \bigcirc\mathcal{K}_{i,j}(C(Pn, i, j, \alpha)))$
(iii)$\Box((\mathcal{K}_{i,j}(CC(Pn, i, j, \beta, \alpha)) \wedge \bigcirc\beta) \rightarrow \bigcirc\mathcal{K}_{i,j}(\neg CC(Pn, i, j, \beta, \alpha)))$

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).

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*:

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

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. 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.

Let us consider as an example a service for purchasing a good.

Example 1. There are two roles: A customer, denoted by C, and a producer, denoted by P. The communicative action of the protocol are: $request(C, P)$, meaning that the customer sends a request for a product, $offer(P, C)$ and $not\_avail(P, C)$, the producer sends an offer or says that the product is not available, $accept(C, P)$ and $refuse(C, P)$, the customer accepts or refuses the offer. Furthermore, as pointed out before, there will be the actions $begin\_Pu(C, P)$ and $end\_Pu(C, P)$ to start and finish the protocol.

[2] We omit the three similar rules with $\mathcal{K}$ replaced by $\mathcal{M}$

As mentioned before, the social state will contain only epistemic fluents. We denote the social knowledge by $\mathcal{K}_{C,P}$, to mean that the knowledge is shared by C and P.

The social state will contain the following fluents, which describe the protocol in an abstract way: *requested*, the product has been requested, *offered*, the product is available and an offer has been sent (we assume that $\neg$*offered* means that the product is not available), *accepted*, the offer has been accepted. The fluent Pu means that the protocol is being executed.

Furthermore, we introduce some base-level commitments (to simplify the notation, in the following we will use $\mathcal{K}^w_{C,P}(f)$ as a shorthand of the formula $\mathcal{K}_{C,P}(f) \vee \mathcal{K}_{C,P}(\neg f)$):

$$C(Pu, C, P, \mathcal{K}_{C,P}(requested))$$
$$C(Pu, P, C, \mathcal{K}^w_{C,P}(offered))$$
$$C(Pu, C, P, \mathcal{K}^w_{C,P}(accepted))$$

We also need the following conditional commitments:

$$CC(Pu, P, C, \mathcal{K}_{C,P}(requested), \mathcal{K}^w_{C,P}(offered))$$
$$CC(Pu, C, P, \mathcal{K}_{C,P}(offered), \mathcal{K}^w_{C,P}(accepted))$$

For instance, the first conditional commitment says that the producer is committed to send an offer, or to say that the product is not available, if a request for the product has been made.

We can now give the action rules for the action of the protocol. We assume all fluents to be undefined in the initial state (i.e., for each fluent f, for each epistemic modality $\mathcal{K}$, $\neg\mathcal{K}f$ and $\neg\mathcal{K}\neg f$ hold in the initial state), except for fluent Pu which will be known to be false. The execution of $begin\_Pu(C, P)$ and $end\_Pu(C, P)$ will have the following effects:

$$\begin{aligned}
&\Box[begin_Pu(C, P)]\mathcal{K}_{C,P}(Pu) \wedge \\
&\quad \mathcal{K}_{C,P}(C(Pu, C, P, \mathcal{K}_{C,P}(requested))) \wedge \\
&\quad \mathcal{K}_{C,P}(CC(Pu, P, C, \mathcal{K}_{C,P}(requested), \mathcal{K}^w_{C,P}(offered))) \wedge \\
&\quad \mathcal{K}_{C,P}(CC(Pu, C, P, \mathcal{K}_{C,P}(offered), \mathcal{K}^w_{C,P}(accepted))) \\
&\Box[end_Pu(C, P)]\mathcal{K}_{C,P}(\neg Pu)
\end{aligned}$$

After starting the protocol, the customer is committed to make a request, and the conditional commitments are created.

The action laws for the remaining actions are the following:

$$\Box[request(C, P)]\mathcal{K}_{C,P}(requested)$$
$$\Box[offer(P, C)]\mathcal{K}_{C,P}(offered) \qquad \Box[accept(C, P)]\mathcal{K}_{C,P}(accepted)$$
$$\Box[not\_avail(P, C)]\mathcal{K}_{C,P}(\neg offered) \qquad \Box[refuse(C, P)]\mathcal{K}_{C,P}(\neg accepted)$$

We can now give the preconditions for the actions of the protocol.

$\Box(\neg\mathcal{K}_{C,P}(\neg Pu) \rightarrow [begin\_Pu(C,P)]\bot)$
$\Box((\neg\mathcal{K}_{C,P}(Pu) \vee \mathcal{K}_{C,P}(requested)) \rightarrow [request(C,P)]\bot)$
$\Box((\neg\mathcal{K}_{C,P}(Pu) \vee \neg\mathcal{K}_{C,P}(requested) \vee \mathcal{K}^w_{C,P}(offered)) \rightarrow [offer(P,C)]\bot)$
$\Box((\neg\mathcal{K}_{C,P}(Pu) \vee \neg\mathcal{K}_{C,P}(requested) \vee \mathcal{K}^w_{C,P}(offered)) \rightarrow [not\_avail(P,C)]\bot)$
$\Box((\neg\mathcal{K}_{C,P}(Pu) \vee \neg\mathcal{K}_{C,P}(offered) \vee \mathcal{K}^w_{C,P}(accepted)) \rightarrow [accept(C,P)]\bot)$
$\Box((\neg\mathcal{K}_{C,P}(Pu) \vee \neg\mathcal{K}_{C,P}(offered) \vee \mathcal{K}^w_{C,P}(accepted)) \rightarrow [refuse(C,P)]\bot)$
$\Box(\neg\mathcal{K}_{C,P}(Pu) \rightarrow [end\_Pu(C,P)]\bot)$

For instance, action $request(C,P)$ cannot be executed if it is not known that the protocol has been started or if it is known that the request has already been achieved (to avoid repeating the action).

A protocol is specified by giving a domain description, defined as follows:

Definition 1. *A domain description D is a pair $(\Pi, \mathcal{C})$ where*

- *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 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 commitment constraints Com_i of all the agents i.*

Given a domain description D, we denote by $Comp(D)$, the completed domain description, the set of formulas: $(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$.

Definition 2. *Given the specification of a protocol by a domain description D, the runs of the system according the protocol are exactly the models of $Comp(D)$.*

Note that protocol "runs" are always finite, while the logic DLTL is characterized by infinite models. 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.

For instance in our example we have the following runs:

$begin\_Pu(C,P); request(C,P); offer(P,C); accept(C,P); end\_Pu(C,P)$
$begin\_Pu(C,P); request(C,P); offer(P,C); refuse(C,P); end\_Pu(C,P)$
$begin\_Pu(C,P); request(C,P); not\_avail(P,C); end\_Pu(C,P)$

4 Reasoning about Web services

Once the interface of a service has been defined by specifying its protocol, several kinds of verification can be performed on it as, for instance, the verification of service compliance with the protocol at runtime, the verification of properties of the protocol and the verification that a given implemented service, whose behavior is known, is compliant with the protocol.

The verification that the interaction protocol has the property φ amounts to show that the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \rightarrow \varphi, \tag{5}$$

is valid, i.e. that all the admitted runs have the property φ.

Verifying that a set of services are compliant with a given interaction protocol at runtime, given the history $\tau = a_1, \ldots, a_n$ describing the interactions of the services (namely, the sequence of communicative messages they have exchanged), amounts to checking if there is a run of the protocol containing that sequence of communications. This can be done by verifying that the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge < a_1; a_2; \ldots; a_n > \top$$

(where i ranges on all the services involved in the protocol) is satisfiable.

In the logic DLTL, a *rigid* protocol like the purchase protocol of Example 1 can be easily represented by means of a regular program, such as the following regular program π_{Pu}:

$begin\_Pu(C, P); request(C, P);$
$((offer(P, C);$
$(accept(C, P) + refuse(C, P)) +$
$not\_avail(P, C));$
$end\_Pu(C, P)$

The correctness of this formulation of the protocol with respect to the formulation given in Example 1 can be verified by proving that all runs of π_{Pu} satisfy the permissions and commitments of the participants, i.e. that the following formula is valid

$$(Comp(\Pi) \wedge Init \wedge \langle \pi_{Pu} \rangle \top) \rightarrow \bigwedge_i (Perm_i \wedge Com_i) \tag{6}$$

where $\langle \pi_{Pu} \rangle \top$ constrains each model to begin with an execution of π_{Pu}.

Further examples of property verification will be given in the next section.

Verification and satisfiability problems can be solved by extending the standard approach for verification 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 nonempty 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 $\rho : prf(\sigma) \to Q$ 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 [12], 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.

A more efficient approach for constructing a Büchi automaton from a DLTL formula making use of a tableau-based algorithm has been proposed in [6]. 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. As for LTL, the number of states of the automaton is, in the worst case, exponential in the size of the input formula, but in practice it is much smaller.

Since the nodes of the graph $\mathcal{G}(\varphi)$ are labeled by sets of formulas, what we actually obtain by the construction is a labeled Büchi automaton, which can be defined by adding to the above definition a *labeling function* $\mathcal{L} : S \to 2^{Lit}$, where *Lit* is the set of all epistemic literals[3]. It is easy to obtain from an accepting run of the automaton a set of models of the given formula, by completing the label of each state in all consistent ways.

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, given a completed domain description

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$$

specifying a protocol, we can construct the corresponding labeled Büchi automaton, such that all runs accepted by the automaton represent runs of the protocol. In [9], we show how to take advantage of the structure of the problems considered in this paper to optimize the construction of the Büchi automaton.

5 Composing Web services

Assume now that we have a service *Sh* for shipping goods, and that the customer wants to reason about the composition of the producer service of the previous section and of this service. For simplicity we assume that the protocol of the shipping service is the same as that of producer service. To distinguish the two protocols we will add the suffix *Pu* or *Sh* to their actions and fluents, while the role of the shipper will be denoted by *S*.

[3] Note that epistemic literals are considered as atomic propositions.

The domain description D_{PS} of the composed service can be obtained by taking the union of the sets of formulas specifying the two protocols: $D_{PS} = D_{Pu} \cup D_{Sh}$. Since we want to reason from the side of the customer, we will replace the epistemic operators $\mathcal{K}_{P,C}$ and $\mathcal{K}_{S,C}$ with $\mathcal{K}_C$, representing the knowledge of the customer. Thus the runs of the composed service PS are given by the interleaving of all runs of the two protocols.

The aim of the customer is to extract from the domain description of PS a *plan* allowing it to interact with the two services. The goal of the plan will be specified by means of a set of constraints *Constr* which will take into account the properties of the composed service. For instance, the customer cannot request an offer to the shipping service if it has not received an offer from the producer. This can be easily expressed by adding a new precondition to the action $request\_Sh(C, S)$:

$$\Box(\neg\mathcal{K}_C(offered\_Pu) \rightarrow [request\_Sh(C, S)]\bot)$$

Other constraints cannot be easily expressed by means of preconditions, since they involve more "global" properties of a run. For instance we expect that the customer cannot accept only one of the offers of the two services. This property can be expressed by the following formula

$$\Diamond\langle accept\_Pu(C, P)\rangle \leftrightarrow \Diamond\langle accept\_Sh(C, S)\rangle$$

stating that the customer must accept both offers or none of them.

Then, the specification of the interaction protocol of the composed service is given by $D_{PS} \cup Constr$, from which the customer will extract the plan. To do this, however, we must first discuss an important aspect of the protocol, i.e. *nondeterminism*.

We assume that, if a protocol contains a point of choice among different communicative actions, the sender of these actions can choose freely which one to execute, and, on the other hand, the receiver cannot make any assumption about which of the actions it will receive. Therefore, from the viewpoint of the receiver, that point of choice is a point of nondeterminism to care about. For instance, the customer cannot know whether the service Pu will reply with *offer_Pu* or *not_avail_Pu* after receiving the request. Therefore the customer cannot simply reason on a single choice of action, but he will have to consider all possible choices of the two services, thus obtaining alternative runs, corresponding to a *conditional plan*. An example of conditional plan is the following[4]

begin_Pu; request_Pu;
  (offer_Pu; begin_Sh; request_Sh;
    (offer_Sh; accept_Pu; accept_Sh; end_Pu; end_Sh +
    not_avail_Sh; refuse_Pu; end_Pu; end_Sh)) +
  (not_avail_Pu; end_Pu).

This plan is represented as a regular program, where, in particular, "+" is the choice operator.

[4] We omit sender and receiver of communicative actions.

Since we are using a linear-time, temporal logic, the constraints in *Constr* can only express properties dealing with a single run. For instance, the run *begin_Pu; request_Pu; offer_Pu; accept_Pu; begin_Sh; request_Sh; offer_Sh; accept_Sh; end_Pu; end_Sh* is correct with respect to the above constraints, since both offers are accepted. However, assume that the customer chooses to execute this plan, and, after executing action *request_Sh*, the shipping service replies with *not_avail_Sh*. At this point there is no other way of continuing the execution, since the customer has already accepted the offer by the producer, while it should have refused it.

The first step for obtaining a *conditional plan* consists in building the Büchi automaton obtained from the domain description D_{PS} and the constraints *Constr*. During the construction of the automaton, we will mark as AND states those states whose outgoing arcs are labeled with actions whose sender is one of the services, such as *offer_Pu* or *not_avail_Pu*[5]. The plan can be obtained by searching the automaton with a forward-chaining algorithm which considers all AND states as branching points of the plan.

In this example, and in many similar cases, the size of the Büchi automaton obtained from the specification of the protocol is small enough to be directly manageable. In this case we might adopt a different approach to the construction of a conditional plan, consisting of "pruning" once and for all the automaton by removing all arcs which do not lead to an accepting state, and all AND states for which there is some outgoing arc not leading to an accepting state. In this way we are guaranteed that, if there is a run $\sigma_1; \mathit{offer\_Sh}; \sigma_2$, where σ_1 and σ_2 are sequences of actions, there must also be a run $\sigma_1; \mathit{not\_avail\_Sh}; \sigma_3$, for some sequence of actions σ_3. Therefore the customer can execute the first part σ_1 of the run, being sure that it will be able to continue with run σ_3 if the shipping service replies with *not_avail_Sh*. In other words, the customer will be able to act by first extracting a linear plan, and begin executing it. If, at some step, one of the services executes an action different from the one contained in the plan, the customer can build a new plan originating from the current state, and restart executing it.

In the construction of the conditional plan, we have taken into account only the nondeterministic actions of the two services. However there are some choices regarding the actions of the customer, such as *accept_Pu* or *refuse_Pu*, that cannot be made at planning time. These nondeterministic choices can also be considered in a conditional plan. In our example we might have the following conditional plan π_{PS}

begin_Pu; request_Pu;
((offer_Pu; begin_Sh; request_Sh;
(offer_Sh;
(accept_Pu; accept_Sh; end_Pu; end_Sh +
refuse_Pu; refuse_Sh; end_Pu; end_Sh) +
not_avail_Sh; refuse_Pu; end_Pu; end_Sh)) +

[5] For simplicity we assume that there is no state whose outgoing arcs are labeled with actions sent and received by the same agent.

(not_avail_Pu; end_Pu))

Note that, in the case of nondeterministic actions of the customer, we are not imposing all choices to be present in the conditional plan, as we did for the actions of the other participants, because some choices might not be possible due to the constraints. For instance, after *accept_Pu* the customer must necessarily execute *accept_Sh*.

A different problem, which can be tackled in our formalism when the conditional plan π_{PS} is given, is that of verifying its correctness with respect to the protocols of the composed services. This requires to verify that, in every run of the conditional plan all the permissions and commitments of the component services are satisfied, and can be done by proving that the formula

$$\bigwedge_k (Comp(\Pi_k) \wedge Init \wedge \langle \pi_{PS} \rangle \top \rightarrow \bigwedge_{k,j} (Perm_j^k \wedge Com_j^k))$$

is valid, where k ranges over the different services and, for each k, j ranges over all the participants of service k. In a similar way, it can be verified that the plan π_{PS} satisfies the constraints *Constr* defined above, by showing the validity of the formula:

$$\bigwedge_k (Comp(\Pi_k) \wedge Init \wedge \langle \pi_{PS} \rangle \top \rightarrow Constr).$$

Up to now the kind of reasoning performed on composed protocols has taken into account only the "public" actions, i.e. the communicative actions of the component protocols. However, in general, the customer should be able to use "private" actions to reason about the information received from the services and to decide what action to execute. Since the information sent by the services will be available only at runtime, such an action should be considered as a nondeterministic action at planning time. We might easily extend our approach to this case by extending the specification of the composed services with "private" actions and fluents of the customer.

The approach described in this section can be applied to the more general problem of building a new service that manages all interactions between the customer and the two services, so that the customer interacts only with the new service through a suitable protocol [19]. Given the protocol *Cu* specifying the interactions between the customer and the new service, the new service can be obtained by putting together the three protocols *Cu, Pu* and *Sh*, and by adding suitable constraints similar to the ones given above. For instance we may state that the offers of each of the two services can be accepted if and only if the customer accepts them:

$$(\Diamond\langle accept\_Pu \rangle \leftrightarrow \Diamond\langle accept\_Cu \rangle) \wedge (\Diamond\langle accept\_Sh \rangle \leftrightarrow \Diamond\langle accept\_Cu \rangle)$$

We can then proceed as before by building the Büchi automaton from the composed protocol and extracting from it a conditional plan, as for instance:

begin_Cu; request_Cu;
begin_Pu; request_Pu;
((offer_Pu; begin_Sh; request_Sh;
((offer_Sh; offer_Cu;
(accept_Cu; accept_Pu; accept_Sh; end_Pu; end_Sh; end_Cu +
refuse_Cu; refuse_Pu; refuse_Sh; end_Pu; end_Sh; end_Cu)) +
not_avail_Sh; not_avail_Cu; refuse_Pu; end_Pu; end_Sh; end_Cu)) +
not_avail_Pu; not_avail_Cu; end_Pu; end_Cu)

This plan can be considered as a specification of the (abstract) behavior of the new service.

6 Conclusions and related work

In this paper we have presented an approach for the specification and verification of interaction protocols in a temporal logic (DLTL). Our approach provides a unified framework for describing different aspects of multi-agent systems. Programs can be 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 are ruled by causal laws, and temporal properties can be expressed by means of temporal formulas. To deal with incomplete information, we have introduced epistemic modalities in the language, to distinguish what is known about the social state from what is unknown. In this framework, various verification problems can be formalized as satisfiability and validity problems in DLTL, and they can be solved by developing automata-based techniques.

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 to reason about composition of Web services. In [8] we have addressed the problem of combining two protocols to define a new more specialized protocol. Here we have shown that service composition can be modeled by taking the formulas giving the domain descriptions of the services, by adding suitable temporal constraints to them, and translating the set of formulas into a Büchi automaton from which a (conditional) plan can be obtained.

The proposal of representing states as sets of epistemic fluent literals is based on [1], which presents a modal approach for reasoning about dynamic domains in a logic programming setting. A similar "knowledge-based" approach has been used to define the PKS planner, allowing to plan under conditions of incomplete knowledge and sensing [16]. PKS generalizes the STRIPS approach, by representing a state as a set of databases that model the agent's knowledge.

The problem of the automated composition of Web services by planning in asynchronous domains is addressed in [19], and extended to the "knowledge level" in [18]. Web services are described in standard process modeling and execution languages, like BPEL4WS, and then automatically translated into a planning domain that models the interactions among services at the knowledge level. The planning technique [19] consists of the following steps. The first step constructs a parallel *state transition system* that combines the given services in a planning domain. The next step consists of formalizing the requirements for the composite service as a goal in a specific language which allows to express extended goals [3]. Finally the planner generates a plan that is translated into a state transition system and into a concrete BPEL4WS process. The planning problem is solved by making use of the state-of-the-art planner MBP.

The approach to Web service composition presented in this paper has analogies with the one presented in [18], particularly with respect to the sequence of steps performed to build the plan. However, the approach of [18] is based on a planning technique derived from model checking for branching-time temporal logic CTL [17], while our approach is based on the dynamic, linear-time, temporal logic DLTL, and on the translation of DLTL formulas into Büchi automata.

In [2] the problem of automatic service composition is addressed assuming that a set of available services (whose behavior is represented by finite state transition systems) is given together with a possibly incomplete specification of the sequences of actions that the client would like to realize. The problem of checking the existence of a composition is reduced to the problem of checking the satisfiability of a PDL formula. This provides an EXPTIME complexity upper bound. In contrast to [2], in our approach client requirements are specified by providing a set of conditions that the target service must satisfy. The composition problem considered in [2] is a generalization of the verification problem we have addressed at the end of section 5 for the case when the protocol of the target service is underspecified and the component e-services that will provide the services required by the client are not known. The extension of our approach to deal with underspecified specifications of the target service will be the subject of further investigation.

References

1. M. Baldoni, L. Giordano, A. Martelli, and Viviana Patti. Reasoning about complex actions with incomplete knowledge: a modal approach. In *Proc. ICTCS'01- LNCS 2202*, 405–425, 2001.
2. D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella and D Calvanese. Syntesis of Underspecified Composite e-Services based on Automated Reasoning. In *Proc. ICSOC'04*, 105–114, 2004.
3. U.Dal Lago, M.Pistore, P.Traverso: Planning with a Language for Extended Goals. *AAAI 2002*, 447-454, 2002.
4. F.Dignum and M.Greaves, "Issues in Agent Communication:An Introduction". In F.Dignum and M.Greaves (Eds.), *Issues in Agent Communication*, LNAI 1916, 1-16, 1999.
5. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *Proc. AAMAS'03*, Melbourne, 520–527, 2003.
6. L. Giordano and A. Martelli. Tableau-based Automata Construction for Dynamic Linear Time Temporal Logic. Annals of Mathematics and Artificial Intelligence, to appear, Springer, 2006.

7. L. Giordano, A. Martelli, and C. Schwind. Verifying Communicating Agents by Model Checking in a Temporal Action Logic. *Proc. Logics in Artificial Intelligence, 9th European Conference, JELIA 2004*, Lisbon, Portugal, Springer LNAI 3229, 57-69, 2004.
8. L. Giordano, A. Martelli, and C. Schwind. Specialization of Interaction Protocols in a Temporal Action Logic. *LCMAS05 (3rd Int. Workshop on Logic and Communication in Multi- Agent Systems)*, ENTCS 157, 4, 1-138, 2006.
9. L. Giordano, A. Martelli and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic Journal of Applied Logic (Special issue on Logic Based Agent Verification), Elsevier, to appear 2006.
10. M. Greaves, H. Holmback and J. Bradshaw. What Is a Conversation Policy?. *Issues in Agent Communication*,LNCS 1916 Springer, 118-131, 2000.
11. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems*, Springer LNAI 2650, 98–112, 2003.
12. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in Annals of Pure and Applied logic, vol.96, n.1-3, 187–207, 1999
13. N.R. Jennings. Commitments and Conventions: the foundation of coordination in multi-agent systems. In *The knowledge engineering review*, 8(3),233–250, 1993.
14. N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols: new trends in agent communication languages. The Knowledge Engineering Review, 17(2):157-179, June 2002.
15. S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-11)*, 77–88, May 2002.
16. R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning (AIPS)*, 212-222, 2002.
17. M.Pistore and P.Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. *Proc. IJCAI'01*, Seattle, 479-484, 2001.
18. M. Pistore, A. Marconi, P. Bertoli and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1252–1259, 2005.
19. M. Pistore, P. Traverso and P. Bertoli. Automated Composition of Web Services by Planning in Asynchronous Domains. *ICAPS 2005*. 2–11, 2005.
20. 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., 359–380, Academic Press, 1991.
21. M. P. Singh. Agent communication languages: Rethinking the principles. IEEE Computer, 31(12), 40–47, 1998.
22. M. P. Singh. A social semantics for Agent Communication Languages. In *Issues in Agent Communication*, Springer LNCS 1916, 31–45, 2000.
23. B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems, In *ICAPS 2003 Workshop on Planning for Web Services*, 28 - 35, Trento, Italy, June 2003.
24. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, 527–534, Bologna, Italy, 2002.