Joint achievement of services' personal goals

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti, Claudio Schifanella
Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185, I-10149 Torino, Italy
Email: {baldoni, baroglio, emarengo, patti, schi}@di.unito.it

***Abstract*—Web service specifications can be quite complex, including various operations and message exchange patterns. In this work, we give a rule-based declarative representation of services, and in particular of WSDL operations, that enables the application of techniques for reasoning about actions and change, that are typical of agent systems. This makes it possible to reason on a rule-based specification of choreography roles and about the selection of possible role players on a goal-driven basis, and allows us to attack the problem of the joint achievement of individual goals for a set of services which animate a choreography.**

I. Introduction

Declarative languages are becoming very important in the Semantic Web, since the focus started to shift from the ontology layer to the *logic layer*, with a consequent need of expressing rules and of applying various forms of reasoning[1], an interest also witnessed by the creation of a W3C working group to define a Rule Interchange Format[2]. Particularly challenging applications concern *web services*.

One of the key ideas behind web services is that services should be amenable to automatic retrieval, thus facilitating their *re-use*. Nevertheless, retrieval cannot yet be accomplished automatically as well and as precisely as desired because the current web service representations (mainly WSDL [1] and BPEL [2]) and the discovery mechanisms are semantically poor and lack of sufficient flexibility. As shown by the framework presented in this paper, *rule-based declarative languages*, and the *reasoning techniques* that they support, supply both an expressive formal semantics and flexibility in the accomplishment of the service selection task.

The need of adding a *semantic layer* to service descriptions is not new. Some approaches, e.g. OWL-S [3] and WSMO [4], propose a richer annotation, aimed at representing *inputs*, *outputs*, *preconditions* and *effects* of the service. Inputs and outputs are usually expressed by ontological terms, while preconditions and effects are often expressed by means of *logic* representations. Preconditions and effects are also used in *design by contract*, originally introduced by Meyer for the Eiffel^{TM} language [5]. Here preconditions are the part of the contract which is to be guaranteed by the client; if this condition is guaranteed in the execution context of a method, then the server commits to guaranteeing that the postcondition holds in the state reached by the execution.

On the other hand, there is often the need of using services not in an individual way but jointly, for executing tasks that none of them alone can accomplish. Semantic annotations alone are not sufficient in this case; it becomes useful to introduce a notion of *goal* [6], [7], [8], which can be used to guide both the selection and the composition of services. The introduction of goals opens the way to the introduction of another abstraction, that of *agent*. Agents include not only the ability of dealing with goals and of performing goal-driven forms of reasoning, but they also show autonomy and proactivity, which are characteristics that help when dealing with open environments, allowing for instance a greater fault tolerance (see [9], [10], [11]).

In this work, we take the perspective of a candidate role player, willing to take part to an interaction, that is ruled by a choreography. We see such an entity as an *enhanced (proactive) service*, made of a declarative description of its behavior, of the operations that it can execute, of its goals [7], and with the capability of reasoning (on its own behavior) so to build *orchestations* that allow the achievement of its goals (proactiveness). So, for instance, a service that wishes to play the role of the "Seller" of a ticket-purchase choreography may have the aim of "selling tickets", guaranteed by playing the choreography role, and have the additional, *private goal* of "loyalizing clients". Notice that personal goals may vary along time, hence, different participations (to the same choreography of a same agent as player of the same role) may be characterized by different personal goals. For example, the flight ticket selling service may change at some point its personal goal, which becomes the "promotion of additional services" offered by the same company. The presence of additional goals, which may not be disclosed to the interacting parties, biases the behavior and the choices of a service. A service will take on a role only after checking the possible achievement of this final condition.

Orchestrations compose the interactions of *sets of* services. However, reasoning on the goals of the orchestrator alone is not sufficient. Indeed, in the context of choreographies that provide a pattern of interaction among services, each service involved has its own goals, that it tries to pursue. The research question is: even though each service has proved that there is a possible way of playing its role, which leads to the achievement of its personal goal, is it possible to guarantee the compatibility of these individual solutions? In other words, will the joint working plan, made of the identified individual solutions, allow the *joint achievement of all the goals* of the interacting parties? A ticket selling service that wants to do some advertisement by sending a newsletter to its clients will

[1] REWERSE NoE, `http://rewerse.net`
[2] `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`

not match the aims of a client which considers this kind of information as spam.

In this paper we describe the first steps of a study about the *joint achievement* of goals of a set of services, whose interaction is ruled by a choreography. The approach relies on a rule-based declarative representation of the choreography roles and of the interacting services, which extends and refines the work in [8], [12], [13], exploiting the results about conservative matching defined in there. The framework enables the application of techniques for reasoning on the effects of playing a role in a choreography, thus checking whether the desired personal goal can be accomplished. The representation is based on the logic programming language described in [14], [15]. Behaviors, as roles and service policies, build upon WSDL-like basic operations, represented as atomic actions with preconditions and effects.

The paper is organized as follows. Section II sets the representation of services and of choreographies that we adopt. Moreover, it explains how it is possible to reason on such a representation in order to allow each service to check the reachability of its goals locally, i.e. by using only the specification of the desired choreography role. Section III tackles the joint achievement of personal goals. A running example is distributed along the pages to better explain the proposed notions and mechanisms. Conclusions end the paper.

II. A THEORETICAL FRAMEWORK FOR REPRESENTING AND REASONING ABOUT SERVICES

In this section, we introduce the notation that we use to represent services and we discuss the problem of verifying a personal goal. The notation, which is a refinement of the proposal in [13], is based on a logical theory for reasoning about actions and change in a *modal logic programming setting* and on the language *DYnamics in LOGic*, described in details in [14]. This language is designed for specifying agents behaviour and for modeling dynamic systems. It is fully set inside the logic programming paradigm by defining programs by sets of Horn-like rules and giving a SLD-style proof procedure. The capability of reasoning about interaction protocols, supported by the language, has already been exploited for customizing web service selection and composition w.r.t. to the user's constraints, based on a semantic description of the services [14]. The language is based on a modal theory of actions and mental attitudes where modalities are used for representing primitive and complex actions as well as the agent beliefs. Complex actions are defined by inclusion axioms [16] and by making use of action operators from dynamic logic, like sequence ";" and test "?".

In this framework, the problem of reasoning amounts either to build or to traverse a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time, due to the application of actions. In general, we cannot assume that the value of each fluent in a state is known: we want to have both the possibility of representing unknown fluents and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent unknown fluents, we use an epistemic operator $\mathbf{B}$, to represent the beliefs an entity has about the world: $\mathbf{B}f$ means that the fluent f is known to be true, $\mathbf{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathbf{B}f$ and $\neg\mathbf{B}\neg f$ hold ($\neg\mathbf{B}f \wedge \neg\mathbf{B}\neg f$). Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*. For the sake of readability, we will add as a superscript of $\mathbf{B}$ the name of the service that has the belief.

Services exhibit interfaces, called port-types, which make a set of operations available to possible clients. In our proposal, a *service description* is defined as a pair $\langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O}$ is a set of basic operations, and $\mathcal{P}$ (*policy*) is a description of the complex behavior of the service. Analogously to what happens for OWL-S composite processes, $\mathcal{P}$ is built upon basic operations and tests, that control the flow of execution.

A. Basic operations

Let us start with *basic operations*. According to the main languages for representing web services, like WSDL and OWL-S, there are four basic kinds of operations [17] (or atomic processes, when using OWL-S terminology [3]):

- *one-way* involves a single message exchange, a client invokes an operation by sending a message to the service;
- *notify* involves a single message exchange, the client receives a message from the service;
- *request-response* involves the exchange of two messages, are initiated by the invoker of the operation, which sends a message to the service and, after that, waits for a response;
- *solicit-response* involves the exchange of two messages, the order of the messages is inverted w.r.t. a request-response, first the invoker waits for a message from the service and then it sends an answer.

A basic operation is described in terms of its *executability preconditions* and *effects*, the former being a set of fluents (introduced by the keyword **possible if**) which must be contained in the service state in order for the operation to be applicable, the latter being a set of fluents (introduced by the keyword **causes**) which will be added to the service state after the operation execution. Syntax for basic operations is:

$$\mathsf{operation}(content) \ \ \mathbf{causes} \ \ E_s \tag{1}$$

$$\mathsf{operation}(content) \ \ \mathbf{possible\ if} \ \ P_s \tag{2}$$

where E_s and P_s, denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while $content$ denotes possible additional data that is required by the operation. Notice that such operations can also be implemented as invocations to other services.

Operations, when executed, trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view, we distinguish between the case when the service is either the initiator (the operation *invoker*) or the servant of an operation (the operation *supplier*) by further decorating the operation name with a notation inspired by [18].

The two views are *complementary*, so if one view includes the act of sending a message, the other correspondingly includes a message reception. With reference to a specific service, $operation^{\gg}$ denotes the operation from the point of view of the invoker, while $operation^{\ll}$ denotes the operation from the point of view of the supplier. The view of operations that is used by *invoker* is given in terms of the operation inputs, outputs, preconditions, and effects as usual for semantic web services [3]. In the next part of this section, inputs and outputs are represented as single messages for simplicity but the representation can easily be extended to sets of exchanged data, as in Example (II-C). In this case, preconditions P_s in (2) and effects E_s in (1) are respectively the conditions required by the operation in order to be invoked, and the expected effects that result from the execution of the operation. For what concerns the view of the *supplier*, also in this case the operation is described in terms of its inputs and outputs. Moreover, we also represent a set of conditions that enable the executability of the operation. In order to distinguish them from the above, in this case we use R_s instead of P_s in (2), calling them *requirements*. Finally, we represent a set of conditions that constitute the *side effects* of the operation. In this case we use S_s instead of E_s in (1). For example, a *buy* operation of a selling service has as a precondition the fact that the invoker has a valid credit card, as inputs the credit card number of the buyer and its expiration date, as output it generates a receipt, and as effect the credit card is charged. From the point of view of the supplier, the requirement to the execution is to have an active connection to the bank, and the side effect is that the store availability is decreased while the service bank account is increased of the perceived amount.

Let us now introduce the formal representation of the four kinds of basic operations (for each operation we report both views) and of complex operations.

1) One-way: In one-way operations, the *invoker* requests an execution which involves sending an information m_{in} to the supplier; the invoker must obviously know the information to send before the invocation (a) (see Table I). The invoker can execute the operation only if the preconditions are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (c), and the invoker will know that it has sent an information to the supplier (b). Using OWL-S terminology, m_{in} represents the *input* of the operation, while P_s and E_s are its preconditions and effects. One-way operations have no output. On the other hand, the *supplier*, which exhibits the one-way operation as one of the services that it can execute, has the requirements R_s (e). The execution of the operation causes the supplier to know the information sent by the invoker (f). We also allow the possibility of having some side effects on the supplier's state. These effects are not to be confused with the operation effects described by IOPE, and have been added for the sake of completeness.

2) Notify: With ref. to Table I, in notify operations, the *invoker* requests an execution which involves receiving an information m_{out} from the supplier. The invoker can execute the operation only if the preconditions are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (c), and the invoker will know the received information (b). Using OWL-S terminology, m_{out} represents the *output* of the operation, while P_s and E_s are its preconditions and effects. Notify operations have no input. The *supplier* must meet the requirements R_s (e). The execution of the operation simply causes the fact that the supplier will know the message to send (f) and that it has sent some information to the invoker (g). As above, we allow the possibility of having some side effects on the supplier's state (h).

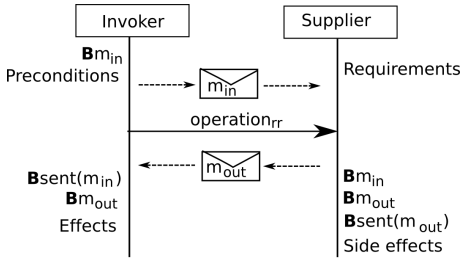

Fig. 1. The request-response basic operation.

3) Request-response: In request-response operations (see Figure 1), the *invoker* requests an execution which involves sending an information m_{in} (the input, according to OWL-S terminology) and then receiving an answer m_{out} from the supplier (the output in OWL-S). The invoker can execute the operation only if the preconditions P_s are satisfied in its current state and if it owns the information to send (a) (see Table I). The execution of the invocation brings about the effects E_s (d), and the fact that the invoker knows that it has sent the input m_{in} to the supplier (b). One further effect of the execution is that the invoker knows the answer returned by the operation (c). This representation abstracts away from the actual message exchange mechanism, which is implemented. Our aim is to reason on the effects of the execution on the mental state of the parties [15]. As for one-way operations, the *supplier* has the requirements R_s to the operation execution (e). It receives an input m_{in} from the invoker (f). The execution produces an answer m_{out} (g), which is sent to the invoker (h). As usual, it is possible to have some side effects on the supplier's state. On the supplier's side, we can notice more evidently the abstraction of the representation from the actual execution process. In fact, we do not model how the answer is produced but only the fact that it is produced.

4) Solicit-response: With ref. to Table I, in solicit-response operations, the *invoker* requests an execution which involves receiving an information m_{out} (the output, according to OWL-S terminology) and then sending a message m_{in} to the supplier (the input in OWL-S). The invoker can execute the invocation only if the preconditions P_s are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (e). The invoker receives a message m_{out} from the supplier (b) then, it produces the input information m_{in} which is sent to the supplier, see (c) and (d). As for notify operations, the

Operation	**Invoker's view**	**Supplier's view**
One-Way	(a) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$ (b) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ (c) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** E_s	(e) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **possible if** R_s (f) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ (g) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** S_s
Notify	(a) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **possible if** P_s (b) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ (c) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** E_s	(e) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **possible if** R_s (f) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ (g) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ (h) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** S_s
Request-response	(a) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$ (b) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ (c) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ (d) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** E_s	(e) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **possible if** R_s (f) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ (g) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ (h) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ (i) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** S_s
Solicit-response	(a) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **possible if** P_s (b) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ (c) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{in}$ (d) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ (e) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** E_s	(f) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **possible if** R_s (g) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ (h) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ (i) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ (l) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** S_s

TABLE I
THE REPRESENTATION OF THE FOUR KINDS OF BASIC OPERATIONS IN THE PROPOSED FRAMEWORK: EACH OPERATION IS GIVEN IN TERMS OF ITS PRECONDITIONS AND EFFECTS, FOR EACH OF THEM WE REPORT BOTH THE INVOKER'S VIEW AND THE SUPPLIER'S VIEW.

supplier must fulfill the requirements R_s (f). The execution causes the supplier to know the information to send (g) and that it has sent such information to the invoker (h). Moreover, it produces also the knowledge of the information m_{in} received from the invoker (i). Side effects on the supplier's state are allowed (l).

B. Service policies and choreography roles

The framework accounts also for *complex behaviors* that require the execution of many operations. A service policy $\mathcal{P}$ is a collection of clauses of the kind:

$$p_0 \ \ \mathbf{is} \ \ p_1, \ldots, p_n \tag{3}$$

where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action (operation), a test action (denoted by the symbol ?), or a procedure call. Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs, and their definitions can be non-deterministic as in Prolog. Complex behaviors are used also for representing choreography roles. Generally speaking, we represent a *choreography* $\mathcal{C}$ as a tuple $(R_1, \ldots, R_n)$ of interacting and complementary *roles*, each role R_i being a subjective view of the interaction that is encoded.

Also a role R is represented by a pair $\langle \mathcal{O}, \mathcal{P} \rangle$. The difference with services is that it composes *specifications* of operations and not implemented operations. The player of each role has to supply appropriate implementations. We call such specifications *unbound operations*. Formally, unbound operations are represented as normal basic operations, in terms of their preconditions and effects. This is one of the advantages of adopting a logic language: it allows handling implementations and specifications in the same way. In our case, both operations and operation specifications are given in terms of their preconditions and effects.

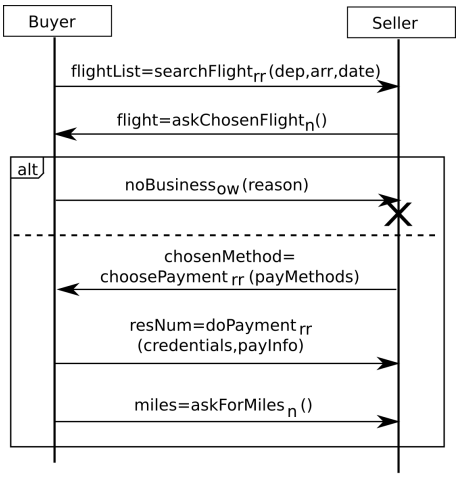

Fig. 2. A simple choreography for reserving a flight, expressed as a UML sequence diagram.

C. Flight-purchase

As an example, let's consider searchFlight, an operation of a flight reservation service, which is offered by a *seller* and can be invoked by a *buyer* to search information about flights with given departure (dep) and arrival locations (arr) plus the date of departure ($date$). From the point of view of the buyer, the operation, which is of kind request-response, is:

(a) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **possible if** $\mathbf{B}^{buyer} dep \wedge \mathbf{B}^{buyer} arr \wedge \mathbf{B}^{buyer} date \wedge \mathbf{B}^{\mathbf{buyer}} \neg \mathbf{sellingStarted}$

(b) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} sent(dep) \wedge \mathbf{B}^{buyer} sent(arr) \wedge \mathbf{B}^{buyer} sent(date)$

(c) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} flightList$

(d) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{\mathbf{buyer}} \mathbf{sellingStarted}$

The inputs of the operation are dep, arr, and $date$, while the output is $flightList$. In this case the set P_s contains only the belief $\mathbf{B}^{buyer} \neg sellingStarted$ (in bold text above) while the set E_s of effects contains the belief $\mathbf{B}^{buyer} sellingStarted$ (in bold text as well).

From the point of view of the supplier, instead, the operation is represented as:

(a) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **possible if** $true$

(b) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$

(c) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} flightList$

(d) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} sent(flightList)$

In this case the sets R_s and S_s of requirements and side effects are empty. The operation expects as input the departure and arrival locations and the date of the flight, and it produces a $flightList$, which it sends to its customer, so after the operation the belief $\mathbf{B}^{seller} sent(flightList)$ will be in its belief state.

buyTicket is, instead, an example of procedure implemented by a possible buyer service:

(a) buyTicket **is**
  $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$;
  $\mathsf{askChosenFlight}^{\ll}_{n}(flight)$;
  evaluateAndBuy.

(b) evaluateAndBuy **is**
  $\mathsf{noBusiness}^{\gg}_{ow}(reason)$.

(c) evaluateAndBuy **is**
  $\mathsf{choosePayment}^{\ll}_{rr}(payMethods, chosenMethod)$;
  $\mathsf{doPayment}^{\gg}_{rr}((credential, payInfo), resNum)$;
  $\mathsf{askForMiles}^{\gg}_{n}(miles)$.

First, it invokes an operation for searching flights that correspond to a given specification ($\mathsf{searchFlight}^{\gg}_{rr}$). After that, it waits for being asked about the preferred flight, chosen among the flights list obtained by the previous operation ($\mathsf{askChosenFlight}^{\ll}_{n}$). This evaluation can give either a negative outcome, hence the interaction is interrupted ($\mathsf{noBusiness}^{\gg}_{ow}$) or the interaction continues with the selection of the payment method ($\mathsf{choosePayment}^{\ll}_{rr}$). Then, the buyer performs the payment ($\mathsf{doPayment}^{\gg}_{rr}$) and, at the end, it is notified about the obtained miles ($\mathsf{askForMiles}^{\gg}_{n}$).

D. Reasoning on goals

In the outlined framework, it is possible to reason about personal goals by means of queries of the form Fs **after** p, where Fs is the *goal* (represented as a conjunction of fluents), that we wish to hold after the execution of a policy p. Checking if a formula of this kind holds corresponds to answering the query: "Is it possible to execute p in such a way that the condition Fs is true in the final state?". When the answer is positive, the reasoning process returns a sequence of atomic actions that allows the achievement of the desired condition. This sequence corresponds to an execution trace of the procedure and can be seen as a *plan* to bring about the goal Fs. This form of reasoning is known as *temporal projection*. Temporal projection fits our needs because, as mentioned in the introduction, in order to perform the selection we need a mechanism that verifies if a goal condition holds after the interaction with the service has taken place. Fs is the set of facts that we would like to hold "after" p.

Reasoning is done by exploiting a goal-directed proof procedure (denoted by "$\vdash$" in the following) designed for the language *DYnamics in LOGic* [15], [14], which supports both temporal projection and planning and allows the proof of existential queries of the kind reported above. The procedure definition constrains the search space. In particular, for what concerns planning, the proof procedure allows the automatic extraction of linear or conditional plans for achieving the goal of interest from an incompletely specified initial state.

Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a service description. The application of temporal projection to $\mathcal{P}$ returns, if any, an execution trace (a linear plan), that makes a goal of interest become true. Let us, then, consider a procedure p belonging to $\mathcal{P}$ and denoting its top level, and denote by Q the query Fs **after** p. Given a state s_0, containing all the fluents that we know as being true in the beginning, we denote the fact that the query Q is successful in the service description by $(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q$. The execution of the above query returns as a side-effect an *execution trace* σ of p; the execution trace σ is a sequence $a_1, \ldots, a_n$ of atomic actions. We denote this by:

$$(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q \text{ w.a. } \sigma \tag{4}$$

where "w.a." stands for *with answer*.

For example, suppose that the initial state of the service $b1$ is $s_0 = \{\mathbf{B}^{buyer} dep, \mathbf{B}^{buyer} arr, \mathbf{B}^{buyer} date, \mathbf{B}^{buyer} deferredPaymentPos, \mathbf{B}^{buyer} \neg sellingStarted, \mathbf{B}^{buyer} credentials\}$, (all the other fluents truth value is "unknown"). This means that $b1$ assumes a date, a departure location, an arrival location, the fact that it is

possible to defer the payment to the departure (at a desk at the airport), and that no selling process has started yet. The goal of $b1$ is to achieve the following condition: $Q = \{\mathbf{B}^{buyer} sellingComplete, \mathbf{B}^{buyer} resNum\}$ **after** buyTicket Intuitively, the buyer expects that, after the interaction, it will have a reservation number as a result.

By reasoning on its policy and by using the definitions of the unbound operations that are given by the choreography, $b1$ can identify an execution trace, that leads to a state where Q holds:

$\sigma =$ searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$;
askChosenFlight$^{\ll}_{n}(flight)$;
choosePayment$^{\ll}_{rr}(payMethods, chosenMethod)$;
doPayment$^{\gg}_{rr}((credential, payInfo), resNum)$;
askForMiles$^{\gg}_{n}(miles)$

This is possible because in a declarative representation specifications are executable. Moreover notice that this execution does not influence the belief about the deferred payment, which persists from the initial through the final state and is not contradicted.

III. Joint achievement of personal goals

So far, we have talked about goals, whose achievement motivates the participation of a service to a choreography. It is, in fact, reasonable that a service may decide of taking on a role only if by doing so it will have the possibility of satisfying its purposes. In our proposal a service takes this decision based on knowledge that includes a public role representation and a representation, given in terms of preconditions and effects, of its own operations, part of which will substitute some specifications contained in the role description. More generally, a choreography is made of many roles, that are played by different services, each of which has its own goal. The question we try to answer here is whether it is possible for this team of enhanced proactive services to reach an agreement about their possible executions so that in the team all of them will achieve their personal goals. As observed in [19], [20] the team can achieve a goal if there is a *joint working plan* that can achieve the goal.

For example, consider a choreography $\mathcal{C} = (R_1, R_2)$ and two services, interested to play respectively R_1 and R_2 for achieving the goals G_1 and G_2. By applying the described reasoning technique, the two services might both find that they can achieve their personal goals, one by following the execution trace σ_1, the other σ_2. The problem is that σ_1 and σ_2 might not be compatible, e.g. one invokes a $operation^{\gg}$, when the other side does not include the corresponding execution of $operation^{\ll}$.

In some cases, such compatible execution traces might not exist. In others, each party may have a set of alternative σ_i available, part of which are indeed compatible with executions traces identified by the partner. The problem, then, becomes to converge to a common solution that allows for the achievement of both goals. Let us discuss these issues, focusing on two-role choreographies. To this aim, let us introduce a few useful notions.

Definition 1 (Compatible execution traces): Let $\sigma = a_0; \ldots; a_n$ and $\sigma' = a'_0; \ldots; a'_n$ be two execution traces, we say that σ is *compatible* to σ' iff for each operation a_i in σ the corresponding a'_i in σ' is its complementary view.

Given an operation a, we denote by $\overline{a}$ its complementary operation. For instance, in Example II-C searchFlight$^{\gg}_{rr}$ is complementary to searchFlight$^{\ll}_{rr}$ and vice versa. We extend the notion of complementarity to execution traces, so the complementary $\overline{\sigma}$ on an execution trace σ is the sequence made of the operations that are respectively complementary to those of σ.

When a service plays a choreography role, it must supply a set of operations that will substitute the corresponding specifications, thus producing a service policy. Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a service description, and let $\mathcal{O}^i_u$ be a subset of $\mathcal{O}$, containing unbound operations that are to be supplied by a same role player S_i. Let $\mathcal{O}_{S_i}$ be the set of such operations, that are bound to the operations in $\mathcal{O}^i_u$. In case S_i is the player of $\langle \mathcal{O}, \mathcal{P} \rangle$, they will be operations decorated by $\ll$, otherwise they will be $\gg$ operations. We represent the binding by the substitution $\theta = [\mathcal{O}_{S_i}/\mathcal{O}^i_u]$ applied to $\langle \mathcal{O}, \mathcal{P} \rangle$. Notice that by $[\mathcal{O}_{S_i}/\mathcal{O}^i_u]$ we identify a set of substitutions $[o/o_u]$ of single service operations to single unbound operations. The application of the substitution is denoted by $\langle \mathcal{O}\theta, \mathcal{P}\theta \rangle$, where every element of $\mathcal{O}^i_u$ is substituted by/bound to an element of $\mathcal{O}_{S_i}$.

When the matching process is applied for selecting a service that should play a role in a choreography, the desire is that *the substitution* (of the service operations to the specifications contained in the choreography) *preserves the properties of interest*, i.e. the goals that could be entailed before by reasoning on each role description separately should be still achievable. Thus, we rely on the notion of *Conservative substitution* by Baldoni et al. in [13].

Let us, now, consider two-role choreography and see how the services S_1 and S_2 can identify a set of compatible traces, whose execution allows the achievement of both their personal goals. The mechanism we are going to describe can be extended to more complicated choreographies. However, we do not discuss the case, due to the lack of space, concerning a number of roles more than two. Suppose that S_1 has identified an execution trace σ that allows the achievement of its goal Fs_1, and it has verified that the substitutions θ^{R_1} and θ^{R_2} are conservative (the two substitutions involve disjoint sets of unbound operations by construction). Therefore, it now has an execution trace $\sigma\theta^{R_1}\theta^{R_2}$ that does not contain unbound operations. Before executing it, its candidate partner S_2 must agree on executing the complementary trace $\overline{\sigma\theta^{R_1}\theta^{R_2}}$. Of course, S_2 might have its own goal Fs_2 which should be achieved with the execution of the top level procedure of the role R_2, that we here call p_2. Therefore, the following conditions must be verified:

1) $\overline{\sigma\theta^{R_1}\theta^{R_2}}$ must be an execution trace of p_2;
2) after executing $\overline{\sigma\theta^{R_1}\theta^{R_2}}$ the goal Fs_2 must hold.

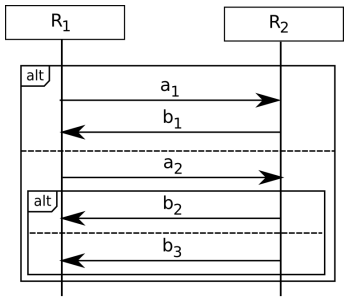

Fig. 3. Role R_1 takes a decision about invoking operation a_1 or a_2 on R_2. In the former case, R_2 will subsequently invoke b_1 over R_1; in the latter, it will take a decision and choose between the invocation of b_2 or the invocation of b_3 on R_1.

The first condition is guaranteed by the assumption that the choreography is well-defined, i.e. roles are interoperable, and that the services that will animate it are conform to the corresponding roles. The expectation that the roles of a choreography are by construction interoperable and that services are conform to them implies that, for any execution trace that a party can execute, its partner has a complementary execution trace. Interoperability is a hot research issue. A discussion about it is out of the scope of this work but the interested reader can take a look at [21], [22], [23], [24].

The second condition is to be verified by S_2 by reasoning on its goal, i.e. by verifying that $\langle R_2\overline{\theta^{R_1}\theta^{R_2}}, s_0\rangle \vdash Fs_2$ **after** $\overline{\sigma\theta^{R_1}\theta^{R_2}}$, where s_0 in this case is the initial state of S_2 and $\overline{\theta^{R_1}\theta^{R_2}}$ represents the substitution obtained from $\theta^{R_1}\theta^{R_2}$ by using the complementary views of the involved operations. Notice that the above reasoning is much simpler than the one executed on p_1 because S_2 only has to check if the goal is satisfied after $\overline{\sigma}$ has been executed. The reasoning applied to p_1, instead, was aimed at identifying such a trace.

To increase the probability of reaching an agreement, we can imagine that S_1 produces a set of alternative execution traces Σ, each of which allows the achievement of Fs_1, and then propose them to S_2, that will restrict them to a set of alternatives $\Sigma' \subseteq \Sigma$, satisfying both goals. It is worth noting that, in general, the set of alternative execution traces might not be finite. Moreover, the derivation ($\vdash$) is semi-decidable unless the choreographies are properly restricted, for instance by focussing onto regular sets [15]. Indeed, many choreographies are of this kind [25]. This interaction between the services, proposed so to allow the joint achievement of their goals, can be seen as a kind of *negotiation* [26], [27]. In case no other preference criterion is introduced, it is *indifferent* which of the execution traces in Σ' will be followed, and it is not necessary to perform any further negotiation step because the choreography is respected by both partners [24], and Σ' is known to both of them. Whatever the initiator of the interaction will be, they know how to behave to allow the mutual achievement of their personal goals, although none of them can make a commitment on a specific execution trace because both take choices that contribute to its definition.

An interesting issue is whether there is any *guarantee* that both partners will stick to execution traces that are in Σ'. In game theory words [26] and considering a choreography as a set of game rules, is Σ' an *equilibrium*? Generally, the answer is no. It might, in fact, be the case that S_2 has some execution trace, that is not contained in Σ' but allows anyway the achievement of its goal: at some point of the execution of an agreed trace, S_2 might decide to continue with an unagreed action, because such action is particularly convenient for it. A related issue is whether a service has a *dominant* strategy, i.e. an execution trace such that all of the alternative courses of action, that its partner has, belong to Σ'. For instance, with reference to Figure 3, let us suppose that Σ' contains the execution traces $a_1; b_1$ and $a_2; b_2$. In this case, the execution trace $a_1; b_1$ is dominant for S_1 because the only possible answer of S_2 is b_1 and the overall trace allows the achievement of both goals. Instead, the execution trace $a_2; b_2$ is not dominant because S_2 has the possibility of deciding to execute b_3 in alternative to b_2 and the trace $a_2; b_3$ does not belong to Σ'. The existence of a dominant strategy depends on the goal that one wants to satisfy, on the choreography, and on the identified substitutions. The derivation that we have proposed is not focussed on the identification of dominant strategies. In general, and differently than what happens for equilibria, the decision whether an execution trace is a dominant strategy can be taken without knowing the goal of the interlocutor. Indeed, a strategy is dominant if any action that the interlocutor can perform, lead to an execution trace which still belong to Σ'. Thus a service which has a dominant strategy has the guarantee to have a way to reach his goal. Therefore, it can be taken by a service individually.

Another interesting problem concerns preference criteria that services may apply in order to rank a set of strategies. Supposing that the two services decide to behave well and to execute a trace which is in the agreement, how can they converge to a best-compromise agreement? One criterion could be to select the trace which entails the minimum number of effects. For example, one can imagine that a buyer of a flight ticket prefers a trace at the end of which it has simply bought the desired ticket w.r.t. one in which it has additionally been registered in the advertisement mailing list of the flight company. This issue will be part of our future works.

As a final remark, negotiation, *does not substitute* the execution of the choreography but it rather concerns checking the possibility of, in this case, achieving all of the goals. The fact that the two services converged to a set of promising execution traces does not imply that, after starting the real execution, they will be able to stick to them. The reason is that those traces were identified by making assumptions on the values returned by tests and conditions, which cannot be known in advance. The actual results of such tests, obtained at execution time, could be different than the assumed ones. The *additional value* of performing the negotiation is double: on the one hand, it is possible to avoid the interaction with partners with which the agreement cannot be reached at all, on the other hand, each partner has the means to understand when

the interaction takes an unagreed path and decide accordingly, for instance concluding that it is better to stop the interaction.

IV. Conclusions

The coordination of a set of autonomous, cooperating agents is well-known and crucial in multi-agent systems research [27]. Solutions proposed in the literature are, for example, coordination as a *post-planning process* [28], where constraints are checked after the plan has been found, and the use of *conversation moderators* [29], which guarantee that achievement of shared objectives.

Along this line, this work tackles the problem of allowing a set of independent services, which must cooperate in the context of a given choreography, to reason about the joint achievement of their goals. The approach that we propose implements a simple form of *negotiation*, inspired by [20], [19]. There are, however, some important differences w.r.t the work by Ghaderi et al. The first is that the behavior of our services is ruled by a choreography, which specifies all the possible interactions that can occur. The assumption is that when services play choreography roles, they commit to keep their behavior adherent to the role specification. Ghaderi et al., instead, do not use or represent choreographies or other kinds of interaction protocols: the reasoning process considers all the possible execution traces that can be composed out of a set of atomic actions. Moreover, even when an execution trace, that allows the joint achievement of goals, is identified there is still the need of adding coordination mechanisms that allow its actual execution. In our case, the necessary coordination that makes the cooperation of the services possible is supplied by the choreography. The other assumption that we make is that the roles that compose a choreography are interoperable. Interoperability is a hot research topic that is orthogonal to the problems faced in this work.

Moreover, in the work by Ghaderi et al. each agent reasons also for the partner, because the two agents share a common goal and a common state. The method, when successful, identifies a set of joint plans, that correspond to preferred strategies for the agents. These are obtained by the iterative elimination of dominated strategies. In our framework each partner does not have knowledge about the goals of its interlocutor, as it is reasonable to suppose for web services; therefore, the execution traces that we identify are not necessarily dominant. For example, a greedy partner may take an action (if the protocol includes it) that is not in the agreement but that allows the immediate achievement of its own goal. This behavior is, however, not convenient over the long run because the former partner knows the choreography and knows the agreed traces, therefore, it has a way of *monitoring* the on-going course of interaction. As soon as it realizes that the partners has deviated from the agreed traces, it can take appropriate action, e.g. interrupt the interaction, enact compensation actions, publish a low reputation rate for the partner. In other words, there is a correspondence to what is known in game theory as a game iterated forever [27].

Some correspondences with the technique "Planning as Model Checking" proposed in [30] can be investigate. In this approach, the planning domain is a semantic model, the planning problem is represented through a set of global state and plan generation is done by checking whether suitable formulas are true in a semantic model. However, deal with web services that have to cooperate in order to reach their own goals seems to be more complicated. Indeed, planning has to be performed by each service on its own, ignoring which are the goals and which will be the substitutions used by the interlocutors. At the end, the chosen plan must be convenient for every service involved. Instead, by means of the mechanism proposed in this paper planning is made only by one service and the resulting set of compatible plans, at the end of the negotiation, allows everyone to achieve its own goal.

Some similarities can be found also with the proposal by Son and Sakama [31], that concerns the process of creating a *joint plan* in a MAS. They define a joint plan as one in which there are some *cooperative actions*, i.e. actions that are mutually requested/offered by agents. Indeed, also the actions involved in a protocol can be seen as cooperative actions, in the sense that a service is not able to achieve its goal on its own, and needs the "cooperation" of other services. The main difference between the two proposals is that a protocol defines not only which actions can be invoked on other services, but also when this can occur. One of the advantages of choreographies is that they limit the number of possible plans, restricting the search space of a joint plan. Moreover, condition 1) in Section III is guaranteed by the fact that whatever execution trace one of the partners focuses on, the other surely has a compatible one, if all of the interacting partners are conformant to the choreography and this is made of a set of interoperable roles.

Some analogies can be found also with the case in which protocols are defined by *social commitments* [32]. Indeed, a commitment is a "promise" of an agent to another on performing some actions. A protocols defined by commitments does not specify when actions are to be taken. However, in this case there is a common knowledge (i.e. the commitments) that constrain somehow the behaviour of the involved parties. This mechanism can guide the reasoning process in a better way. Indeed the reasoning is not on the belief that one agent has on the others –e.g. I believe that if he has provide this actions he will do it–, but is on the obligation one agent has taken: I know that he has taken the obligation of performing this action when I will ask him. Of course, this does not mean that the action will be executed when required. Indeed, actions are defined in terms of preconditions. Thus, its is possible that a condition is not satisfied when the action should be executed.

References

[1] W3C, "Web services description language (WSDL) version 2.0 part 1: Core language." [Online]. Available: http://www.w3.org/TR/wsdl20/

[2] OASIS, "Web services business process execution language version 2.0, working draft." [Online]. Available: http://www.oasis-

open.org/committees/download.php/17355/wsbpel-specification-draft.doc
[3] OWL-S Coalition. [Online]. Available: http://www.daml.org/services/owl-s/
[4] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres, *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer.
[5] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.
[6] M. Pistore, L. Spalazzi, and P. Traverso, "A minimalist approach to semantic annotations for web processes compositions." in *ESWC*, ser. LNCS, Y. Sure and J. Domingue, Eds., vol. 4011. Springer, 2006, pp. 620–634.
[7] M. B. van Riemsdijk and M. Wirsing, "Goal-Oriented and Procedural Service Orchestration. A Formal Comparison," in *AWESOME007*, Durham, UK, September 2007.
[8] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Reasoning on choreographies and capability requirements," *International Journal of Business Process Integration and Management*, vol. 2, no. 4, pp. 247–261, 2007.
[9] G. Caire, D. Gotta, and M. Banzi, "WADE: A software platform to develop mission critical applications exploting agents and workflows," in *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, 12-16 May 2008, pp. 29–36.
[10] M. Piunti, A. Ricci, and A. Santi, "SOA/WS Applications using Cognitive Agents working in CArtAgO Environments," in *Decimo Workshop Nazionale "Dagli Oggetti agli Agenti" (WOA 2009)*, Parma, 2009.
[11] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta, "CooWS: Adaptive BDI agents meet service-oriented computing," in *Proceedings of the Int. Conference on WWW/Internet*, 2005, pp. 205–209.
[12] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Service selection by choreography-driven matching," in *Emerging Web Services Technology*, ser. Whitestein Series in Software Agent Technologies and Autonomic Computing, T. Gschwind and C. Pautasso, Eds. Birkhäuser, September 2008, vol. II, ch. 1, pp. 5–22.
[13] M. Baldoni, C. Baroglio, V. Patti, and C. Schifanella, "Conservative reuse ensuring matches for service selection," in *Proc. of the 6th European Workshop on Multi-Agent Systems (EUMAS)*, 2008.
[14] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction protocols for customizing web service selection and composition," *JLAP, special issue on Web Services and Formal Methods*, vol. 70, no. 1, pp. 53–73, 2007.
[15] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443
[16] M. Baldoni, "Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension," Ph.D. dissertation, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998.
[17] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*. Springer, 2004.
[18] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Synthesis of Underspecified Composite e-Service bases on Automated Reasoning," in *Proc. of ICSOC04*. ACM, 2004, pp. 105–114.
[19] H. Ghaderi, H. Levesque, and Y. Lespérance, "Towards a logical theory of coordination and joint ability," in *Proc. of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07)*, 2007, pp. 532–534.
[20] ——, "A logical theory of coordination and joint ability," in *Proc. of AAAI'07*, 2007, pp. 421–426.
[21] S. K. Rajamani and J. Rehof, "Conformance checking for models of asynchronous message passing software," in *CAV*, ser. LNCS, vol. 2404. Springer, 2002, pp. 166–179.
[22] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella, "When are two web services compatible?" in *TES 2004*, ser. LNCS, vol. 3324. Springer, 2005, pp. 15–28.
[23] M. Bravetti and G. Zavattaro, "Contract based multi-party service composition," in *FSEN*, ser. LNCS, vol. 4767. Springer, 2007, pp. 207–222.
[24] M. Baldoni, C. Baroglio, A. Chopra, N. Desai, V. Patti, and M. Singh, "Choice, interoperability, and conformance in interaction protocols and service choreographies," in *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
[25] Foundation for Intelligent Physical Agents, "http://www.fipa.org."
[26] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. MIT Press, July 1994.
[27] M. Wooldridge, *An Introduction to Multiagent Systems*. John Wiley & Sons, March 2002.
[28] J. R. Steenhuisen, C. Witteveen, A. ter Mors, and J. Valk, "Framework and Complexity Results for Coordinating Non-cooperative Planning Agents," in *MATES*, ser. Lecture Notes in Computer Science, vol. 4196. Springer, 2006, pp. 98–109.
[29] C. Sibertin-Blanc and N. Hameurlain, "Participation Components for Holding Roles in Multiagent Systems Protocols," in *Engineering Societies in the Agents World*, ser. Lecture Notes in Computer Science, vol. 3451, August 2005, pp. 60–73.
[30] F. Giunchiglia and P. Traverso, "Planning as Model Checking," in *Recent Advances in AI Planning*, ser. LNCS. Springer, 2000, pp. 1–20.
[31] T. C. Son and C. Sakama, "Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming," in *DALT: Declarative Agent Languages and Technologies*, ser. LNAI. Budapest, Hungary: Springer, May 2009.
[32] M. P. Singh, "Agent communication languages: Rethinking the principles," in *Communication in Multiagent Systems*, ser. Lecture Notes in Computer Science, M.-P. Huget, Ed., vol. 2650. Springer, 2003, pp. 37–50.