Verifying Agents' Conformance with Multiparty Protocols

Laura Giordano¹ and Alberto Martelli²

¹ Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria

² Dipartimento di Informatica, Università di Torino, Torino

Abstract. The paper deals with the problem of agents conformance with multiparty protocols. We introduce a notion of conformance of a set of k agents with a multiparty protocol with k roles, which requires the agents to be interoperable and to produce correct executions of the protocol. We introduce conditions that enable each agent to be independently verified with respect to the protocol. We assume that protocols are specified in a temporal action theory and we show that the problem of verifying the conformance of an agent with a protocol can be solved by making use of automata based techniques. Protocols with nonterminating computations, modeling reactive agents, can also be captured in this framework.

1 Introduction

In an open environment, the interaction of agents is ruled by interaction protocols on which agents commonly agree. An important issue, in this regard, concerns agent conformance with the protocol. Although agent policy may somehow deviate from the behavior dictated by the protocol, in some cases we want, nevertheless, to regard the policy as being compatible with the protocol. A related issue concerns the *interoperability* of agents in an open environment. The need for conditions to guarantee that a set of agents may interact properly, has led to the introduction of different notions of *compatibility* among agents [4] as well as to the definition of notions of *conformance* of an agent with a protocol [2,5,11,17].

In this paper, we define a notion of agent conformance for the general case of multiparty protocols. This notion must assure that a set of agents, that are conformant with a protocol, interoperate (in particular, they do not get stuck) and that their interactions produce correct executions of the protocol.

In our proposal, the specification of agents and protocols is given in a temporal action theory [9,13], by means of temporal constraints, and the communication among agents is synchronous. Protocols are given a *declarative specification* consisting of: (i) the specification of communicative actions by means of their effects and preconditions on the social state which, in particular, includes commitments; (ii) a set of temporal constraints, which specify the wanted interactions (and, under this respect, our approach to protocol specification is similar to the

one proposed in *DecSerFlow* [21]). Protocols with nonterminating computations, modeling reactive services [8], can also be captured in this framework.

We define a multiparty protocol P with k roles, by separately specifying the behavior of all roles $P_1, \dots, P_k$ in the protocol. We then introduce a notion of *interoperability* among a set of agents, which guarantees the agents to interact properly. More precisely, each agent can freely choose among its possible emissions without the computation getting stuck.

Given a multiparty protocol P , we define a notion of *conformance of a set of agents* $A_1, \dots, A_k$ with P : agents $A_1, \dots, A_k$ interoperate and their interaction only produces runs of the protocol P . Verifying the conformance of a set of agents all together, however, is not feasible in an open environment, as, in general, the internal behavior of all agents participating in a protocol is not known. The verification of each agent participating in the protocol must be done independently.

In this paper, we introduce a definition of *conformance of a single agent* A_i (*playing role i*) with the protocol P . We will see that verifying an agent A_i with respect with its role P_i is not sufficient to guarantee the interoperability of a set of conformant agents in the multiparty case, unless a rather narrow notion of conformance is adopted. Indeed, our notion of conformance of an agent A_i with a protocol P is defined by comparing the runs of agent A_i , when executed in the context of the protocol, and the runs of P itself. We prove that a set of agents which are independently conformant with the protocol P are guaranteed to be interoperable and to produce correct executions of P .

Starting from a specification of the protocol in a temporal logic, the problem of verifying the conformance of an agent with a protocol can be solved by making use of an automata based approach. In particular, interoperability can be checked by working on the Büchi automaton which can be extracted from the logical specification of the protocol.

2 Protocol Specification

The specification of interaction protocols we adopt is based on Dynamic Linear Time Temporal Logic (DLTL) [15], a linear time temporal logic which extends LTL by allowing the until operator to be indexed by programs in Propositional Dynamic Logic (PDL). DLTL allows until formulas of the form $a\mathcal{U}^\pi\beta$, where the program $\pi \in \text{Prg}(\Sigma)$ is a regular expression built from a set Σ of atomic actions. More precisely, $\text{Prg}(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$, where $a \in \Sigma$ and π_1, π_2, π range over $\text{Prg}(\Sigma)$.

As for LTL, DLTL models are infinite linear sequences of worlds (propositional interpretations), each one reachable from the initial world by a finite sequence τ of actions in the alphabet Σ . More precisely, a model $M = (\sigma, V)$ consists of an infinite sequence of actions σ over Σ (the sequence of actions executed from the initial world) and a valuation function V , defining the interpretation of propositions at each world τ (where τ is a prefix of σ).

In the following, we denote by $\text{prf}(\sigma)$ the set of all finite prefixes of σ (the worlds) and, for each regular program π , we denote by $[[\pi]]$ the set of finite sequences associated with π . Given a model $M = (\sigma, V)$, a finite word $\tau \in \text{prf}(\sigma)$ and a formula α , the *satisfiability of a formula α at τ in M* , written $M, \tau \models \alpha$, is defined as usual for the classical connectives. Moreover:

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

A formula $\alpha \mathcal{U}^\pi \beta$ is true at a world 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 $[\pi] \alpha \equiv \neg \langle \pi \rangle \neg \alpha$. When π is Σ^* (representing all finite actions sequences), we replace $\langle \pi \rangle$ with $\diamond$ and $[\pi]$ with $\square$. Furthermore, the $\bigcirc$ (next) operator can be defined as $\bigcirc \alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$. As shown in [15], $\text{DLTL}(\Sigma)$ is strictly more expressive than $\text{LTL}(\Sigma)$. The satisfiability and validity problems for DLTL are PSPACE complete problems [15].

In this paper, we make use of the Product version of DLTL , $\text{DLTL}^\otimes$ [14], which allows to describe the behavior of a network of sequential agents which coordinate their activities by performing common actions together. There are k agents $1, \dots, k$, and a *distributed alphabet* $\tilde{\Sigma} = \{\Sigma_i\}_{i=1}^k$, a family of (possibly non-disjoint) alphabets, with each Σ_i a non-empty, finite set of actions (Σ_i is the set of actions which require the participation of agent i).

Atomic propositions are introduced in a local fashion, by introducing a non-empty set of atomic propositions $\mathcal{P}$. For each atomic proposition $p \in \mathcal{P}$ and agent i , p_i represents the “local” view of the proposition p at i , and is evaluated in the local state of agent i . The formulas of the language are obtained as the boolean combination of the formulas of $\text{DLTL}_i^\otimes$ which can be constructed on the alphabet (actions and propositions) of each agent i , using the modalities $\mathcal{U}_i^\pi$, $\langle \pi \rangle_i$, $[\pi]_i$, $\bigcirc_i$, $\diamond_i$ and $\square_i$. A $\text{DLTL}^\otimes$ model is a pair (σ, V) , where $V = \{V_i\}_{i=1}^k$ is a family of valuation functions, one for each agent i . The satisfiability of the formulas of $\text{DLTL}_i^\otimes$ is evaluated by making use of V_i and of the projection $\sigma|_i$ of σ to Σ_i (where $\sigma|_i$ is the sequence obtained by erasing from σ all occurrences of symbols that are not in Σ_i).

We illustrate how a protocol can be specified in this framework trough the specification of a *Purchase protocol*.

Example 1. We have three roles: the *merchant* (mr), the *customer* (ct) and the *bank* (bk). ct sends a request to mr ; mr replies with an offer or by saying that the requested good is not available. If ct receives the offer, it may either accept the offer and send a payment request to bk , or refuse the offer. If ct accepts the offer, then mr delivers the goods. If ct requires bk to pay mr , bk sends the payment. ct can send the request for payment to bk even before it has received the goods.

In this example, all actions are communicative actions: *sendRequest*, *sendOffer*, *sendNotAvail*, *sendAccept*, *sendRefuse*, *sendPaymentReqest*, *sendPayment*, each

one belonging to the action alphabet of the sender and of the receiver. For instance, action *sendRequest* is both in Σ_{ct} (as *ct* is the sender of the request) and in Σ_{mr} (as *mr* is the receiver of the request). Communication is synchronous: roles communicate by synchronizing on the execution of communicative actions.

The Purchase protocol *Pu* is given by specifying separately the protocols of the three participating roles: P_{ct} , P_{mr} and P_{bk} . The role P_i in the protocol is specified by a *domain description* D_i , which is a pair $(\Pi_i, \mathcal{C}_i)$, where Π_i is a set of formulas describing the effects and preconditions of the actions (the action theory) of role i , and $\mathcal{C}_i$ is a set of *constraints* that the executions of role i must satisfy. The approach is a generalization of the one proposed in [13].

Let us define, for instance, the *domain description* $D_{mr} = (\Pi_{mr}, \mathcal{C}_{mr})$ of the merchant. We adopt a social approach where an interaction protocol is specified by describing the effects of communicative actions on the social state. The social state contains the domain specific fluents describing observable facts concerning the execution of the protocol (*request*, the customer has requested a quote, *accepted*, the customer has accepted the quote, etc.), but also special fluents to model *commitments* (and conditional commitments) among the roles [22,20]: $C(i, j, \alpha)$ says that role i is committed to role j to bring about α . Furthermore, a *conditional commitments* $CC(i, j, \beta, \alpha)$ says that role i is committed to role j to bring about α , if the condition β is brought about.

The action theory Π_{mr} consists of *action laws*, *causal laws*, *precondition laws*, and an *initial state*. *Action laws* $\mathcal{AL}_{mr}$ describe the effects of the execution of actions on the state. The action laws:

$$\begin{aligned} &\Box_{mr}([sendOffer]_{mr}(offer \wedge CC(mr, ct, accept, goods))) \\ &\Box_{mr}(request \rightarrow [sendOffer]_{mr}\neg request) \end{aligned}$$

say that: when *mr* sends the quote for the good, then it commits to send the goods if *ct* accepts the request; and when *mr* sends the quote for the good, if there is a request, the request is cancelled.

Causal laws $\mathcal{CL}_{mr}$ are intended to express “causal” dependencies (or ramifications) among fluents. In this framework they are used to rule the dynamics of commitments. For instance, the *causal law*:

$$\Box_{mr}(\bigcirc_{mr}\alpha \rightarrow \bigcirc_{mr}\neg C(i, j, \alpha))$$

says that a commitment to bring about α is cancelled when α holds. Other causal laws are needed for dealing with conditional commitments.

Precondition laws $\mathcal{PL}$ have the form: $\Box(\alpha \rightarrow [a]\perp)$, meaning that the execution of an action a is not possible if α holds. The precondition law

$$\Box_{mr}(\neg request \rightarrow [sendOffer]_{mr}\perp)$$

says that an offer cannot be sent if a request has not been issued.

The *initial state* $\mathcal{IS}$ of the protocol defines the initial value of all the fluents. Here, we assume that the initial state is complete.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by action execution are assumed to persist unaltered

to the next state. To cope with the *frame problem* [18] we use a completion construction $Comp$, which is applied given a domain description, introduces frame axioms in the style of the successor state axioms proposed by Reiter [19]. The completion construction $Comp$ is only applied to the action laws and to the causal laws [13]. Thus Π_{mr} is defined as $Comp(\mathcal{AL} \wedge \mathcal{CL}) \wedge \mathcal{PL} \wedge \mathcal{IS}$.

The second component $\mathcal{C}_{mr}$ of the domain description P_{mr} defines constraints as arbitrary temporal formulas of $DLTL_{mr}$. For instance, to model the fact that the merchant cannot send more than one offer, we introduce the constraint: $\neg \Diamond_{mr} \langle sendOffer \rangle_{mr} \Diamond_{mr} \langle sendOffer \rangle_{mr} \top$.

We are interested in those execution of the purchase protocol in which all commitments of all the roles have been fulfilled. In particular, for each commitment $C(i, j, \alpha)$ of which the merchant is a debtor or a creditor (i.e., $i = mr$ or $j = mr$), we add in $\mathcal{C}_{mr}$ the constraint: $\Box_{mr}(C(i, j, \alpha) \rightarrow \Diamond_{mr}(\alpha \vee \neg C(i, j, \alpha)))$, meaning that a commitment has to be fulfilled unless it is cancelled.

Given $D_{mr} = (\Pi_{mr}, \mathcal{C}_{mr})$ as defined above, we let $P_{mr} = \Pi_{mr} \wedge \mathcal{C}_{mr}$. Once the protocols P_{ct} , P_{mr} and P_{bk} have been defined, the specification Pu of the Purchase protocol can be given as: $Pu = P_{ct} \wedge P_{mr} \wedge P_{bk}$. The *runs* of the protocol are then defined to be the linear models of Pu . They are all the runs that can be obtained by interleaving the actions of the runs of P_{ct} , P_{mr} and P_{bk} , while synchronizing on common actions. By projecting the runs of the protocol Pu to the alphabets of the participating roles, we get runs of each role P_{ct} , P_{mr} and P_{bk} . As the properties we will consider in this paper regard only the sequence of communicative action exchanged between agents, in the following, we will consider protocol runs as infinite sequences of actions, and disregard worlds.

Note that protocol runs are always infinite, as logic $DLTL$ is characterized by infinite models and we assume that all agents have infinite runs. Therefore, infinite protocols can be easily modelled in our framework. For instance, to model the infinite protocol in which the customer repeatedly issues a request, we can add to the specification above the constraint: $\Box_{ct} \Diamond_{ct} \langle sendRequest \rangle_{ct} \top$, requiring that the message request is issued infinitely many times. When we want to model terminating protocols, as the Purchase protocol above, we assume the domain description of each role of the protocol to be suitably extended with an action $noop_i$ which does nothing and which can be executed forever after termination of the protocol. Hence, in the following, we will assume that, for all i , the i -th projection of a run σ of a protocol P is an infinite run $\sigma|_i$ of P_i .

A protocol specification similar to the one above has been used in [12] to deal with problem of service composition. As a difference, here we use the product version of $DLTL$ so to specify the role of each role P_i independently, while in [12] a global specification of the protocol was given, including all roles.

A protocol defined as above, is not guaranteed to be well behaved, and we must impose some constraints on its structure to guarantee that the different roles participating in the protocol interoperate. To define the interoperability of a set of roles, we will put conditions on the computations of the interacting roles, which can be described by making use of Büchi automata.

2.1 Automata Based Verification

As usual for LTL, a DLT formula can be mapped into a Büchi automaton so that the accepting runs of the automaton correspond to the models of the formula.

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 $\sigma \in \Sigma^\omega$ be an infinite word on Σ . Then a run of $\mathcal{B}$ over σ is a mapping $\rho : prf(\sigma) \rightarrow Q$ such that $\rho(\varepsilon) \in Q_{in}$ and $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ 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 $\tau \in prf(\sigma)$. Informally, a run is accepting if it goes infinitely many times through a final state.

As described in [15], the satisfiability problem for DLT can be solved in deterministic exponential time, as for LTL. Given a domain description $(\Pi, \mathcal{C})$, a corresponding Büchi automaton can be obtained such that all runs accepted by the automaton represent runs of the protocol, and vice versa. An algorithm for constructing on-the-fly a Büchi automaton from a DLT formula has been proposed in [10], by generalizing the tableau-based algorithm for LTL. In general, this Büchi automaton is non-deterministic.

Let $P = P_1 \wedge \dots \wedge P_k$ be a protocol such that each role P_i is specified by a nondeterministic Büchi automaton $\mathcal{M}_i = (Q_i, \rightarrow_i, Q_i^{in}, F_i)$, whose accepting runs provide all the correct executions of the role. We assume that these automata have been “pruned” by eliminating all the states which do not occur on any accepting run. This can be achieved by starting from the accepting states, and by propagating backwards the information on the states for which a path to an accepting state exists.

The interactions of $P_1, \dots, P_k$ can be described by the runs of a product automaton $\mathcal{M}$ over $\tilde{\Sigma}$. The states of $\mathcal{M}$ will be k-tuples $\langle s_1, \dots, s_k \rangle$ of states of the automata $\mathcal{M}_1, \dots, \mathcal{M}_k$. Let $Q^{\mathcal{M}} = Q^1 \times \dots \times Q^k$ be the set of global states of $\mathcal{M}$. The transitions of $\mathcal{M}$ correspond to the execution of shared actions. The i-local transition relations induce a global transition relation $\rightarrow^{\mathcal{M}} \subseteq Q^{\mathcal{M}} \times \Sigma \times Q^{\mathcal{M}}$ as follows:

$$q \xrightarrow{a}_{\mathcal{M}} q' \text{ iff } q[i] \xrightarrow{a}_i q'[i], \text{ for each } i \in Ag(a)$$

$$q[i] = q'[i], \text{ for each } i \notin Ag(a)$$

where $q[i]$ denotes the i th component of $q = \langle q_1, \dots, q_k \rangle$, and $Ag(a)$ is the set of agents sharing action a . Moreover, $Q^{in} \subseteq Q_1^{in} \times \dots \times Q_k^{in}$ is the set of global initial states of $\mathcal{M}$.

A run of $\mathcal{M}$ over $\sigma \in \Sigma^\infty$ is a mapping $\rho : \text{prf}(\sigma) \rightarrow Q^{\mathcal{M}}$ such that $\rho(\varepsilon) \in Q^{\text{in}}$ and $\rho(\tau) \xrightarrow{a}_{\mathcal{M}} \rho(\tau a)$ for each $\tau a \in \text{prf}(\sigma)$. The run is *accepting* if, for all $i = 1, \dots, k$, $\sigma|_i$ is infinite and $\rho(\tau)[i] \in F_i$ for infinitely many $\tau \in \text{prf}(\sigma)$. The runs of $\mathcal{M}$ describe the interleaving of the executions of $P_1, \dots, P_k$, synchronizing on common actions.

Observe that, in an accepting run of $\mathcal{M}$, each $\mathcal{M}_i$ is executing a loop containing at least an accepting state in F_i . We will call such a loop in $\mathcal{M}_i$ an *accepting loop for P_i* . Vice-versa, a loop in $\mathcal{M}_i$ which does not contain any state in F_i will be called a *non accepting loop for P_i* . In the following section we introduce a notion of interoperability of a set of roles $P_1, \dots, P_k$ by putting conditions on (accepting and non accepting) runs of $\mathcal{M}$ and of the $\mathcal{M}_i$'s.

3 Interoperability

Let us consider an alternative specification P'_{ct} of the customer role, according to which, after sending a request, the customer has to wait for an offer and it does not expect to receive from the merchant the answer “goods is not available”. In such a case, the customer role P'_{ct} would not interact properly with the merchant role P_{mr} as defined in the previous section. If the merchant, after receiving a request from the customer, chooses to reply with *sendNotAvail*, the computation gets stuck, as the customer role P'_{ct} cannot receive this message.

This example shows that, as the different roles of the protocol are defined separately, some requirement is needed to guarantee that such roles interact properly, so that the protocol as a whole is well defined. In particular, the interaction of the roles in the protocol should not produce deadlock. Similarly, we want to avoid infinite executions in which some role P_i of the protocol is not executing an accepting run as, either, from some point onwards, P_i does not execute any action, or P_i is executing infinitely many actions, but on a non accepting run for P_i .

In the following, given the roles $P_1, \dots, P_k$ of a protocol, as introduced in Section 2, we say that roles $P_1, \dots, P_k$ are *interoperable* when they are free of choosing their actions at each step avoiding deadlock and non accepting executions. Let us point out that here we are considering in two different ways the nondeterministic choices concerning emissions (the customer can accept or refuse an offer) and those concerning receptions (the customer can receive the messages *sendOffer* or *sendNotAvail*). As usual in agent and web service applications, we assume that, in the first case, the choice is internal to the role (internal non determinism), while, in the second case, the choice is external to the agent and depends on the environment, namely on the interleaving of actions of partner agents (external non determinism). Hence, we postulate that a role can choose which message to send among the messages it can send in a state (the customer can decide whether to accept or refuse an offer), but that it cannot choose which message to receive among the messages he can receive in a state (the customer waits for *sendOffer* or *sendNotAvail*, but it cannot choose which one it will receive).

We can now define a notion of interoperability. We will denote by $m(i, j)$ the communicative action m sent from i to j . Let $\pi_i = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_v} q_v$ be the prefix of a run of $\mathcal{M}_i$. To model the fact that each P_i must be able to choose which action to execute after π_i , we introduce a function $choice(P_i, \pi_i)$, whose value is defined as follows: either $choice(P_i, \pi_i) = m(i, j)$, where $m(i, j)$ is a send action that can be executed after π_i on an accepting run of $\mathcal{M}_i$ (i.e., there is an accepting run of P_i with prefix $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_v} q_v \xrightarrow{m(i, j)} q_{v+1}$); or the value $choice(P_i, \pi_i) = receiveR$, where $R = \{m_1(j_1, i), \dots, m_n(j_n, i)\}$ contains all the receive actions that can be executed after π_i on an accepting run of $\mathcal{M}_i$. In the last case, P_i expects to receive a message from another agent after π_i but it doesn't know which one it will receive among those messages $\{m_1(j_1, i), \dots, m_n(j_n, i)\}$ it is able to receive after π_i . Observe that the choice of agent P_i in π_i may depend on the state q_v but also on the sequence of actions $a_1, a_2, \dots, a_v$ executed by P_i up to q_v .

While we have assumed that agents can choose among the messages they can send, we have postulated that they cannot decide which message they will receive among those they are able to receive in a given state.

As a matter of notation, in the following, we say that σ is an *execution of* P when there is a (not necessarily accepting) run $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots$ of $\mathcal{M}$ over $\sigma = a_1, a_2, \dots$ (representing a possible computation of $P_1, \dots, P_k$). We say that π is a *finite execution of* P when π is a finite prefix of an execution of P . We say that σ is an *accepting run of* P when there is an accepting run of $\mathcal{M}$ over σ .

We say that σ is an execution of $P_1 \dots P_k$ *respecting the function "choice"* if σ is an execution of $P_1 \dots P_k$ such that, for each prefix $\pi m(i, j)$ of σ , it holds that $choice(P_i, \pi|_i) = m(i, j)$ and $choice(P_j, \pi|_j) = receiveR$, with $m(i, j) \in R$.

We say that a *choice function is fair for* P_i if there is no execution of P_i respecting the choice function, in which P_i executes infinitely many times a non-accepting loop of $\mathcal{M}_i$, although there is a send action $m(i, j)$ that P_i can execute in some state of the loop leading outside the loop. A choice function is *fair* if it is fair for all the P_i 's. We say that σ is a *fair execution of* $P_1 \dots P_k$ if σ is an execution of $P_1 \dots P_k$ respecting a fair choice function.

In essence, according to a fair choice function, role P_i cannot choose to execute infinitely many times a non accepting loop, if it can exit the loop by executing a send action: P_i cannot be willing to execute infinitely many times a non-accepting loop. The interoperability of a set of roles is defined as follows:

Definition 1. $P_1 \dots P_k$ are interoperable if the following conditions hold:

- (i) For any function $choice$ and any finite execution π of $P_1 \dots P_k$, there exists an action $m(i, j)$, such that $choice(P_i, \pi|_i) = m(i, j)$ and $choice(P_j, \pi|_j) = receiveR$ (with $m(i, j) \in R$), and $m(i, j)$ is the first action executed after π on an execution σ of $P_1 \dots P_k$ with prefix π^1 .
- (ii) For each fair $choice$, each infinite execution σ of $P_1 \dots P_k$ respecting $choice$, is an accepting run of $P_1 \dots P_k$.

¹ Remember that $\pi|_i$ is the projection of π on the alphabet of P_i .

According to the above definition, any prefix obtained by the execution of $P_1, \dots, P_k$ can be extended by executing a new action according to the *choice* function. In particular, each agent can choose which action it wants to execute at each stage of the computation and, whatever the choice might be, the computation does not get stuck (condition(i)) and, eventually, each agent can execute his choice in an accepting run (condition (ii)).

Observe that the choice of a role P_i after a sequence π of execution steps only depends on the actions of P_i up to that point (namely, on $\pi|_i$). Hence, the choice of P_i is left unchanged by the execution of communicative actions not involving P_i . As a consequence, the choice of agent P_i remains unchanged until eventually it is fulfilled by the execution of the chosen action.

The conditions above guarantee that the choice of an agent cannot be delayed forever by the choices of other agents. The following proposition follows easily from the definition of interoperability above:

Proposition 1. (a) *For each fair choice functions, for each finite executions π of $P_1 \dots P_k$, if $\text{choice}(P_i, \pi|_i) = m(i, j)$ (or $\text{choice}(P_i, \pi|_i) = \text{receive}R$), then there is an accepting run σ of $P_1 \dots P_k$ respecting the choice function, such that σ has prefix $\pi\pi'm(i, j)$ (respectively, $\pi\pi'm(j, i)$, with $m(j, i) \in R$), where π' does not contain actions of P_i .*
 (b) *If $P_1 \dots P_k$ are interoperable, then $P_1 \dots P_k$ have an accepting run.*

Proof. By condition (i), the finite execution π can be extended to an infinite execution σ respecting the fair choice function. By condition (ii) the fair execution σ respecting the fair choice function is an accepting run of $P_1 \dots P_k$. As P_i must execute infinitely many actions on σ , it will eventually execute an action after π according to his choice in π . Namely, P_i will eventually execute $m(i, j)$ (respectively, $m(j, i)$) as its first action after π . The proof of point (2) is similar.

Let us consider the example in Figure 1. Role P_1 can repeatedly either send message $m(1, 2)$ to P_2 or message $m(1, 3)$ to P_3 . Role P_1 can go on choosing to send $m(1, 2)$ to P_2 , so that P_3 is not executing any action. This choice is fair. P_2 and P_3 have the only (fair) choice of executing the receive $m(1, 2)$ and the receive $m(1, 3)$, respectively. P_1, P_2, P_3 are not interoperable. According to the above fair choices, P_1 and P_2 can go on exchanging message $m(1, 2)$, and they

Fig. 1.

Fig. 2.

produce an execution σ which is an accepting run of P_1 and of P_2 , but is not an accepting run of P_3 (P_3 does not execute any action on σ). σ is a fair execution of P_1, P_2, P_3 , but it is not a run of P_1, P_2, P_3 . Condition (ii) fails.

As a different example, let us consider the example in Figure 2. The runs of P_2 are those sequences obtained by repeating infinitely many times: a (nonempty) finite sequence of actions $m_1(1, 2)$ followed by an action $m_2(1, 2)$. P_1 and P_2 can go on by exchanging the message $m_1(1, 2)$, thus producing a fair execution σ' (in each state, P_2 has the fair choice of executing a receive, but it cannot control which message is received). While for P_1 σ' is an accepting run, for P_2 it is not. Although P_2 executes infinitely many actions in σ' , it does not execute a run. P_1 and P_2 are not interoperable, as condition (ii) is not satisfied.

4 Conformance

Let $A_1, \dots, A_k$ be a set of agents. The specification of each agent A_i can be either given through a logical specification as the one introduced in Section 2 for roles, or by introducing the automaton describing the possible behaviors of the agent. We assume that the actions of each agent A_i are deterministic and that the automaton describing its possible behaviors is a deterministic Büchi automaton. The notions of *execution* of $A_1, \dots, A_k$ and of *accepting run* of $A_1, \dots, A_k$ are defined as for $P_1, \dots, P_k$ in Section 3.

Given a protocol $P = P_1 \wedge \dots \wedge P_k$ with k roles, we define the *conformance* of a set of agents $A_1, \dots, A_k$ with P , as follows:

Definition 2. *Agents $A_1, \dots, A_k$ are conformant with P if:*

- (a) $A_1, \dots, A_k$ are interoperable, and
- (b) all accepting runs of $A_1, \dots, A_k$ are accepting runs of the protocol P .

In this section we want to introduce a notion of *conformance of a single agent A_i with the protocol P* , so that the conformance of each A_i , proved independently, guarantees the conformance of the overall set of agents $A_1, \dots, A_k$ with P , according to Definition 2.

Given the definition of interoperability given in the previous section, this notion of conformance can be based on the policy: *less emissions and more receptions* [4,2,11]. Consider, for instance, a customer agent A_{ct} whose behavior differs from that of the role “customer” of protocol P_u as follows: whenever it

receives an offer from the merchant, it always accepts it; after accepting the offer it expects to receive from P_{mr} either *sendGoods* or *cancelDelivery*. Although the behavior of A_{ct} and that of the corresponding role of the protocol are different, we could consider however the agent to be conformant with the protocol, since the customer can choose which messages to send, and thus it is not forced to send all the messages required by the protocol. Also, the agent can receive more messages than those required by the protocol, since these receptions will never be executed.

Let $\mathcal{M}_i$ be the automaton specifying role P_i and $\mathcal{M}_i^A$ the automaton specifying agent A_i . We can formulate the policy “less emissions and more receptions” with the following condition. Let π be a finite execution common to A_i and P_i :

(C1) [Correctness of emissions of A_i]

If $\pi m(i, j)$ is a finite execution of A_i (where $m(i, j)$ is a send action of A_i), then $\pi m(i, j)$ is a finite execution of P_i .

(C2) [Completeness of receptions of A_i]

If $\pi m(j, i)$ is a finite execution of P_i (where $m(j, i)$ is a receive action of P_i), then $\pi m(j, i)$ is a finite execution of A_i .

Unfortunately, the policy “less emissions and more receptions” only works for two-party protocols, as shown in the next example.

Example 2. Consider protocol Pu . The customer ct , at some point, may accept the offer of mr and require bk to send the payment to mr . Assume that mr has the requirement that it can receive the payment from bk only after it has received the acceptance of the offer from ct . Namely, the specification of P_{mr} contains the constraint:

$$(*) \quad \Box_{mr} (\neg \text{accepted} \rightarrow [\text{sendPayment}]_{mr} \perp).$$

According to the protocols of ct and bk the message *sendPayment* can be sent from bk to mr either before or after the message *sendAccept* is sent from ct to mr . It is clear that, although ct and bk do not put constraints on the order in which they send the acceptance of the offer and the payment to mr , in the overall protocol Pu they are forced to respect the constraint of the merchant, and only the runs in which *sendAccept* is executed before *sendPayment* are accepted as runs of Pu .

Let us now consider an agent A_{mr} , playing the role of the merchant, whose behavior is the following: either it receives a message *sendAccept* followed by a message *sendPayment*, or receives a message *sendPayment* followed by a message *sendAccept*. Agent A_{mr} allows for more receptions with respect to its role P_{mr} (in fact, A_{mr} receives the additional message *sendPayment* followed by a message *sendAccept*). When A_{mr} interacts with a customer and a bank agents behaving as stipulated by P_{ct} and P_{bk} , it may produce an execution in which *sendPayment* comes before *sendAccept*, which is not a run of protocol Pu (as it does not satisfy the constraint $(*)$ above). Conversely, if we consider a variant Pu' of the Pu protocol in which the roles of the bank P'_{bk} and of the customer P'_{ct} coordinate their executions so that they execute *sendAccept* before

sendPayment, the interaction of agent A_{mr} with the other roles cannot produce the unwanted execution in which *sendPayment* comes before *sendAccept*.

In this example, A_{mr} can be regarded to be non-conformant with protocol Pu , but conformant with the protocol Pu' , although P_{mr} is the same in both protocols. The case in which A_{mr} is non-conformant with Pu , is similar to the example discussed in [17], where the problem of conformance checking is analyzed for models of asynchronous message passing software. The solution adopted in [17] is that of requiring that an agent A_i cannot do more receptions than those established by protocol P_i , so that: A_i can do less emission and exactly the same receptions as stated by P_i . In the example above, this would correspond to take A_{mr} as being non-conformant. We believe that this policy is too restrictive: as we see from Example 2, the conformance of A_{mr} depends on the overall protocol, including other roles. In the following, we propose a definition of the conformance of an agent A_i with respect to the overall protocol P , rather than to its role P_i . We are ready to admit additional receptions in A_i , if we are sure that such receptions cannot give rise to unwanted executions when A_i interacts with agents respecting protocol P .

In the following, besides referring to the executions and runs of a protocol P , we need to refer to the executions and runs of an agent A_i in the context of the protocol P . We will denote by $P[A_i]$ the set of roles/agents $P_1, \dots, P_{i-1}, A_i, P_{i+1}, \dots, P_k$. $P[A_i]$ represents the protocol obtained from P by replacing role P_i with agent A_i . Also, we will refer to the executions (accepting runs) of $P_1, \dots, P_{i-1}, A_i, P_{i+1}, \dots, P_k$ as executions (accepting runs) of $P[A_i]$.

To guarantee that an agent A_i is conformant with a protocol P , we need to introduce, besides (C1) and (C2) above, further conditions which ensure that A_i interacts properly with the other roles in the protocol P :

- (C3) [*Interoperability*] $P[A_i]$ interoperate
- (C4) [*Correctness of the receipts of A_i in the context of P*] All accepting runs of $P[A_i]$ are accepting runs of P .

Condition (C3) says that $P[A_i]$ is interoperable, that is A_i interacts with $P_1, \dots, P_{i-1}, P_{i+1}, \dots, P_n$ so that each role can make its choices without the computation getting stuck or ending up in non accepting loops.

Condition (C4) requires that the executions of A_i are *correct* when A_i is interacting with other agents respecting the protocol P . In particular, this ensures that although A_i can execute more receptions than P_i , such additional receptions are not executed when A_i interacts with the other roles in P .

Observe that condition (C4) can be equivalently expressed in the logic by saying that the formula $P[A_i] \rightarrow P$ has to be valid.

According to the above definition of conformance, the merchant agent A_{mr} in Example 2 is conformant with Pu' , while A_{mr} is non-conformant with protocol Pu . In fact, although $Pu[A_{mr}]$ is interoperable (and, in particular, A_{mr} can interact with other agents executing the protocol without getting stuck), there is a run of $Pu[A_{mr}]$, in which *sendPayment* comes before *sendAccept*, that is not a run of Pu (and therefore (C4) is violated). This last case shows that the

interoperability of A_i with the other roles in P is not sufficient to guarantee the correctness of the resulting runs with respect to P .

Are conditions (C1) to (C4) sufficient to guarantee the conformance of an agent with a protocol? What we expect is that, given an interoperable protocol P and a set of agents $A_1, \dots, A_k$, if each agent A_i is conformant with P (according to (C1)...(C4)) then the agents $A_1, \dots, A_k$ interoperate and their accepting runs are runs of P .

Consider the roles P_1 and P_2 and the agents A_1 and A_2 in Figure 3. P_1 and P_2 interoperate. A_1 has the same receptions and less emissions than P_1 (the send action $m_3(1,2)$ is not present in A_1). A_1 is conformant with P (according to conditions (C1),..., (C4)). In particular, A_1 and P_2 interoperate: for any fair choice function, P_2 cannot produce the infinite execution $\sigma = m_1(1,2), m_2(2,1), m_1(1,2), m_2(2,1), \dots$, as P_2 must eventually execute the send action $m_4(2,1)$. Similarly, A_2 is conformant with P and it has the same receptions and less emissions than P_1 (it misses the send action $m_4(2,1)$). However, A_1 and A_2 do not interoperate: A_1 and A_2 have the only choice of executing σ and this is a fair choice for A_1 and A_2 . However, σ is not a run of A_1, A_2 , and this violates interoperability condition (ii).

Observe that the choice of executing σ is fair for A_1 , as A_1 has not the choice of executing any action to exit the non accepting loop in σ . Instead, the execution σ is not fair for P_1 , as P_1 can choose to exit the non accepting loop by executing the send action $m_3(1,2)$. We introduce the following condition:

- (C5) For each σ execution of both A_i and P_i , if σ is a fair execution of A_i , then σ is a fair execution of P_i .

Condition (C5) is violated by A_1 (and by A_2) in Figure 3. The choice of executing σ is fair for A_1 , as there is no send action that agent A_1 can execute to exit from the non accepting loop. Instead, the choice of executing σ is not fair for P_1 which can execute action $m_3(1,2)$ to exit from the non accepting loop. The verification of condition (C5) requires reasoning on non accepting loops in the automata of A_i and P_i . The idea is that, although A_i can contain less

Fig. 3.

emissions than P_i , it must contain at least those emissions allowing to go out from non accepting loop and to go on with an accepting run, if this is possible for P_i .

Finally, we introduce the following conditions to deal with infinite computations, which requires that, when A_i and P_i produce a common action sequence, then either they both accept it or they both do not accept it.

- (C6) For all σ that are both executions of A_i and of P_i , σ is an accepting run of A_i if and only if σ is an accepting run of P_i .

The notion of conformance of an agent with a protocol is then defined as follows:

Definition 3. *[Conformance of agent A_i with protocol P] A_i is conformant with a protocol $P = P_1 \wedge \dots \wedge P_k$ if the conditions (C1), ..., (C6) above are satisfied.*

Let $P = P_1 \wedge \dots \wedge P_k$ be an interoperable protocol. P has at least one accepting run. Let an execution of $A_1, \dots, A_k$ be a run of the product automaton $\mathcal{A}$ defined as the product automaton $\mathcal{M}$ in Section 2.1. Let an accepting run of $A_1, \dots, A_k$ be an accepting run of the product automaton $\mathcal{A}$. We can prove that, given k agents $A_1, \dots, A_k$, if each A_i is conformant with P according to Definition 3, then the accepting runs of $A_1, \dots, A_n$ are accepting runs of protocol P and $A_1, \dots, A_n$ interoperate.

Theorem 1. *Let $A_1, \dots, A_k$ be k agents. If, for each $i = 1, \dots, k$, agent A_i is conformant with protocol P according to Definition 3, then agents $A_1, \dots, A_k$ are conformant with P according to Definition 2.*

In order to prove the result above, we prove the following lemmas. Lemma 1 says that any finite execution π of a set of agents $A_1, \dots, A_k$, with each A_i conformant with P , is a prefix of a run of P .

Lemma 1. *Assume that, for all $i = 1, \dots, k$, agent A_i is conformant with protocol $P = P_1 \wedge \dots \wedge P_k$ according to Definition 3. If there is a finite execution π of $A_1, \dots, A_k$, then there is an accepting run ρ of P with prefix π .*

Proof. We prove that there is an accepting run of P with prefix π by induction on the length l of π . If $\pi = \epsilon$, the theorem holds, as there exists an accepting run of P , by Proposition 1(b).

For the inductive case, let $\pi m(i, j)$ be an execution of $A_1, \dots, A_k$ of length $l + 1$. We show that there is an accepting run of P with prefix $\pi m(i, j)$. By inductive hypothesis, there is an accepting run of P with prefix π . As A_i can execute action $m(i, j)$ after $\pi|_i$, by the correctness of the send of A_i with respect to P_i , condition (C1), also P_i can execute action $m(i, j)$ just after $\pi|_i$.

From the hypothesis we know that A_j can receive $m(i, j)$ as its first action after π . As P_i can emit $m(i, j)$ as its first action after π , A_j and P_i can synchronize to execute $m(i, j)$ after π in $P[A_j]$. Hence, $\pi m(i, j)$ is a finite execution of $P[A_j]$. Let us consider any fair choice function such that the finite execution $\pi m(i, j)$

respects the choice function. By (C3) $P[A_j]$ interoperate and, by Proposition 1, $\pi m(i, j)$ can be extended to an accepting run ρ of $P[A_j]$ respecting the choice function. By (C4), ρ is also a run of P . Hence, there is an accepting run of P with prefix $\pi m(i, j)$. $\square$

Lemma 1 can be generalized to prove that, if there is a finite execution π of $P[A_1, \dots, A_h]$ (with $h \leq k$), then there is an accepting run ρ of P with prefix π . We can now prove the following interoperability result.

Lemma 2. *Assume that, for all $i = 1, \dots, k$, agent A_i is conformant with protocol P according to definition 3. If the roles of P , $P_1 \dots P_k$, are interoperable, then also $A_1 \dots A_k$ are interoperable.*

Proof. Assume that $P_1 \dots P_k$, are interoperable. To show that $A_1 \dots A_k$ are *interoperable*, we have to prove that $A_1 \dots A_k$ satisfy conditions (i) and (ii) of Definition 1.

Let us prove (i). Let $choice_A$ be any choice function for $A_1 \dots A_k$, and let π be any finite execution of $A_1 \dots A_k$. We want to prove that there exists an action $m(i, j)$, such that $choice_A(A_i, \pi|_i) = m(i, j)$ and $choice_A(A_j, \pi|_j) = receiveR$ (with $m(i, j) \in R$), and $m(i, j)$ can be executed by $A_1 \dots A_k$ just after π .

By Lemma 1, π is prefix of an accepting run σ of $P_1 \dots P_k$. Therefore, for all i , P_i has a run $\sigma|_i$ with prefix $\pi|_i$. We want to exploit the fact that $P_1 \dots P_k$ are interoperable and show that $choice_A$ can be taken as the choice function for the P_i 's in π . Let I be the set of all i such that $choice_A(A_i, \pi|_i) = m(i, j)$, that is, all i such that agent A_i wants to execute a send after π .

Let $i \in I$. As $choice_A(A_i, \pi|_i) = m(i, j)$, then A_i can execute $m(i, j)$ after $\pi|_i$ on some of its runs. By the correctness of the send of A_i with respect to P_i , condition (C1), also P_i can choose to execute the send action $m(i, j)$ after $\pi|_i$. Hence, we can take $m(i, j)$ as the choice of P_i after π .

Let $j \notin I$, that is $choice_A(A_j, \pi|_j) = receiveR$. We want to show that also P_j can choose to execute a receive action after π . As π is a finite execution of $A_1 \dots A_k$ and the prefix of an accepting run of $P_1 \dots P_k$, then π is a finite execution of $P[A_j]$. By (C3), $P[A_j]$ interoperate. Let us consider any fair choice function ch' for $P_1 \dots P_k$ such that $ch'(A_j, \pi|_j) = choice_A(A_j, \pi|_j) = receiveR$. By interoperability of $P[A_j]$ there is an accepting run ρ of $P[A_j]$ with prefix $\pi \pi' m(i, j)$, where $m(i, j) \in R$ and π' does not contain actions of agent j . By (C4) ρ is an accepting run of P . As P_i executes the receive $m(i, j)$ as its first action after π in ρ , P_i can choose to execute a receive after π .

Now, let $choice_P$ be a choice function for $P_1 \dots P_k$, such that $choice_P(P_i, \pi|_i) = choice_A(A_i, \pi|_i)$. By the interoperability of $P_1 \dots P_k$ there is an action $m(i, j)$ which can be executed by the P_i 's after π according to the choice function $choice_P$. Action $m(i, j)$ is the choice of agent A_i after π , so that A_i can send $m(i, j)$ after π . Since P_j receives message $m(i, j)$ after π , by the conformance of A_j to P , condition (C2), also agent A_j can receive message $m(i, j)$ after π . Hence, $m(i, j)$ can be executed by $A_1 \dots A_k$ after π according to the choice function $choice_A$. This concludes the proof of point (i).

Let us prove (ii). We have to prove that, for each fair *choice*, each infinite execution σ of $A_1, \dots, A_k$ respecting *choice*, is an accepting run of $A_1, \dots, A_k$.

Let σ be an infinite execution of $A_1, \dots, A_k$ respecting *choice*. It is easy to see that σ is also an execution of P . In fact, by Lemma 1 we know that each prefix π of an execution of $A_1, \dots, A_k$ is also a prefix of a run of P . Hence, σ is also an execution of $P_1, \dots, P_k$. Given condition (C5), the *choice* function, which is fair for $A_1, \dots, A_k$, must also be a fair choice function for $P_1, \dots, P_k$. As P is interoperable and the *choice* function is fair for $P_1, \dots, P_k$, by the interoperability condition (ii), σ is an accepting run of P . Observe that, for all i , $\sigma|_i$ is both an execution of A_i and of P_i . By condition (C6), as $\sigma|_i$ is an accepting run of P_i , $\sigma|_i$ must also be an accepting run of A_i . It follows that σ is an accepting run of $A_1, \dots, A_k$. $\square$

Lemma 3. *Assume that, for all $i = 1 \dots k$, agent A_i is conformant with protocol P according to definition 3. If σ is an accepting run of $A_1 \dots A_k$, then σ is an accepting run of P .*

Proof. By induction on h , we prove that: If σ is an accepting run of $P[A_1, \dots, A_h, A_{h+1}]$, then σ is an accepting run of $P[A_1, \dots, A_h]$. For $h = 0$, by (C4), if σ is an accepting run of $P[A_1]$, then σ is an accepting run of P .

For $h > 0$, assume that σ is an accepting run of $P[A_1, \dots, A_h, A_{h+1}]$ and that σ is not an accepting run of $P[A_1, \dots, A_h]$. It must be that $\sigma|_i$ is an accepting run of A_{h+1} , while $\sigma|_i$ is not an accepting run of P_{h+1} .

Observe that, by (C1), all the send actions of A_{h+1} in σ are correct and can be executed by P_{h+1} , but A_{h+1} could receive a message $m(j, h+1)$ in σ which cannot be received by P_{h+1} . If this is not the case, σ is an execution of $P[A_1, \dots, A_h]$ and, by (C6), σ is an accepting run of $P[A_1, \dots, A_h]$.

If there is a receive $m(j, h+1)$ in σ which is not executable by P_{h+1} , then there must be a prefix $\pi m(j, h+1)$ of an execution of $P[A_1, \dots, A_h, A_{h+1}]$ which is not a prefix of an accepting run of P (as it is not a prefix of an accepting run of P_i). This contradicts Lemma 1. $\square$

Observe that, although the interoperability of A_i with P (condition (C3)), guarantees that A_i is able to receive the messages sent to it from the other roles in P , it does not enforce condition (C2). If (C2) were omitted in Definition 3, Theorem 1 would not be provable. In fact, in such a case, it might occur that, although agent A_1 is interoperable with other roles in P and the same holds for agent A_2 , the two agents A_1 and A_2 do not interoperate with each other and with the other roles in P .

Let, for instance, P be an interoperable protocol with 4 roles, where each role has the following runs:

$$\begin{aligned} P_1 &: \{m(1, 3), m(1, 4)\} \\ P_2 &: \{m(2, 4), m(2, 3)\} \\ P_3 &: \{m(1, 3), m(2, 3) + m(2, 3), m(1, 3)\} \\ P_4 &: \{m(1, 4), m(2, 4) + m(2, 4), m(1, 4)\} \end{aligned}$$

Let us consider now the agents A_1, A_2, A_3, A_4 , such that $A_1 = P_1$, $A_2 = P_2$, while A_3 and A_4 have the following runs:

$$A_3 : \{m(2, 3), m(1, 3)\} \qquad A_4 : \{m(1, 4), m(2, 4)\}$$

It can be seen that A_3 satisfies conditions (C1) and (C3), but not (C2), and the same holds for A_4 . The agents A_1, A_2, A_3, A_4 do not interoperate, since A_1 and A_2 must emit respectively $m(1, 3)$ and $m(2, 4)$ as their first message, but none of them can be received by A_3 or A_4 .

Observe that some conditions in the definitions of interoperability and conformance can be easily verified, as for instance, the interoperability condition (i), which requires to check some conditions on all the states of the product automaton: from each state it must always be possible to execute some action according to the choices of the agents. This verification requires polynomial time in the size of the product automaton $\mathcal{M}$. However, the verification of other conditions, like the interoperability condition (ii) and of conditions (C4) and (C5) is rather complex. For instance, the verification of condition (ii) requires to execute a check on all loops in the automaton $\mathcal{M}$, that are non accepting loops for some role P_i . Also, the verification of condition (C4), that all the runs of $P[A_i]$ are runs of P , requires to check language inclusion between two non deterministic ω automata. This problem is, in the general case, PSPACE-hard [7].

We may wonder whether the addition of simplifying assumption may lead to a simplified definition of the notions of conformance and interoperability. Several simplifications naturally arise: the assumption that the agents have the same receptions as the corresponding roles; the assumption that the role and agent automata are deterministic; the assumption that all protocol are finite and that finite automata can be used rather than Büchi automata to model protocols.

Concerning the first simplification, if agent A_i has that same receptions as P_i , conditions (C1) and (C3) can be replaced by the single condition (C1') All the runs of A_i are runs of P_i stating that both emissions and receptions of A_i are correct with respect to P_i . Verifying this condition, requires to verify the validity of the entailment $A_i \rightarrow P_i$, which for DLTL (as for LTL) is a problem in PSPACE-hard. With this simplification however condition (C5) is still needed to guarantee interoperability, as shown, for instance by the example in Figure 3.

If we restrict our consideration to protocols described by deterministic Büchi automata, rather than to non deterministic ones, the verification of some condition becomes simpler (as, for instance, the verification of (C4)), although the conditions for interoperability and conformance remain unaltered.

In the case we are considering finite state automaton, some conditions, like (C6), are not needed. Also, the interoperability conditions (i) and (ii) gets simplified. However, to guarantee interoperability of the agents which are individually conformant with the protocol, some condition playing the role of (C5) is still needed, as shown by the example in Figure 3, which can be easily adjusted for finite state automata.

5 Related Work

The paper deals with the problem of agents conformance with multiparty protocols. The notion conformance we have introduced guarantees the interoperability of a set of agents which are conformant with the protocol. We have assumed that the specification of the protocol is given in a temporal action logic and we have shown that the verification of conformance can be done independently for each agents, by making use of automata-based techniques. The proposed approach deals with both terminating and non-terminating protocols.

The paper generalizes the proposal in [11], where a notion of conformance for two-party protocols has been defined based on the policy “less emissions and more receptions”. Such policy, however, is not sufficient to guarantee stuck-freeness in the multiparty case. Indeed, the notion of conformance we propose here requires in addition: 1) the correctness of the receipts an agent can do, when it is interacting with other roles of the protocol, and 2) the interoperability of the agent with the other roles of the protocol. The notion of conformance proposed here is also stronger than the notion of compliance in [13]. There, an agent A_i is said to be compliant with a protocol P if, in all interactions of A_i with other roles in the protocol, A_i satisfies its commitments and permissions. This condition essentially corresponds to the correctness condition (C2) in Definition 3. The notion of compliance in [13] does not guarantee stuck-freeness.

Several other proposals have been put forward in the literature for dealing with agent and agent conformance and interoperability.

In [4], several notions of *compatibility* and *substitutability* among agents have been analyzed, in which agents are modelled by Labelled Transition Systems, communication is synchronous, and models are deterministic. Substitutability is related to the notion of conformance. [4] introduces two distinct notions of substitutability (related with conformance): a first one, based on the policy “less emissions and more receptions”, which does not preserve deadlock-freeness, and a second more restrictive one requiring “the same emissions and receptions”. Agent executions are always terminating.

In [2] an automata based approach is used for conformance verification, by taking into account the asymmetry between messages that are sent and messages that are received. Agents and protocols are represented as deterministic finite automata, and protocols have only two roles. The approach has been extended to the multiparty case in [3], which also accounts for the case of nondeterministic agents and roles producing the same interactions but having different branching structures. Such a case cannot be handled in the framework in [2] as well as in our framework, due to the fact that our approach is based on a trace semantics. A similar approach is also used in [1], where an abductive framework is used to verify the conformance of agents to a choreography with any number of roles. As a difference with the above proposals, our proposal deals with protocols with infinite runs and guarantees stuck-freeness also in the multiparty case.

In [17] a notion of conformance is defined to check if an implementation model I conforms with a signature S , in the case both I and S are CCS processes, and communication is asynchronous. The policy “less emissions and the same

receptions" is introduced to guarantee stuck-freeness. Our approach provides a solution to guarantees stuck-freeness without requiring an agent to have the same receptions as its role in the protocol.

The notions of conformance, coverage and interoperability are defined in [5]. A distinctive feature of that formalization is that the three notions are orthogonal to each other. Conformance and coverage are based on the semantics of runs and relies on the notion of run subsumption, concerning the single agent and its role in the protocol. Interoperability among agents is based upon the idea of blocking and depends on the computation that the agents can jointly generate. The paper only considers two-party protocols and agents with finite runs.

In [6] a notion of constitutive interoperability is proposed, which "abstracts from the process-algebraic notion of interoperability and makes commitment alignment the sole criterion" so to capture the business meaning of the business processes. As said in [6], this notion is complementary to a notion of regulative interoperability, which takes into consideration message order, occurrence and data flow. The work we have presented here falls within the context of regulative interoperability.

In [16] Web Services are modelled as MAS and model checking is used for verifying temporal epistemic properties of OWL specifications. An OWL specification of a service is mapped to ISPL (the language of the model checker MCMAS), and a coloring of states as compliant (green) or non compliant (red) is assumed for the verification. The paper does not provide formalization of contracts (or protocols) and it does not provide techniques for automatically computing, for given a service, the states which are compliant or non compliant with a contract.

[8] focuses on the realizability problem of a framework for modeling and specifying the behaviors of *reactive electronic services*. In that framework, services communicate by asynchronous message passing, and are modeled by means of Büchi automata. The authors show that not every conversation protocol is realizable in the framework, and give some realizability conditions and show that each conversation protocol satisfying those conditions is realizable.

Acknowledgement

The work was partially supported by Regione Piemonte, Project ICT4LAW.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M.: An abductive framework for a-priori verification of web agents. In: Principles and Practice of Declarative Programming (PPDP 2006). ACM Press, New York (2006)
2. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Verification of protocol conformance and agent interoperability. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS, vol. 3900, pp. 265–283. Springer, Heidelberg (2006)
3. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: A Priori Conformance Verification for Guaranteeing Interoperability in Open Environments. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 339–351. Springer, Heidelberg (2006)

4. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are two web-agents compatible, VLDB-TES (2004)
5. Chopra, A.K., Singh, M.P.: Producing Compliant Interactions: Conformance, Coverage, and Interoperability. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS, vol. 4327, pp. 1–15. Springer, Heidelberg (2006)
6. Chopra, A.K., Singh, M.P.: Constitutive Interoperability. In: Proc. of the 7th Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 797–804 (2008)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
8. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
9. Giordano, L., Martelli, A., Schwind, C.: Verifying Communicating Agents by Model Checking in a Temporal Action Logic. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 57–69. Springer, Heidelberg (2004)
10. Giordano, L., Martelli, A.: Tableau-based Automata Construction for Dynamic Linear Time Temporal Logic. Annals of Mathematics and Artificial Intelligence 46(3), 289–315 (2006)
11. Giordano, L., Martelli, A.: Verifying Agent Conformance with Protocols Specified in a Temporal Action Logic. In: Basili, R., Paziienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 145–156. Springer, Heidelberg (2007)
12. Giordano, L., Martelli, A.: Web Service Composition in a Temporal Action Logic. In: 4th Int. Workshop on AI for Service Composition, AICS 2006, Riva del Garda, August 28 (2006)
13. Giordano, L., Martelli, A., Schwind, C.: Specifying and Verifying Interaction Protocols in a Temporal Action Logic. Journal of Applied Logic (Special issue on Logic Based Agent Verification) 5(2007), 214–234 (2007)
14. Henriksen, J.G., Thiagarajan, P.S.: A product Version of Dynamic Linear Time Temporal Logic. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 45–58. Springer, Heidelberg (1997)
15. Henriksen, J.G., Thiagarajan, P.S.: Dynamic Linear Time Temporal Logic. Annals of Pure and Applied logic 96(1-3), 187–207 (1999)
16. Lomuscio, A., Qu, H., Solanki, M.: Towards verifying compliance in agent-based web service compositions. In: AAMAS 2008, pp. 265–272 (2008)
17. Rajamani, S.K., Rehof, J.: Conformance checking for models of asynchronous message passing software. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 166–179. Springer, Heidelberg (2002)
18. Reiter, R.: Knowledge in Action. MIT Press, Cambridge (2001)
19. Reiter, R.: The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In: Lifschitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pp. 359–380. Academic Press, London (1991)
20. Singh, M.P.: A social semantics for Agent Communication Languages. In: Dignum, F.P.M., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 31–45. Springer, Heidelberg (2000)
21. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)
22. Yolum, P., Singh, M.P.: Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In: AAMAS 2002, Bologna, Italy, pp. 527–534 (2002)