Specialization of Interaction Protocols in a Temporal Action Logic

Laura Giordano [1,2]

Dipartimento di Informatica
Università del Piemonte Orientale
Alessandria, Italy

Alberto Martelli [1,3]

Dipartimento di Informatica
Università di Torino
Torino, Italy

Camilla Schwind [4]

MAP, CNRS
Marseille, France

Abstract

Temporal logics are well suited for the specification and verification of systems of communicating agents. In this paper we adopt a social approach to agent communication, where communication is described in terms of changes to the social state, and interaction protocols in terms of permissions and commitments among agents. In particular, we make use of a temporal action theory, where a protocol is defined as a set of temporal constraints, which specify the effects and preconditions of the communicative actions on the social state. The paper addresses the problem of combining two protocols to define a new more specialized protocol, and presents a notion of protocol specialization which is based on the well known notion of stuttering equivalence between runs. Moreover, the paper studies sufficient conditions (verifiable from the protocol specification) which guarantee that the combination of two protocols is legal.

Key words: reasoning about actions and change, temporal reasoning, multiagent systems

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.

The specification of interaction protocols has been traditionally done by making use of finite state machines, but the transition net approach has been soon recognized to be too rigid to allow for the flexibility needed in agent communication [18,8]. 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 [22], is the social approach [1,4,11,18]. 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 it is well suited for dealing with "open" multiagent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

In this paper we deal with the problem of protocol specialization: A protocol can be specialized by combining it with other protocols. The definition of new protocols by aggregating existing ones is interesting from the point of view of protocol design [17], as refinement and aggregation provide notions of abstraction which are useful to enable the reuse of existing protocols, whose properties are well understood. This is particularly needed for the design of E-Business protocols, as "realistic business settings will need an endless variety of business processes" [17]. 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 [19,23].

The form of composition between protocols that we consider in this paper is that of replacement of an (abstract) action of a protocol with a call to another protocol which achieves the same effects as that action. Such replacement

[1] This research has been partially supported by the project PRIN 2003 "Logic-based development and verification of multi-agent systems"

[2] Email: `laura@mfn.unipmn.it`

[3] Email: `mrt@di.unito.it`

[4] Email: `Camilla.Schwind@map.archi.fr`

operation has some relation with the notion of “splicing into” introduced in [17], where protocols are defined as transition systems, and we refer to the conclusions for a comparison with this work. As an example of replacement, in the following we will consider the specialization of a purchase protocol by replacing the abstract action of “sending goods” with a call to a shipping protocol which achieves the effect of sending the goods from the sender to the receiver.

Our approach to deal with composition and specialization of protocols is based on a temporal logic specification of the protocols. More precisely, it relies on a theory for reasoning about communicating agents we have developed in [7], which is based on Dynamic Linear Time Temporal Logic ($DLTL$) [12]. DLTL extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. The specification of a protocol is given by means of a theory for reasoning about action developed in [6] which is based on DLTL and which allows reasoning with incomplete initial states and dealing with postdiction, ramifications as well as with nondeterministic actions. The action theory allows a simple formalization of the communicative actions in terms of their effects and preconditions as well as the specification of an interaction protocol to constrain the behaviors of autonomous agents. Several verification tasks can be performed on the formalization, including the verification of the compliance of an agent to the protocol, and they can be formalized either as validity or as satisfiability problems in DLTL. The verification can be automated by making use of Büchi automata. In particular, a tableau-based algorithm has been presented in [5] for constructing on-the-fly 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 and satisfiability is PSPACE complete.

In order to verify that the composition of two protocols is legal, the paper defines a notion of specialization between protocols in terms of *stuttering-equivalence* [3] between runs, which, we believe, provides a reasonable notion of abstraction among protocols. Moreover, the paper studies sufficient conditions under which the composition of a protocol P with another protocol P’ (which is called within P) provides a specialization of P. Such conditions can be statically verified on the specification of the protocols P and P'.

The schedule of the paper is the following. In Section 2 we introduce the temporal logic DLTL. In Section 3 we describe how a protocol can be specified in the temporal action theory, using a Purchase protocol as the running example. In Section 4 we introduce the notion of protocol combination by replacement. In Section 5 we define a notion of specialization between protocols in terms of stuttering equivalence between runs. Moreover, we define sufficient conditions under which the composition of two protocols results in a specialization of the original protocol. The last section concludes the paper.

2 Dynamic Linear Time Temporal Logic

In this section we shortly define the syntax and semantics of DLTL as introduced in [12]. In DLTL the next state modalities are 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 $\omega = \{0, 1, 2, \ldots\}$ and let ϵ denote the empty word. 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 mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined as follows:

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

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

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

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

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as 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 $\varepsilon \leq \tau'' < \tau'$[5], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word

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

$\tau \in prf(\sigma)$ 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:

$\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$,
$\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$,
$\Diamond\alpha \equiv \top\mathcal{U}\alpha$,
$\Box\alpha \equiv \neg\Diamond\neg\alpha$,

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). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

3 Protocol Specification

In the social approach an interaction protocol is specified by describing the effects of communicative actions on the social state, in particular by specifying the permissions for executing an action and the commitments that arise as effects on an action execution.

Let us shortly recall the action theory developed in [6] that we use for the specification of interaction protocols.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluents*. 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 $(\Pi, \mathcal{C})$, where Π is a set of *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*.

Action laws in Π have the form: $\Box(\alpha \rightarrow [a]\beta)$, with $a \in \Sigma$ and α, β arbitrary formulas, meaning that executing action a in a state where precondition α holds causes the effect β to hold.

Causal laws in Π have the form: $\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc\gamma)$, meaning that if α holds in a state and β holds in the next state, then γ also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents.

Constraints in $\mathcal{C}$ are arbitrary temporal formulas of *DLTL*. In particular, the set of constraints includes *precondition laws* of the form: $\Box(\alpha \rightarrow [a]\bot)$, meaning 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.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state.[6] To cope with the *frame problem*, the laws in Π, describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in $\mathcal{C}$ and given a special treatment. In [6], we defined a completion construction which, given a domain description, introduces frame axioms in the style of the successor state axioms introduced by Reiter [21]. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. In the following we call $Comp(\Pi)$ the completion of a set of laws Π.

We now provide a specification of a **Purchase protocol**. We have two agents: the merchant (*mr*) and the customer (*ct*). The protocol begins with the customer requesting a quote for some desired goods, followed by the merchant sending the quote. If the customer accepts the quote, then the merchant delivers the goods. After receiving the payment, the merchant forwards a receipt to the customer.

The two agents share all the communicative actions, which are: *sendQuote*, *sendGoods*, *sendReceipt* whose sender is the merchant; *sendRequest*, *sendAccept* and *sendPayment*, whose sender is the customer. Communication is synchronous: the agents communicate by synchronizing on the execution of communicative actions.

The social state will contain the following domain specific fluents: *request* (the customer has requested a quote), *quote* (the merchant has sent the quote), *accept* (the customer has accepted the quote), *goods* (the merchant has sent the goods), *paid* (the customer has made the payment), *receipt* (the merchant has sent the receipt). Such fluents describe observable facts concerning the execution of the protocol.

We also introduce special fluents to represent *base-level commitments* of the form $C(i, j, \alpha)$, meaning that agent i is committed to agent j to bring about α, where α is an arbitrary formula, and *conditional commitments* of the form $CC(i, j, \beta, \alpha)$ (agent i is committed to agent j to bring about α, if the condition β is brought about). The two kinds of base-level and conditional commitments we allow are essentially those introduced in [24]. Such choice is different from the one in [10], where agents are committed to execute an action rather than to achieve a condition.

For modeling the example we introduce the commitments

$$C(mr, ct, goods),\ C(mr, ct, receipt),\ C(ct, mr, paid)$$

and the conditional commitments

$$CC(mr, ct, accept, goods),\ CC(mr, ct, paid, receipt),\ CC(ct, mr, goods, paid).$$

Some reasoning rules have to be defined for cancelling commitments when they have been fulfilled and for dealing with conditional commitments. We

[6] As a difference with [6], here we assume for simplicity that all fluents are frame with respect to all actions.

introduce the following *causal laws*:

$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i,j,\alpha))$$
$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg CC(i,j,\beta,\alpha))$$
$$\Box((CC(i,j,\beta,\alpha) \land \bigcirc\beta) \rightarrow (\bigcirc(C(i,j,\alpha) \land \neg CC(i,j,\beta,\alpha))))$$

A commitment (or a conditional commitment) to bring about α is cancelled when α holds, and a conditional commitment $CC(i,j,\beta,\alpha)$ becomes a base-level commitment $C(i,j,\alpha)$ when β has been brought about.

Let us now describe the effects of communicative actions by the following *action laws*.

Actions of the merchant:

$$\Box([sendQuote](quote \land CC(mr,ct,accept,goods) \land CC(mr,ct,paid,receipt)))$$
$$\Box([sendGoods]goods)$$
$$\Box([sendReceipt]receipt)$$

Actions of the customer:

$$\Box([sendRequest]request)$$
$$\Box([sendAccept]accept \land CC(ct,mr,goods,paid))$$
$$\Box([sendPayment]paid)$$

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

We add two additional action laws:

$$\Box([StartPu]Pu)$$
$$\Box([EndPu]\neg Pu).$$

Actions *StartPu* and *EndPu* start and end the purchase protocol, so that the fluent *Pu* is true during the execution of the protocol. Though, in general one might assume that the protocol is started by executing the first communicative action (and, in a flexible protocol, more then one action could be executed as the first communicative action of a protocol), here, for simplicity, we have introduced the special action *StartPu* for starting the protocol. We assume that the customer is the sender of the *StartPu* action and that any of the participants to the purchase protocol can deliberately terminate the conversation by executing an *EndPu* action. Also we assume that, after a protocol is started, it will be eventually terminated by one of the participants. Hence, the following constraint holds

$$\Box([StartPu]\Diamond\langle EndPu\rangle\top). \tag{1}$$

The permissions to execute communicative actions in each state are determined by social facts. We represent them by precondition laws. We assume that all the actions of the purchase protocol have as a precondition the fact *Pu*. The *precondition laws* for the merchant are the following ones:

$$\Box(\neg Pu \lor \neg request \lor quote \rightarrow [sendQuote]\bot)$$
$$\Box(\neg Pu \lor \neg paid \lor receipt \rightarrow [sendReceipt]\bot)$$

$$\Box(\neg Pu \rightarrow [EndPu]\bot)$$

(2) $$\Box(\neg Pu \vee \neg accept \vee goods \rightarrow [sendGoods]\bot)$$

meaning, for instance, that the action *sendQuote* cannot be executed if the request has not been done by the customer, and similarly for *sendReceipt* and *sendGoods*. Furthermore all actions cannot be executed if their effect has already been achieved, to avoid executing them several times.

The precondition laws for the actions of the customer are the following ones:

$$\Box(\neg Pu \vee request \rightarrow [sendRequest]\bot)$$
$$\Box(\neg Pu \vee \neg quote \vee accept \rightarrow [sendAccept]\bot)$$
(3) $$\Box(\neg Pu \vee \neg goods \vee paid \rightarrow [sendPayment]\bot)$$
$$\Box(Pu \rightarrow [StartPu]\bot)$$
$$\Box(\neg Pu \rightarrow [EndPu]\bot)$$

The last two laws mean that the purchase protocol can be started only if it is not yet under execution and it can be ended only if the protocol is under execution.

We will denote by $Perm_i$ (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.

We assume that all fluents are false in the *initial state* $Init_{Pu}$ of the protocol, i.e. $Init_{Pu} = \{\neg f_k,$ for all the fluents f_k, with $f_k \neq Pu\}$. Thus we introduce the following action law

$$\Box[StartPu](\bigwedge Init_{Pu} \wedge Pu).$$

We are interested in those execution of the purchase protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i,j,\alpha)$ will be fulfilled within the execution of the purchase protocol by the constraint:

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

(actually commitments should have a further parameter to explicitly refer to the protocol). We 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 he is the debtor.

Given the above rules, the domain description $D_{Pu} = (\Pi, \mathcal{C})$ of Purchase protocol is defined as follows: Π is the set of the action and causal laws given above, and $\mathcal{C} = \{\bigwedge_i(Perm_i \wedge Com_i)\} \cup \{(1)\}$ is the set of constraints containing the permissions $Perm_i$ and the commitments Com_i of all the agents, as well as constraint (1). The completed domain description $Comp(D_{Pu})$ is the set of formulas $(Comp(\Pi) \wedge \mathcal{C})$. The runs of the system according the protocol are the linear models of $Comp(D_{Pu})$. We will refer to such runs as *the runs of protocol Pu*.

Observe that in these protocol runs all permissions and commitments have been fulfilled. Therefore, even if action $EndPu$ has no preconditions, it can only be executed in a state which does not contain any commitment, since otherwise the protocol would terminate without satisfying the constraints on commitments.

If Com_j were not included for some agent j, the runs might contain commitments which have not been fulfilled by j. This option is used in [7] in the verification of the compliance of agent j, where the conditions in Com_j and the permissions $Perm_j$ have to be verified for agent j, given the specification of the behaviour of the agent and the assumption that all other agents are respecting their commitments and permissions. We refer to [7] for a description of the verification problems which can be addressed with this formalization of agent protocols.

4 Protocol Specialization

In this section we define how two protocols can be combined in a new more specialized protocol. Consider the specification of the purchase protocol. The action *sendGoods* can be regarded as an abstract action whose effect is that of transferring the goods from the merchant to the customer. It is clear that in a real setting, this task might require several steps to be executed. One could think of solving this task by making use of a shipping service and replace the abstract action *sendGoods* by a call to a **Shipping protocol**.

Let us consider the following specification of the shipping protocol. For simplicity we assume that the names of all the fluents used within the shipping protocol are different from those used in the purchase protocol. The shipper s delivers goods from merchant m to the customer c.
Action Laws:

$\Box[Start_{Sh}(m,c,s)](Sh \wedge Init_{Sh})$
$\Box[requestQuote](req_{Sh} \wedge CC(m,s,goods_{atC}, paid_{Sh}))$
$\Box[sendQuote](quote_{Sh} \wedge C(s,m,goods_{atC}))$
$\Box(goods_{atM} \to [deliverGoods(m,s)]goods_{atS})$
$\Box(goods_{atS} \to [deliverGoods(s,c)]goods_{atC})$
$\Box(quote_{Sh}(q) \to [sendMoney(m,s,q)]paid_{Sh})$
$\Box[End_{Sh}(m,c,s)](end_{Sh} \wedge \neg Sh)$
$\Box(\bigcirc goods_{atI} \to \bigcirc\neg goods_{atJ})$, for $I \neq J$

Preconditions:

$\Box(\neg Sh \vee req_{Sh} \to [requestQuote]\bot)$
$\Box(\neg Sh \vee \neg req_{Sh} \vee quote_{Sh} \to [sendQuote]\bot)$
$\Box(\neg Sh \vee \neg goods_{atM} \vee \neg req_{Sh} \to [delivGoods(m,s)]\bot)$
$\Box(\neg Sh \vee \neg goods_{atS} \to [deliverGoods(s,c)]\bot)$
$\Box(\neg Sh \vee \neg goods_{atC} \to [End_{Sh}(m,c,s)]\bot)$

The set $Init_{Sh}$ contains the commitments $C(m,s,req_{Sh})$ (if the merchant starts protocol Sh, he is committed to request a quote) and $CC(s, m, req_{Sh}, quote_{Sh})$ (the shipper is committed to send a quote, if the merchant sends a request).

The last causal law describes the mutual exclusion between the values of the three fluents $goods_{atM}$, $goods_{atS}$ and $goods_{atC}$. The goods may be either at the merchant site, or at the shipper site, or at the customer site, but in no more than one site at a time. In the initial state $Init_{Sh}$ of the protocol all fluents are taken to be false, except for $goods_{atM}$, $goods_{atC}$, whose values depend on the environment in which the protocol is executed. Moreover, we assume $goods_{atS}$ to be false in $Init_{Sh}$.

As for the purchase protocol, also for the shipping protocol we assume that, after the protocol is started, it will eventually be terminated by the execution of an End action. Hence, the following constraint holds

$$\Box([StartSh]\Diamond\langle EndSh\rangle\top). \tag{4}$$

The domain description $D_{Sh} = (\Pi_{Sh}, \mathcal{C}_{Sh})$ of the Shipping protocol contains the set Π_{Sh} of action laws and causal laws and the set of constraints $\mathcal{C}_{Sh} = \{\bigwedge_i (Perm_i \wedge Com_i)\} \cup \{(4)\}$ containing the permissions and commitments of all agent in the protocol and constraint (4) above. We will denote by $Comp(D_{Sh})$ the completed domain description, that is, the set of formulas $(Comp(\Pi_{Sh}) \wedge \mathcal{C}_{Sh})$.

To determine if we can replace the action *sendGoods* by a call to the shipping protocol, we must have a general specification of the global effects and preconditions of the protocol, defining which properties of the environment may be affected by the execution of the protocol, and under which condition. The global effects of the shipping protocol can be described by the following law:

$$\Box(goods_{atM} \rightarrow [Start_{Sh}(m, c, s)]\Diamond goods_{atC})$$

saying that if the goods are at m before the start of the shipping protocol, then, after its start, a state will be eventually reached in which the goods have been delivered to the customer. Such property is a global properties of the shipping protocol, and can be derived from the specification of the protocol given above. Namely, the formula:

$$Comp(D_{Sh}) \rightarrow (\Box(goods_{atM} \rightarrow [Start_{Sh}(m, c, s)]\Diamond goods_{atC}))$$

is valid.

As concerns the global preconditions of the shipping protocol, it is clear that the action $Start_{Sh}$ starting the protocol can always be executed. However, in all the execution of the protocol satisfying the permissions and commitments of all agents, the execution of the $Start_{Sh}$ action may occur in a state only if $goods_{atM}$ holds. The formula

$$Comp(D_{Sh}) \rightarrow (\Box(\neg goods_{atM} \rightarrow [Start_{Sh}(m, c, s)]\bot))$$

is valid.

If we want to compose the Purchase protocol above with the Shipping protocol, we have to combine their specifications D_{pu} and D_{sh}. Moreover, we have to replace the action *sendGoods* in the Purchase protocol with a call to the shipping protocol, by removing the action laws for action *sendGoods*, and replacing the precondition law (2) for action *sendGoods* with a corresponding precondition law for the action $Start_{Sh}$, which starts the execution of the shipping protocol. Hence, the precondition law (2) in P must be replaced by the following one:

$$\Box(\neg Pu \vee \neg accept \vee goods \rightarrow [Start_{Sh}(m, c, s)]\bot)$$

meaning that the action $Start_{Sh}$ can be executed only if the customer has accepted the quote and the goods have not already been delivered.

While there are some properties of the social state that are only significant within the shipping protocol, and they are used to determine the correct behaviour of the protocol (consider, for instance, the fluent $quote_{Sh}$), other fluents used within the shipping protocol describe properties of the environment in which the protocol is executed and they relate to fluents used in the purchase protocol. For instance, the location of the goods, represented by the fluents $goods_{atM}$, $goods_{atS}$, $goods_{atC}$ in the shipping protocol, is represented by the single fluent *goods* in the purchase protocol. *goods* means that goods are at C, so that when *goods* is false either $goods_{atM}$ or $goods_{atS}$ must be true. If we want to combine two protocols, the relation among such fluents can be made explicit by an interface $Intf$: containing the following causal laws:

$\Box(\bigcirc goods \rightarrow \bigcirc goods_{atC})$
$\Box(\bigcirc goods_{atC} \rightarrow \bigcirc goods)$
$\Box(\bigcirc\neg goods \rightarrow \bigcirc\neg goods_{atC})$
$\Box(\bigcirc\neg goods_{atC} \rightarrow \bigcirc\neg goods)$
$\Box(\bigcirc goods_{atJ} \rightarrow \bigcirc\neg goods)$, for $J = M, S$
$\Box(\bigcirc\neg goods \wedge \neg goods_{atS} \rightarrow \bigcirc goods_{atM})$
$\Box(\bigcirc\neg goods \wedge \neg goods_{atM} \rightarrow \bigcirc goods_{atS})$

This interface describes how a change to the values of fluents $goods_{atM}$, $goods_{atS}$, $goods_{atC}$ causes a change to the value fluent *goods*, and vice-versa.

Given such an interface, from the specification of the shipping protocol it follows that, when the shipping protocol is called in a state in which goods have not been delivered, eventually a state is reached where the goods have been delivered, i.e., the formula

(5) $$(Comp(D_{sh}) \wedge Intf) \rightarrow \Box(\neg goods \rightarrow [Start_{Sh}(m, c, s)]\Diamond goods)$$

is valid. This fact suggests to replace the action *sendGoods* in P with a call to the shipping protocol.

In general, given two protocols P and P', with their associated specifications D_P and $D_{P'}$, we define their composition by replacement as follows (we assume that the specification of P and P' make use of different fluents, and that an interface $Intf$ has been defined among them):

Definition 4.1 A **protocol P is combined with protocol P' by replacement at action A** , by the following sequence of steps:

(1) remove from D_P the action laws for A;

(2) replace in D_P the remaining occurrences of action A with the action $StartP'$ (which is a call to an execution of protocol P');

(3) add to the specification D_P thus modified the specification $D_{P'}$ of protocol P' and the interface $Intf$ between P and P'.

We denote the resulting protocol by $P(P'/A)$.

Observe that, in general, the number of agents involved in the protocol $P(P'/A)$ may be bigger than the number of agents involved in the protocol P. In the example, the shipper agent was not involved in the original purchase protocol.

Also observe that this way of combining the purchase and the shipping protocols dose not impose any constraint on the order in which the payment actions are executed. The merchant can pay the shipper (by executing action $sendMoney(m, s)$) either before or after the customer has paid the merchant (by executing action $sendPayment$). As expected, however, due to the precondition law (3), the customer will not send his payment before the shipper has delivered the goods.

However, we could impose the requirement that the merchant can pay the shipper only after the customer has made his payment, by introducing the following constraint:

$\Box(\neg paid \rightarrow [sendMoney(m, s)]\bot)$

The addition of a constraint to the protocol is an operation of **refinement** of the protocol, which can eliminate some of the runs of the protocol. In the following we define conditions under which the operation of composition of two protocols by replacement and the operation of refinement of a protocol are legal.

We have argued that, for the composition of two protocols to be meaningful, the protocol P' must at least provide on the environment the same effect as action A in the specification of the protocol P (in the case above, we require that in each execution of the shipping protocol eventually the goods is delivered at C). However, this is not enough to guarantee that the composition of two protocols by "replacement" is legitimate. Consider, for instance, a protocol P' which fulfills all the commitments of the protocol P in a different order with respect to the order expected by protocol P, or, instead, which makes unexpected changes to the values of some fluent of P. In these cases we do not want to accept the composition of P and P' as a legal one.

In general, the protocol P' might interfere unexpectedly with protocol P and we want to be able to establish if the combination of P and P' by "replacement" is legitimate. We regard the combination as legitimate when

$P(P'/A)$ is a specialization of P.

5 Properties of Protocol Specialization

In this section we define a notion of specialization between protocols, which is needed to establish whether a protocol $P(P')$ obtained by combining two protocols P and P', is well behaved, in the sense that it specializes the (more abstract) protocol P. Later we will also introduce conditions on the specification of the protocols P and P' which are sufficient for the composition $P(P')$ to be a specialization of P.

The notion of specialization between protocols is defined in terms of *stuttering-equivalence* [15,3] between runs of the protocols, which in our case are linear models. We refer to the definition in [20]. Let AP be a subset of propositions.

Definition 5.1 Two linear models (σ_1, V_1) and (σ_2, V_2) are **stuttering equivalent** (with respect to AP) if there are two infinite sequences $0 \leq i_1 \leq i_2 \leq \dots$ and $0 \leq j_1 \leq j_2 \leq \dots$ such that for every $k \geq 0$, the valuations

$$V_1(\sigma_1^{i_k}), V_1(\sigma_1^{i_k+1}), \dots, V_1(\sigma_1^{i_{k+1}-1}),$$
$$V_2(\sigma_2^{j_k}), V_2(\sigma_2^{j_k+1}), \dots, V_2(\sigma_2^{j_{k+1}-1})$$

are identical on the subset of propositions AP, where, given $\sigma = a_0, a_1, a_2, \dots$, $\sigma^h = a_0, a_1, a_2, \dots, a_h$ is the prefix of σ containing its first h symbols.

Intuitively, two linear models are stuttering equivalent if they can be partitioned into finite blocks of repeated, or stuttered states, and corresponding blocks are equivalent in the two linear models: the states in the k-th block of one model have the same valuation as the states in the k-th block of the other model on propositions from AP.

When comparing protocols, we compare their runs, which, as we have seen in section 3, are linear model of their specification. Let $\mathcal{P}_2$ be the propositions occurring in the specification of protocol P_2.

Definition 5.2 Protocol P_1 is a **specialization** of protocol P_2 if for each run (σ_1, V_1) of P_1 there is a run (σ_2, V_2) of P_2 such that (σ_1, V_1) is *stuttering equivalent* to (σ_2, V_2) with respect to the subset of propositions $AP = \mathcal{P}_2$.

We say that P_1 is a specialization of protocol P_2 when for each run of P_1 there is a run of P_2 which is equivalent to it if we only consider the propositions of the (more abstract) protocol P_2. The notion of stuttering equivalence allows consecutive states with the same evaluation on the propositions of P_2 to be identified, thus ignoring in the runs of P_1 the actions which do not affect the value of the propositions in P_2.

The above notion of protocol specialization is more restrictive then the one proposed by Yolum and Singh in [17], as here we require that for every run of P_1, there is a stuttering equivalent run of P_2 in which all the changes to the fluents of P_2 occur exactly in the same order. Instead, the notion of subsumption between runs in [17] requires to find, for each run σ_1 of P_1 a run σ_2 of P_2 such that all the states of σ_2 also occur in σ_1 (modulo a state-similarity function), and they occur in the same relative order in the two runs. No constraint is put on the valuation of intermediate states, which may occur in σ_1 in between states corresponding to σ_2 states. Stuttering equivalence imposes that the value of the fluents occurring in P_2 cannot change in such intermediate states. As a further difference, the state similarity function used in [17] compares states on the bases of the commitments they contain, while we consider all the fluents used in P_2.

The notion of protocol specialization can be used to define when the composition $P(P'/A)$ of two protocols is legitimate.

Definition 5.3 Given two protocols P and P', their **combination** $P(P'/A)$ **is legitimate** when $P(P'/A)$ is a specialization of P.

Hence we accept that in the specialized protocol $P(P'/A)$ some behaviour of the original protocol P is omitted, but we require that all the runs of $P(P'/A)$ are runs of the original protocol P (modulo stuttering equivalence).

According to this definition, we regard as legal the combination $Pu(Sh/sendGoods)$ of the purchase protocol and the shipping protocol above. However, if we consider a different shipping protocol Sh', in which the shipper brings back and forth several times the goods between the merchant and the customer before leaving them to the customer, we would not consider (according to the definition above) the combination $Pu(Sh'/sendGoods)$ as a legal one: such protocol, in fact, may have runs in which the value of the fluent *goods* changes several times before eventually taking the value *true* as expected. Such runs are not stuttering equivalent to any run of Pu, as in Pu the value of fluent *goods* changes value from $false$ to *true* only once.

Under some respects, using stuttering equivalence for defining when the combination $P(P'/A)$ of two protocols is legitimate might appear to be too restrictive. Consider, for instance, the case when action A of protocol P has two effects α and β. Then, a protocol P' which achieves both α and β can be combined with P so that the combination $P(P'/A)$ is legitimate (according to definition 5.3) only if P' achieves α and β simultaneously. If P' achieves, for instance, α and β in the order, then $P(P'/A)$ may have runs which are not stuttering-equivalent to any run of P (in which the execution of action A makes α and β simultaneously true).

On the other hand, using stuttering equivalence among runs to define when a protocol P' is a specialization of another protocol P has the important advantage that all the stutter invariant properties that can be proved for P are also properties of P'. For this reason we have adopted the notion of stuttering equivalence in the definition of protocol specialization, rather than introducing a weaker notion of equivalence among runs.

It was proved by Peled in [20] that every stutter-invariant property expressible in (propositional) LTL can be expressed in LTL(U) (the fragment of LTL with $\mathcal{U}$ as the only temporal operator). As a consequence of this result all the properties of a protocol P which can be expressed in LTL(U) also hold for the protocol $P(P'/A)$, if the composition $P(P'/A)$ is legitimate.

The problem of deciding stuttering equivalence on finite states Kripke structures is known to be solvable in polynomial time [9]. However, we want to provide sufficient conditions which guarantee that, given two protocols P and P', their combination $P(P'/A)$ is legitimate. Such conditions must be verifiable from the specification of the two protocols P and P'.

The intuition is that we can combine P and P' by replacing action A in P by a call to P' only when, in each state in which action A can be executed: (1) protocol P' produces the same effects as action A; (2) protocol P' does not affect any other fluent in the specification of P.

Let $\Box(\beta_1 \rightarrow [A]l_1), \ldots, \Box(\beta_n \rightarrow [A]l_n)$ be all the action laws describing the immediate effects of action A (the l_i's may be either positive or negative literals). Given a subset Γ of action laws for A, we denote by E_Γ the set of literals l_i which are in the consequents of the laws in Γ and by $Cond_\Gamma$ the set of formulas β_i which are in the antecedents of the laws in Γ. Moreover, for all the fluents f which are positively or negatively affected by the actions laws for A, let $\beta_1^f, \ldots, \beta_k^f$ be the conditions occurring in the antecedents of all the action laws whose effect is f and let $\gamma_1^f, \ldots, \gamma_h^f$ be the conditions occurring in the antecedents of all the action laws whose effect is $\neg f$. Finally, let $\Box(\xi_1 \rightarrow [A]\bot), \ldots, \Box(\xi_r \rightarrow [A]\bot)$ be all the precondition laws for action A. In the following we denote by $Poss_A$ the conjunction $\neg\xi_1 \wedge \ldots \wedge \neg\xi_r$. $Poss_A$ is true in all the states in which A is executable.

Definition 5.4 We say that protocol P' **achieves the same effects as action A in P** if:

- for all the subsets Γ of action laws for A , the formula
  $Comp(D_{P'}) \wedge Intf \rightarrow \Box(Poss_A \wedge Cond_\Gamma \rightarrow [Start_{P'}]\neg E_\Gamma \mathcal{U}(E_\Gamma \wedge E_\Gamma \mathcal{U} End_{P'}))$
  is valid;
- for all the fluents f, the formulas
  $Comp(D_{P'}) \wedge Intf \rightarrow \Box(Poss_A \wedge \bigwedge \neg\beta_i^f \wedge \bigwedge \neg\gamma_i^f \wedge f \rightarrow [Start_{P'}] f\mathcal{U}(End_{P'} \wedge f)$
  $Comp(D_{P'}) \wedge Intf \rightarrow \Box(Poss_A \wedge \bigwedge \neg\beta_i^f \wedge \bigwedge \neg\gamma_i^f \wedge \neg f$
  $\rightarrow [Start_{P'}] f\mathcal{U}(End_{P'} \wedge \neg f)$
  are valid.

The first item can be explained as follows: All the action laws for A which have their conditions β_i jointly true before the start of protocol P' will eventually have their effects l_i (simultaneously) true during the execution of protocol P', and these effects will remain true until the end of P'. The second items says that: each fluent f which might be affected by action A, remains unchanged during the execution of P', if none of the conditions β_i^f and γ_i^f in the antecedent of its action laws holds in the state before the execution of the

$StartP'$ action.

Let us characterize the fluents which are not affected by the execution of a protocol P as those fluents whose value persists through the execution of the protocol.

Definition 5.5 We say that protocol P' **does not affect the fluents** $F_1, \ldots, F_n$ **in** P if, for all $i = 1, \ldots, k$, the formulas

$$Comp(D_{P'}) \wedge Intf \rightarrow \Box(F_i \rightarrow [Start_{P'}]F_i\mathcal{U}(End_{P'} \wedge F_i)) \text{ and}$$
$$Comp(D_{P'}) \wedge Intf \rightarrow \Box(\neg F_i \rightarrow [Start_{P'}]\neg F_i\mathcal{U}(End_{P'} \wedge \neg F_i))$$

are valid.

Let $\mathcal{P}_{\overline{A}}$ the set of fluents of P which do not occur in the effect of any action law for A in D_P (they are not affected by A in P).

We can now state the following proposition:

Proposition 5.6 *If (i) P' achieves the same effects as action A and (ii) P' does not affect the fluents $\mathcal{P}_{\overline{A}}$ in P, then the protocol $P(P')$ is a specialization of protocol P.*

Considering the example above, we have seen that, the shipping protocol has the property that, when executed in a state in which $\neg goods$ holds, it eventually makes $goods$ true (see (5)). It is easy to see that, after the shipping protocol has made $goods$ true, $goods$ remains true until the end of the protocol. Hence, the formula

$$Comp(D_{sh}) \wedge Intf \rightarrow$$
$$\Box(\neg goods \rightarrow [Start_{Sh}(m, c, s)]\Diamond\neg goods\mathcal{U}(goods \wedge goods\mathcal{U}End_{Sh})$$

is valid. This is the only effect of the shipping protocol on the fluents of the purchase protocol. The value of all the other fluents persist through the execution of the purchase protocol. We can thus conclude that the combination $Pu(Sh/sendGoods)$ of the two protocols is a specialization of the purchase protocol Pu.

6 Conclusions

In this paper we have introduced a notion of protocol specialization and composition. Two protocols P and P' can be combined by replacing an action A of P by a call to the protocol P'. To determine if the resulting protocol $P(P'/A)$ is legal (it is a specialization of the original protocol P), we have introduced a notion of specialization between protocols, which is based on the well-known notion of stuttering-equivalence. We have defined sufficient conditions, verifiable on the specification of the protocols, which guarantee that the composition of two protocols is legal.

As we have mentioned in the introduction, the replacement operation we have introduced has some relation with the notion of "splicing into" introduced in [17], where, protocols are defined as transition systems (similar in spirit to finite state machines). Splicing into a protocol P by another protocol P' requires to find two states of P between which P' can be spliced into. The operation of replacement we propose is more liberal: to replace the execution of an action A within a protocol P with a call to a protocol P', action A is replaced by an action $StartP'$, which initiates execution of P'. The (sub-)protocol P' called within P is not required to complete its execution within a given step of P, but its actions can be interleaved with other actions in P, provided that the temporal constraints ruling the two protocols are satisfied. As a further difference, we have already mentioned in the previous section that the notion of specialization between protocols we propose is more restrictive than the one proposed in [17].

The notion of stuttering-equivalence we have used to define protocol specialization, could be used to define a notion of equivalence between protocols, and it would be interesting to compare such notion with the concepts of equivalence introduced in [13], which explores several alternative definitions of equivalence between protocols.

In this paper we have introduce a notion of combination of protocols which allows more specialized protocols to be defined. The problem is related to the problem of service composition and coordination, which is being addressed in the Web services communities. In particular, in [23] planning techniques have been adopted for automatically generating composed web services. Though in this paper we have only addressed a specific kind of protocol combination we argue that our approach can be extended to cope with problems like services composition.

Formal specification and verification of web service protocols has been also faced in [14], which describes a specification in TLA+ [16] of the Web Services Atomic Transaction protocol, which was verified with a model checker. This protocol can be considered as a horizontal protocol [2], i.e. a protocol defining a common infrastructure (middleware) independent of the application area. However, in principle, the same approach could be adopted also to deal with vertical protocols, i.e. protocols related to specific business areas, which are mainly concerned with the semantic aspects of the exchanged messages and with the set of correct conversations, as those we deal with in this paper.

We just want to mention that the verification of protocol properties from a protocol specification as a temporal domain description in DLTL, can be automated by making use of Büchi automata [5]. The construction of the automata 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 automata is, in the worst case, exponential in the size of the input formula and satisfiability is PSPACE-complete.

References

[1] M. Alberti, D. Daolio and P. Torroni. Specification and Verification of Agent Interaction Protocols in a Logic-based System. *SAC'04*, March 2004.

[2] G.Alonso, F.Casati, H.Kuno, V.Machiraju. Web Services, Springer, 2004.

[3] E.M.Clarke, O.Grumberg, and D. Peled, Model Checking, MIT Press, 2000.

[4] N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *Proc. AAMAS'03*, Melbourne, pp. 520–527, 2003.

[5] L. Giordano and A. Martelli. On-the-fly Automata Construction for Dynamic Linear Time Temporal Logic. *TIME 04*, June 2004.

[6] L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In *The Logic Journal of the IGPL*, Vol. 9, No. 2, pp. 289-303, March 2001.

[7] L. Giordano, A. Martelli, and C. Schwind. Verifying Communicating Agents by Model Checking in a Temporal Action Logic. JELIA 2004, Lisbon, Portugal, September 27-30, 2004, pp. 57-69.

[8] M. Greaves, H. Holmback and J. Bradshaw. What Is a Conversation Policy?. *Issues in Agent Communication*,LNCS 1916 Springer, pp. 118-131, 2000.

[9] J.F. Groote and F.W. Vaandrager An efficient algorithm for branching bisimulation and stuttering equivalence. In Proceedings of ICALP, Warwick, LNCS 443, pages 626-638, 1990.

[10] F. Guerin. Specifying Agent Communication Languages. PhD Thesis, Imperial College, London, April 2002.

[11] F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems*, Springer LNAI 2650, pp. 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, pp.187–207, 1999

[13] M.W Johnson and P. Mc Burney and S. Parsons. When are two protocols the same? In *Communication in Multiagent Systems*, pages 253-268, LNAI 2650, Springer, 2003.

[14] James E. Johnson, David E. Langworthy, Leslie Lamport and Friedrich H. Vogt, Formal Specification of a Web Services Protocol, proceedings of the First International Workshop on Web Services and Formal Methods (WS-FM 2004),Pisa , 2004.

[15] L. Lamport. What Good Is Temporal Logic? Information Processing 83, R. E. A. Mason, ed., Elsevier Publishers (1983), 657-668.

[16] Leslie Lamport. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley Professional, 2003.

[17] A.U. Mallya and M.P. Singh. A Semantic Approach for Designing E-Business Protocols. In *Proceedings of the Third International Conference on Web Services (ICWS04)*, pp. 742-745, July 2004.

[18] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols: new trends in agent communication languages. In *The Knowledge Engineering Review*, 17(2):157-179, June 2002.

[19] 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)*, May, 2002.

[20] D. Peled and T. Wilke. Stutter-Invariant Temporal Properties are Expressible without the Next-time Operator. In *Inf. Process. Lett.* 63(5): 243-246 (1997).

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

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

[23] P.Traverso and M.Pistore. Automated Composition of Semantic Web Services into Executable Processes. Proc. Third International Semantic Web Conference (ISWC2004), November 9-11, 2004, Hiroshima, Japan.

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