Available online at www.sciencedirect.com

The Journal of Logic and Algebraic Programming 70 (2007) 53–73

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

Reasoning about interaction protocols for customizing web service selection and composition [☆]

Matteo Baldoni [*], Cristina Baroglio, Alberto Martelli, Viviana Patti

Dipartimento di Informatica – Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy

Abstract

This work faces the problem of automatic selection and composition of web services, discussing the advantages that derive from the inclusion, in a web service declarative description, of the high-level communication protocol, that is used by the service for interacting with its partners, allowing a rational inspection of it. The approach we propose is set in the Semantic Web field of research and inherits from research in the field of multi-agent systems. Web services are viewed as software agents, communicating by predefined sharable interaction protocols. A logic programming framework based on modal logic is proposed, where the protocol-based interactions of web services are formalized and the use of reasoning about actions and change techniques (planning) for performing the tasks of selection and composition of web services in a way that is personalized w.r.t. the user request is enabled. We claim that applying reasoning techniques on a declarative specification of the service interactions allows to gain flexibility in fulfilling the user preference in the context of a web service matchmaking process.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Semantic Web services; Reasoning about actions; Interaction protocols; Personalization; Agent logic programming

1. Introduction

Web services are an emergent paradigm for implementing business collaborations, across and within corporation boundaries [1]. Workflow research and technology have found in web services a natural field of application which opens interesting and challenging perspectives. Web services, however, also raised the attention of other research communities, in particular the two respectively studying the Semantic Web and multi-agent systems (MAS for short). The work presented in this paper is set transversely across these three fields. The aim is, basically, to show with a practical example the possibility and the benefits of cross-fertilization of these three areas, which indeed show interesting convergence points.

[☆] This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 "Specification and Verification of Agent Interaction Protocols" national project.

[*] Corresponding author. Tel.: +39 011 6706711; fax: +30 011 751603.

E-mail addresses: baldoni@di.unito.it (M. Baldoni), baroglio@di.unito.it (C. Baroglio), mrt@di.unito.it (A. Martelli), patti@di.unito.it (V. Patti).

1567-8326/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2006.05.005

Concerning web services, this paper focuses on a central issue: studying declarative descriptions aimed at allowing forms of automated interoperation that include, on the one hand, the automation of tasks like matchmaking and execution of web services, on the other, the automation of service *selection and composition* in a way that is customized w.r.t. the *user's goals* and *needs*, a task that can be considered as a form of *personalization* [2]. Indeed, selection and composition not always are to be performed on the sole basis of general properties of the services themselves and of their interactive behavior, such as the category of the service or the functional compositionality of a set of services, but they should also take into account the user's intentions (and purposes) which both motivate and constrain the search or the composition. As a quick example, consider a web service that allows buying products, alternatively paying cash or by credit card: a user might have preferences on the form of payment to enact. In order to decide whether or not buying at this shop, it is necessary to single out the specific course of interaction that allows buying cash. This form of personalization can be obtained by applying *reasoning techniques* on a description of the service process. Such a description must have a well-defined meaning for all the parties involved. In this issue it is possible to distinguish three necessary components: first, web services capabilities must be represented according to some declarative formalism with a well-defined semantics, as also recently observed by van der Aalst [1]; second, automated tools for reasoning about such a description and performing tasks of interest must be developed; third in order to gain flexibility in fulfilling the user's request, reasoning tools should represent such requests as *abstract goals*.

The approach that we propose is to exploit results achieved by the community that studies *logic for agent systems* and, in particular, *reasoning about actions and change*. Indeed, the availability of semantic information about web resources enables the application of reasoning techniques, such as ontology reasoning, constraint reasoning, non-monotonic reasoning, and temporal reasoning [3], whose use would allow the design of systems that, being able of autonomous decisions, can adapt to different users and are open to interact with one another. In particular, we propose to use techniques for *reasoning about actions* for performing the automatic selection and composition of web services, in a way that is customized w.r.t. the users' request, by reasoning on the *communicative behavior* of the services. Communication can, in fact, be considered as the behavior resulting from the application of a special kind of actions: *communication actions*. The reasoning problem that this proposal faces can intuitively be described as looking for an answer to the question "Is it possible to make a deal with this service respecting the user's goals?". Given a logic-based representation of the service policies and a representation of the customer's needs as abstract goals, expressed by a logic formula, logic programming reasoning techniques are used for understanding if the constraints of the customer fit in with the policy of the service.

This proposal inherits from the experience of the research community that studies MAS and, in particular, logic-based formalizations of interaction aspects. Indeed, communication has intensively been studied in the context of formal theories of agency [4,5] and a great deal of attention has been devoted to the definition of standard agent communication languages (ACL), e.g. FIPA [6] and KQML [7]. Recently, most of the efforts have been devoted to the definition of formal models of interaction among agents, that use *conversation protocols*. The interest for protocols is due to the fact that they improve the interoperability of the various components (often separately developed) and allow the verification of compliance to the desired standards. Given the abstraction of web services as entities, that communicate by following predefined, public and sharable interaction protocols, we have studied the possible benefits provided by a *declarative description* of their communicative behavior, in terms of personalization of the service selection and composition. The approach models the interaction protocols provided by web services by a set of logic clauses, thus at high (not at network) level. A description by policies is definitely richer than the list of input and output, precondition and effect properties usually taken into account for the matchmaking (see Section 5). Moreover having a logic specification of the protocol, it is possible to reason about the effects of engaging specific conversations, and, on this basis, to perform many tasks in an automatic way. Actually, the proposed approach can be considered as a *second step* in the matchmaking process, which narrows a set of already selected services and performs a *customization* of the interaction with them.

For facing the problem of describing and reasoning about conversation protocols, we have extended the agent language DyLOG [8] by introducing a communication kit, that is presented in this article. DyLOG is an agent programming language, based on a modal logic for reasoning about actions and beliefs, which has already been used in the development of adaptive web applications [9]. It allows the specification of the behavior of rational agents, and it supplies mechanisms for reasoning about it. In Section 3 we present an *extension* for dealing with communication. This extension is based on an agent theory, in which agents have *local* beliefs about the world and about the mental states of the other agents, and where communications are modelled as actions that operate on such beliefs. This account of communication aims at coping with two main aspects: the *change in the state* that encodes an agent's beliefs, caused by a communicative act, and the *decision strategy* used by an agent for answering to a received communication. To these aims, the semantics of primitive speech acts is described in terms of effects on the mental state, both in the case in which the agent is the sender and in the case in which it is the recipient, and, in the line of [10], conversation protocols are used as decision procedures. Each agent has a *subjective perception* of the on-going conversations and, guided by the protocol, it makes hypothetical assumptions on the other agents' answers. In the web service application context we exploit such a feature by including in the knowledge base of an agent (the requester) a description of the potentially interesting services. This description is given by logic rules expressing their communicative policies from the point of view of the requester. The language provides a goal-directed proof procedure that supports reasoning on communication and allows an agent to reason about the interaction that it is going to enact *before* it actually occurs, with the aim of proving properties of the possible executions.

In Section 2 we will locate our proposal in the context of current research on web services, while in Sections 2.1 and 4 we will show by means of examples how it is possible to use the new tools offered by the language for representing and reasoning on the conversation policies of the web services for personalizing the retrieval, the composition of, and the interaction with services. In this perspective, this article integrates and extends the work in [11,12].

2. Context and perspectives

In the last years distributed applications over the World-Wide Web have obtained wide popularity and uniform mechanisms have been developed for handling computing problems which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments have begun to coalesce around the web service paradigm, where a service can be seen as a component available over the web. Each service has an interface that is accessible through standard protocols and that describes the *interaction capabilities* of the service. It is possible to develop new applications over the web by *combining and integrating* existing web services.

In this scenario, two needs have inspired recent research [13]: the first is the need of developing *programming languages* for implementing the behavior of the single participants involved in an interaction, the other is the need of developing *modelling languages* for describing processes and their *interaction protocols*, abstracting from details concerning the internal behavior of the single components. At the moment of writing this article, the language BPEL4WS (BPEL for short [14]) has emerged as the standard for specifying the business processes of single services and it allows writing a *local* view of the interaction that should take place, i.e. the interaction from the point of view of the process. Its authors envisioned the use of the language both as an *execution language*, for specifying the actual behavior of a participant in a business interaction, and as a modelling language, for specifying the interaction at an abstract level, once again from the perspective of the service being described. If, on a hand, BPEL as an execution language is extremely successful, BPEL as a language for modelling abstract interactions substantially did not succeed [1]. Its limit is that it does not allow to perform the analysis of the described process, but the capability of performing this analysis is fundamental to the real implementation of those sophisticate forms of flexibility and composition that one expects in the context of the web. For achieving such a flexibility and enable automatic devices to use a web resource, the latter must bear some public information about itself, its structure, the way in which it is supposed to be used, and so forth. This information should be represented according to some conventional formalism which relies on *well-founded models*, upon which it is possible to define access and usage mechanisms. To meet these requirements, the attention of part of the community has focussed on capturing the behavior of BPEL in a formal way, and many translations of BPEL into models supporting analysis and verification (process algebras, petri nets, finite state machines) are currently under investigation [15,16]. For what concerns the *specification of interaction protocols*, instead, there is a growing agreement on the fact that the local point of view is not sufficient and that a *global* view of the interaction to occur should be expressed. This level of representation is captured by the concept of *choreography*. Choreographies are expressed by using specific languages, like the W3C proposal WS-CDL [17]. We will discuss about choreographies further below in the section.

In parallel, the World-Wide Web Consortium (W3C), has given birth to the *Semantic Web* initiative [18] (see also [19,20]) which is centered on providing a common framework, that allows resources to be shared and reused across application, enterprise, and community boundaries. In order to be *machine-processable*, information must be given a well-defined meaning; one underling assumption is that its representation has a *declarative format*. Research in the Semantic Web area is giving very good results for what concerns the intelligent retrieval and use of documents, nevertheless, research on Semantic Web services is still at its beginning. The current W3C proposal for describing Semantic Web services is the language OWL-S. Like BPEL, OWL-S allows the description of possibly composite processes from a *local* perspective. Noticeably, the sets of operators that the two languages supply, and by which the processes can be composed, though not fully identical, show a consistent intersection. The main difference between the two languages seems to be that BPEL is more focussed on the *message exchange* between the service and its parties, while OWL-S is more focussed on the *process advertisement* and the *process structure*. Another similarity is that both languages allow the definition of executable specifications. For the sake of completeness, OWL-S is but a proposal and alternative initiatives are being conducted, like WSMO [21]. It is also interesting to notice that in the Semantic Web context, there is currently no proposal of a concept close to that of "choreography", the questions of service effective executability and interoperability are open, and the proposed matchmaking techniques are still simple and quite far from fully exploiting the power of shareable semantics.

We have observed two interesting convergence points, one between the vision reported in [13] and results of research in multi-agent systems, the other between research in multi-agent systems and that in the Semantic Web. The former is that also in the case of agent systems, research has identified in message exchange the key to interoperation, even in the case in which agents are not rational. In particular, in an agent system it is possible to distinguish the *global interaction schema* that the group of agents should realize, the interaction protocol (analogous to a choreography), from a declarative representation of the interactive *behavior of the single* agents, their interaction policies (executable specifications). Protocols are public and shared by the agents; policies can be made public. Similarly to what postulated by the Semantic Web, both descriptions are supposed as being declarative, with a well-defined meaning and a well-founded semantics. Moreover, agent systems are commonly supposed as being open and constituted by heterogeneous members. In this context, besides giving a global view of communication, protocols can also be considered as a specification, in an interlanguage (koine), that can be used – even run-time – for selecting agents that should interact in some task. Partners which agree on using a global interaction protocol, can cooperate even if their local interactive behaviors are implemented by using different programming languages. The respect of the global rules guarantees the interoperability of the parties (i.e. their capability of *actually* producing an interaction), and that the interactions will satisfy expected requirements. Of course in this context verifying that the local implementations conform to the global specification becomes crucial.

On the other hand, having a declarative representation with a strong formal basis makes it possible to apply *reasoning techniques* for achieving goals of interest or prove properties of interest, the formalization basically depending on the reasoning tasks to be tackled. In particular, in order to introduce that degree of flexibility that is necessary for personalizing the selection and composition of web services in an open environment, in such a way that the user's goals and needs are satisfied, it is necessary to reason on the interactive behavior that the service can enact by using a declarative language that both supports the *explicit coding* of interaction policies, and that refers to a formal model for reasoning on *knowledge attitudes* like goals and beliefs. As we will show in the following examples, this kind of formalization allows the selection of specific courses of interaction among sets of possible alternatives, based on the user's likings and constraints, a selection process that can be extended to the case of service composition. To our knowledge, none of the most widely diffused languages for representing (semantic) web services has all these characteristics, so for proposing these advanced forms of selection it is necessary to borrow from the neighboring area of multi-agent systems. We have chosen the DyLOG language for writing the abstract specifications of the service behaviors because the language shows all these characteristics.

2.1. A scenario

Let us now introduce a simple scenario that will help us in showing how, by reasoning about the communicative behavior of web services, it is possible to personalize their fruition. The scenario encompasses two kinds of web services for taking reservation: services to book tables at restaurants and services to book seats at cinemas. In this framework it is quite easy to invent queries of various complexity. Not all of them can be answered by the basic keyword-based retrieval mechanisms (see related works in Section 5) with sufficient precision. To the aims of this example it is sufficient to suppose that there are only two restaurant booking services (we will simply call them *restaurant1* and *restaurant2*) and two cinema booking services (*cinema1* and *cinema2*). On a keyword basis, the two cinemas have the same description as well as the two restaurants but all the services have a different interactive behavior with their clients. Figs. 1 and

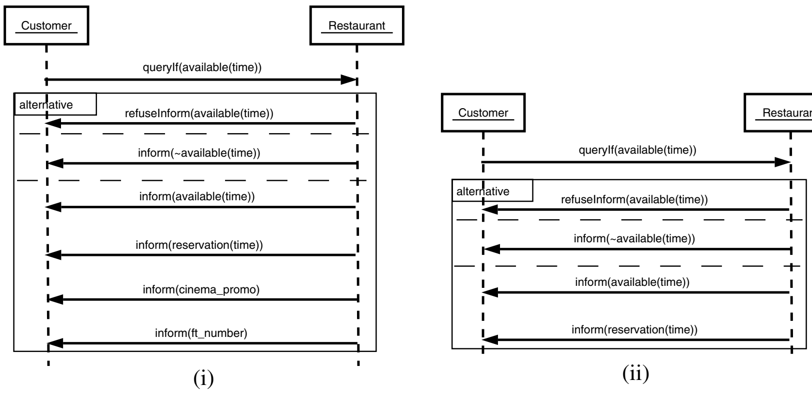

Fig. 1. The AUML sequence diagrams represent the interactions occurring between the customer and each of the restaurant web services is followed by (i) *restaurant1* and (ii) *restaurant2*.

2 report their interaction protocols, represented as AUML [22,23] sequence diagrams. In particular, *restaurant1* takes part to a promotion campaign, by which each customer, who made a reservation by the internet service, receives a free ticket for a movie (Fig. 1(i)). *Restaurant2* does not take part to this initiative (Fig. 1(ii)). On the side of cinemas, *cinema2* accepts reservations to be paid cash but no free tickets (Fig. 2(iv) whereas *cinema1* accepts the promotional tickets and, as an alternative, it also takes reservations by credit card (Fig. 2(iii)).

One of the simplest tasks that can be imagined is web service retrieval, in this case the query is a description of the desired service (e.g. "cinema booking service"). However, what to do when the user is not simply interested in a generic kind of service but s/he would like to find services that besides being of that kind also show other characteristics? For instance to find a cinema booking service that does not require to confirm the reservation by sending a credit card number. While in the former case a keyword-based description would have been sufficient, and the answer would have included both *cinema1* and *cinema2*, in the latter case more information about the service is necessary. How to consider *cinema1* that requires either the credit card or a promotional free ticket? Even more interesting is the case in which the answer to the user's goal requires to combine the executions of two (or more) independent web services. For instance, let us suppose that the user would like to organize an evening out by, first, having dinner at some nice restaurant and, then, watching a movie; moreover, s/he heard that in some restaurants it is possible to gain a free ticket for a movie and s/he would like to profit of this possibility but only if it is not necessary to use the credit card because s/he does not trust internet connections very much. If, on one hand, searching for a cinema or a restaurant reservation service is a task that can be accomplished by any matchmaking approach, the conditions "look for promotions" and "do not use my credit card" can be verified only by a rational inspection of the communication with the services. This verification should be done before the actual interaction. Obviously, the only combination that satisfies all the user's requests is to reserve a table at *restaurant1*, and then make a reservation at *cinema1*. This solution can be found only enacting a reasoning mechanism that, based on the description of the interaction protocols, selects *restaurant1* and *cinema1* and, in particular, it selects the course of action that makes use of the free ticket.

In order to perform this kind of tasks, in this work we set the web service selection and composition problems in a multi-agent perspective. The idea is to interpret web services as agents, each bearing a public communication protocol (described in a declarative format), and to delegate the task of selection (or composition) to a *personalization agent*. Such an agent receives a description of the user's desires and goals, interacts with other systems that perform a less informed retrieval of services, and, in order to find a suitable solution to the user's request, applies a reasoning mechanism to the communication protocols of the services. The protocols of such services, that have been identified by the less informed retrieval systems, are supposed to be represented in a specification language, such as AUML [22,23]. We assume that this representations are translated in DyLOG procedures that are included in the knowledge base of the personalization agent. Given this representation, in order to accomplish the reasoning tasks that have been described, we propose the use of procedural planning techniques, in which the possible executions of a sequence of protocols are evaluated, with the aim of deciding if they meet or not the requirements or under which assumptions. In particular, we will exploit the planning capabilities offered by the DyLOG agent programming language, which is described in Section 3.

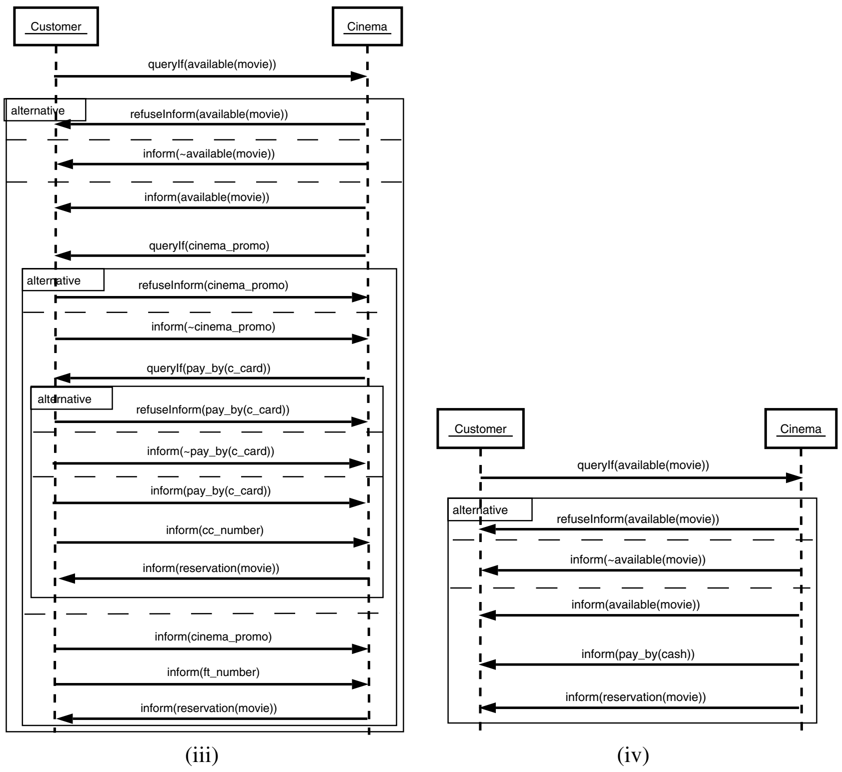

Fig. 2. In this case, the AUML sequence diagrams represent the possible interactions between the customer and each cinema web service: (iii) represents the protocol followed by *cinema1*; (iv) is followed by *cinema2*.

Of course, the agent cannot be sure that once the plan is executed it will not fail because the planning process is performed before the interaction; so, for instance, it cannot know if there will actually be a free table left at the restaurant. No currently existing selection mechanism can guarantee this but it would be interesting to study the relations between replanning techniques and the compensation techniques developed for long-time transactions (see Section 5 for further discussion). The advantage of the proposed approach is that it leaves out services, that would in no case allow the achievement of the user's goals respecting all the requirements, as well as it allows the exact identification of those permitted courses of interaction that satisfy them.

Present web service technology is quite primitive w.r.t. the framework we propose, and the realization of the picture sketched above would require an integration, in the current tools for web service representation and handling, of knowledge representation languages – in the line of those developed in the Semantic Web area – and of techniques for reasoning and dealing with communication, inspired by those studied in the area of MAS. Even if a real integration is still far from being real for the sake of concreteness, let us describe our *vision* of the steps to be taken toward the realization.

In our view, public descriptions of interaction protocols in the scenario above (represented by AUML diagrams) can be mapped in public descriptions of choreographies (e.g. WS-CDL-like descriptions). A choreography defines a global view of the protocol followed by a certain service, e.g. the *cinema service*, for accomplishing the cooperative task of booking a cinema ticket. A *costumer* service, that in principle is interested to participate to the cooperation for booking a ticket for its user, *translates* such a description in the declarative language DyLOG[1] and uses reasoning techniques, supported by the language, plus its local knowledge on the user's preferences for checking whether the contract, defined by the choreography, satisfies its user. Such reasoning *conditions* the costumer's decision of selecting the cinema service as a partner in a real interaction for accomplishing a task. Of course the outcome of the reasoning is

[1] This process requires the selection of a proper ontology, see Section 5.

meaningful under the following assumption: the implementation of the cinema service behavior (that could be written in an execution language like BPEL) must be *conformant* w.r.t. the choreography specification that is used as input of the reasoning. Verifying the conformance and the interoperability of web services to a global protocol definition (to be provided at the choreography level) is definitely one of the hot topics at the moment. In [24] we started to attack the problem by proposing a framework, based on the theory of *formal languages*, where both the global protocol and the web service behavior are expressed by using finite state automata. Instead in [25] choreography and orchestration are formalized by using two process algebras and conformance takes the form of a bisimulation-like relation.

3. A declarative agent language for reasoning about communication

Logic-based, executable languages for agent specification have been deeply investigated in the last years [26–28,8,29]. In this section we introduce the main features of DyLOG, our agent language for reasoning about actions and changes, and, in particular, we present the extension for dealing with communication (the CKit, Section 3.2), which will be explained with more details and with examples. Section 3.3 presents the semantics of the CKit based on a non-monotonic solution to deal with the persistency problem, while Section 3.4 presents the reasoning mechanisms that will be used in Section 4. The proof theory is reported in Appendix A.

DyLOG is a logic programming language for modeling rational agents, based on a modal theory of actions and mental attitudes. In this language *modalities* are used for representing *actions* while *beliefs* model the agent's internal state. The language refers to a *mentalistic* approach, which is also adopted by the standard FIPA-ACL [6], where communicative actions affect the internal mental state of the agent. The mental state of an agent is described in terms of a consistent set of *belief formulas*. The modal operator $\mathbf{B}^{ag_i}$ models the beliefs of the agent ag_i. The modal operator $\mathbf{M}^{ag_i}$ is defined as the dual of $\mathbf{B}^{ag_i}$ ($\mathbf{M}^{ag_i}\varphi \equiv \neg\mathbf{B}^{ag_i}\neg\varphi$); intuitively it represents the fact that agent ag_i considers φ possible. The language allows also dealing with *nested beliefs*, which allow the representation of what an agent thinks about the other agents' beliefs, and make reasoning on how they can be affected by communicative actions possible. DyLOG accounts both for *atomic* and *complex actions*, or procedures. Atomic actions are either *world* actions, that is actions which affect the world, or *mental* actions, i.e. sensing or communicative actions which only modify the agent's beliefs. For each world action and for each agent the modalities $[a^{ag_i}]$ and $\langle a^{ag_i}\rangle$ are defined: $[a^{ag_i}]\varphi$ denotes the fact that the formula φ holds after *every* execution of a performed by agent ag_i, while $\langle a^{ag_i}\rangle\varphi$, represents the *possibility* that φ holds after the action has been executed by the agent. A modality $Done(a^{ag_i})$ is also introduced for expressing that a (a communicative act or a world action) has been executed. Last but not least, the modality $\Box$ (box) denotes formulas that hold in all the possible agent mental states. The formalization of *complex actions* draws considerably from dynamic logic for the definition of action operators like sequence (;), ruled by $\langle a; b\rangle\varphi \equiv \langle a\rangle\varphi$, test (?), $\langle\psi?\rangle\varphi \equiv \psi \wedge \varphi$ and non-deterministic choice ($\cup$), $\langle a \cup b\rangle\varphi \equiv \langle a\rangle\varphi \vee \langle b\rangle\varphi$ but, differently than [30], DyLOG refers to a *Prolog-like* paradigm and procedures are defined as recursive Prolog-like clauses. Analogously to what done in the case of atomic actions, for each procedure p, the language contains also the *universal* and *existential modalities* $[p]$ and $\langle p\rangle$. All the modalities of the language are normal; $\Box$ is reflexive and transitive and its interaction with action modalities is ruled by the axiom $\Box\varphi \supset [a^{ag_i}]\varphi$, that is, in ag_i's mental state φ will hold after every execution of any action performed by the agent. The epistemic modality $\mathbf{B}^{ag_i}$ is serial, transitive and Euclidean. The interaction of $Done(a^{ag_i})$ with other modalities is ruled by the axioms $\varphi \supset [a^{ag_i}]Done(a^{ag_i})\varphi$ and $Done(a^{ag_j})\varphi \supset \mathbf{B}^{ag_i}Done(a^{ag_j})\varphi$ (*awareness*), with $ag_i = ag_j$ when a^{ag_i} is not a communicative action.

3.1. The agent theory

DyLOG agents are considered as individuals, each with its *subjective* view of a dynamic domain. The framework does not model the real world but only the internal dynamics of each agent in relation to the changes caused by actions. An agent's *behavior* is specified by a *domain description* that includes:

1. the agent's *belief state*;
2. *action* and *precondition laws* that describe the effects and the preconditions of atomic world actions on the executor's mental state;
3. *sensing axioms* for describing atomic sensing actions;
4. *procedure axioms* for describing complex behaviors.

Let us denote by the term *belief fluent* F, a belief formula $\mathbf{B}^{ag_i} L$ or its negation, where L, the *belief argument*, is a *fluent literal* (f or $\neg f$), a *done fluent* ($Done(a^{ag_i})\top$ or its negation), or a belief fluent of *rank 1* ($\mathbf{B}l$ or $\neg\mathbf{B}l$). In this latter case the symbol l is an *attitude-free* fluent, that is a fluent literal or a done fluent.

Intuitively, the *belief state* contains what an agent (dis)believes about the world and about the other agents. It is a complete and consistent set of *rank 1* and 2 belief fluents. A belief state provides, for each agent, a three-valued interpretation of all the possible belief arguments L, that can either be *true*, *false*, or *undefined* when both $\neg\mathbf{B}^{ag_i} L$ and $\neg\mathbf{B}^{ag_i}\neg L$ hold. $\mathbf{U}^{ag_i} L$ expresses the ignorance of ag_i about L.

World actions are described by their preconditions and effects on the *actor*'s mental state; they trigger a revision process on the actor's beliefs. Formally, *action laws* describe the conditional effects on ag_i's belief state of an atomic action a, executed by ag_i itself. They have the form:

$$\Box(\mathbf{B}^{ag_i} L_1 \wedge \cdots \wedge \mathbf{B}^{ag_i} L_n \supset [a^{ag_i}]\mathbf{B}^{ag_i} L_0) \tag{1}$$

$$\Box(\mathbf{M}^{ag_i} L_1 \wedge \cdots \wedge \mathbf{M}^{ag_i} L_n \supset [a^{ag_i}]\mathbf{M}^{ag_i} L_0) \tag{2}$$

Law (1) states that if ag_i believes the preconditions to an action a in a certain epistemic state, after a execution, ag_i will also believe the action's effects. (2) states that when the preconditions of a are unknown to ag_i, after the execution of a, ag_i will consider unknown also its effects.[2] *Precondition laws*, instead, specify mental conditions that make an atomic action executable in a state. An agent ag_i can execute a when the precondition fluents of a are in its belief state. More formally:

$$\Box(\mathbf{B}^{ag_i} L_1 \wedge \cdots \wedge \mathbf{B}^{ag_i} L_n \supset \langle a^{ag_i}\rangle\top) \tag{3}$$

Sensing actions produce knowledge about fluents; they are defined as non-deterministic actions, with unpredictable outcome, formally modelled by a set of *sensing axioms*. Each sensing action s has associated a set $dom(s)$ of literals (its *domain*). When ag_i executes s, it will know which of such literals is true.

$$[s]\varphi \equiv \Big[\bigcup_{l\in dom(s)} s^{\mathbf{B}^{ag_i} l}\Big]\varphi \tag{4}$$

$\cup$ is the choice operator of dynamic logic and $s^{\mathbf{B}^{ag_i} l}$, for each $l \in dom(s)$, is an *ad hoc* primitive action, that probes one of the possible outcomes of the sensing. Such primitive actions are ruled by the simple action clauses:

$$\Box(\mathbf{B}^{ag_i} l_1 \wedge \cdots \wedge \mathbf{B}^{ag_i} l_n \supset \langle s^{\mathbf{B}^{ag_i} l}\rangle\top) \tag{5}$$

$$\Box(\top \supset [s^{\mathbf{B}^{ag_i} l}]\mathbf{B}^{ag_i} l) \tag{6}$$

$$\Box(\top \supset [s^{\mathbf{B}^{ag_i} l}]\mathbf{B}^{ag_i}\neg l') \tag{7}$$

for each $l' \in dom(s)$, $l \neq l'$. Clause (5) means that after any sequence of world actions, if the set of literals $\mathbf{B}^{ag_i} l_1 \wedge \cdots \wedge \mathbf{B}^{ag_i} l_n$ holds, then the action $s^{\mathbf{B}^{ag_i} l}$ can be executed. The other ones describe the effects of $s^{\mathbf{B}^{ag_i} l}$: in any state, after the execution of $s^{\mathbf{B}^{ag_i} l}$, l is believed (6), while all the other fluents belonging to $dom(s)$ are believed to be false (7). Note that the binary sensing action on a fluent l, is a special case of sensing where the associated finite set is $\{l, \neg l\}$.

Complex actions specify complex behaviors by means of *procedure definitions*, built upon other actions. Formally, a complex action is a collection of *inclusion axiom schema* of the modal logic, of form:

$$\langle p_0\rangle\varphi \subset \langle p_1; p_2; \ldots; p_m\rangle\varphi \tag{8}$$

p_0 is a procedure name, ";" is the *sequencing operator* of dynamic logic, and the p_i's, $i \in [1, m]$, are procedure names, atomic actions, or test actions. Procedure definitions may be *recursive* and procedure clauses can be executed in a *goal-directed* way, similarly to standard logic programs.

3.2. Communication

A *communication theory* has been integrated in the general agent theory by adding further axioms and laws to the agents' domain description. It consists of *speech acts*, *get-message actions* and *conversation policies*.

[2] Laws of form (2) allow actions with non-deterministic effects, that may cause a *loss* of knowledge, to be specified.

Speech acts are atomic actions of the form $\mathsf{speech\_act}(sender, receiver, l)$ where $sender$ and $receiver$ are agents and l is either a fluent literal or a done fluent. Since agents have a personal view of the world, the way in which an agent's beliefs are modified by the execution of a speech act depends on the *role* that it plays. For this reason, speech act specification is twofold: one definition holds when the agent is the *sender*, the other when it is the *receiver*. Speech acts are modelled by generalizing the action and precondition laws of world actions, so to enable the representation of the effects of communications that are performed by other agents. When the agent is the sender, the precondition laws contain some *sincerity condition* that must hold in the agent's mental state. When ag_i is the receiver, the action is supposed as *always* executable (ag_i has no control over a communication performed by another agent). This representation allows agents to *reason about* conversation effects. Hereafter are a few examples of speech act, as defined in DyLOG: they are the specification of the *inform*, *queryIf*, and *refuseInform* FIPA-ACL primitive speech acts.

$\mathsf{inform}(Self, Other, l)$
(a) $\Box(\mathbf{B}^{Self} l \wedge \mathbf{B}^{Self}\mathbf{U}^{Other} l \supset \langle \mathsf{inform}(Self, Other, l)\rangle\top)$
(b) $\Box([\mathsf{inform}(Self, Other, l)]\mathbf{M}^{Self}\mathbf{B}^{Other} l)$
(c) $\Box(\mathbf{B}^{Self}\mathbf{B}^{Other} authority(Self, l) \supset [\mathsf{inform}(Self, Other, l)]\mathbf{B}^{Self}\mathbf{B}^{Other} l)$
(d) $\Box(\top \supset \langle \mathsf{inform}(Other, Self, l)\rangle\top)$
(e) $\Box([\mathsf{inform}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{B}^{Other} l)$
(f) $\Box(\mathbf{B}^{Self} authority(Other, l) \supset [\mathsf{inform}(Other, Self, l)]\mathbf{B}^{Self} l)$
(g) $\Box(\mathbf{M}^{Self} authority(Other, l) \supset [\mathsf{inform}(Other, Self, l)]\mathbf{M}^{Self} l)$

Clause (a) represents the *executability preconditions* for the action $\mathsf{inform}(Self, Other, l)$: it specifies those mental conditions that make this action executable in a state. Intuitively, it states that $Self$ can execute an inform act only if it believes l ($\mathbf{B}^{Self}$ models the beliefs of agent $Self$) and it believes that the receiver ($Other$) does not know l. According to clause (b), the agent also considers possible that the receiver will adopt its belief (the modal operator $\mathbf{M}^{Self}$ is the dual of $\mathbf{B}^{Self}$ by definition), although it cannot be sure that this will happen by the *autonomy assumption*. Nevertheless, if agent $Self$ thinks to be considered by the receiver a *trusted authority* about l, it is also confident that $Other$ will adopt its belief, clause (c). Since executability preconditions can be tested only on the mental state of $Self$, when $Self$ is the receiver the action of informing is considered as *always* executable, clause (d). When $Self$ is the receiver, the effect of an inform act is that $Self$ will believe that l is believed by the sender ($Other$), clause (e), but $Self$ will adopt l as an own belief only if it thinks that $Other$ is a trusted authority, clause (f).

$\mathsf{queryIf}(Self, Other, l)$
(a) $\Box(\mathbf{U}^{Self} l \wedge \neg\mathbf{B}^{Self}\mathbf{U}^{Other} l \supset \langle \mathsf{queryIf}(Self, Other, l)\rangle\top)$
(b) $\Box(\top \supset \langle \mathsf{queryIf}(Other, Self, l)\rangle\top)$
(c) $\Box([\mathsf{queryIf}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{U}^{Other} l)$

By queryIf an agent asks another agent if it believes that l is true. To perform a queryIf act, $Self$ must ignore l and it must believe that the receiver does not ignore l, clause (a). After a queryIf act, the receiver will believe that the sender ignores l.

$\mathsf{refuseInform}(Self, Other, l)$
(a) $\Box(\mathbf{U}^{Self} l \wedge \mathbf{B}^{Self} Done(\mathsf{queryIf}(Other, Self, l))\top$
$\supset \langle \mathsf{refuseInform}(Self, Other, l)\rangle\top)$
(b) $\Box(\top \supset \langle \mathsf{refuseInform}(Other, Self, l)\rangle\top)$
(c) $\Box([\mathsf{refuseInform}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{U}^{Other} l)$

By refuseInform an agent refuses to give an information it was asked for. The refusal can be executed only if the sender ignores l and it believes that the receiver previously queried it about l, clause (a). A refusal by some other agent is considered as always possible, clause (b). After a refusal the receiver believes that the sender ignores l.

Get-message actions are used for *receiving* messages from other agents. Since, from the agent perspective, they correspond to *queries* for an external input, they are modelled as a special kind of sensing actions, whose outcome

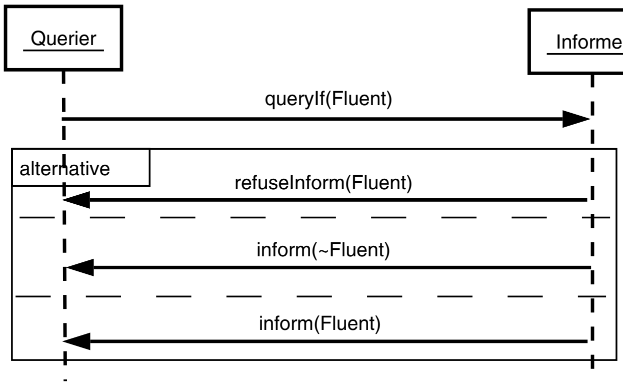

Fig. 3. The AUML sequence diagram represents the communicative interactions occurring between the *querier* and the *informer* in the yes_no_query protocol.

is unpredictable. The main difference w.r.t. normal sensing actions is that they are defined by means of speech acts performed by the *interlocutor*. Formally, get_message actions are defined by axiom schemas of the form:

$$[\mathsf{get\_message}(ag_i, ag_j, l)]\varphi \equiv \Big[\bigcup_{\mathsf{speech\_act} \in \mathbf{C}_{\mathsf{get\_message}}} \mathsf{speech\_act}(ag_j, ag_i, l) \Big]\varphi \quad (9)$$

$\mathbf{C}_{\mathsf{get\_message}}$ is a finite set of speech acts, which are all the possible communications that agent ag_i expects from agent ag_j in the context of a given conversation. A get_message action does not have a domain of mental fluents associated to it, the information is obtained by looking at the effects of such speech acts on ag_i's mental state.

Conversation protocols specify patterns of communication; they define the context in which speech acts are executed [10] and are modelled by means of *procedure axioms* having the form (8). Since agents have a subjective perception of the communication, each protocol has as many procedural representations as the possible *roles* in the conversation. We will call *policy* each such role-centric implementation.

Let us consider, for instance the yes_no_query protocol reported in Fig. 3, a simplified version of the FIPA Query Interaction Protocol [31]. The protocol has two complementary views, one to be followed for performing a query ($\mathsf{yes\_no\_query}_Q$) and one for responding ($\mathsf{yes\_no\_query}_I$). Let us show how it would be possible to implement it in DyLOG.

(a) $\langle \mathsf{yes\_no\_query}_Q(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{queryIf}(Self, Other, Fluent); \mathsf{get\_answer}(Self, Other, Fluent)\rangle\varphi$

(b) $[\mathsf{get\_answer}(Self, Other, Fluent)]\varphi \equiv$
$[\mathsf{inform}(Other, Self, Fluent) \cup \mathsf{inform}(Other, Self, \neg Fluent) \cup$
$\mathsf{refuseInform}(Other, Self, Fluent)]\varphi$

Trivially, in $\mathsf{yes\_no\_query}_Q$ agent $Self$ performs a queryIf speech act then it waits for the answer of agent $Other$. The definitions of get_answer and get_start (the latter is reported hereafter, axiom (f)) are instances of the get_message axiom. Intuitively, the right hand side of get_answer represents all the possible answers expected by agent $Self$ from agent $Other$ about $Fluent$, in the context of a conversation ruled by the $\mathsf{yes\_no\_query}_Q$ conversation policy.

(c) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$
$\mathbf{B}^{Self} Fluent?; \mathsf{inform}(Self, Other, Fluent)\rangle\varphi$

(d) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$
$\mathbf{B}^{Self} \neg Fluent?; \mathsf{inform}(Self, Other, \neg Fluent)\rangle\varphi$

(e) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$

$\mathbf{U}^{Self} Fluent?$; $\mathsf{refuseInform}(Self, Other, Fluent)\rangle\varphi$

The $\mathsf{yes\_no\_query}_I$ protocol specifies the behavior of the agent $Self$, when it plays the role of the informer, waiting for a query from $Other$ and, then, replying in accordance to its beliefs on the query subject. Last but not least, rule (f) reports the axiom by which $\mathsf{get\_start}$ is defined:

(f) $[\mathsf{get\_start}(Self, Other, Fluent)]\varphi \equiv [\mathsf{queryIf}(Other, Self, Fluent)]\varphi$

It is a renaming of $\mathsf{queryIf}$.

We are now in condition to define the *communication kit*, denoted by CKit^{ag_i}, of an agent ag_i as the triple $(\Pi_{\mathbf{C}}, \Pi_{\mathbf{CP}}, \Pi_{\mathbf{S}_{get}})$, where $\Pi_{\mathbf{C}}$ is the set of simple action laws defining ag_i's speech acts, $\Pi_{\mathbf{S}_{get}}$ is the set of axioms that specify ag_i's $\mathsf{get\_message}$ actions and $\Pi_{\mathbf{CP}}$ is the set of procedure axioms specifying its conversation protocols.

A *domain description* defining an agent ag_i is, then, a triple $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$, where CKit^{ag_i} is the agent's communication kit, S_0 is ag_i's initial set of belief fluents, and Π is a specification of the agent's non-communicative behavior. It is a triple $(\Pi_{\mathbf{A}}, \Pi_{\mathbf{S}}, \Pi_{\mathbf{P}})$, where $\Pi_{\mathbf{A}}$ is the set of ag_i's world action and precondition laws, $\Pi_{\mathbf{S}}$ is the specification of a set of sensing action, $\Pi_{\mathbf{P}}$ a set of axioms that define complex actions.

3.3. Dealing with persistency

In the DyLOG framework, a non-monotonic solution is adopted to deal with the persistency problem. More precisely, an abductive semantics is proposed for the language, in which abductive assumptions are used to model the persistency of beliefs fluents, from a state to the following one, when an action is performed. The solution is a generalization of [8], that allows one to deal also with nested beliefs and communicative actions, and consists in maximizing persistency assumptions about epistemic fluents after the execution of action sequences. In particular any belief fluent F which holds in a given state is assumed to persist through an action, unless it is inconsistent to assume so, i.e. unless $\neg F$ holds after the action execution.

Notice that belief states are *inconsistent* when they contain either a belief $\mathbf{B}^{ag_i} l$ and its negation, or the belief formulas $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i}\neg l$, or the belief formulas $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_j}\neg\mathbf{B}^{ag_i} l$. However, from the *seriality* of the $\mathbf{B}^{ag_i}$ operators, the following general formula schema for the *rank 2* beliefs holds in the defined logic for any two agents ag_i and ag_j (actually, the general schema for any rank of nesting holds):

$$\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\neg\varphi \supset \neg\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\varphi \tag{10}$$

This property guarantees that when an inconsistency arises "locally" in the beliefs ascribed from ag_i to some other agent, the beliefs of ag_i itself will be inconsistent. Therefore, in case of a nested epistemic fluent $\mathbf{B}^{ag_i}\mathbf{B}^{ag_j} l$, the persistency is *correctly blocked* when a locally inconsistent fluent $\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\neg l$ becomes true after an action execution, because $\neg\mathbf{B}^{ag_i}\mathbf{B}^{ag_j} l$ can be derived from (10).

Given these considerations, the semantics is defined according to the method used by Eshghi and Kowalski in the definition of the abductive semantics for negation as failure [32]. A new set of atomic propositions of the form $\mathbb{M}[a_1]\cdots[a_m]F$ are defined as *abducibles*.[3] Their meaning is that the fluent expression F can be assumed to hold in the state obtained by the execution of the primitive actions $a_1, \ldots, a_m$. Each abducible can be assumed to hold, if it is consistent with the domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ and with the other assumed abducibles. Then, we add to the axiom system, that characterizes the logic defined by the domain description, the *persistency axiom schema*:

$$[a_1]\cdots[a_{m-1}]F \wedge \mathbb{M}[a_1]\cdots[a_{m-1}][a_m]F \supset [a_1]\cdots[a_{m-1}][a_m]F$$

where $a_1, \ldots, a_m$ are primitive actions and F is a belief fluent. It means that if F holds after $a_1, \ldots, a_{m-1}$, and it can be assumed to persist after action a_m (i.e., it is consistent to assume $\mathbb{M}[a_1]\cdots[a_m]F$), one can conclude that F holds after the sequence of actions $a_1, \ldots, a_m$.

[3] Notice that here $\mathbb{M}$ is not a modality. $\mathbb{M}\alpha$ denotes a new atomic proposition. $\mathbb{M}\alpha$ means "α is consistent", analogously to default logic.

3.4. Reasoning about conversations

Given a domain description, we can reason about it and formalize the *temporal projection* and the *planning* problem by means of existential queries of form:

$$\langle p_1\rangle\langle p_2\rangle\cdots\langle p_m\rangle Fs \tag{11}$$

where each p_k, $k = 1,\dots,m$ may be an (atomic or complex) action executed by ag_i or an external speech act, that belongs to CKit^{ag_i}. By the word *external* we denote a speech act in which our agent plays the role of the receiver. Checking if a query of form (11) succeeds corresponds to answering the question "Is there an execution trace of the sequence $p_1,\dots,p_m$ that leads to a state where the conjunction of belief fluents Fs holds for agent ag_i?" In case all the p_k's are atomic actions, it amounts to predict if the condition of interest will be true after their execution. In case complex actions are involved, the execution trace that is returned in the end is a *plan* to bring about Fs. The procedure definition constrains the search space.

A special case is when the procedure is a *conversation protocol*. In this case and by applying these same reasoning techniques, the agent will be able to predict how a conversation can affect its mental state and also to produce a conversation that will allow it achieve a communicative goal of interest. In this process $\mathsf{get\_message}$ actions are treated as sensing actions, whose outcome is not known at planning time – agents cannot read each other's mind, so they cannot know in advance the answers that they will receive. For this reason all of the possible alternatives are to be taken into account. This can be done because of the existence of the protocol. The extracted plan will, then, be *conditional*, in the sense that for each $\mathsf{get\_message}$ action and for each sensing action it will contain as many branches as possible action outcomes. Each path in the resulting tree is a *linear plan* that brings about the desired condition Fs. More formally:

(1) an action sequence $\sigma = a_1;\dots;a_m$, with $m \geq 0$, is a conditional plan;
(2) if $a_1;\dots;a_m$ $(m \geq 0)$ is an action sequence, $s \in \mathbf{S}$ is a sensing action, and $\sigma_1,\dots,\sigma_t$ are conditional plans, then $\sigma = a_1;\dots;a_m;s;((\mathbf{B}^{ag_i}l_1?);\sigma_1 \cup\cdots\cup (\mathbf{B}^{ag_i}l_t?);\sigma_t)$ where $l_1,\dots,l_t \in dom(s)$, is a conditional plan;
(3) if $a_1;\dots;a_m$ $(m \geq 0)$ is an action sequence, $g \in \mathbf{S}$ is a $\mathsf{get\_message}$ action, and $\sigma_1,\dots,\sigma_t$ are conditional plans, then $\sigma = a_1;\dots;a_k;g;((\mathbf{B}^{ag_i}Done(c_1)\top?);\sigma_1 \cup\cdots\cup (\mathbf{B}^{ag_i}Done(c_t)\top?);\sigma_t)$ where $c_1,\dots,c_t \in \mathbf{C}_g$, is a conditional plan.

In some applications it is actually possible to extract a conditional plan, that leads to the goal *independently* from the answers of the interlocutor. This mechanism will be used for web service selection in Section 4.2. An alternative is to look for a linear plan that leads to the goal, given some *assumptions* on the received answers. This approach does not guarantee that at *execution time* the services will stick to the planned conversation, but it allows finding a feasible solution when a conditional plan cannot be found. This is actually the case of web service composition (Section 4.3). If we compose *restaurant1* and *cinema1*, it is possible to find a conversation after which the user's desires about the credit card and about the use of promotional tickets are satisfied. However, the success of the plan depends on information that is known only at execution time (availability of seats) and that we *assumed* during planning. In fact, if no seat is available the goal of making a reservation will fail. The advantage of reasoning about protocols, in this latter situation, is that the information contained in the protocol is sufficient to exclude *a priori* a number of compositions that will never satisfy the goal. For instance, *restaurant1* plus *cinema2* does not permit to exploit a promotion independently from the availability of seats. The proof procedure that allows to reason about conversation protocols is a natural evolution of [8] and is described in Appendix A; it is goal-directed and based on negation as failure (NAF). NAF is used to deal with the persistency problem for verifying that the complement of a mental fluent is not true in the state resulting from an action execution, while in the modal theory we adopted an abductive characterization. The proof procedure allows agents to find *linear* plans for reaching a goal from an incompletely specified initial state. The soundness can be proved under the assumption of e-consistency, i.e. for any action the set of its effects is consistent [33]. The extracted plans always lead to a state in which the goal condition Fs holds.

4. Reasoning about conversations for web service selection and composition

This section reports a few examples aimed at showing the utility of a declarative representation of conversation protocols and policies in a Semantic Web framework. The scenario we refer to is the one introduced in Section 2.1. The web services that are involved can be classified in two categories, depending on their function: restaurant web services and cinema web services. The former allows a user to book a table at a given restaurant, the latter to book a seat at a given cinema. The interaction, however, is carried on in different ways. In particular, the services accept different forms of payment and only part of them allow users to benefit of promotions. The section is structured in the following way. First of all, we focus on knowledge representation and present the protocols used by the web services involved in the scenario. Afterwards, the two tasks of web service selection and web service composition will be tackled, showing how personalization plays an important role in both cases.

4.1. Representing conversation protocols in DyLOG

Let us begin with describing the *protocols*, that are followed by the web services. Such protocols allow the interaction of two agents (the service and the customer), so each of them encompasses two complementary views: the view of the web service and the view of the customer. Each view corresponds to an agent conversation policy, which is represented as a DyLOG procedure but for the sake of brevity, we report only the view of the customer. It is easy to see how the structures of the procedure clauses correspond to the sequence of AUML operators in the sequence diagrams. The subscripts next to the protocol names are a writing convention for representing the role that the agent plays; so, for instance, Q stands for *querier*, and C for *customer*. The customer view of the restaurant protocols is the following:

(a) $\langle \mathsf{reserv\_rest\_1}_C(Self, Service, Time)\rangle\varphi \subset$
$\quad\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Time))$;
$\quad\quad \mathbf{B}^{Self} available(Time)?$;
$\quad\quad \mathsf{get\_info}(Self, Service, reservation(Time))$;
$\quad\quad \mathsf{get\_info}(Self, Service, cinema\_promo)$;
$\quad\quad \mathsf{get\_info}(Self, Service, ft\_number)\rangle\varphi$

(b) $\langle \mathsf{reserv\_rest\_2}_C(Self, WebS, Time)\rangle\varphi \subset$
$\quad\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Time))$;
$\quad\quad \mathbf{B}^{Self} available(Time)?$;
$\quad\quad \mathsf{get\_info}(Self, WebS, reservation(Time))\rangle\varphi$

(c) $[\mathsf{get\_info}(Self, Service, Fluent)]\varphi \subset [\mathsf{inform}(Service, Self, Fluent)]\varphi$

Procedure (a) is the protocol procedure that describes the communicative behavior of the first restaurant: the customer asks if a table is available at a certain time, if so, the restaurant informs it that a reservation has been taken and that it gained a promotional free ticket for a cinema ($cinema\_promo$), whose code number (ft_number) is returned. The get_message action get_info and the protocol $\mathsf{yes\_no\_query}_Q$ have already been explained in Section 3. Procedure (b), instead, describes the communicative behavior of the second restaurant: the interaction is similar to the previous case but the restaurant does not take part to the promotion so the customer does not get any free ticket for the cinema. Clause (c) shows how get_info can be implemented as an inform act executed by the service and having as recipient the customer. The question mark amounts to check the value of a fluent in the current state; the semicolon is the sequencing operator of two actions. On the other hand, the cinema protocols are as follows:

(c) $\langle \mathsf{reserv\_cinema\_1}_C(Self, Service, Movie)\rangle\varphi \subset$
$\quad\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Movie))$;
$\quad\quad \mathbf{B}^{Self} available(Movie)?$;
$\quad\quad \mathsf{yes\_no\_query}_I(Self, Service, cinema\_promo)$;
$\quad\quad\quad \neg\mathbf{B}^{Self} cinema\_promo?$;
$\quad\quad\quad \mathsf{yes\_no\_query}_I(Self, Service, pay\_by(c\_card))$;
$\quad\quad\quad\quad \mathbf{B}^{Self} pay\_by(c\_card)?$;
$\quad\quad\quad\quad \mathsf{inform}(Self, Service, cc\_number)$;
$\quad\quad\quad\quad \mathsf{get\_info}(Self, Service, reservation(Movie))\rangle\varphi$

(d) $\langle \mathsf{reserv\_cinema\_1}_C(Self, Service, Movie)\rangle\varphi \subset$
$\quad\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Movie))$;
$\quad\quad \mathbf{B}^{Self} available(Movie)?$;
$\quad\quad \mathsf{yes\_no\_query}_I(Service, Self, cinema\_promo)$;

$\mathbf{B}^{Self} cinema\_promo?$;
$\mathsf{inform}(Self, Service, ft\_number)$;
$\mathsf{get\_info}(Self, Service, reservation(Movie))\rangle\varphi$

(e) $\langle \mathsf{reserv\_cinema\_2}_C(Self, WebS, Movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Movie))$;
$\mathbf{B}^{Self} available(Movie)?$;
$\mathsf{get\_info}(Self, WebS, pay\_by(cash))$;
$\mathsf{get\_info}(Self, WebS, reservation(Movie))\rangle\varphi$

Supposing that the desired movie is available, the first cinema alternatively accepts credit card payments, clause (c), or promotional tickets, clause (d). The second cinema, instead, accepts only cash payments, clause (e).

In the following, the selection and composition tasks, that the personalization agent *pa* can accomplish, are described. As explained in Section 2.1, these tasks are aimed at refining a previous selection, performed by a retriever. In the example that we are using, when *pa* will start the personalization task the following list of candidate services will already be available in the DyLOG knowledge base as well as the protocols that they follow:

$\mathbf{B}^{pa} service(restaurant, restaurant1, reserv\_rest\_1_C)$
$\mathbf{B}^{pa} service(restaurant, restaurant2, reserv\_rest\_2_C)$
$\mathbf{B}^{pa} service(cinema, cinema1, reserv\_cinema\_1_C)$
$\mathbf{B}^{pa} service(cinema, cinema2, reserv\_cinema\_2_C)$

The agent will personalize the interaction with the services, according to the requests of the user, dismissing services that do not fit. To this aim, it will reason about the procedures $select\_service$ or $comp\_services$, reported hereafter.

4.2. Web service selection by reasoning about interaction

Web service selection by means of reasoning about a service conversation policy, amounts to answering to the query "Is there a possible conversation among those allowed by the service protocol, after which a condition of interest holds?". In the scenario depicted in Section 2.1, an example is the desire of the user of avoiding credit card payments over the web (for instance when booking a cinema ticket for the movie $akira$). Let us suppose that a preliminary selection has identified *cinema1* and *cinema2*, whose conversation policies are described in the previous section, clauses (c)–(e), as cinemas that show $akira$. A further selection based on *reasoning* can begin by reasoning about the (Prolog-like) procedure $select\_service$:

$\langle select\_service(TypeService, Name, Data)\rangle\varphi \subset$
$\langle \mathbf{B}^{pa} service(TypeService, Name, Protocol)?\ ;\ Protocol(pa, Name, Data)\ \rangle\varphi$

Let us consider the query expressing the fact that the personalization agent wants to select a cinema booking service, that allows a dialogue by which the credit card number of the user is not communicated to the service, that is, the condition that must hold in *pa*'s mental state after the procedure's execution is $\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$. This condition is a nested belief, that is, a belief about the knowledge of another agent (the cinema service). This is the query:

$\langle select\_service(cinema, C, akira)\rangle\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number)$

C is a variable that ranges over the set of cinema names in the knowledge base. This query, as we will show, succeeds with answer C equal to *cinema2*.

Let us begin with considering *cinema1* ($Protocol$ will be equal to $\mathsf{reserv\_cinema\_1}_C$). This service will be selected if it is possible to answer the query $\langle \mathsf{reserv\_cinema\_1}_C(pa, cinema1, akira)\rangle\mathbf{B}^{pa}\neg\mathbf{B}^{cinema1} cc\_number$. It is easy to see from the protocol specification that such an interaction is possible, given that the user owns a free ticket. However, let us suppose that this is not the case and that the initial mental state of the agent contains the beliefs: $\mathbf{B}^{pa} cc\_number$ i.e. the agent knows the user's credit card number which is not to be used, $\neg\mathbf{B}^{pa} cinema\_promo$, the user does not have a free ticket, $\neg\mathbf{B}^{pa} pay\_by(c\_card)$, the agent is an authority about the method of payment. Moreover, suppose that *pa* also has some hypothesis about the knowledge of the cinema services, like $\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$ (with C varying on $\{cinema1, cinema2\}$), the services do not already know the credit card number. Of course, initially the agent will also believe that no ticket for $akira$ has been booked yet ($\mathbf{B}^{pa}\neg booked(akira)$).

When the agent reasons about the protocol execution, the test $\mathbf{B}^{pa} cinema\_promo?$ fails as well as the subsequent test $\mathbf{B}^{pa} pay\_by(c\_card)?$ fails. As a result, *cinema1* is not selected, and $select\_service$ considers the other option, i.e. *cinema2*. In this case the analogous query succeeds and an execution trace of the protocol $\mathsf{reserv\_cinema\_2}_C$ is returned as a result. It is the following *conditional dialogue plan* between pa and the *cinema2*:

$\mathsf{queryIf}(pa, cinema2, available(akira));$
$((\mathbf{B}^{pa} Done(\mathsf{inform}(cinema2, pa, akira))\top?);$
$\mathsf{get\_info}(pa, cinema2, pay\_by(cash));$
$(\mathbf{B}^{pa} Done(\mathsf{inform}(cinema2, pa, pay\_by(cash)))\top?);$
$\mathsf{get\_info}(pa, cinema2, reservation(akira));$
$(\mathbf{B}^{pa} Done(\mathsf{inform}(cinema2, pa, reservation(akira)))\top?) \cup$
$(\mathbf{B}^{pa} Done(\mathsf{inform}(cinema2, pa, \neg akira))\top?) \cup$
$(\mathbf{B}^{pa} Done(\mathsf{refuseInform}(cinema2, pa, akira))\top?))$

Notice that no communication involves the belief $\mathbf{B}^{pa}\neg\mathbf{B}^{cinema2} cc\_number$, which persists from the initial state; thus, at the end of each execution branch the user's desire of keeping the credit card number secret is satisfied.

4.3. Web service composition by reasoning about interaction

The other task of interest is web service composition. Again, suppose that a preliminary selection has already been accomplished, resulting in the a set of candidate services listed at the end of Section 4.1. In this case, the user wants to book a table at a restaurant plus a cinema entrance profiting of the promotion. For accomplishing this task the personalization agent reasons about the procedure $comp\_services$ (a possible implementation is reported below), that sketches the general composition-by-sequencing of a set of services, based on their interaction protocols.

$\langle comp\_services([\,])\rangle\varphi \subset true$
$\langle comp\_services([[TypeService, Name, Data]|Services])\rangle\varphi \subset$
$\langle \mathbf{B}^{pa} service(TypeService, Name, Protocol)?\,;$
$Protocol(pa, Name, Data)\,;\, comp\_services(Services)\rangle\varphi$

Intuitively, $comp\_services$ builds a sequence of protocols so that it will be possible to reason about them as a whole. Let us now consider the query:

$\langle comp\_services([[restaurant, R, dinner], [cinema, C, akira]])\rangle$
$(\mathbf{B}^{pa} cinema\_promo \wedge \mathbf{B}^{pa} reservation(dinner) \wedge$
$\mathbf{B}^{pa} reservation(akira) \wedge \mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number \wedge \mathbf{B}^{pa}\mathbf{B}^{C} ft\_number)$

that amounts to determine if it is possible to compose the interaction with a restaurant and a cinema web services, reserving a table for dinner ($\mathbf{B}^{pa} reservation(dinner)$) and booking a ticket for the movie $akira$ ($\mathbf{B}^{pa} reservation(akira)$), exploiting a promotion ($\mathbf{B}^{pa} cinema\_promo$). Credit card is not to be used ($\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$), the free ticket is to be spent ($\mathbf{B}^{pa}\mathbf{B}^{C} ft\_number$). The agent initial mental state contains beliefs about the user's credit card number ($\mathbf{B}^{pa} cc\_number$), the desire to avoid using it ($\neg\mathbf{B}^{pa} pay\_by(credit\_card)$), and the fact that the agent is an authority about the form of payment. Further assumptions are that no reservation for dinner nor for the movie has been taken yet, $\mathbf{B}^{pa}\neg reservation(dinner)$ and $\mathbf{B}^{pa}\neg reservation(akira)$, that pa does not have a free ticket for the cinema, $\neg\mathbf{B}^{pa} cinema\_promo$, and some hypothesis on the interlocutor's mental state, e.g. the belief fluent $\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$ (with C in $\{cinema1, cinema2\}$), meaning that the cinema booking services do not already know the credit card number. In this context, the query succeeds with answer R equal to *restaurant1* and C equal to *cinema1*, and the agent builds the following execution trace of $comp\_services$ $([[restaurant, restaurant1, dinner], [cinema, cinema1, akira]])$:

$\mathsf{queryIf}(pa, restaurant1, available(dinner))\,;$
$\boxed{\mathsf{inform}(restaurant1, pa, available(dinner))\,;}$
$\mathsf{inform}(restaurant1, pa, reservation(dinner))\,;$
$\mathsf{inform}(restaurant1, pa, cinema\_promo)\,;$
$\mathsf{inform}(restaurant1, pa, ft\_number)\,;$
$\mathsf{queryIf}(pa, cinema1, available(akira))\,;$

$\boxed{\mathsf{inform}(cinema1, pa, available(akira))}$;
$\mathsf{queryIf}(cinema1, pa, cinema\_promo)$;
$\mathsf{inform}(pa, cinema1, cinema\_promo)$;
$\mathsf{inform}(pa, cinema1, ft\_number)$;
$\mathsf{inform}(cinema1, pa, reservation(akira))$

This is a dialogue plan that is made of a conversation between pa and $restaurant1$ followed by one between pa and $cinema1$, instances of the respective conversation protocols, after which the desired condition holds. The linear plan, will, lead to the desired goal given that some *assumptions* (above outlined with a box) about the provider's answers hold. For instance, that there is a free seat at the cinema, a fact that can be known only at execution time. Assumptions occur when the interlocutor can respond in different ways depending on its internal state. It is not possible to know in this phase which the answer will be, but since the set of the possible answers is given by the protocol, it is possible to identify the subset that leads to the goal. In the example they are answers foreseen by a yes_no_query protocol. Returning such assumptions to the designer is also very important to understand the correctness of the implementation w.r.t. the chosen speech act ontology.

5. Conclusions and related works

The work presented in this article is set in the Semantic Web field of research and faces some issues related to web service selection and composition. The basic idea is to consider a service as a software agent and the problem of composing a set of web services as the problem of making a set of software agents cooperate within a multi-agent system (or MAS). This interpretation is, actually, quite natural, although somewhat new to the web service community, with a few exceptions [34,35]. In particular, we have studied the possible benefits provided by the introduction of an explicit (and declarative) description of the communicative behavior of the web services in terms of *personalization* of the service fruition and of the *composition* of a set of services. Indeed, a web service must follow some possibly non-deterministic procedure aimed at getting/supplying all the necessary information. So far, however, standard languages for *semantic* web service description do not envision the possibility of separating the communicative behavior from the rest of the description. On the contrary, communication plays a central role in languages for describing workflows and business processes, like BPEL. Unfortunately, such languages do not have a formalization that enables the automation, based on reasoning, of the tasks that are the focus of this work [36,37].

The idea of setting web service selection and composition in the Semantic Web rather than in the WWW (that is of representing web services according to a formal model that enables automated forms of reasoning), is motivated by a growing agreement, in the web service research community, on the need of finding representation models that abstract away from the specific languages (like BPEL) used to specify business processes, and which allow the analysis of properties of the interactions that are executed [16]. The introduction of choreographies as global schemas of interaction (and of languages like WS-CDL [17]) as well as the distinction between this global view from the local perspective of the single services are a first consequence of such needs. At the moment of writing, in the Semantic Web area no concept equivalent (or at least close to) choreographies has been defined yet, nevertheless, the languages proposed for web service representation share with BPEL the characteristic of representing the local view of the interaction, that each single process should have. The interesting point is that such languages have been designed expressly with the aim of supplying abstract, formal representations. Another general agreement is on the principle that the richer the semantic information that is used, the higher the precision of the answer to a query, where *precision* is the extent to which only the items "really of interest" are returned [38]. Depending on the kind of resources and of tasks of interest it is necessary to identify the right piece of information to use. In the approach that we have proposed the user's goals and needs have this role, and to be matched, they require the rational inspection of the interactive behavior of the services, which consequently is to be represented in an explicit way.

The approach, proposed in this paper for performing the inspection, is based on techniques for reasoning about actions and change, in which the communicative behavior of the services is modelled as a complex action (a procedure) based on speech acts. In this framework the selection and composition tasks can be interpreted as the process of answering the questions "Is it possible to interact with this service in such a way that this condition will hold afterwards?" and "Is it possible to execute these services in sequence in such a way that this condition will hold afterwards?". Of course the effective achievement of the goal might depend on conditions, that are necessary to the completion of the interaction but that will be known only at execution time. For instance, in order to book a table at a restaurant it is necessary that there is at least one free table. We have not tackled this problem yet, although it would be very interesting to integrate in the approach that we have proposed a mechanism for dealing with *failure* (at execution time) and *replanning*. This form of reasoning is necessary to arrive to real applications and it could take into account also degrees of preference explicitly expressed by the user. Such criteria could be used to achieve a greater flexibility, by relaxing some of the constraints in case no plan can be found or when a plan execution fails. In particular, in our opinion it would be extremely interesting to study *integrations* of replanning techniques with *compensation techniques*. Indeed, in the research community that studies planning the interest is mainly posed on the efficiency of the planner and replanning, if any, does not exploit any explicit representation of actions for undoing things, with a few exceptions [39]. Given the current state, whatever it is, forward actions are searched for reaching the goal. The process might have, as a side effect, the undoing of some partial achievement but this will be an occasional outcome. The main achievement of the proposed approach, however, is that the presence of public protocol/policy specifications gives the agent the possibility to *set up* plans. Even though this plan might be subject to assumptions (e.g. that a table will be available), the agent will *know in advance* if it is worthwhile to interact with that partner. In a way, the approach that has been proposed can be seen as a sieve that allows agents to brush off a number of partners before the interaction. Nevertheless, it is something more than a sieve because it also allows the identification of *courses of interaction* that the agent is willing to perform. In this perspective, it is actually a valuable tool for personalizing the interaction with web services according to the goals of the user.

Our proposal can be considered as an approach based on the process ontology, a *white box* approach in which part of the behavior of the services is available for a rational inspection. In this case, the deductive process exploits more semantic information: in fact, it does not only take into account the pre- and post-conditions, as above, it also takes into account the complex behavior (the communicative behavior) of the service. The idea of focussing on abstract descriptions of the communicative behavior is, actually, a novelty also with respect to other proposals that are closer to the agent research community and more properly set in the Semantic Web research field. The work that is surely the closest to ours is the one by the DAML-S (now OWL-S) coalition, that designed the language OWL-S [40]. An OWL-S service description has three conceptual levels: the profile, used for advertising and discovery, where inputs, outputs, preconditions and effects are enumerated, the process model, a declarative description of the structure of a service, and the grounding, that describes how an agent can access the service by means of low-level (SOAP) messages. To our aims, the most interesting component of an OWL-S web service description is the process model, which describes a service as atomic, simple (viewed as atomic) or composite, in a way inspired by the language GOLOG and its extensions [30,28,41,40]. However, to our knowledge, no approach for reasoning about the process model has been proposed yet. In the works by McIlraith et al. (e.g. [42]), indeed the most relevant for us, the idea is always to compose services that are viewed as *atomic*, and the composition is merely based on their preconditions and effects, exploiting techniques derived from the Situation Calculus. In particular, the precondition and effect, input and output lists are flat; no relation among them can be expressed, so it is impossible to understand if a service can follow alternative courses of interaction, among which one could choose. Indeed, the advantage of working at the protocols level is that by reasoning about protocols agents can personalize the interaction by selecting a course that satisfies user- (or service-) given requirements. This process can be started before the actual interaction takes place.

Of course, other approaches to matchmaking have been proposed and are being investigated. For instance, the frame-based approaches, like the UDDI registry service for WSDL web services [13]. In this case two sets of textual property values are compared, a service description and a query (i.e. the description of a desired service); both descriptions are based on partially pre-enumerated vocabularies of service types and properties. Close to frame-based approaches is the set-based modelling approach proposed in the WSMO service discovery. In this case the sets of objects to be compared are the set of user's goals (intended as the information that the user wants to receive as output) and the set of objects delivered after a web service execution. The two sets might be inter-related in some way by ontologies, which are considered as domain models. Actually, four different degrees of match are formalized, depending on additional properties that should hold and a complete formalization based on description logics is explained [43]. A refinement of this proposal, in which more elaborated descriptions are taken into account, is proposed as a second step. At this higher level of formalization, services are considered as relations on an abstract state-space and are described in terms of inputs, outputs, preconditions, effects, assumptions, and post-conditions; in this extension exploits of interest are identified by applying transaction logics. Another category is that of deductive retrieval (also known as "reuse by contract"), well described in [44]. Here web services, interpreted as software components, are seen as black-boxes, whose description relies on the concept of Abstract Data Type (or ADT). The semantics of an ADT is given by a set of logic axioms. Part of the approaches in this family (the "plug in match" approaches [45]) use these logic axioms to identify the pre- and post-conditions to the execution of the web service. In this case also queries are represented by a set of pre- and post-conditions. The decision of whether a service matches a given query depends on the truth value of the formula $(pre_Q \supset pre_{WS}) \cap (post_Q \supset post_{WS})$, where pre_Q and $post_Q$ are the pre- and post-conditions of the query and pre_{WS} and $post_{WS}$ are the pre- and post-conditions of the service. Many works should be cited in this line of research, like NORA/HAMRR [46], feature-based classification [47], LARKS [48], SWS matchmaker [49], up to PDDL-based languages (PDDL stands for "Planning Domain Definition Framework" [50]) like the proposal in [51]. To our knowledge none of these works is based on reasoning on an explicit representation of the interactive behavior of the services, even in the case in which PDDL is used, the idea is to consider web services as atomic actions.

Appendix A. Goal-directed proof procedure for DyLOG

This appendix presents the proof procedure used to build linear plans, making assumptions on sensing actions and on external communicative actions. Then, a variant that builds conditional plans is introduced, where all the possible values returned by sensing and by incoming communications are taken into account. For the sake of brevity, we do not report in this paper the demonstrations. Actually, these are very similar to those for DyLOG without communication kit, see [8,52] for details.

Appendix A.1. Linear plan extraction

A query (see Section 3.4) of the form $\langle p_1; p_2; \ldots; p_n\rangle Fs$ succeeds if it is possible to execute in the given order $p_1, p_2, \ldots, p_n$, starting from the current state, in such a way that Fs holds in the resulting belief state. Since a state can be represented by the sequence of atomic actions performed for reaching it, in general, we write:

$$a_1, \ldots, a_m \vdash \langle p_1; p_2; \ldots; p_n\rangle Fs \text{ with answer (w.a.) } \sigma$$

where $a_1, \ldots, a_m$ represents the current state, to mean that the query can be proved with answer σ in the current state and from the domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$. The answer σ is an execution trace $a_m, \ldots, a_{m+k}$ of $p_1, \ldots, p_n$ in the current state. We denote by ε the initial mental state.

1) $$\frac{\overline{a}_{1\cdots m} \vdash \langle p'_1; \ldots; p'_{n'}; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle p; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } p \in \mathbf{P} \text{ and } \langle p\rangle\varphi \subset \langle p'_1; \ldots; p'_{n'}\rangle\varphi \in \Pi_{\mathbf{P}} \cup \Pi_{\mathbf{CP}}$$

2) $$\frac{\overline{a}_{1\cdots m} \vdash Fs' \quad \overline{a}_{1\cdots m} \vdash \langle \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle (Fs')?; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}$$

3) $$\frac{\overline{a}_{1\cdots m} \vdash Fs' \quad \overline{a}_{1\cdots m}, a \vdash \langle \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle a; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } a \in \mathbf{A} \cup \mathbf{C}, \text{ and } \Box(Fs' \supset \langle a\rangle\top) \in \Pi_{\mathbf{A}} \cup \Pi_{\mathbf{C}}$$

4) $$\frac{\overline{a}_{1\cdots m} \vdash \langle s^{\mathbf{B}^{ag_i}l}; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle s; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } s \in \mathbf{S} \text{ and } l \in dom(s)$$

5) $$\frac{\overline{a}_{1\cdots m} \vdash \langle c; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle g; \overline{p}_{2\cdots n}\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } g \in \mathbf{S}get \text{ and } [g]\varphi \equiv [\textstyle\bigcup_{c \in \mathbf{C}_g} c]\varphi$$

6) $$\frac{\overline{a}_{1\cdots m} \vdash Fs}{\overline{a}_{1\cdots m} \vdash \langle \varepsilon\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } \sigma = a_1; \ldots; a_m$$

Fig. A.1. Rules (1–6) of the goal-directed proof procedure for DyLOG. $\overline{a}_{1\cdots m}$ and $\overline{p}_{2\cdots n}$ stands for $a_1, \ldots, a_m$ and $p_2, \ldots, p_n$, respectively. l denotes a fluent literal or a done fluent while L denotes l or a belief fluent of rank 1.

$$6)\quad \frac{\overline{a}_{1\cdots m} \vdash Fs}{\overline{a}_{1\cdots m} \vdash \langle\varepsilon\rangle Fs \text{ w. a. } \sigma} \qquad \text{where } \sigma = a_1;\ldots;a_m$$

$$7)\quad \frac{}{\overline{a}_{1\cdots m} \vdash \top}$$

$$8a)\quad \frac{\overline{a}_{1\cdots m-1} \vdash Fs'}{\overline{a}_{1\cdots m} \vdash F} \qquad \begin{array}{c}\text{where } m > 0 \text{ and}\\ \Box(Fs' \supset [a_m]F) \in \Pi_{\mathbf{A}}\end{array}$$

$$8b)\quad \frac{}{\overline{a}_{1\cdots m} \vdash F} \qquad \text{if } a_m = s^F$$

$$8c)\quad \frac{\textbf{not } \overline{a}_{1\cdots m} \vdash \neg F \;\; \overline{a}_{1\cdots m-1} \vdash F}{\overline{a}_{1\cdots m} \vdash F} \qquad \text{where } m > 0$$

$$8d)\quad \frac{}{\varepsilon \vdash F} \qquad \text{if } F \in S_0$$

$$9)\quad \frac{\overline{a}_{1\cdots m} \vdash Fs' \;\; \overline{a}_{1\cdots m} \vdash Fs''}{\overline{a}_{1\cdots m} \vdash Fs' \wedge Fs''}$$

$$10)\quad \frac{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} L}{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} L}$$

$$11)\quad \frac{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i}\mathbf{B}^{ag_i} l} \qquad 11')\quad \frac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i}\mathbf{M}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} l}$$

$$12)\quad \frac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i}\mathbf{M}^{ag_i} l} \qquad 12')\quad \frac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i}\mathbf{B}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} l}$$

$$13)\quad \frac{\overline{a}_{1\cdots m} \vdash Done(a)\top}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} Done(a)\top} \qquad 14)\quad \frac{}{\overline{a}_{1\cdots m} \vdash Done(a_m)\top}$$

Fig. A.2. Rules (7–14) of the goal-directed proof procedure for DyLOG. $\overline{a}_{1\ldots m}$ and $\overline{p}_{2\ldots n}$ stands for $a_1, \ldots, a_m$ and $p_2, \ldots, p_n$, respectively. l denotes a fluent literal or a done fluent while L denotes l or a belief fluent of rank 1.

$$\text{4-bis)}\quad \frac{\forall l_k \in \mathbf{F},\; \overline{a}_{1\cdots m} \vdash \langle s^{\mathbf{B}^{ag_i} l}; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } a_1;\ldots;a_m; s^{\mathbf{B}^{ag_i} l}; \sigma'_k}{\overline{a}_{1\cdots m} \vdash \langle s; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } a_1;\ldots;a_m; s; (\bigcup_{k=1\ldots t}(\mathbf{B}^{ag_i} l_k?); \sigma'_k)}$$

$$\text{5-bis)}\quad \frac{\forall c_k \in \mathbf{C}_g,\; \overline{a}_{1\cdots m} \vdash \langle c_k; \overline{p}_{2\ldots n}\rangle Fs_i \text{ w. a. } a_1;\ldots;a_m; c_k; \sigma'_k}{\overline{a}_{1\cdots m} \vdash \langle g; \overline{p}_{2\ldots n}\rangle Fs_i \text{ w. a. } a_1;\ldots;a_m; g; (\bigcup_{k=1\ldots t}(\mathbf{B}^{ag_i} Done(c_k)\top?); \sigma'_k)}$$

Fig. A.3. A variant of the proof procedure for extracting conditional plans. In (4-bis) $s \in \mathbf{S}$ and $\mathbf{F} = \{l_1, \ldots, l_t\} = dom(s)$; in (5-bis) $g \in \mathbf{S}$ and $\{c_1, \ldots, c_t\} = \mathbf{C}_g$.

The first part of the proof procedure, rules (1–6) in Fig. A.1, deals with the execution of complex actions, sensing actions, primitive actions and test actions. The proof procedure reduces the complex actions in the query to a sequence of primitive actions and test actions, then it verifies if the execution of primitive actions is possible and if test actions are successful. To do this, it reasons about the execution of a sequence of primitive actions from the initial state and computes the values of fluents at different states. During a computation, a state is represented by a sequence of primitive actions $a_1, \ldots, a_m$. The value of fluents at a state is not explicitly recorded but it is computed when needed. The second part of the procedure, rules (7–14), allows the values of mental fluents in an agent ag_i state to be determined.

Let us briefly comment the rules. To execute a *complex action* p the modality $\langle p\rangle$ is non-deterministically replaced with the modality in the antecedent of a suitable axiom, rule (1). To execute a *test action* $(Fs)?$, the value of Fs is checked in the current state; if Fs holds, the test action is eliminated otherwise the computation fails, rule (2). To execute a *primitive action* a, first precondition laws are checked to verify if the action is possible. If they hold, the computation moves to a new state in which the action has been performed, rule (3). To execute a *sensing action* s, rule (4), we non-deterministically replace it with one of the primitive actions which define it, that, when it is executable, will cause $\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_i} \neg l'$, for each $l' \in dom(s)$, with $l \neq l'$. Rule (5) deals with get_message actions: a get_message *action* g is non-deterministically replaced with one of the external communicative actions which define it.

Rule (6) deals with the case when no more actions are to be executed. The desired sequence of primitive actions has already been determined so, to check if Fs is true after it, rules (7-14) in Fig. A.2 are used. An *epistemic fluent* F

holds at a state $a_1, \dots, a_m$ if either F is an immediate effect of action a_m (rule 8a); or action a_m is a primitive action s^F (introduced to model the sensing action s), whose effect is to add F to the state (rule 8b); or F holds in the previous state $a_1, \dots, a_{m-1}$ and it persists after a_m, rule (8c); or $a_1, a_2, \dots, a_m$ is the initial state and F already holds in it, rule (8d). Notice that rule (8c) allows to deal with the *frame problem*: F persists from a state to the next one unless the executed action a_m makes $\neg F$ true, i.e. it persists if $\neg F$ fails from $a_1, a_2, \dots, a_m$. In this rule **not** represents *negation as failure*. Rule (9) deals with *conjunction*. Rule (10) allows $\mathbf{M}^{ag_i} l$ to be concluded from $\mathbf{B}^{ag_i} l$, this is justified by the property of seriality of the belief modality. Rules (11) and (11′) have been introduced for coping with *transitivity of beliefs*. Rules (12) and (12′) tackle their *euclideaness*. Rules (13) and (14) have been introduced to provide *awareness* of the action execution.

Under the assumption of *e-consistency*, i.e. for every set of action laws for a given action which may be applied in the same state, the set of their effects is consistent [33], of the domain description and of consistency of the initial situation, the proof procedure is *sound* w.r.t. the non-monotonic semantics. First, it is necessary to show that the proof procedure is sound and complete w.r.t. the monotonic Kripke semantics; then, it is possible to show the soundness of the non-monotonic part.

Finally, let $\langle p_1; \dots; p_n \rangle Fs$ be an existential query and σ the answer returned by one of its successful derivations. It is possible to show that σ is effectively an execution trace of $p_1; \dots; p_n$, that is, given a domain description, $\langle \sigma \rangle Fs \supset \langle p_1; \dots; p_n \rangle Fs$. Moreover, σ is a *legal* sequence of atomic actions, it can actually be executed, and it always leads to a state in which Fs holds, i.e. the $\langle \sigma \rangle \top$ and $[\sigma] Fs$ hold.

Appendix A.2. Building conditional plans

Let us now introduce a variant of the proof procedure presented above which, given a query $\langle p_1; p_2; \dots; p_n \rangle Fs$, computes a *conditional plan* σ. All the executions in σ are possible behaviors of the sequence $p_1; p_2; \dots; p_n$. The new proof procedure is obtained by replacing rules (4) and (5) in Fig. A.1 (to handle sensing actions and get message actions, respectively) with rules (4-bis) and (5-bis) in Fig. A.3. As a difference with the previous case, when a sensing action is executed, the procedure now considers all the possible outcomes of the action, so that the computation splits in more branches. The resulting plan will contain a branch for each value that leads to success. The same holds for the get_message actions, which indeed are treated as a special case of sensing.

References

[1] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, N. Russell, H.M.W. Verbeek, P. Wohed, Life after BPEL? in: Proc. of WS-FM’05, LNCS, vol. 3670, Springer, 2005, pp. 33–50 (invited speaker).

[2] M. Baldoni, C. Baroglio, N. Henze, Personalization for the Semantic Web, in: Reasoning Web, LNCS Tutorial, vol. 3564, Springer, 2005, pp. 173–212.

[3] G. Antoniou, M. Baldoni, C. Baroglio, R. Baungartner, F. Bry, T. Eiter, N. Henze, M. Herzog, W. May, V. Patti, S. Schaffert, R. Schidlauer, H. Tompits, Reasoning methods for personalization on the semantic web, Ann. Math. Comput. Teleinformatics (AMCT) 2 (1) (2004) 1–24.

[4] F. Dignum, M. Greaves, Issues in agent communication, in: Issues in Agent Communication, LNCS, vol. 1916, Springer, 2000, pp. 1–16.

[5] F. Dignum (Ed.), Advances in Agent Communication Languages, LNAI, vol. 2922, Springer-Verlag, 2004.

[6] FIPA, Communicative act library specification, Tech. rep., FIPA (Foundation for Intelligent Physical Agents), 2002.

[7] T. Finin, Y. Labrou, J. Mayfield, KQML as an Agent Communication Language, in: J. Bradshaw (Ed.), Software Agents, MIT Press, 1995.

[8] M. Baldoni, L. Giordano, A. Martelli, V. Patti, Programming Rational Agents in a Modal Action Logic, Ann. Math. Artif. Intell. 41 (2–4) (2004), 207–257 (Special issue on Logic-Based Agent Implementation).

[9] M. Baldoni, C. Baroglio, V. Patti, Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions, Artificial Intelligence Rev. 22 (1) (2004) 3–39.

[10] A. Mamdani, J. Pitt, Communication protocols in multi-agent systems: a development method and reference architecture, in: Issues in Agent Communication, LNCS, vol. 1916, Springer, 2000, pp. 160–177.

[11] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, Reasoning about self and others: communicating agents in a modal action logic, in: Proc. of ICTCS’2003, LNCS, vol. 2841, Springer, 2003, pp. 228–241.

[12] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, Reasoning about interaction protocols for web service composition, in: Bravetti and Zavattaro [15], pp. 21–36, Electronic Notes in Theoretical Computer Science, vol. 105.

[13] A. Barros, M. Dumas, P. Oaks, A critical overview of the web services choreography description language(ws-cdl), Business Process Trends. Available from: <http://www.bptrends.com>.

[14] BPEL4WS, 2003. Available from: <http://www-106.ibm.com/developerworks/library/ws-bpel>.

[15] M. Bravetti, G. Zavattaro (Eds.), Proc. of the 1st Int. Workshop on Web Services and Formal Methods (WS-FM 2004), Elsevier Science Direct, 2004, Electronic Notes in Theoretical Computer Science, vol. 105.
[16] M. Bravetti, L. Kloul, G. Zavattaro (Eds.), Proc. of the 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005), LNCS, vol. 3670, Springer, 2005.
[17] WS-CDL, 2004. Available from: <http://www.w3.org/tr/2004/wd-ws-cdl-10-20041217/>.
[18] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American.
[19] REWERSE Network of Excellence, 2004. Available from: <http://www.rewerse.org>.
[20] G. Antoniou, F. van Harmelen, A Semantic Web Primer, MIT Press, 2004.
[21] WSMO, 2005. Available from: <http://www.wsmo.org/>.
[22] J.H. Odell, H. Van Dyke Parunak, B. Bauer, Representing agent interaction protocols in UML, in: Agent-Oriented Software Engineering, Springer, 2001, pp. 121–140. Available from: <http://www.fipa.org/docs/input/f-in-00077/>.
[23] Foundation for InteroPerable Agents, Fipa modeling: Interaction diagrams, Tech. rep., working Draft Version 2003-07-02, 2003.
[24] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, C. Schifanella, Verifying the conformance of web services to global interaction protocols: a first step, in: Proc. of 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005, LNCS, vol. 3670, 2005, pp. 257–271.
[25] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, Choreography and Orchestration: a synergic approach for system design, in: Proc. the 3rd Int. Conf. on Service Oriented Computing, 2005.
[26] K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, V. Subrahmanian, IMPACT: a platform for collaborating agents, IEEE Intell. Systems 14 (2) (1999) 64–72.
[27] M. Fisher, A survey of concurrent METATEM – the language and its applications, in: D. Gabbay, H. Ohlbach (Eds.), Proc. of the 1st Int. Conf. on Temporal Logic, ICTL'94, LNAI, vol. 827, Springer-Verlag, 1994, pp. 480–505.
[28] G.D. Giacomo, Y. Lesperance, H. Levesque, Congolog, a concurrent programming language based on the situation calculus, Artificial Intelligence 121 (2000) 109–169.
[29] J. Leite, A. Omicini, P. Torroni, P. Yolum (Eds.), Int. Workshop on Declarative Agent Languages and Technology, New York City, NY, USA, 2004. Available from: <http://centria.di.fct.unl.pt/∼jleite/dalt04>.
[30] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R.B. Scherl, GOLOG: a logic programming language for dynamic domains, J. Logic Programming 31 (1997) 59–83.
[31] FIPA, Fipa 2000, fipa Query Interaction Protocol Specification, Tech. rep., FIPA (Foundation for Intelligent Physical Agents), November 2000.
[32] K. Eshghi, R. Kowalski, Abduction compared with negation by failure, in: Proc. of ICLP'89, The MIT Press, Lisbon, 1989.
[33] M. Denecker, D. De Schreye, Representing incomplete knowledge in abduction logic programming, in: Proc. of ILPS'93, The MIT Press, Vancouver, 1993.
[34] J. Bryson, D. Martin, S. McIlraith, L.A. Stein, Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web, 2002. Available from: <citeseer.nj.nec.com/bryson02agentbased.html>.
[35] K. Sycara, Brokering and matchmaking for coordination of agent societies: a survey, in: A. Omicini et al. (Ed.), Coordination of Internet Agents, Springer, 2001.
[36] D.J. Mandell, S.A. McIlraith, Adapting BPEL4WS for the Semantic Web: the bottom-up approach to web service interoperation, in: Proc. of the 2nd Int. Semantic Web Conference (ISWC2003), Sanibel Island, FL, 2003.
[37] J. Koehler, B. Srivastava, Web service composition: current solutions and open problems, in: ICAPS 2003 Workshop on Planning for Web Services, 2003, pp. 28–35.
[38] M. Klein, A. Bernstein, Searching for services on the semantic web using process ontologies, in: Proc. of the Int. Semantic Web Working Symposium (SWWS), Stanford University, California, USA, 2001.
[39] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso, Planning and monitoring web service composition, in: Proc. of Planning and Monitoring Web Service Composition Artificial Intelligence: Methodology, Systems, Application (AIMSA) 2004, 2004.
[40] OWL-S, http://www.daml.org/services/owl-s/1.1/, 2004.
[41] S. McIlraith, T. Son, Adapting Golog for programming the Semantic Web, in: 5th Int. Sympos. on Logical Formalization of Commonsense Reasoning, 2001, pp. 195–202.
[42] S. Narayanan, S.A. McIlraith, Simulation, verification and automated composition of web services, Honolulu, Hawaii, USA, 2002, pp. 77–88.
[43] U. Keller, R.L.A. Polleres, I. Toma, M. Kifer, D. Fensel, D5.1 v0.1 wsmo web service discovery, Tech. rep., WSML deliverable, 2004.
[44] J. Peer, Towards automatic web service composition using ai planning techniques, http://www.mcm.unisg.ch/org/mcm/web.nsf/staff/jpeer, 2003.
[45] A.M. Zamreski, J.M. Wing, Specification matching of software components, in: Proc. of the 3rd ACM SIGSOFT Sympos. on the Foundations of Software Eng., 1995.
[46] B. Fischer, J. Schumann, NORA/HAMRR: Making deduction-based software component retrieval practical, in: Proc. of CADE-14 Workshop on Automated Theorem Proving in Software Engineering, 1997.
[47] J. Penix, P. Alexander, Efficient specification-based component retrieval, Automated Software Engineering.
[48] K. Sykara, S. Widoff, M. Klusch, J. Lu, LARKS: dynamic matchmaking among heterogeneous software agents in cyberspace, Autonomous Agents and Multi-Agent Systems.
[49] R.M.V. Haarslev, Description of the racer system and its applications, Stanford, CA, USA, 2001.
[50] D. MacDermott, AI planning systems competition, AI Magazine 21 (2) (2000) 35–55.
[51] J. Peer, M. Vokovic, A proposal for a semantic web service description format, in: Proc. of the European Conf. on Web Services (ECOWS'04), Springer-Verlag, 2004.
[52] V. Patti, Programming Rational Agents: a Modal Approach in a Logic Programming Setting, Ph.D. thesis, Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy, 2002. Available from: <http://www.di.unito.it/∼patti/>.