

Reasoning about interaction protocols for web service composition

Matteo Baldoni, Cristina Baroglio,
Alberto Martelli, and Viviana Patti^{1,2}

*Dipartimento di Informatica
Università degli Studi di Torino
C.so Svizzera, 185, I-10149 Torino (Italy)*

Abstract

In this work, we face the problem of web service composition, arguing the importance of the inclusion, in a web service description, of the high-level communication protocol used by a service to interact with a client. The work is set in the same multi-agent research area from which DAML-S is derived: reasoning about actions and about the change produced by actions on the world. In this perspective web services are viewed as actions, either simple or complex, characterized by preconditions and effects. In our proposal, interaction is interpreted as the effect of communicative action execution, so that it can be reasoned about.

Key words: Please list keywords from your paper here, separated by commas.

1 Introduction

Recent years witnessed a rapid evolution of the concept of world-wide web. In less than ten years we passed from the concept of web as a means for exchanging (HTML) documents to the concept of web as a means for accessing to (interactive) *web services* [14]. Research in this field is very much alive. As the huge amount of information on the web urged the development of standard languages for representing the semantics behind the HTML (e.g. RDF [21], OWL [19]), recently some attempt to standardize the description of web services has been carried on (DAML-S [8], WSDL [23]). The use of standard descriptions is aimed at allowing the automatic discovery of web services, their automatic execution and monitoring, and (the task we will focus

¹ Partially supported by MIUR Cofin 2003 “Logic-based development and verification of multi-agent systems” national project.

² Email: {baldoni,baroglio,mrt,patti}@di.unito.it

on in this paper) *automatic composition*. While the WSDL initiative is mainly carried on by the commercial world, with the aim of standardizing registration, look-up mechanisms and interoperability, DAML-S is more concerned with providing greater expressiveness to service description in a way that can be reasoned about [6]. In particular, a service description has three conceptual levels: the *profile*, used for advertising and discovery, the *process model*, that describes how a service works, and the *grounding*, that describes how an agent can access the service. In particular, the process model describes a service as atomic, simple or composite in a way inspired by the language Golog and its extensions [12,10,15]. In this perspective, a wide variety of agent technologies based upon the *action metaphor* can be used. In fact, we can view a service as an action (atomic or complex) with preconditions and effects, that modifies the state of the world and the state of agents that work in the world. The process model can, then, be viewed as the description of such an action; therefore, it is possible to design agents, which apply techniques for reasoning about actions and change to web service process models for producing new, composite, and customized services.

This work is set in a multi-agent framework in which the web service is an agent that communicates with other agents in a FIPA-like Action Communication Language (ACL); the web service behavior can be expressed as a *conversation protocol*, which describes the communications that can occur with other agents. Indeed, the web service must follow some possibly non-deterministic procedure aimed at getting/supplying all the necessary information. We already proved that by reasoning on the (explicitly given) conversation protocols followed by web services we achieve a better personalization of the service fruition [1]; in the current work, we show that this is true also in the case in which services are to be composed for solving the desired task. As an example, we will describe a rational agent, that is requested to organize a day out for a given user, making a reservation both to a restaurant and to a cinema, according to user given constraints.

We faced the problem of describing and reasoning about conversation protocols in an *agent logic programming* setting by using the modal action and belief framework of the language DyLOG, introduced in [4,3]. Integrated in the language, a communication kit [20,2] allows an agent to reason about the interactions that it is *going to* enact for proving if there is a possible execution of the protocol, after which a set of beliefs of interest (or goal) will be true in the agent mental state. Such a form of reasoning implies making assumptions about the mental state of *other* agents, the ones ours wishes to interact with. We consider a conversation protocol as a (non-deterministic) procedure that specifies the complex communicative behavior of a *single* agent, based upon simpler, FIPA-like communicative acts; in a communication protocol, an agent can either play the part of the *initiator* or of the *responder*.

2 Reasoning about conversations for web service composition

Let us consider a software agent, whose task is to search for web services, according to a user's requirements; we will refer to it as *pa* (personal assistant). As a novelty w.r.t. the work in [1], *pa* composes the found services with the aim of solving complex tasks. For example, let us suppose that the user wants to spend a day out by going to a restaurant and then to a cinema. He wants to make a reservation at both places and he is a little restrictive about the possible alternatives. He wants to see a specific movie (e.g. Nausicaa) and he wishes to benefit of some promotion on the cinema ticket but he is not eager to communicate his credit card number on the internet. If, on one hand, searching for a cinema or a restaurant reservation service is a task that can be accomplished on the basis of a set of characteristic keywords, stored in a registry system used for advertisement, the other kinds of condition (look for promotions, do not use credit card) can be verified only by reasoning about the way in which the web service operates and, in particular, about the interaction protocol that it follows.

To complete the example, suppose that two restaurants and two cinemas are available (see Fig. 1 for the AUML graphs representing their protocols), but only *restaurant1* takes part to a promotion campaign, by which it gives to each customer, who made a reservation by the internet service, a *free ticket* for a movie. On the side of cinemas, suppose that *cinema2* accepts reservations but no free ticket, whereas *cinema1* accepts promotional tickets, allowing a user to make reservations by sending the credit card number or (alternatively) by accepting to pay cash later. Fig. 1 (iii) is the part of protocol followed by *cinema1* in case of credit card payment, (iv) is the other case.

In a multi-agent framework, especially in open environments, when a communication is enacted, agents exchange their communication protocol [13]. The advantage of having a protocol exchange at the high-level of the interaction is that by doing so agents can reason about the change caused by a conversation to their own belief state. A rational model of communicative acts requires also the agent to make rational assumptions about the change caused to its interlocutor's beliefs [5,11]. So, for instance, an agent will communicate a piece of information only if it believes that the recipient ignores it; after sending the information it will make assumptions about the augmented recipient's set of beliefs and act consequently. In our example, *pa* should choose the first restaurant because in this way it can benefit of the promotion and obtain a free ticket for the cinema. It should also choose the first cinema but carry on a conversation in which the credit card is not requested. In order to perform this kind of reasoning, it is necessary to formalize (again: at a high-level) the communicative acts and the interaction protocols. We did it by using the DyLOG language, briefly introduced in the next section.

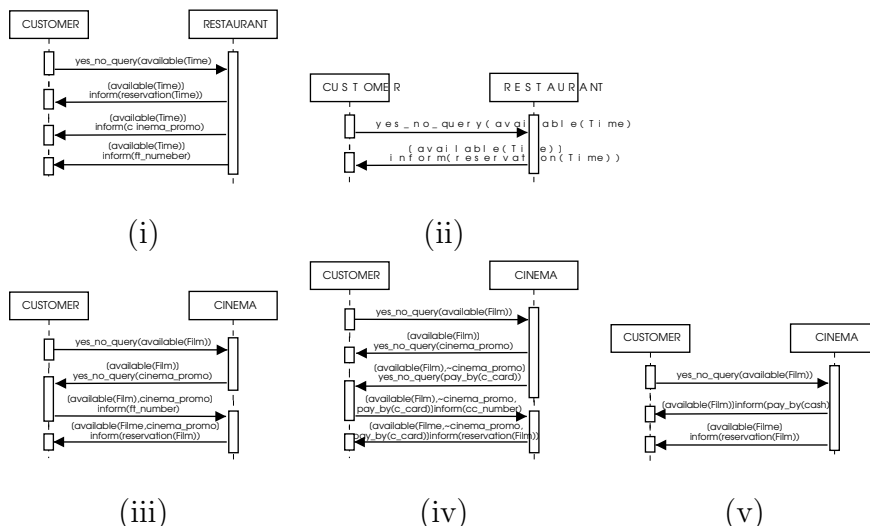


Fig. 1. The AUML graphs [16] represent the communicative interactions occurring between the customer (pa) and each of the web services; (i) is followed by *restaurant1*, (ii) by *restaurant2*, (iii) and (iv) are the two parts of the protocol followed by *cinema1*; (v) is followed by *cinema2*.

3 Introduction to the agent language

DyLOG is an agent language defined in a modal logic framework, that accounts both for atomic and complex actions, or procedures. In the following the fundamentals of the language are briefly introduced, for a more complete description see [2]. Atomic actions are either world actions, affecting the world, or mental actions, i.e. sensing or communicative actions which only affect the agent beliefs. For each world action and for each agent we define the modalities $[a^{ag_i}]\alpha$, meaning that the formula α holds after every execution of a by ag_i , and $\langle a^{ag_i} \rangle \alpha$, representing the possibility that α holds after the action execution. We also introduce a modality $Done(a^{ag_i})$ for expressing that a (a communicative act or a world action) has been executed. The modality \square 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, test and non-deterministic choice but, differently than [12], we refer to a *Prolog-like* paradigm and procedures are defined as (recursive) Prolog-like clauses. For each procedure p , the language contains also the universal and existential modalities $[p]$ and $\langle p \rangle$.

The mental state of an agent is described in terms of a consistent set of *belief formulas*. The modal operator \mathcal{B}^{ag_i} models the beliefs of ag_i while \mathcal{M}^{ag_i} , defined as the dual of \mathcal{B}^{ag_i} ($\mathcal{M}^{ag_i}\varphi \equiv \neg\mathcal{B}^{ag_i}\neg\varphi$), intuitively means that ag_i considers φ possible. It is possible to deal also with nested beliefs, to represent what other agents believe and reason on how they can be affected by communicative actions.

The properties of the modalities are described in [2].

3.1 The agent theory

The *behavior* of an agent is specified by a domain description that includes: (1) the agent *belief state*; (2) action and precondition laws that describe the *atomic world actions* effects on the executor’s mental state; (3) sensing axioms for describing *atomic sensing actions*; (4) procedure axioms for describing *complex behaviors*. In our framework agents are individuals, each with its *subjective* view of a dynamic domain. We do not model the real world but only the internal dynamics of each agent in relation to the changes caused by actions. The *belief state* of an agent intuitively contains what it (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 fluent* F is a belief formula $\mathcal{B}^{ag_i}L$ or its negation; L denotes a *belief argument*³). 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 $\neg\mathcal{B}^{ag_i}L$ and $\neg\mathcal{B}^{ag_i}\neg L$ hold); $\mathcal{U}^{ag_i}L$ 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. *Precondition laws*, instead, specify mental conditions that make a world action (or a communicative act) executable in a state. An agent can execute a when the precondition fluents of a are in its belief state.

Sensing Actions produce knowledge about fluents; they are defined as non-deterministic actions, with unpredictable outcome, formally modelled by a set of *sensing axioms*. If we associate to each sensing action s a set $dom(s)$ of literals (domain), when ag_i executes s , it will know which of such literals is true.

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

$$(1) \quad \langle p_0 \rangle \varphi \subset \langle p_1; p_2; \dots; p_m \rangle \varphi$$

p_0 is a procedure name and the p_i ’s ($i = 1, \dots, m$) are either procedure names, atomic actions, or test actions; the operator “;” is the sequencing operator of dynamic logic. 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 each agent domain description.

³ i.e. 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 ($\mathcal{B}l$ or $\neg\mathcal{B}l$). We use l for denoting attitude-free fluents: a fluent literal or a done fluent.

Speech acts are atomic actions of form $\text{speech_act}(sender, receiver, l)$: $sender$ and $receiver$ are agents and l is either a fluent literal or a done fluent. They can be seen as special mental actions, affecting both the sender's and the receiver's mental state. In our model we focus on the *internal representation*, that agents have of speech acts: ag_i 's belief changes depend on the role of the agent. Speech act specification is, then, twofold: one definition holds when the agent is the sender, the other when it is the receiver. They 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 on ag_i mental state. This representation allows agents to *reason about* conversation effects.

Hereafter is an example of specification of the semantics of the *inform* FIPA-ACL primitive speech act, as defined in our framework:

$\text{inform}(Self, Other, l)$

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

(a) says that an inform act can be executed if the sender believes l and believes that the receiver does not know l . If $Self$ is the sender it thinks possible, but it cannot be sure -autonomy assumption (b)-, that $Other$ will adopt its belief. If it believes that $Other$ considers it a trusted *authority* about l , it is confident that $Other$ will adopt its belief (c). When $Self$ is the receiver, it believes that l is believed by the sender $Other$ (e) but it adopts l as an own belief only if it thinks $Other$ is a trusted authority (f)-(g).

Get-message actions are used for *receiving* messages from other agents. We model them as a special kind of sensing actions, because from the agent perspective they correspond to queries for an external input, whose outcome 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, we use get_message actions defined by an axiom schema of the form:

$$(2) \quad [\text{get_message}(ag_i, ag_j, l)]\varphi \equiv [\bigcup_{\text{speech_act} \in \mathcal{C}_{\text{get_message}}} \text{speech_act}(ag_j, ag_i, l)]\varphi$$

Intuitively, $\mathcal{C}_{\text{get_message}}$ is a finite set of speech acts, which are all the possible communications that ag_i expects from ag_j in the context of a given conversation. We do not associate to a get_message action a domain of mental fluents, but we calculate the information obtained by looking at the effects of such speech acts on ag_i 's mental state.

We assume individual speech acts to take place in the context of predefined conversation protocols [13] that specify communication patterns. Since agents

have a subjective perception of the communication, each protocol has as many procedural representations as the possible roles in the conversation.

We can define the *communication kit* of an agent ag_i , CKit^{ag_i} , as the triple $(\Pi_C, \Pi_{CP}, \Pi_{S_{get}})$, where Π_C is the set of simple action laws defining ag_i 's primitive speech acts, $\Pi_{S_{get}}$ is a set of axioms for ag_i 's `get_message` actions and Π_{CP} is the set of procedure axioms specifying the ag_i 's conversation protocols. In this extension of the DyLOG language, we define as *Domain Description* for agent ag_i , a triple $(\Pi, \text{CKit}^{ag_i}, S_0)$, where CKit^{ag_i} is ag_i communication kit, S_0 is the initial set of ag_i 's belief fluents, and Π is a tuple (Π_A, Π_S, Π_P) , where Π_A is the set of ag_i 's world action and precondition laws, Π_S is a set of axioms for ag_i 's sensing actions, Π_P a set of axioms that define complex actions.

3.3 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:

$$(3) \quad \langle p_1 \rangle \langle p_2 \rangle \dots \langle p_m \rangle Fs$$

Each p_k , $k = 1, \dots, m$ in (3) may either 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 (3) succeeds corresponds to answering to the question “is there an execution trace of p_1, \dots, p_m leading to a state where the conjunction of belief fluents Fs holds for ag_i ?”. Such an execution trace is a *plan* to bring about Fs . The procedure definition constrains the search space.

In presence of communication, the problem of reasoning about *conversation protocols* is faced (a conversation is a sequence of speech acts) because in the case in which p_1, \dots, p_m are conversation protocols, by answering to the query (3) we find a conversation after which some desired condition Fs holds.

In the current application we aim at finding a conversation, that allows to achieve a desired goal, which is an instance of the composition of many protocols. Conversations, however, usually contain *get-message* actions for receiving (unpredictable) messages from other agents. Since their outcome is unknown at planning time because agents cannot know in advance the answers that they will receive, we treat them as sensing actions. Nevertheless, the existence of protocols allows us to make assumptions on such messages and find out if there is a possible conversation that leads to the goal. Of course it is not possible to guarantee that at *execution time* the services will attain to the planned conversation because part of it may depend on information that is owned by the service providers and that will be known during the actual interaction. For instance, let us consider again the example of Section 2. 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 is that the information contained in the protocol is sufficient to exclude a number of compositions that will never satisfy the goal (e.g. *restaurant1* plus *cinema2* does not permit to exploit a promotion independently from the availability of seats).

The proof procedure is a natural evolution of [4,3] and is described in [20,2]; 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 [9]. The extracted plans always lead to a state in which the goal condition Fs holds.

4 A rational personal assistant

In this section we describe the personal assistant *pa*. This agent composes a set of web services to satisfy the user's requests. We imagine the search and composition process as divided in two steps. The first step (not described in this work) is *keyword-based* and is aimed at restricting the attention to a little set of services, possibly equivalent, extracted from a registry system. The second step is a further selection based on *reasoning*. During this step the agent personalizes the interaction with the services according to the requests of the user and dismisses services that do not fit. To this aim, the agent reasons about the procedure *comp_services* (a possible implementation of it is reported below), that sketches the general composition-by-sequencing of a set of services, feeding in the interaction protocols (*service*(*TypeService*, *Name*, *Protocol*)), that we suppose explicitly contained in the service descriptions identified by the first step.

$$\begin{aligned} &\langle \text{comp_services}([\])\rangle\varphi \supset \text{true} \\ &\langle \text{comp_services}([\text{TypeService}, \text{Name}, \text{Data}]\text{Services})\rangle\varphi \supset \\ &\quad \langle \mathcal{B}^{pa} \text{service}(\text{TypeService}, \text{Name}, \text{Protocol}) ; \text{Protocol}(pa, \text{Name}, \text{Data}) ; \\ &\quad \text{comp_services}(\text{Services})\rangle\varphi \end{aligned}$$

Intuitively, *comp_services* builds the sequence of protocols to apply for interacting with a set of services, so that it will be possible to reason about the whole. The presented implementation is quite simple but is sufficient as an example and generally it could be any prolog-like procedure. Before explaining the kind of reasoning that can be applied, let us describe the *protocols*, that are followed by the web services of our example. Such protocols allow the interaction of two agents, so each of them has two complementary views: the view of the web service and the view of the customer, i.e. *pa*. In the following we will report (written in DyLOG) the view that *pa* has of the protocols.

$$(a) \langle \text{reserv_rest_1}_C(\text{Self}, \text{WebS}, \text{Time})\rangle\varphi \subset$$

$$\langle \text{yes_no_query}_Q(\text{Self}, \text{WebS}, \text{available}(\text{Time})) ; \mathcal{B}^{\text{Self}} \text{available}(\text{Time})? ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{reservation}(\text{Time})) ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{cinema_promo}) ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{ft_number}) \rangle \varphi$$

(b) $\langle \text{reserv_rest_2}_C(\text{Self}, \text{WebS}, \text{Time}) \rangle \varphi \subset$
 $\langle \text{yes_no_query}_Q(\text{Self}, \text{WebS}, \text{available}(\text{Time})) ; \mathcal{B}^{\text{Self}} \text{available}(\text{Time})? ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{reservation}(\text{Time})) \rangle \varphi$

(c) $[\text{get_info}(\text{Self}, \text{WebS}, \text{Fluent})] \varphi \subset [\text{inform}(\text{WebS}, \text{Self}, \text{Fluent})] \varphi$

Procedures (a) and (b) respectively describe the customer-view of the protocols followed by the two considered restaurants. The customer asks if a table is available at a certain time, if so, the service informs the customer that a reservation has been taken. The first restaurant also informs the customer that it gained a promotional free ticket for a cinema (*cinema_promo*) and it returns a code number (*ft_number*).

(d) $\langle \text{reserv_cinema_1}_C(\text{Self}, \text{WebS}, \text{Film}) \rangle \varphi \subset$
 $\langle \text{yes_no_query}_Q(\text{Self}, \text{WebS}, \text{available}(\text{Film})) ; \mathcal{B}^{\text{Self}} \text{available}(\text{Film})? ; \\ \text{yes_no_query}_I(\text{Self}, \text{WebS}, \text{cinema_promo}) ; \neg \mathcal{B}^{\text{Self}} \text{cinema_promo}? ; \\ \text{yes_no_query}_I(\text{Self}, \text{WebS}, \text{pay_by}(\text{c_card})) ; \mathcal{B}^{\text{Self}} \text{pay_by}(\text{c_card})? ; \\ \text{inform}(\text{Self}, \text{WebS}, \text{cc_number}) ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{reservation}(\text{Film})) \rangle \varphi$

(e) $\langle \text{reserv_cinema_1}_C(\text{Self}, \text{WebS}, \text{Film}) \rangle \varphi \subset$
 $\langle \text{yes_no_query}_Q(\text{Self}, \text{WebS}, \text{available}(\text{Film})) ; \mathcal{B}^{\text{Self}} \text{available}(\text{Film})? ; \\ \text{yes_no_query}_I(\text{Self}, \text{WebS}, \text{cinema_promo}) ; \mathcal{B}^{\text{Self}} \text{cinema_promo} ; \\ \text{inform}(\text{Self}, \text{WebS}, \text{ft_number}) ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{reservation}(\text{Film})) \rangle \varphi$

(f) $\langle \text{reserv_cinema_2}_C(\text{Self}, \text{WebS}, \text{Film}) \rangle \varphi \subset$
 $\langle \text{yes_no_query}_Q(\text{Self}, \text{WebS}, \text{available}(\text{Film})) ; \mathcal{B}^{\text{Self}} \text{available}(\text{Film})? ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{pay_by}(\text{cash})) ; \\ \text{get_info}(\text{Self}, \text{WebS}, \text{reservation}(\text{Film})) \rangle \varphi$

Clauses (d)–(f) encode two protocols: (d) and (e) are the protocol followed by *cinema1*, (f) is the protocol followed by *cinema2*. Supposing that the desired movie is available, *cinema1* alternatively accepts credit card payments (d) or promotional free tickets (e). *Cinema2*, instead, accepts cash payments but it does not take part to the promotion campaign.

Let us now consider the query:

$$\langle \text{comp_services}([\text{restaurant}, R, \text{dinner}], [\text{cinema}, C, \text{nausicaa}]) \rangle \\ (\mathcal{B}^{\text{pa}} \text{cinema_promo} \wedge \mathcal{B}^{\text{pa}} \text{reservation}(\text{dinner}) \wedge \mathcal{B}^{\text{pa}} \text{reservation}(\text{nausicaa}) \\ \wedge \mathcal{B}^{\text{pa}} \neg \mathcal{B}^C \text{cc_number} \wedge \mathcal{B}^{\text{pa}} \mathcal{B}^C \text{ft_number})$$

that amounts to determine if it is possible to compose the interaction with a restaurant web service and a cinema web service, so that it is possible to reserve a table for dinner ($\mathcal{B}^{pa}reservation(dinner)$) and to book a cinema ticket for the movie Nausicaa ($\mathcal{B}^{pa}reservation(nausicaa)$), exploiting a promotional ticket ($\mathcal{B}^{pa}cinema_promo$). The user also specifies that no credit card is to be used ($\mathcal{B}^{pa}\neg\mathcal{B}^Ccc_number$), instead the obtained free ticket is to be spent ($\mathcal{B}^{pa}\mathcal{B}^Cft_number$), i.e. pa believes that after the whole conversation the chosen cinema service will know the number of the ticket given by the selected restaurant but it will not know the user's credit card number.

Let us suppose that pa has the following list of available services:

$\mathcal{B}^{pa}service(restaurant, restaurant1, reserv_rest_1C)$
 $\mathcal{B}^{pa}service(restaurant, restaurant2, reserv_rest_2C)$
 $\mathcal{B}^{pa}service(cinema, cinema1, reserv_cinema_1C)$
 $\mathcal{B}^{pa}service(cinema, cinema2, reserv_cinema_2C)$

then the query succeeds with answer R equal to $restaurant1$ and C equal to $cinema1$. This means that there is first a conversation between pa and $restaurant1$ and, then, a conversation between pa and $cinema1$, that are instances of the respective conversation protocols, after which the desired condition holds. Agent pa works on behalf of a user, thus it knows the user's credit card number ($\mathcal{B}^{pa}cc_number$) and his desire to avoid using it in the current transaction ($\neg\mathcal{B}^{pa}pay_by(credit_card)$). It also believes to be an authority about the form of payment and about the user's credit card number and it believes that the other agents are authorities about what they communicate by inform acts. The initial mental state will also contain the fact that pa believes that no reservation for dinner nor for *nausicaa* has been booked yet, $\mathcal{B}^{pa}\neg reservation(dinner)$ e $\mathcal{B}^{pa}\neg reservation(nausicaa)$, the fact that pa does not have a free ticket for the cinema yet, $\neg\mathcal{B}^{pa}cinema_promo$, and some hypothesis on the interlocutor's mental state, e.g. the belief fluent $\mathcal{B}^{pa}\neg\mathcal{B}^{cinema1}cc_number$, meaning that the web service does not already know the credit card number.

In this context, the agent builds the following execution trace of *comp_services* ($[[restaurant, R, dinner], [cinema, C, nausicaa]]$):

queryIf($pa, restaurant1, available(dinner)$) ;
inform($restaurant1, pa, available(dinner)$) ;
inform($restaurant1, pa, reservation(dinner)$) ;
inform($restaurant1, pa, cinema_promo$) ;
inform($restaurant1, pa, ft_number$) ;
queryIf($pa, cinema1, available(nausicaa)$) ;
inform($cinema1, pa, available(nausicaa)$) ;
inform($cinema1, pa, cinema(C)$) ;
queryIf($cinema1, pa, cinema_promo$) ;
inform($pa, cinema1, cinema_promo$) ;
inform($cinema1, pa, pay_by(free_ticket)$) ;
inform($pa, cinema1, ft_ticket$) ;

`inform(cinema1, pa, reservation(nausicaa))`

We can easily see that there is no other execution trace of *comp_services* that satisfies the goals. This means that, in the search space defined by this procedure, it is not possible to use with success any other composition of the services (e.g. *restaurant1* with *cinema2*) and this does not depend on the possible outcomes of conversations. Observe that arbitrary compositions of the services, i.e. compositions that cannot be found as executions of *comp_services*, may satisfy the user’s goal; in order to find them one could use a *general-purpose planner*. However, there are situations in which one has a *general schema* for the desired solution, which is helpful for reducing the search time.

5 Related Work and Conclusions

In this work we have shown some possible benefits of reasoning about communication protocols in the task of web service composition. In particular, we have focussed on services with a representation a la DAML-S that, by making explicit the preconditions and the effects of a service, practically assimilates it to an action. In this perspective an approach that exploits mechanisms for reasoning about actions and change, like ours, seems particularly suitable.

Web service composition in the DAML-S context has been faced also by other researchers. The most relevant work in this line has been carried on by the équipe of Sheila McIlraith at Stanford [6]. This work differs from ours in the fact that, while we compose web services by applying procedural planning to possibly complex actions (i.e. procedures that encode conversation protocols), in the work by McIlraith et al. the composed services correspond to atomic actions only and composition is based on their preconditions and effects. The advantage of working at the protocols level is that by reasoning about them agents can personalize the interaction by selecting a course that satisfies user-given (or service given) requirements. Otherwise, since the IOPE lists (inputs, outputs, preconditions and effects) are flat and do not allow the representation of relations among them, it would be unfeasible to select a service that allows different actual interactions. This process can be started before the actual interaction takes place and can be exploited also for web service search.

The idea of working on conversation protocols derives from the experience of the authors in the multi-agent research area, where agents commonly exchange communication protocols before interacting. We based our work in a modal action logic framework and used the agent logic programming language DyLOG. By exploiting nested beliefs we took a subjective representation of conversation protocols, in which an agent makes rational assumptions on its interlocutor’s state of mind. Notice that, since we are only interested in reasoning about the local mental state’s dynamics, our approach differs from other logic-based approaches to communication in multi-agent systems, as the one taken in Congolog [22], where communicative actions affect the global

state of a multi-agent system.

From a different perspective, if we interpret web services as agents, we can find a wide literature about coordination models and languages, e.g. [7], that supply the abstractions necessary for ruling the interaction of agents with a common goal. A well-known coordination model is TuCSoN [17], which exploits tuple centers and the logic-based language ReSpecT [18] for agent coordination. Tuple centers are characterized by a reactive behavior, that specifies how the center responds to a communication event. However, the chain reaction is *transparent* to the communicating agents, which perceive responses as single-step state transitions of the tuple center. Also in this context it would be interesting to have a mechanism for reasoning about the behavior of the tuple center.

*****FUTURE WORK: pianificazione / esecuzione -; fallimento / replanning*****

References

- [1] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction for personalizing web service fruition. In G. Armano, F. De Paoli, A. Omicini, and E. Vargiu, editors, *Proc. of WOA 2003: Dagli oggetti agli agenti, sistemi intelligenti e computazione pervasiva*, Villasimius (CA), Italy, September 2003. Pitagora Editrice Bologna.
- [2] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about self and others: communicating agents in a modal action logic. In *Proc. of ICTCS'2003*, volume 2841 of *LNCS*, pages 228–241. Springer, 2003.
- [3] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*. To appear.
- [4] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Reasoning about Complex Actions with Incomplete Knowledge: A Modal Approach. In *Proc. of ICTCS'2001*, volume 2202 of *LNCS*, pages 405–425. Springer, 2001.
- [5] P. Bretier and D. Sadek. A rational agent as the kernel of a cooperative spoken dialogue system: implementing a logical theory of interaction. In J.P. Müller, M. Wooldridge, and N.R. Jennings, editors, *Intelligent Agents III, proc. of ECAI-96 Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, volume 1193 of *LNAI*. Springer-Verlag, 1997.
- [6] J. Bryson, D. Martin, S. McIlraith, and L. A. Stein. Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web, 2002.
- [7] P. Ciancarini and J. Wooldridge, editors. *Agent Oriented Software Engineering (AOSE 2000)*, volume 1957 of *LNCS*. Springer-Verlag, 2000.

- [8] DAML-S. <http://www.daml.org/services/daml-s/0.9/>. version 0.9, 2003.
- [9] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abduction Logic Programming. In *Proc. of ILPS '93*, Vancouver, 1993. The MIT Press.
- [10] G. De Giacomo, Y. Lesperance, and H.J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.
- [11] A. Herzig and D. Longin. Beliefs dynamics in cooperative dialogues. In *Proc. of AMSTEOLOGUE 99*, 1999.
- [12] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Programming*, 31:59–83, 1997.
- [13] A. Mamdani and J. Pitt. Communication protocols in multi-agent systems: A development method and reference architecture. In *Issues in Agent Communication*, volume 1916 of *LNCIS*, pages 160–177. Springer, 2000.
- [14] J. Maurer, editor. *ACM Queue*. ACM, march 2003.
- [15] S. McIlraith and T. Son. Adapting Golog for Programmin the Semantic Web. In *5th Int. Symp. on Logical Formalization of Commonsense Reasoning*, pages 195–202, 2001.
- [16] James H. Odell, H. Van Dyke Parunak, and Bernhard Bauer. Representing agent interaction protocols in UML. In *Agent-Oriented Software Engineering*, pages 121–140. Springer, 2001. <http://www.fipa.org/docs/input/f-in-00077/>.
- [17] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3), 1999. Special Issue on Coordination Mechanisms and Patterns for Web Agents.
- [18] Andrea Omicini and Enrico Denti. Formal ReSpecT. In Maria Chiara Meo, editor, *2000 Joint Conf. on Declarative Programming (AGP'00)*, La Habana (Cuba), 4–7 December 2000.
- [19] OWL. <http://www.w3c.org/tr/owl-guide/>. 2003.
- [20] V. Patti. *Programming Rational Agents: a Modal Approach in a Logic Programming Setting*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 2002. Available at <http://www.di.unito.it/~patti/>.
- [21] RDF. <http://www.w3c.org/tr/1999/rec-rdf-syntax-19990222/>. 1999.
- [22] S. Shapiro, Y. Lespance, and H. J. Levesque. Specifying communicative multi-agent systems. In *Agents and Multi-Agent Systems - Formalisms, Methodologies, and Applications*, volume 1441 of *LNAI*, pages 1–14. Springer-Verlag, 1998.
- [23] WSDL. <http://www.w3c.org/tr/2003/wd-wsdl12-20030303/>. version 1.2, 2003.