Reasoning about interaction for personalizing web service fruition

Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
Tel. +39 011 6706711 — Fax. +39 011 751603
E-mail: {baldoni,baroglio,mrt,patti}@di.unito.it

***Abstract*—In this work, we argue the importance of including in web service descriptions also the high-level communication protocol, used by a service to interact with a client. We will motivate this claim by setting web services in a multi-agent framework, in the same line of that research area from which the DAML-S language derived. In our proposal, interaction is interpreted as the effect of action execution; we will show a typical situation in which the ability of reasoning about interactions and conversations is fundamental for choosing a web service or for personalizing the service fruition.**

I. Introduction

Recent years witnessed a rapid evolution of the way in which information is made accessible to users via the world-wide web. In less than ten years we passed from static HTML pages to interactive pages (developed in languages like DHTML) to dynamically built pages containing information extracted from data bases, and to (interactive) *web services* [18]. Research applied to the web world is very much alive. The huge amount of available information, for instance, urged the development of standard languages for representing semantic knowledge, in the beginning in the form of flat meta-data (e.g. RDF [25]) and then in more structured ways by allowing the definition of ontologies (DAML+OIL [10], OWL [23]). In the very same way in which these languages allow one to capture the semantics behind the contents of a web page, very recently some attempt to standardize the descriptions of web services, which we can consider as dynamic entities with a behavior, has been carried on (DAML-S [9], WSDL [27]).

While WSDL is an initiative mainly carried on by the commercial world, aimed at 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 [7]. In particular, it structures a service description in three conceptual areas: the *profile*, that characterizes the service to the purpose of advertising and discovery, the *process model*, that describes how it 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 [16], [14], [19]; the purpose is to allow automatic discovery of web services, automatic execution and monitoring, and automatic composition, by exploiting a wide variety of agent technologies based upon the *action metaphor*. We can, in fact, view a service as an action (atomic or complex) with preconditions and effects, that can modify the state of the world and the state of agents that work in the world. The process model can be viewed as a description of such an action; we can, then, think to design agents, which apply techniques for reasoning about actions and change to web service process models, for producing new, composite, and customized services.

We set our work 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); in this context, the web service behavior can be expressed as a *conversation protocol*, which describes the communications that can occur between the service and its client. Indeed, the web service must follow some possibly non-deterministic procedure, aimed at getting/supplying all the necessary information. This is true also in the case in which the service has a reactive behavior. We claim that a better personalization can be achieved by allowing agents to reason also about the conversation protocols followed by web services; to this aim, the protocol must be part of the web service description. As an example, consider a *user personal assistant*, a rational agent, that must book a ticket at a cinema where they show Akira but is requested by the user not to communicate his credit card number. If we take a closer look to the two requests, we can notice that while the first identifies the main *goal* to achieve, the second constrains the way in which the *interaction* between the web service and the agent should be carried on. If the agent had a description of the conversation behavior of the service, it could *reason about* the possible outcoming interactions *before* they actually take place, either verifying if the interaction may be *personalized* so to satisfy all the user's requirements, or, when this is not possible, the agent could decide to search for another service provider. A specific interaction occurring between a provider and a client can, then, be considered as a *conversation*, determined by the communication protocol.

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 [5]. Integrated in the language, a communication kit [24] allows an agent to reason also about the interactions that it is going to enact for proving if there is a possible execution of the *communication 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 the *other* agent, the one ours wishes to interact with. We consider a conversation protocol as a (possibly 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 sender or of the receiver of a message.

DyLOG has already been used with success in web applications for implementing a virtual tutor that helps students to build personalized study curricula [1], [3], [4], based on the description of courses viewed as actions (an application that bears many analogies with web service process model description and usage).

II. Reasoning about conversations for web service personalization

Let us consider a software agent that is the user personal assistant described in the introduction; in the following we will refer to it as pa. The task of the agent is to invoke a web service that must satisfy the user's requirements, not only in the kind of service it provides but also in the kind of interaction taken. Suppose, in particular, that pa is asked to book a ticket at a cinema where they show the movie Akira, avoiding to communicate to the web service the user's credit card number. Suppose also that

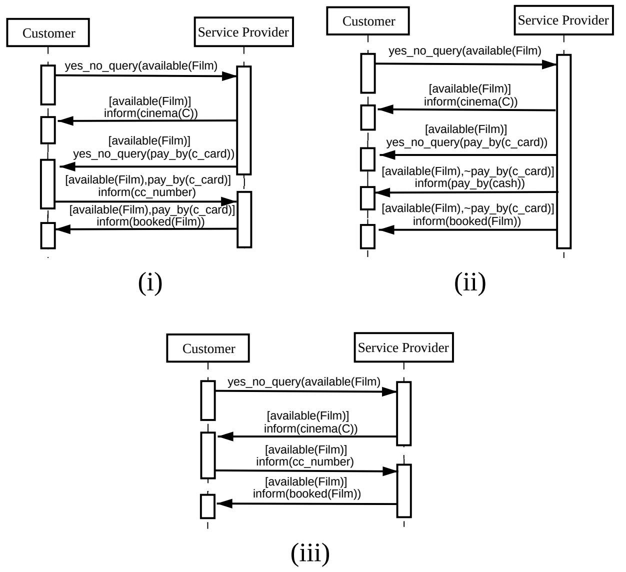

Fig. 1. The three AUML graphs [20] represent the communicative interactions occurring between the customer (pa) and the provider; (i) and (ii) are followed by $click\_ticket$, (iii) is followed by all_cinema. Formulas among square brackets represent conditions on the execution of the speech act.

two cinema booking services are available, called $click\_ticket$ and all_cinema respectively, that apply two different interaction protocols, one permitting both to book a ticket to be paid later by cash (Fig. 1 (i)) and to buy it by credit card (Fig. 1 (ii)), the other allowing only ticket purchase by credit card (Fig. 1 (iii)). In a multi-agent framework, when a communication is enacted, agents exchange their communication protocol [17]. The advantage of having a protocol exchange at the high-level of interaction is that by doing so agents can reason about the change caused by a conversation to their own belief state and make rational assumptions about the change caused to the other agent beliefs. Thus agents can decide, before starting the actual interaction, if that interaction leads to fulfill their goals. In our example, pa could for instance choose between the two cinema booking providers, selecting $click\_ticket$ because its interaction protocol allows conversations in which the user's credit card number is not requested.

In order to perform this kind of reasoning, it is necessary to formalize (at high-level) the communicative acts and the interaction protocols. Moreover, a rational model of communicative acts requires the agent to make assumptions about its interlocutor's beliefs [6], [15]; for instance, the agent communicates a piece of information if it believes the other agent to be ignorant about it. In the following sections we will briefy describe the DyLOG language and, afterwards, we will use it to develop agent pa.

III. The agent language: an introduction

The agent language accounts both for atomic and complex actions, or procedures. Atomic actions are either world actions, affecting the world, or mental actions, i.e. sensing or communicative actions which only affect the agent beliefs. The set of atomic actions consists of the set $\mathcal{A}$ of the world actions, the set $\mathcal{C}$ of communicative acts, and the set $\mathcal{S}$ of sensing actions. For each atomic action a and agent ag_i we introduce the modalities $[a^{ag_i}]$ and $\langle a^{ag_i}\rangle$. $[a^{ag_i}]\alpha$ means that α holds after every execution of action a by agent ag_i; $\langle a^{ag_i}\rangle\alpha$ means that there is a possible execution of a (by ag_i) after which α holds. For each atomic action a in $\mathcal{A} \cup \mathcal{C}$ we also introduce a modality $Done(a^{ag_i})$ for expressing that a has been executed. $Done(a^{ag_i})\alpha$ is read "a has been executed by ag_i; before its execution, α was true"[1]. The modality $\Box$ denotes formulas that hold in all the possible agent mental states. Our formalization of complex actions draws considerably from dynamic logic for the definition of action operators like sequence, test and non-deterministic choice. However, differently than [16], we refer to a *Prolog-like* paradigm: procedures are defined by means of (possibly 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*. We enriched the belief state of a DyLOG agent by allowing also nested beliefs, for representing what other agents believe and reasoning on how they can be affected by communicative actions. We use the modal operator $\mathcal{B}^{ag_i}$ to model the beliefs of agent ag_i. The modality $\mathcal{M}^{ag_i}$ is defined as the dual of $\mathcal{B}^{ag_i}$ ($\mathcal{M}^{ag_i}\varphi \equiv \neg\mathcal{B}^{ag_i}\neg\varphi$). Intuitively $\mathcal{M}^{ag_i}\varphi$ means that ag_i consider φ possible.

All the modalities of the language are normal; $\Box$ is reflexive and transitive, its interaction with action modalities is ruled by $\Box\varphi \supset [a^{ag_i}]\varphi$. The epistemic modality $\mathcal{B}^{ag_i}$ is serial, transitive and euclidean. The interaction of the $Done(a^{ag_i})$ modality

[1] $Done(a^{ag_i})\top$ is read "the action a has been executed by agent ag_i".

with other modalities is ruled by: $\varphi \supset [a^{ag_i}]Done(a^{ag_i})\varphi$ and $Done(a^{ag_j})\varphi \supset \mathcal{B}^{ag_i}Done(a^{ag_j})\varphi$ (awareness), with $ag_i = ag_j$ when $a^{ag_i} \notin \mathcal{C}$.

A. The agent theory

In the line of [5] the *behavior* of an agent ag_i can be specified by a domain description, which includes, besides a specification of the agent *belief state*: (1) action and precondition laws for describing the *atomic world actions* in terms of their preconditions and effects on the executor's mental state; (2) sensing axioms for describing *atomic sensing actions*; (3) procedure axioms for describing *complex behaviors*.

Belief state Agents are individuals, each having a mental state: its *subjective* point of view on a dynamic domain. Then, we do not model the real world but only the internal dynamics of each agent in relation to the changes caused by actions. A mental state is a set of belief formulas (*belief state*), intuitively it contains what ag_i (dis)believes about the world and about the other agents. A belief state is a complete and consistent set of rank 1 and 2 belief fluents, where a *belief fluent* F is a belief formula $\mathcal{B}^{ag_i}L$ or its negation. L denotes a *belief argument*, 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. Consistency is guaranteed by the seriality of the $\mathcal{B}^{ag_i}$ modalities[2]. In essence a belief state provides, for each agent ag_i, a three-valued interpretation of all the possible belief arguments L: each L is either *true*, *false*, or *undefined* when both $\neg\mathcal{B}^{ag_i}L$ and $\neg\mathcal{B}^{ag_i}\neg L$ hold. In the following we use $\mathcal{U}^{ag_i}L$ for expressing 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 $a \in \mathcal{A}$, executed by ag_i itself. They have the form:

$$\Box(\mathcal{B}^{ag_i}L_1 \wedge \ldots \wedge \mathcal{B}^{ag_i}L_n \supset [a^{ag_i}]\mathcal{B}^{ag_i}L_0) \qquad (1)$$

$$\Box(\mathcal{M}^{ag_i}L_1 \wedge \ldots \wedge \mathcal{M}^{ag_i}L_n \supset [a^{ag_i}]\mathcal{M}^{ag_i}L_0) \qquad (2)$$

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[3]. *Precondition laws* specify mental conditions that make an action in $\mathcal{A} \cup \mathcal{C}$ executable in a state. They have form:

$$\Box(\mathcal{B}^{ag_i}L_1 \wedge \ldots \wedge \mathcal{B}^{ag_i}L_n \supset \langle a^{ag_i}\rangle\top) \qquad (3)$$

ag_i can execute a when the precondition fluents of a are in ag_i's 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:

$$[s]\varphi \equiv [\bigcup_{l \in dom(s)} s^{\mathcal{B}^{ag_i}l}]\varphi \qquad (4)$$

$\cup$ is the choice operator of dynamic logic and $s^{\mathcal{B}^{ag_i}l}$, for each $l \in dom(s)$, is an *ad hoc* primitive action, that probes one of the possible outcomes of the sensing.

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

$$\langle p_0\rangle\varphi \subset \langle p_1; p_2; \ldots; p_m\rangle\varphi \qquad (5)$$

p_0 is a procedure name and the p_i's ($i = 1, \ldots, m$) 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.

B. Communication

The integration of a *communication theory* in the general agent theory is obtained by adding further axioms and laws to ag_i's domain description. In this section we will introduce a communication kit that allows the specification of communicative behaviors.

Speech Acts Communication primitives are atomic actions, described in terms of preconditions and effects on the agent mental state. They have 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. Such actions can be seen as special mental actions, affecting both the sender's and the receiver's mental state. In our model we focused on the *internal representation*, that agents have of each speech act, by specifying ag_i's belief changes both when it is the sender and when it is the receiver. They are modelled by generalizing the action and precondition laws of form (1), (2), and (3), so to allow the representation of the effects of communications performed by other agents on ag_i mental state. Such a representation provides the capability of *reasoning about* conversation effects.

Speech act specification is, then, twofold: one definition holds when the agent is the sender, the other when it is the receiver. In the first case, the precondition laws contain some *sincerity condition* that must hold in the agent mental state. When ag_i is the receiver, the action is *always* executable. Let us consider some primitive speech acts from the standard agent communication language FIPA-ACL, and let us define them and their semantics within our framework:

$\mathsf{inform}(sender, receiver, l)$

a) $\Box(\mathcal{B}^{ag_i}l \wedge \mathcal{B}^{ag_i}\mathcal{U}^{ag_j}l \supset \langle\mathsf{inform}(ag_i, ag_j, l)\rangle\top)$

b) $\Box([\mathsf{inform}(ag_i, ag_j, l)]\mathcal{M}^{ag_i}\mathcal{B}^{ag_j}l)$

c) $\Box(\mathcal{B}^{ag_i}\mathcal{B}^{ag_j}authority(ag_i, l) \supset$
$[\mathsf{inform}(ag_i, ag_j, l)]\mathcal{B}^{ag_i}\mathcal{B}^{ag_j}l)$

d) $\Box(\top \supset \langle\mathsf{inform}(ag_j, ag_i, l)\rangle\top)$

e) $\Box([\mathsf{inform}(ag_j, ag_i, l)]\mathcal{B}^{ag_i}\mathcal{B}^{ag_j}l)$

f) $\Box(\mathcal{B}^{ag_i}authority(ag_j, l) \supset [\mathsf{inform}(ag_j, ag_i, l)]\mathcal{B}^{ag_i}l)$

[2]A belief state is *not consistent* when it contains: a belief $\mathcal{B}^{ag_i}l$ and its negation, or the belief formulas $\mathcal{B}^{ag_j}\mathcal{B}^{ag_i}l$ and $\mathcal{B}^{ag_j}\mathcal{B}^{ag_i}\neg l$, or the belief formulas $\mathcal{B}^{ag_j}\mathcal{B}^{ag_i}l$ and $\mathcal{B}^{ag_j}\neg\mathcal{B}^{ag_i}l$.

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

g) $\Box(\mathcal{M}^{ag_i} authority(ag_j, l) \supset [\mathsf{inform}(ag_j, ag_i, l)]\mathcal{M}^{ag_i} l)$

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

$\mathsf{queryIf}(sender, receiver, l)$
a) $\Box(\mathcal{U}^{ag_i} l \wedge \neg\mathcal{B}^{ag_i}\mathcal{U}^{ag_j} l \supset \langle \mathsf{queryIf}(ag_i, ag_j, l)\rangle\top)$
b) $\Box(\top \supset \langle \mathsf{queryIf}(ag_j, ag_i, l)\rangle\top)$
c) $\Box([\mathsf{queryIf}(ag_j, ag_i, l)]\mathcal{B}^{ag_i}\mathcal{U}^{ag_j} l)$

By queryIf ag_i asks ag_j if it believes that l is true. To perform a queryIf act, ag_i must ignore l and it must believe that the receiver does not ignore l (a). After a queryIf act, the receiver will believe that the sender ignores l.

$\mathsf{refuseInform}(sender, receiver, l)$
a) $\Box(\mathcal{U}^{ag_i} l \wedge \mathcal{B}^{ag_i} Done(\mathsf{queryIf}(ag_j, ag_i, l))\top \supset$
$\langle \mathsf{refuseInform}(ag_i, ag_j, l)\rangle\top)$
b) $\Box(\top \supset \langle \mathsf{refuseInform}(ag_j, ag_i, l)\rangle\top)$
c) $\Box([\mathsf{refuseInform}(ag_j, ag_i, l)]\mathcal{B}^{ag_i}\mathcal{U}^{ag_j} l)$

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. After a refusal the receiver believes that the sender ignores l.

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:

$$[\mathsf{get\_message}(ag_i, ag_j, l)]\varphi \equiv$$
$$[\bigcup_{\mathsf{speech\_act} \in \mathcal{C}_{\mathsf{get\_message}}} \mathsf{speech\_act}(ag_j, ag_i, l)]\varphi \qquad (6)$$

Intuitively, $\mathcal{C}_{\mathsf{get\_message}}$ 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 the speech acts in $\mathcal{C}_{\mathsf{get\_message}}$ on ag_i's mental state.

Conversation protocols We suppose individual speech acts to take place in the context of predefined conversation protocols [17] that specify communication patterns. Each agent has a subjective perception of the communication with other agents, for this reason each protocol has as many procedural representations as the possible roles in the conversation. Let us consider, for instance the yes_no_query protocol reported in Fig. 2, a simplified version of the FIPA Query Interaction Protocol [13]. The protocol has two complementary views, one to be followed

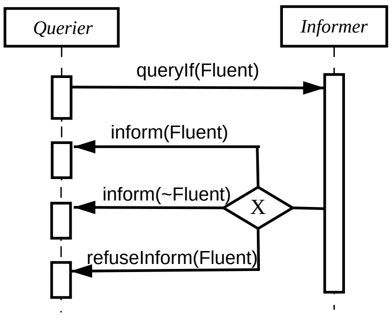

Fig. 2. The AUML graph represents the communicative interactions occurring between the *querier* and the *informer* in the yes_no_query protocol.

for making a query ($\mathsf{yes\_no\_query}_Q$) and one for responding ($\mathsf{yes\_no\_query}_I$). In the following get_answer and get_start definitions are instances of the get_message axiom.

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

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

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 $\mathsf{yes\_no\_query}_Q$ protocol.

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

The $\mathsf{yes\_no\_query}_I$ protocol specifies the behavior of the agent $Self$, that waits a query from $Other$; afterwards, it replies according to its beliefs on the query subject. get_start is a get_message action ruled by the following axiom:

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

We can define the *communication kit* of an agent ag_i, CKit^{ag_i}, as the triple $(\Pi_{\mathcal{C}}, \Pi_{\mathcal{CP}}, \Pi_{\mathcal{S}get})$, where $\Pi_{\mathcal{C}}$ is the set of simple action laws defining ag_i's primitive speech acts, $\Pi_{\mathcal{S}get}$ is a set of axioms for ag_i's get_message actions and $\Pi_{\mathcal{CP}}$ 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 $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$, 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 $(\Pi_{\mathcal{A}}, \Pi_{\mathcal{S}}, \Pi_{\mathcal{P}})$, where $\Pi_{\mathcal{A}}$ is the set of ag_i's world action and precondition laws, $\Pi_{\mathcal{S}}$ is a set of axioms for ag_i's sensing actions, $\Pi_{\mathcal{P}}$ a set of axioms that define complex actions.

C. Reasoning about conversations

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

$$\langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_m \rangle Fs \tag{7}$$

Each p_k, $k = 1, \ldots, m$ in (7) 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). By checking if a query of form (7) succeeds we can cope with the planning problem. In fact this corresponds to answering to the question "is there an execution trace of $p_1, \ldots, p_n$ 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 planning problem turns into the problem of reasoning about *conversation protocols*, where a conversation is a sequence of speech acts. This allows, for instance, an agent to *investigate* the possible changes to its mental state, produced by a specific conversation, or if a conversation is an instance of some predefined protocol [12].

Since conversation protocols represent conversation schemas that guide the communicative behavior of the agent, by answering to the query (7) we find a conversation, which is an instance of the protocol, after which the desired condition Fs holds. In this process we treat get_message actions as sensing actions, whose outcome cannot be known at planning time. Since agents cannot read each other's mind, 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; we can foresee them because of the existence of the protocol. Therefore, the extracted plan will be *conditional*, in the sense that for each get_message 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, a conditional plan σ is either:

- an action sequence $a_1; \ldots; a_m$, with $m \geq 0$;
- if $a_1; \ldots; a_m$ ($m \geq 0$) is an action sequence, $s \in \mathcal{S}$ is a sensing action, and $\sigma_1, \ldots, \sigma_t$ are conditional plans then $a_1; \ldots; a_m; s; ((\mathcal{B}^{ag_i} l_1?); \sigma_1 \cup \ldots \cup (\mathcal{B}^{ag_i} l_t?); \sigma_t)$, where $l_1, \ldots, l_t \in dom(s)$;
- if $a_1; \ldots; a_m$ ($m \geq 0$) is an action sequence, $g \in \mathcal{S}$ is a get_message action, and $\sigma_1, \ldots, \sigma_t$ are conditional plans then $a_1; \ldots; a_k; g; ((\mathcal{B}^{ag_i} Done(c_1)\top?); \sigma_1 \cup \ldots \cup (\mathcal{B}^{ag_i} Done(c_t)\top?); \sigma_t)$, where $c_1, \ldots, c_t \in \mathcal{C}_g$.

The proof procedure is a natural evolution of the work in [5], and is described in [2]; it is a goal-directed proof procedure, based on negation as failure (NAF). NAF is used to deal with the persistency problem to verify 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 [24]. The proof procedure allows agents to find *linear* and *conditional* plans for achieving 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 [11]. Intuitively, this is necessary to grant that persistency of fluents depends only on action effects and the values that the same fluents had in the previous state (see [5], [24]). Moreover, the extracted plans always lead to a state in which the desired condition Fs holds, for all the possible results of the sensing actions.

IV. A PERSONAL ASSISTANT FOR BOOKING CINEMA TICKETS

Let us consider again the personal assistant introduced in Section II, its aim is to look for a cinema booking service that satisfies the user's requests. The two web services, $click\_ticket$ and all_cinema, respectively follow the interaction protocols get_ticket_1 and get_ticket_2. The difference between them is that the former permits both to book a ticket to be paid later by cash and to buy it by credit card, while the latter allows only ticket purchase by credit card. Let us suppose that such protocols are part of the DAML-S descriptions of $click\_ticket$ and all_cinema. Since these protocols are meant to allow the interaction of two agents, each of them has two complementary views: the view of the web service and the view of the client, i.e. pa. Intuitively, if one of the two agents plays the part of the sender of a piece of information, the other should play the part of the receiver. In the following we will report the views –written in DyLOG– that pa has of the two protocols. We refer to them as $\mathsf{get\_ticket\_1}_C$ and $\mathsf{get\_ticket\_2}_C$. Let us suppose that pa knows the credit card number (cc_number) of the user but it is requested not to use it. Let us see how pa reasons on the *way* in which conversations will be carried on.

(a) $\langle \mathsf{get\_ticket\_1}_C(Self, WebS, Film)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Film));$
$\mathcal{B}^{Self} available(Film)?\ ;$
$\mathsf{get\_info}(Self, WebS, cinema(C));$
$\mathsf{yes\_no\_query}_I(Self, WebS, pay\_by(credit\_card));$
$\mathcal{B}^{Self} pay\_by(credit\_card)?\ ;$
$\mathsf{inform}(Self, WebS, cc\_number);$
$\mathsf{get\_info}(Self, WebS, booked(Film))\rangle\varphi$

(b) $\langle \mathsf{get\_ticket\_1}_C(Self, WebS, Film)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Film));$
$\mathcal{B}^{Self} available(Film)?\ ;$
$\mathsf{get\_info}(Self, WebS, cinema(C));$
$\mathsf{yes\_no\_query}_I(Self, WebS, pay\_by(credit\_card));$
$\neg\mathcal{B}^{Self} pay\_by(credit\_card)?\ ;$
$\mathsf{get\_info}(Self, WebS, pay\_by(cash));$
$\mathsf{get\_info}(Self, WebS, booked(Film))\rangle\varphi$

(c) $\langle \mathsf{get\_ticket\_1}_C(Self, WebS, Film)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Film));$
$\neg\mathcal{B}^{Self} available(Film)?\rangle\varphi$

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

Protocol $\mathsf{get\_ticket\_1}_C$ works in the following way: the personal assistant ($Self$) is supposed to begin the interaction. After checking if the requested movie is available in some cinema by the $\mathsf{yes\_no\_query}_Q$ protocol, it should wait for an information (get_info) from the provider ($WebS$) about which cinema shows it. Then the form of payment is defined: (a) defines the interaction that occurs when the tickets are paid by credit card (see Fig. 1(i)); (b) is selected when $\neg\mathcal{B}^{Self} pay\_by(credit\_card)$ is contained in pa mental state (which is our case), leading to book a ticket to be paid by cash (see Fig. 1(ii)). In both cases a confirmation of the ticket booking is returned to the pa. Clause (c) tackles the case in which the movie is not available. Clause (d) describes get_info, which is a get_message action. Let us now consider the query:

$$\langle \mathsf{get\_ticket\_1}_C(pa, click\_ticket, akira)\rangle \\ \mathcal{B}^{pa}\neg\mathcal{B}^{click\_ticket} cc\_number$$

that amounts to determine if there is a conversation between pa and $click\_ticket$ about the movie $akira$, that is an instance of the conversation protocol get_ticket_1$_C$, after which the service does not know the credit card number of the user. Agent pa works on the behalf of a user, thus it knows the user's credit card number ($B^{pa} cc\_number$) and his desire not to use 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 that $click\_ticket$ is an authority about cinema and tickets. This is represented by the beliefs: $\mathcal{B}^{pa} authority(pa, cc\_number)$ and $\mathcal{B}^{pa} authority(click\_ticket, booked(akira))$. The initial mental state will also contain the fact that pa believes that no ticket for $akira$ has been booked yet, $\mathcal{B}^{pa}\neg booked(akira)$, and some hypothesis on the interlocutor's mental state, e.g. the belief fluent $\mathcal{B}^{pa}\neg B^{click\_ticket} cc\_number$, meaning that the web service does not already know the credit card number. Suppose, now, that the ticket is available; since pa mental state contains the belief $\neg\mathcal{B}^{pa} pay\_by(credit\_card)$, when it reasons about the protocol execution, the test on $\mathcal{B}^{pa} pay\_by(credit\_card)?$ fails. Then clause (b) is to be followed, leading pa to be informed that it booked a ticket, $\mathcal{B}^{pa} booked(akira)$, which is supposed to be paid cash. No communication involves the belief $\mathcal{B}^{pa}\neg B^{click\_ticket} cc\_number$, which persists from the initial state. Even when the ticket is not available or the movie is not known by the provider, the interaction ends without consequences on the fluent $\mathcal{B}^{pa}\neg B^{click\_ticket} cc\_number$. After the briefly described reasoning process, the agent finds an execution trace of get_ticket_1$_C$, which corresponds to a *personalized conditional dialogue plan* between itself and the provider $click\_ticket$, always leading to satisfy the user goal of not giving the credit card number:

```
$\mathsf{queryIf}(pa, click\_ticket, akira);$
    $((\mathcal{B}^{pa} Done(\mathsf{inform}(click\_ticket, pa, akira))\top?);$
        $\mathsf{get\_info}(pa, click\_ticket, cinema(C));$
        $(\mathcal{B}^{pa} Done(\mathsf{inform}(click\_ticket, pa, cinema(C)))\top?);$
        $\mathsf{get\_info}(pa, click\_ticket, pay\_by(credit\_card)));$
        $(\mathcal{B}^{pa} Done(\mathsf{queryIf}(click\_ticket, pa,$
            $pay\_by(credit\_card)))\top?);$
        $\mathsf{inform}(pa, click\_ticket, \neg pay\_by(credit\_card));$
        $\mathsf{get\_info}(pa, click\_ticket, pay\_by(cash));$
        $(\mathcal{B}^{pa} Done(\mathsf{inform}(click\_ticket, pa, pay\_by\_cash))\top?);$
        $\mathsf{get\_info}(pa, click\_cinema, booked(akira));$
        $(\mathcal{B}^{pa} Done(\mathsf{inform}(click\_ticket, pa, booked(akira)))\top?) \cup$
    $(\mathcal{B}^{pa} Done(\mathsf{inform}(click\_ticket, pa, \neg akira))\top?) \cup$
    $(\mathcal{B}^{pa} Done(\mathsf{refuseInform}(click\_ticket, pa, akira))\top?))$
```

Let us now considering get_ticket_2$_C$, the protocol followed by all_cinema:

(e) $\langle \mathsf{get\_ticket\_2}_C(Self, WebS, Film)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Film));$
$\mathcal{B}^{Self} available(Film)?\ ;$
$\mathsf{get\_info}(Self, WebS, cinema(C));$
$\mathsf{inform}(Self, WebS, cc\_number);$
$\mathsf{get\_info}(Self, WebS, booked(cinema\_ticket, Film))\rangle\varphi$

(f) $\langle \mathsf{get\_ticket\_2}_C(Self, WebS, Film)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Film));$
$\neg\mathcal{B}^{Self} available(Film)?\rangle\varphi$

We can easily see that in this case, the query $\langle \mathsf{get\_ticket\_2}_C(pa, all\_cinema, akira)\rangle\ \mathcal{B}^{pa}\neg\mathcal{B}^{all\_cinema} cc\_number$ fails because, the protocol allows only one interaction sequence, containing the action inform(pa, all_cinema, *cc_number*) that causes the mental state of pa to contain $\mathcal{B}^{pa}\mathcal{B}^{all\_cinema}$ *cc_number* (see Fig. 1(iii)).

V. Conclusions

In this work we have shown the possible benefits of enriching the (DAML-S) web service description with the explicit representation of the conversation protocol followed by the service: by reasoning about such protocols, agents can personalize the interaction by selecting an interaction course that satisfies user-given requirements. This process can be started before the actual interaction takes place and can be exploited also for web service search. The idea of declaring the conversation protocol derives from the experience of the authors in the multi-agent research area, where agents commonly exchange communication protocols before interacting [17][4]. We based our work in a modal action logic framework and we used the agent logic programming language DyLOG, that includes a communication kit based on FIPA-like communicative acts. The semantics of communicative acts is described in terms of their effects on the mental state of both of the sender and the recipient; 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 [26], 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. [8], that supply the abstractions necessary for ruling the interaction of agents, aimed at satisfying a common goal. A well-known example of coordination model is TuCSoN [21], which exploits tuple centers and the logic-based language ReSpecT [22] 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 eyes of 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.

References

[1] M. Baldoni, C. Baroglio, A. Chiarotto, and V. Patti. Programming Goal-driven Web Sites using an Agent Logic Language. In I. V. Ramakrishnan, editor, *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 60–75, Las Vegas, Nevada, USA, 2001. Springer.

[4] For a discussion of our work in this context see [2].

[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*, LNCS. Springer, 2003. to appear.

[3] M. Baldoni, C. Baroglio, and V. Patti. Applying logic inference techniques for gaining flexibility and adaptivity in tutoring systems. In C. Stephanidis, editor, *Proceedings of the 10th International Conference on Human-Computer Interaction (HCII 2003)*, Crete, Grece, 2003. Lawrence Erlbaum Associates. To appear.

[4] M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. Using a Rational Agent in an Adaptive Web-based Tutoring System. In P. Brusilovsky, N. Henze, and E. Millan, editors, *Proc. of Workshop on Adaptive System for Web-based Education, 2nd Int. Conf. on Adaptve Hypermedia and Adaptive Web Based Systems (AH 2002), Selected Papers*, pages 43–55, Malaga, Spain, 2002.

[5] 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.

[6] 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.

[7] 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.

[8] P. Ciancarini and J. Wooldridge, editors. *Agent Oriented Software Engeneering (AOSE 2000)*, volume 1957 of *LNCS*. Springer-Verlag, 2000.

[9] DAML-S. http://www.daml.org/services/daml-s/0.9/. version 0.9, 2003.

[10] DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html. 2001.

[11] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abduction Logic Programming. In *Proc. of ILPS '93*, Vancouver, 1993. The MIT Press.

[12] F. Dignum and M. Greaves. Issues in agent communication. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 1–16. Springer, 2000.

[13] FIPA. FIPA 2000. Technical report, FIPA (Foundation for Intelligent Physical Agents), November 2000.

[14] 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.

[15] A. Herzig and D. Longin. Beliefs dynamics in cooperative dialogues. In *Proc. of AMSTELOGUE 99*, 1999.

[16] 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.

[17] 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 *LNCS*, pages 160–177. Springer, 2000.

[18] J. Maurer, editor. *ACM Queue*. ACM, march 2003.

[19] 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.

[20] 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/.

[21] 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.

[22] Andrea Omicini and Enrico Denti. Formal ReSpecT. In Maria Chiara Meo, editor, *2000 Joint Conference on Declarative Programming (AGP'00)*, La Habana (Cuba), 4–7 December 2000.

[23] OWL. http://www.w3c.org/tr/owl-guide/. 2003.

[24] 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/`.

[25] RDF. http://www.w3c.org/tr/1999/rec-rdf-syntax-19990222/. 1999.

[26] 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.

[27] WSDL. http://www.w3c.org/tr/2003/wd-wsdl12-20030303/. version 1.2, 2003.