Preserving player's goals: a choreography-driven matchmaking approach

Matteo Baldoni, Cristina Baroglio, Alberto Martelli,
Viviana Patti, and Claudio Schifanella
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,mrt,patti,schi}@di.unito.it

***Abstract*—An agent interaction protocol, a service choreography, can quite naturally be interpreted as an alliance of parties, which cooperate to achieve a goal. On the other hand, each participant entered the alliance moved by goals of its own, which it would like to fulfill by playing one of the roles. The achievement of the shared and of the specific goals depend both on the interaction schema, that is captured by the choreography, and on the participant's capabilities, where by this word we mean the skills of the participant, the actions that it can execute. We show in this paper that the choice of which capabilities to use cannot rely totally on local criteria, as instead it is commonly done by the approaches to matchmaking, but it must take into account the choreography/protocol. This happens whenever the match is not exact, e.g. when plugin match is used. We also describe an extended plugin match that takes into account also the constraints given by the choreography for performing the capability selection.**

I. Introduction

Web services have a platform-independent nature, that endeavors enterprises to develop new business processes by combining existing services, retrieved over the web. Web service composition is still much of a costly and manual process, which is made more and more difficult by the growing width of the space to search. Hence, the need of methods for reducing the search space and for making compositions in an automatic way. This direction has been suggested in [1], where a UML specification of a business process was used to abstract the description of a composition away from the specification of the actually composed services. This abstract specification defined a *model*, used for driving the retrieval and the composition task. The idea of capturing the overall schema of interaction of a set of entities is exploited also in other areas, like multi-agent systems. In this context, the role of the abstract specification is played by the so-called "interaction protocol" [2], while actions take the place of services..

Kolp, Giorgini, and Mylopoulos have investigated [3] the possibility of using real world organizational structures as a metaphor for defining MAS architectures. The derived architectures are evaluated w.r.t. a set of quality attributes, amongst which predictability and adaptability. One of the studied human organizational structures is "strategic alliance". A strategic alliance links specific facets of a group of organizations and is defined with the purpose of achieving an overall, shared goal. The organizations within the alliance, however, remain independent and have control over the assigned tasks. Each actor, however, has to reconcile and adjust its own views with the policies of the organization. This is particularly true in the case of "co-optation", a special kind of strategic alliance, in which the partners become part of a newly founded organization, hence having, on the one hand, a commitment on pursuing the alliance goals and interests, and, on the other hand, their own and specific goals to reach.

Among the many quality attributes that the authors define, the following are particularly interesting w.r.t. our work.

- *Coordinability*: agents are not really useful if they cannot coordinate. Coordination is used to distribute expertise, information, etc. among agents, which depend on one another. *Cooperativity* is a form of coordination. Cooperation is achieved either communicative or non-communicative.
- *Modularity*: it increases the efficiency of task execution, results in higher flexibility and reduces the communication overhead, although, on the other hand it constrains inter-module communication.
- *Predictability*: agents have many degrees of freedom in the way they undertake action, in their domains. The capability of predicting the behaviour of the individuals is important when we need to aggregate such individuals in an organization.

It is quite natural, then, in the case of multi-agent systems and of (web) services, to interpret an interaction protocol, or a choreography, as the specification of an alliance (in particular, a joint venture or a co-optation), because they specify a coordination pattern based on communication. Roles can be considered as modules that capture an activity within the schema, constraining the interaction of the partners. The fact that a partner takes a role in a choreography guarantees that the partner will behave as expected (predictability of the behavior). The choreography/protocol can be seen as an alliance of independent partners. This alliance is aimed at pursuing a goal, that is subscribed by all the participants, but each partner has also *its own* goals, that motivate its taking part to the alliance. The achievement of the shared goal and of the specific agent's goal not only depends upon the choreography-given schema of interaction but also on the skills that each agent has, i.e. by each agent's specific *capabilities*. Indeed, every agent has control over the ways for accomplishing the assigned tasks. Thus, before an agent subscribes the alliance, there is a need to check if its capabilities match with those that are requested by a role, i.e. if its capabilities allow it to achieve its goal in the context of the given choreography. It is implicit that the choreography (the protocol) specify in some way the necessary capabilities. In [4], we have shown that there is the need of enriching the choreography specification by introducing the concept of *capability requirement*. A capability requirement expresses an operation that a peer should be able to perform at some specific point of the choreography.

(Web) services share many facets with multi-agent systems [5]. The introduction of the concept of "choreography" (and of languages like WS-CDL [6]) has opened new perspectives on the way an abstract specification of a system of services should be described. Choreographies can be used to build policies that some peer will execute. In order for a policy to be "playable" by a peer, the peer must have the requested capabilities. For instance, it must have some means for producing or retrieving (e.g. by contacting a third service) a piece of information to send. This approach can be extended by considering different kinds of actions (e.g. communicative actions) or a different granularity (e.g. agents, services, or other software components). More in details, a role specification in a choreography can be used to produce a *policy skeleton*, which is to be completed by *substituting* capabilities to capability requirements, so to make it executable. The substitution can be defined by applying a *matching process* between the abstract specification given by capability requirements and the available capabilities. In general, it is unlikely to have capabilities that perfectly match the requirements; the retrieval process will identify capabilities which *slightly differ* from the specification. If one wants to use them anyway, rather than writing new software, it is necessary to verify that the policy obtained after the substitution still allows the achievement of the goal, which is not granted anymore [4].

The task of retrieving capabilities that match given requirements is analogous to the task of service discovery. We can, then, think to use the same techniques, e.g. [7], [8], [9], [10]. In particular, in this work we focus on the matches proposed by Zaremski and Wing in their seminal work [8], where various kinds of relaxed match are proposed. We show that none of the matches (but the so-called exact pre/post match) guarantee that a synthesized policy, in which capabilities have been selected according to them, will still allow to reach the goal of interest. The reason is that they take into account only the "local" information given by the capability requirement and do not consider constraints posed by the choreography ("global" constraints). We also show how to integrate the *plugin* match in the context given by a choreography in such a way that the goal is preserved by the substitution.

The article is organized as follows. In Section II we recall the matches introduced in [8], and explain their relations with a choreography and with the goals. Section III introduces a simple representation for services and choreographies, that is based on a declarative language. Section IV shows that the local matches alone do not guarantee the preservation of the goal, and it also shows how to integrate the plugin match so to produce substitutions that preserve the goal. We will introduce the notion of *conservative substitution*. Conclusions and related works end the paper.

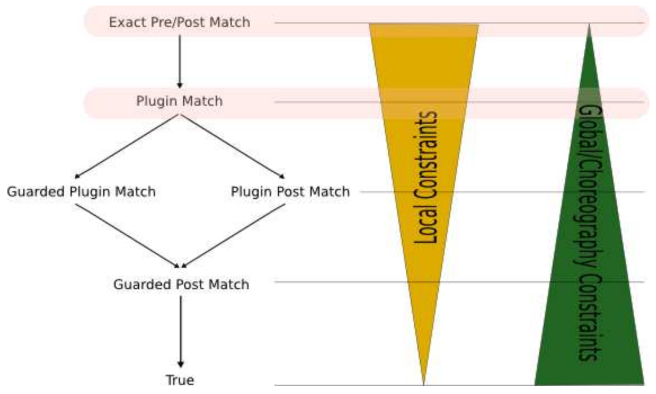

Fig. 1. The lattice of the different local matches: on top the strongest. Our claim is that the local and global constraints are related; the stronger the local match, the weaker the global constraints.

II. Capability match: local vs. global properties

We suppose, in the line of previous work [4], that choreographies are enriched with additional descriptions of those actions, that peers must be able to perform for playing roles (*capability requirements*). Capability requirements are used to select the specific capabilities that are necessary to build an executable policy. As mentioned in the introduction, this selection can be done by applying matching techniques that are analogous to those used for service discovery. Zaremski and Wing [8] propose a formal specification to describe the behavior of software components. Each software component has precondition S_{pre} and postcondition S_{post} written as predicates in first-order logic. Requirements are coherently specified as having precondition R_{pre} and postcondition R_{post}. Five kinds of relaxed match between R and S are defined:

- EM (*Exact Pre/Post Match*): $R_{pre} \Leftrightarrow S_{pre} \wedge R_{post} \Leftrightarrow S_{post}$
- PIM (*Plugin Match*): $R_{pre} \Rightarrow S_{pre} \wedge S_{post} \Rightarrow R_{post}$
- POM (*Plugin Post Match*): $S_{post} \Rightarrow R_{post}$
- GPIM (*Guarded Plugin Match*): $R_{pre} \Rightarrow S_{pre} \wedge ((S_{pre} \wedge S_{post}) \Rightarrow R_{post})$
- GPOM (*Guarded Post Match*): $((S_{pre} \wedge S_{post}) \Rightarrow R_{post})$

Exact pre/post match states the equivalence of R and S. *Plugin match* is weaker: S must only be behaviorally equivalent to R when plugged-in to replace R. *Plugin post match* relaxes the former: only the postcondition is considered. *Guarded matches* focus on guaranteeing that the desired postcondition holds when the precondition of S holds, not necessarily in general. The different matches can be organized according to a lattice [8], that we have reported in Fig. 1.

In our application domain, R will be a capability requirement, while S will be a capability. Capability requirements are contextualized in some choreography. Capabilities are specific software components and depend on the player of a choreography role: they are matched against requirements in the process of checking if a player can play a certain role, by selecting –at the same time– its right capabilities. In general, since the final aim is software reuse, it will be quite difficult to retrieve an exact match for a capability requirement. More likely (and more interesting) is the case when one of the other four degrees of match hold.

All these matches have been defined for the retrieval of single components, and have a *local* nature, i.e. they compare a requirement to a software specification (in our case, a capability) independently of the context of usage (in our work, the service choreography). In other words, the software specification must respect some constraints. Relaxing the exact match means relaxing these "local constraints" (see Fig. 1). On the other hand, a choreography defines the *global execution context*, in which capability requirements are immersed. Intuitively, the selection of a capability (for replacing a capability requirement) should *preserve* those properties of the choreography that motivated its choice, in particular, the *goal* for which it was chosen. In the case of the *exact match*, the whole verification is done *locally*. Due to the fact that it is a kind of equivalence, matching exactly a requirement is a sufficient condition to preserve the goal. As we will see in Section IV, the other kinds of match do not give this guarantee. It becomes, therefore, necessary to *add some constraints* by using the available source of global information: the choreography.

Our claim (see Fig. 1) is that the more relaxed is the local match, the stronger should be the compensation supplied at the global level. The extreme is given by the bottom of the lattice: the match that returns always *true*. In this case, the choice of the capabilities could be performed, for instance, by randomly choosing capabilities and by substituting the to the requirements while simulating the execution of the policy. When the goal is not verified by the current choice, a *backtracking* mechanism allows the revision. The whole process *relies on* the choreography. Checking global constraints can be expensive but it is possible to reduce the costs by limiting the attention to those capability requirements which belong to the execution traces, which actually allow to achieve the goal.

III. Reasoning about capabilities

For what concerns the representation of choreographies and specific peers, in order to abstract from the specific language (e.g. WS-CDL, WSDL) and from the details of the implementation, we adopt a *declarative* representation and focus on the study of the properties of interest.

Each choreography is made of a set of *interacting roles*. It can be described as a set of subjective views of the interaction that is encoded, each corresponding to one of the roles. We call the implementation of each role a *policy*. We will represent both *roles* and policies by means of the *declarative language* DyLOG [11], by interpreting interactions among services, capabilities and capability requirements as *actions*, and by using *reasoning about actions* for making predictions about the effects of role and policies executions.

DyLOG has been developed as a language for programming agents and is based on a logical theory for reasoning about actions and change in a modal logic programming setting. DyLOG is equipped with a communication kit for dealing with interactions, and has already been used for customizing Web service composition [12]. An agent's behavior is described in a non-deterministic way by giving the set of actions that it can perform. Each action can have preconditions to its application and cause some effects. Given this view of actions, we can think to the problem of reasoning as the act of building or of traversing a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time. Such properties encode the information that flows during the execution of the agent actions. In DyLOG we do not assume that the value of each fluent in a state is known: it is possible to represent unknown fluents and to reason about the execution of actions on incomplete states. We introduced an epistemic operator $\mathcal{B}_i$, to represent the beliefs that an entity i has about the world: $\mathcal{B}_i f$ means that the fluent f is believed to be true by the entity i, $\mathcal{B}_i \neg f$ means that the fluent f is believed to be false. A fluent f is undefined, $u_i(f)$, when both $\neg\mathcal{B}_i f$ and $\neg\mathcal{B}_i \neg f$ hold. Thus each fluent in a state can have one of the three values: *true, false* or *unknown*.

In a DyLOG description of a service role (or policy) the interactions between the service and its interlocutor(s) can be defined in terms of communicative actions performed by the service (*speech acts*) and *get-message actions*. A *speech act* is an atomic action of form *performative(sender, receiver, content)*, where *performative* is the kind of speech act (e.g. inform), *sender* and *receiver* are the name of the interacting peers, while *content* is a *fluent literal* representing the piece of information that is passed by its execution. The set of all performatives is supposed to be *shared by the two parties*. *Get-message* actions allow to represent the reception of information and to reason about the outcome of the speech acts performed by the interlocutor. The range of possible incoming speech acts is supposed to be finite: the interlocutor is supposed to use a performative out of a finite and predefinite set to produce its answer within a choreographed interaction. *Capability requirements*/*capabilities* in a service role/policy are represented as (possibly communicative) atomic actions.

Complex behaviors can be specified in DyLOG by means of *procedures*, Prolog-like clauses built upon the other kind of actions mentioned. We represent the behavior of both *roles* and *policies* by DyLOG procedures [1]. Intuitively, a *role* is a procedure that combines speech acts, get-message acts, *capability requirements* and procedure calls, and a *policy* is a procedure combining speech acts, get-message acts, *capabilities* and procedure calls.

A *role* in a choreography can, therefore, be specified as a quadruple of the form $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where:

[1]Since our focus is to study the preservation of global properties, we will assume that the sets of terms used for representing speech acts and capabilities are the same in the choreography and in the peer description.

1) $\mathcal{S}_\mathcal{A}$ is a set of *speech acts*, represented as [2]:

$\mathsf{performative}(sender, receiver, l)$
$\quad$ **causes** $\{E_1, \ldots, E_n\}$
$\mathsf{performative}(sender, receiver, l)$
$\quad$ **possible if** $\{P_1, \ldots, P_t\}$

where E_i, and P_j are respectively: the fluents that are obtained as effect of the speech act, and the precondition to the execution of the performative.

2) $\mathcal{G}_\mathcal{A}$ is a set of *get-message actions*, they are represented as: $\mathsf{receive\_act}(receiver, sender, [l_1, \ldots, l_n])$ **receives** $\mathcal{I}$, where $\mathcal{I}$ is a set of alternative speech act, that can be received by the executor of receive_act; each speech act in $\mathcal{I}$ has an element in $[l_1, \ldots, l_n]$ as content.

3) $\mathcal{CR}$ is a set of capability requirements, they are modeled as atomic actions and are represented as:

c **causes** $\{E_1, \ldots, E_m\}$
c **possible if** $\{P_1, \ldots, P_t\}$

where c is the name of the required capability and the semantics of the clauses is the same as above. We will use the functions $\mathsf{Effs}(c) = \{E_1, \ldots, E_m\}$ and $\mathsf{Precs}(c) = \{P_1, \ldots, P_t\}$ to return the effects and the preconditions of c. The same functions apply also to speech acts.

4) $\mathcal{P}$ encodes the behavior for the role; it is represented as a collection of clauses of the kind p_0 **is** $p_1, \ldots, p_n$ $(n \geq 0)$, where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action, a *get-message* action, a test action, or a procedure name (i.e. a procedure call). Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs, and their definitions can be non-deterministic as in Prolog.

Policies are defined in a way that is analogous to role descriptions. Let $\mathcal{C}$ be the set of capabilities of a peer, then, a *policy* is quadruple $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{S}_\mathcal{A}$, $\mathcal{G}_\mathcal{A}$, and $\mathcal{P}$ are defined as above.

Example 1: As an example, let us introduce a choreography (enriched with capability requirements) that rules a simple room reservation protocol with two roles: the *buyer* wants to book a room at the hotel managed by the *seller*. Figure 2 depicts the interaction between the two roles: first the buyer sends to the seller the date for the room reservation; then, the seller must have the capability of performing a *reserveRoom* action, and inform the buyer about the room price. The buyer checks the price, by performing an *evaluatePrice* action. Then, it informs the seller about the results of this evaluation: it can either decide to refuse the offer and conclude the interaction or it can inform the seller about the desired payment mode (*cash* or *credit card*). At this point, the seller must have the capability

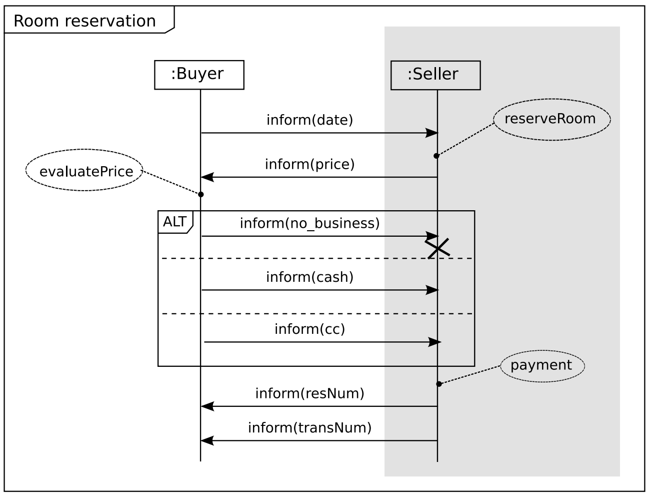

Fig. 2. The Room Reservation Protocol, represented by means of UML sequence diagrams, and enriched with capability requirements (oval elements).

of performing the *payment* action, and finalize the business transaction. Finally it notifies the buyer the reservation and transaction numbers.

Let us focus on the *seller* role description [3] $R_{seller} = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where $\mathcal{P} = \{\mathsf{booking}, \mathsf{finalize\_reservation}\}$, $\mathcal{S}_\mathcal{A} = \{\mathsf{inform}(s,\ b, price),\ \mathsf{inform}(s,\ b, resNum),\ \mathsf{inform}(s\ ,b,\ transNum)\}$, $\mathcal{G}_\mathcal{A} = \{\mathsf{receive\_date}(s, b, date),$ $\mathsf{receive\_evaluation}(s,\ b,\ [no\_business,\ cash, cc])\}$, $\mathcal{CR} = \{\mathsf{reserve\_room}_{\mathsf{CR}}, \mathsf{payment}_{\mathsf{CR}}\}$. The procedures in $\mathcal{P}$ are described by the following clauses:

$\mathsf{booking}$ **is** $\mathsf{receive\_date}(s, b, date)$,
$\quad \mathsf{reserve\_room}_{\mathsf{CR}}, \mathsf{inform}(s, b, price)$,
$\quad \mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc])$,
$\quad \mathsf{finalize\_reservation}$
$\mathsf{finalize\_reservation}$ **is** $\mathcal{B}no\_business?$
$\mathsf{finalize\_reservation}$ **is** $\mathsf{payment}_{\mathsf{CR}}, \mathsf{inform}(s, b, resNum)$,
$\quad \mathsf{inform}(s, b, transNum)$

The get_message actions in $\mathcal{G}_\mathcal{A}$ are described by:

$\mathsf{receive\_date}(s, b, date)$ **receives** $[\mathsf{inform}(b, s, date)]$
$\mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc])$
$\quad$ **receives** $[\mathsf{inform}(b, s, no\_business)$ **or**
$\quad \mathsf{inform}(b, s, cash)$ **or** $\mathsf{inform}(b, s, cc)]$

The capability requirements in $\mathcal{CR}$:

$\mathsf{reserve\_room}_{\mathsf{CR}}$ **causes** $\{\mathcal{B}price\}$
$\mathsf{reserve\_room}_{\mathsf{CR}}$ **possible if** $\{\mathcal{B}date\}$
$\mathsf{payment}_{\mathsf{CR}}$ **causes** $\{\mathcal{B}transNum, \mathcal{B}resNum\}$
$\mathsf{payment}_{\mathsf{CR}}$ **possible if** $\{\mathcal{B}PcashSupported,$
$\quad \mathcal{B}PccSupported\}$

Finally, the semantics of the $\mathsf{inform}(sender, receiver, l)$ actions in $\mathcal{S}_\mathcal{A}$ and $\mathcal{G}_\mathcal{A}$ is given by the rules (for more details see [12]):

$\mathsf{inform}(s, b, l)$ **possible if** $\{\mathcal{B}_s l\}$
$\mathsf{inform}(s, b, l)$ **causes** $\{\}$
$\mathsf{inform}(b, s, l)$ **possible if** $\{\}$

[2]In DyLOG the semantics of speech acts is inspired to the standard semantics of FIPA Communicative Acts [13]. Therefore speech acts are characterized in terms of (a) feasibility preconditions denoting the ability of the speaker to perform the act and (b) the desired and rational *perlocutionary effects* of the utterance. See [11] for more details.

[3]In the following examples all the beliefs refer to the seller mental state, thus, for sake of readability we will omit to index the modal operator $\mathcal{B}$.

$\mathsf{inform}(b, s, l)$ **causes** $\{\mathcal{B}_s l\}$

Intuitively, the first two clauses state that I (the seller) can execute an inform act only if I believe l; the execution of the action will modify the interlocutor's mental state, while do not have any effects on my mental state. The last two clauses describe what happen in my mental state when I am the receiver of the information. In this case, since I am not the actor, the action of informing is considered *always* executable; moreover I will adopt l as my own belief. ■

In DyLOG, it is possible to perform a form of reasoning known as *temporal projection*, by means of *existential* queries of the form: Fs **after** p, where p is a policy name and Fs is a conjunction of fluents. Checking if a formula of this kind holds corresponds to answering the query "Is there an execution trace of p that leads to a state in which Fs is true?". By execution trace we mean a sequence of atomic actions, i.e. speech acts and capabilities (capability requirements). When the answer is positive, such sequence is a plan to bring about Fs. This plan can be *conditional* because whenever a *get-message* action is involved none of the possible answers from the interlocutor can be excluded. In other words, we will have a different execution branch for every option.

Let us consider a role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$. We can apply temporal projection to $\mathcal{P}$ to find an execution trace, that makes a goal of interest become true. Let us, then, consider a procedure p belonging to $\mathcal{P}$, and denote by G the DyLOG query: Fs **after** p, where Fs is the set of fluents that we want to be true after the execution of p. Given a state S_0, containing all the fluents that we know as being true in the beginning, we will denote the fact that G is successful in R_d by:

$$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$$

The execution of the above query returns as a side-effect an *execution trace* σ of p. The execution trace σ can either be *linear*, i.e. a terminating sequence $a_1, \ldots, a_n$ of atomic actions, or it can be *conditional*, when the procedure contains get-message actions. Intuitively, by this mechanism it is possible to verify, by reasoning about the choreography, if the role allows for an execution after which a condition of interest holds.

Example 2: In the context of the Example 1, let us consider the goal:

$$G = \{\mathcal{B}transNum, \mathcal{B}resNum\} \ \textbf{after} \ \mathsf{booking}$$

where the initial state S_0 contains the fluents $\{\mathcal{B}PcashSupported,\ \mathcal{B}PccSupported\}$, while all the other fluents are unknown. There are two possible execution traces that lead to a state where G holds, hereafter we report one of them:

$\sigma = \mathsf{inform}(b, s, date); \mathsf{reserve\_room_{CR}};$
$\mathsf{inform}(s, b, price); \mathsf{inform}(b, s, cc); \mathsf{payment_{CR}};$
$\mathsf{inform}(s, b, resNum); \mathsf{inform}(s, b, transNum).$

■

A *policy* can be built from a *role description* by substituting capability requirements with a set of capabilities of a peer that should play the role. If we denote by $\mathcal{C}$ the capabilities of the peer, by $\mathcal{CR}$ the capability requirements, and by θ the substitution $[\mathcal{C}/\mathcal{CR}]$, the policy built from the role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ will be $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle$. Given a policy description $P_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle$, a goal $G = Fs$ **after** p, and an initial state S_0, we can verify if G is successful in P_d by:

$$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle, S_0) \vdash G$$

Intuitively, this allows to verify, by reasoning about the peer description, if the policy allows for an execution that brings about the condition of interest.

IV. Choreography-driven match

When the matching process is applied for selecting a capability that is part of a role specification, the desire is that the selected capability preserves the properties of the specification. Generally, the matchmaking process will result in a set of alternative θ_i because each capability requirement has a set of matching capabilities. The selected θ not only must satisfy the matching rules but it must also be *conservative*, i.e. it must guarantee that those *goals*, that can be achieved by reasoning on the *role specification*, will be achieved also after the *substitution*. Then, the following implication must hold:

Definition 1 (Conservative substitution): Let $\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ be a role description, S_0 the initial state, and G the goal of interest. Suppose that the following relation holds:

$\exists \sigma, \theta = [\mathcal{C}/\mathcal{CR}_\sigma],\ \mathcal{CR}_\sigma \subseteq \mathcal{CR}\ s.t.$
$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. $\sigma \Rightarrow$
$(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta \rangle, S_0) \vdash G$ w.a. $\sigma\theta$

where σ is an execution trace which makes the goal true when reasoning at the level of the choreography, and θ is a substitution $\mathcal{CR}_\sigma \rightarrow \mathcal{C}$, where $\mathcal{CR}_\sigma \subseteq \mathcal{CR}$, $\mathcal{CR}_\sigma = \{cr \in \mathcal{CR} \mid cr \text{ occurs in } \sigma\}$. In this case, the substitution θ is conservative.

Notice that we are interested in a substitution θ that involves only the capability requirements contained in the execution trace σ, which is, therefore, used to select the requirements to be matched. The substitution θ is obtained by applying one of the matching rules, described in Section II, that we here rephrase as follows (c represents a single capability and cr a single capability requirement):

- EM (*Exact Pre/Post Match*): $\mathsf{Precs}(cr) = \mathsf{Precs}(c) \wedge \mathsf{Effs}(cr) = \mathsf{Effs}(c)$
- PIM (*Plugin Match*): $\mathsf{Precs}(cr) \supseteq \mathsf{Precs}(c) \wedge \mathsf{Effs}(c) \supseteq \mathsf{Effs}(cr)$
- POM (*Plugin Post Match*): $\mathsf{Effs}(c) \supseteq \mathsf{Effs}(cr)$
- GPIM (*Guarded Plugin Match*): $\mathsf{Precs}(cr) \supseteq \mathsf{Precs}(c) \wedge ((\mathsf{Precs}(c) \cup \mathsf{Effs}(c)) \supseteq \mathsf{Effs}(cr))$
- GPOM (*Guarded Post Match*): $((\mathsf{Precs}(c) \cup \mathsf{Effs}(c)) \supseteq \mathsf{Effs}(cr))$

For short, we will respectively denote by θ_{EM}, θ_{PIM}, θ_{POM}, θ_{GPIM}, θ_{GPOM}, the substitutions obtained by applying the five degrees of match. For simplicity we will call a substitution obtained by applying the plugin match a PIM substitution, the one obtained by applying Exact Pre/Post match an EM substitution, and so on for the other kinds. It is immediate to see that any substitution, obtained by applying the *exact pre/post match*, satisfies Definition 1. In other words, the local constraints are sufficient to guarantee the property (see Fig. 1). However this is not true for the other kinds of match.

Theorem 1: The class of PIM, POM, GPIM and GPOM substitutions are not conservative.

Proof: The proof is given by a counterexample.

Let us consider a role description $R_d = \langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle$, where $\mathcal{P} = \{p \ \ \mathbf{is} \ \ cr_1, \ a\}$, $\mathcal{S_A} = \{a\}$, $\mathcal{G_A}$ is empty, and the capability requirement cr_1 in $\mathcal{CR}$ and the speech act a in $\mathcal{S_A}$ are described by [4]:

$$\begin{array}{ll} cr_1 \ \ \mathbf{causes} \ \ \{\mathcal{B}l_1\} & a \ \ \mathbf{causes} \ \ \{\mathcal{B}l_2\} \\ cr_1 \ \ \mathbf{possible\ if} \ \ true & a \ \ \mathbf{possible\ if} \ \ \{\mathcal{B}l_1, \mathcal{B}l_3\} \end{array}$$

Assuming as goal $G = \mathcal{B}l_2$ **after** p, where the initial state contains $\mathcal{B}l_3$ while all the other fluents are unknown, the reasoning process will generate the execution trace $\sigma = cr_1; a$ for achieving G. If we consider the set of capabilities $\mathcal{C} = \{c_1\}$:

$$\begin{array}{l} c_1 \ \ \mathbf{causes} \ \ \{\mathcal{B}l_1, \mathcal{B}\neg l_3\} \\ c_1 \ \ \mathbf{possible\ if} \ \ true \end{array}$$

By applying the substitution $\theta = \{[c_1/cr_1]\}$ we obtain the new policy $\mathcal{P}\theta = \{p \ \ \mathbf{is} \ \ c_1, \ a\}$. However, by using this policy, the query $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \vdash G$ does not succeed: in fact, the additional effect $\mathcal{B}\neg l_3$ of the capability c_1 inhibits the executability of the speech act a. On the other hand, it is easy to check that θ is an instance of all the kinds of substitutions that we have listed, i.e. it is a PIM substitution as well as a POM substitution, a GPIM and a GPOM substitution. ■

This example witnesses that working at the level of the local constraints is not sufficient. Our claim is that, in general, in order for a substitution to be conservative, it must take into account not only the *local* aspects but also the *overall structure*, encoded by the choreography. The locality of the matches used in the matchmaking phase, indeed, seriously limits the possibility of re-using software (services) by selecting and composing it in an automatic way.

Let us now focus on the *plug-in match* (PIM), which is one of the most used and which immediately follows the exact match in the lattice (therefore it is the strongest of the flexible matches). We show that, by introducing appropriate constraints at the level of the choreography, it is possible to guarantee the selection of conservative substitutions. To this aim, we take into account the *dependencies* between actions, which produce as effects fluents, that are used as preconditions by subsequent action. Intuitively, the idea is to verify that the "causal chain" which allows the execution of the sequence of actions, is not broken by the differences between capabilities and capability requirements, as instead happens in the example. The obvious hypothesis is that we have a choreography and that we know that it allows to achieve the goal of interest, i.e. that there is an execution σ of the role specification, which allows the achievement of the goal. We will use this trace for defining the additional properties for the match.

Let us start by introducing the notions that define dependencies between actions and dependency sets for fluents. Consider a role description $R_d = \langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle$ and suppose that, given the initial state S_0, the goal $G = Fs$ **after** p succeeds, thus obtaining as answer the successful sequence of actions $\sigma = a_1; a_2; \ldots; a_n$, which is an execution trace of p.[5] We denote by $\overline{\sigma}$ the sequence of actions $a_0; a_1; a_2; \ldots; a_n; a_{n+1}$, where a_0 and a_{n+1} are two *fictitious* actions that will be used respectively to represent the initial state S_0 and the set of fluents Fs, which must hold after σ. That is, we assume a_0 has no precondition and $\mathsf{Effs}(a_0) = S_0$, and that a_{n+1} has no effect but $\mathsf{Precs}(a_{n+1}) = Fs$.

Consider two indexes i and j, such that $j < i$, $i, j = 0, \ldots, n+1$. We say that *in* $\overline{\sigma}$ *the action* a_i *depends on* a_j *for the fluent* $\mathcal{B}l$, written $a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i$, iff $\mathcal{B}l \in \mathsf{Effs}(a_j)$, $\mathcal{B}l \in \mathsf{Precs}(a_i)$, and there is not a k, $j < k < i$, such that $\mathcal{B}l \in \mathsf{Effs}(a_k)$. Given a fluent $\mathcal{B}l$ and a sequence of actions σ, we can, therefore, define the *dependency set* of $\mathcal{B}l$ as $\mathsf{Deps}(\mathcal{B}l, \sigma) = \{(j, i) \mid a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i\}$.

Let $[c/c_r]$ be a specific substitution of a capability requirement with a capability, that is contained in θ_{PIM}, we say that a fluent $\mathcal{B}l \in \mathsf{Effs}(c) - \mathsf{Effs}(c_r)$ (i.e. an additional effect of the capability c w.r.t. the effects of the capability requirement cr) is an *uninfluential fluent* w.r.t. the sequence $\sigma\theta_{PIM}$ iff for all pairs $(j, i) \in \mathsf{Deps}(\mathcal{B}\neg l, \sigma)$, identifying by k the position of c_r in σ, we have that $k < j$ or $i \leq k$, Intuitively, this means that the fluent will not break any dependency between the actions which involve the inverse fluent because either it will be overwritten or it will appear after its inverse has already been used. Note that σ and $\sigma\theta_{PIM}$ have the same length and are identical as sequences of actions but for the fact that in the latter capabilities substitute capability requirements. For this reason, we can reduce to reasoning on σ for what concerns the action positions. A substitution θ_{PIM} is called *uninfluential* iff for any substitution $[c/c_r]$ in θ_{PIM}, all beliefs in $\mathsf{Effs}(c) - \mathsf{Effs}(c_r)$ are uninfluential fluents w.r.t. σ. Now we are in position to prove that a substitution which exploits the *plugin match* and which is also *uninfluential*, is conservative.

Theorem 2: Let G be a goal and let $R_d = \langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle$ a role description. If $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ and there is an uninfluential substitution $\theta_{PIM} = [\mathcal{C}/\mathcal{CR}_\sigma]$, $\mathcal{CR}_\sigma \subseteq \mathcal{CR}$ then $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \vdash G$ w.a. $\sigma\theta_{PIM}$.

Proof: The proof is by absurd and it uses the proof theory introduced in [11]. Let us assume that $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ but $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \not\vdash G$ w.a. $\sigma\theta_{PIM}$. Since, by hypothesis, for any substitution $[c/c_r]$ in θ_{PIM}, $\mathsf{Effs}(c) \subseteq \mathsf{Effs}(cr)$ holds, there exists a fluent F such that $a_0, a_1, \ldots, a_{i-1} \vdash F$ but $(a_0, a_1, \ldots, a_{i-1})\theta_{PIM} \not\vdash F$, where $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ and $F \in \mathsf{Precs}(a_i)$. Now, since $a_0, a_1, \ldots, a_{i-1} \vdash F$, there exists $j \leq i-1$,

[4]In the following, for the sake of readability, we will omit the indexing of the modal operator $\mathcal{B}$ when it is clear that the beliefs belong to the same role.

[5]In this work we focus on linear plans. Conditional plans can be tackled by considering each path separately.

such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in \mathsf{Effs}(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_{PIM} \not\vdash F$, that is $F \notin \mathsf{Effs}(a_j\theta_{PIM})$. This is absurd due to the hypothesis that θ_{PIM} is an uninfluential substitution. ■

Example 3: Let us refer to the running example introduced in Section III and let us consider the set of capabilities $\mathcal{C} = \{\mathsf{reserve\_room_{C1}}, \mathsf{reserve\_room_{C2}}, \mathsf{payment_C}\}$:

$\mathsf{reserve\_room_{C1}}$ **causes** $\{\mathcal{B}\neg PccSupported, \mathcal{B}price\}$
$\mathsf{reserve\_room_{C1}}$ **possible if** $\{\mathcal{B}date\}$
$\mathsf{reserve\_room_{C2}}$ **causes** $\{\mathcal{B}freeDinner, \mathcal{B}price\}$
$\mathsf{reserve\_room_{C2}}$ **possible if** $\{\mathcal{B}date\}$
$\mathsf{payment_C}$ **causes** $\{\mathcal{B}transNum, \mathcal{B}resNum\}$
$\mathsf{payment_C}$ **possible if** $\{\mathcal{B}PcashSupported, \mathcal{B}PccSupported\}$

By choosing the *plugin match* as matching rule, there are two possible substitutions called θ'_{PIM} and θ''_{PIM} respectively:

$\theta'_{PIM} = \{[\mathsf{reserve\_room_{C1}}/\mathsf{reserve\_room_{CR}}], [\mathsf{payment_C}/\ \mathsf{payment_{CR}}]\}$,
$\theta''_{PIM} = \{[\mathsf{reserve\_room_{C2}}/\mathsf{reserve\_room_{CR}}], [\mathsf{payment_C}/\ \mathsf{payment_{CR}}]\}$.

While $\mathsf{payment_C}$ exactly matches $\mathsf{payment_{CR}}$, $\mathsf{reserve\_room_{C1}}$ and $\mathsf{reserve\_room_{C2}}$ slightly differ from the requirement. By applying the substitution θ'_{PIM} we obtain the set of policies $\mathcal{P}\theta'_{PIM}$:

$\mathsf{booking}$ **is** $\mathsf{receive\_date}(s, b, date), \mathsf{reserve\_room_{C1}}, \mathsf{inform}(s, b, price), \mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc]), \mathsf{finalize\_reservation}$
$\mathsf{finalize\_reservation}$ **is** $\mathcal{B}no\_business?$
$\mathsf{finalize\_reservation}$ **is** $\mathsf{payment_C}, \mathsf{inform}(s, b, resNum), \mathsf{inform}(s, b, transNum)$

Differently than in Example 2, by using the resulting policies, the query $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle, S_0) \vdash G$ does not succeed: in fact, the additional effect $\mathcal{B}\neg PccSupported$ of the capability $\mathsf{reserve\_room_{C1}}$ inhibits the executability of the capability $\mathsf{payment_C}$. On the other hand, we observe that the application of the other substitution θ''_{PIM}, provides the agent with a set of policies $(\mathcal{P}\theta''_{PIM})$ that allows to satisfy the query $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle, S_0) \vdash G$. Thus, θ''_{PIM} represents an *uninfluential substitution.* ■

The verification that a substitution is uninfluential involves the derivation σ, and it is based on checking whether the chains of dependencies between actions for the various fluents are not interrupted by some opposite fluent. Obviously, if the domain is such that no fluent, once asserted, can be negated, any θ_{PIM} will be conservative. This can be verified statically on the choreography and the set of capabilities, by checking that every fluent (that appears as effect of some action) is always positive or negative, including the initial state and the goal in the verification. Indeed, the application domains in which actions produce *knowledge* are of this kind. One example is given by e-learning applications where the capabilities supply knowledge elements that are either supplied or used as prerequisites.

V. Conclusions and related works

In this work we have studied the relation between the matchmaking and the achievement of a goal in an interaction ruled by a choreography. We have proved that local matches (but the exact match) do not preserve the goal when capabilities are substituted to capability requirements. It is necessary to introduce a verification that involves the choreography definition. We argue that the more relaxed are the local matches, the stricter must be the the global verification. As an example, we have presented the integrated approach in the case for the plugin match.

In the agent framework, the adoption of an interaction policy has been proposed in CooBDI and Coo-AgentSpeak [14], [15]. These works extend the BDI (*Belief, Desire, Intention*) model in such a way that agents are enabled to exchange plans. This mechanism is activated when the agent cannot find a plan, for pursuing a goal of interest, by just exploiting its own capabilities. The ideas behind the CooBDI theory have been implemented by means of web services technologies, leading [16] to the development of CooWS agents. Another recent work is the one by [17]. Here, in the setting of the DALI language, agents can cooperate by exchanging sets of rule that can either define a procedure, or constitute a module for coping with some situation, or be just a segment of a knowledge base. Agents have reasoning techniques that enable them to evaluate how useful the new knowledge is. Nevertheless, these techniques cannot be directly imported in the context of service-oriented computing. The reason is that, while in agent systems it is not a problem to discover during the interaction that an agent does not own all the necessary actions, in service composition it is necessary that all the actors are known before the interaction takes place.

In [18] (inspired by JACK [19] and extended in [20]), the term “capability” is used for identifying the “ability to react rationally towards achieving a particular goal” in the BDI framework. An agent has the capability to achieve a goal if its plan library contains a plan for reaching the goal. Therefore, an agent’s goals and intentions are constrained to be compatible with its capabilities.

For what concerns (web) services and matchmaking, it is not easy to be exhaustive. The matches proposed in [8] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [9] propose four degrees of match (exact, plugin, subsumes, and fail) that are computed on the ontological relations of the outputs of an advertisement for a service and a query.

WSMO (Web Service Modeling Ontology) [10] is an organizational framework for semantic web services. As such, it does not suggest a specific matching rule, which is up to the specific implementations. However, the authors propose in [21] an approach that is based on [8] and on [22], which, in turn, is based upon [9]. More recently, a WSMO matchmaker has been proposed in [23], which combines several aspects: type matching, relation matching, constraint matching, parameter matching, intentional matching. Last but not least, in [7] a multi-level evaluation model is proposed, for deciding whether two services are composable. This is done through four levels of control (quality, dynamic semantics, static semantics, and syntax). Dynamic semantics is the name given to the matches of [8]. None of these approaches relates the matching with the possible context of application of the sought services, even WSMO which, as a framework, includes the possibility of composing orchestrations of services. On the other hand, so far we have not yet tackled the integration of ontological reasoning in our work. This is surely an interesting extension that we will face soon, given that all these proposals as well as ours have the same kernel, and we expect similar results.

The idea of synthesizing a policy from an abstract specification (a choreography) is also stated in [24], where it is observed that services are often conceived so as to be delivered individually, while there is a growing need of reusing this software, either by composing services or by tailoring a composition to some specific client. In [25] a tool for service (activity in the paper) coordination and evaluation is introduced, based on the MetaFrame open tool coordination environment. Differently than in our approach, there is no specification of a choreography as we have used here but the desired behavior is given in terms of global constraints. Temporal logic is used to express both the constraints and the goal to achieve, enabling the automatic synthesis of a composition of activities.

Finally, works like [26], [27] propose approaches for goal-driven service composition based on planning. However, this task is accomplished without reference to any choreography. In particular, in [26] the composition phase and the semantic reasoning phase (carried on on inputs and outputs) are separated and the latter is performed on a local basis only.

ACKNOWLEDGMENT

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. Claudio Schifanella is partially supported by the fellowship program "Fondazione CRT - Progetto Lagrange" (cf. http://www.progettolagrange.it).

REFERENCES

[1] B. Örriens, J. Yang, and M. Papazoglou, "Model driven service composition," in *ICSOC 2003*, 2003.

[2] M. P. Huget and J. Koning, "Interaction Protocol Engineering," in *Communication in Multiagent Systems*, ser. LNAI 2650. Springer, 2003, pp. 179–193.

[3] M. Kolp, P. Giorgini, and J. Mylopoulos, "Multi-agent architectures as organizational structures," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 1, 2006.

[4] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Reasoning on choreographies and capability requirements," *International Journal of Business Process Integration and Management*, 2007, to appear.

[5] M. Singh and M. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley and sons, Ltd., 2005.

[6] WS-CDL, "http://www.w3.org/tr/ws-cdl-10/."

[7] B. Medjahed and A. Bouguettaya, "A multilevel composability model for semantic web services," *IEEE Trans. on KDE*, vol. 17, no. 7, pp. 954–968, 2005.

[8] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Transactions on SEM*, vol. 6, no. 4, pp. 333–369, 1997.

[9] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of ISWC '02*. Springer, 2002, pp. 333–347.

[10] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres, *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer.

[11] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *AMAI*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443

[12] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction protocols for customizing web service selection and composition," *JLAP, special issue on Web Services and Formal Methods*, vol. 70, no. 1, pp. 53–73, 2007.

[13] F. for Intelligent Physical Agents, "FIPA communicative act library specification, 2002." [Online]. Available: http://www.fipa.org/repository/aclspecs.html

[14] D. Ancona and V. Mascardi, "Coo-BDI: Extending the BDI Model with Cooperativity," in *Proc. of the 1st Declarative Agent Languages and Technologies Workshop (DALT'03), Revised Selected and Invited Papers*, J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, Eds. Springer, 2004, pp. 109–134, lNAI 2990.

[15] D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini, "Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange," in *Proc. of AAMAS 2004*. ACM press, 2004, pp. 698–705.

[16] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta, "CooWS: Adaptive BDI agents meet service-oriented computing," in *Proceedings of the Int. Conference on WWW/Internet*, 2005, pp. 205–209.

[17] A. T. S. Costantini, "Learning by knowledge exchange in logical agents," in *Proc. of WOA 2005: Dagli oggetti agli agenti, simulazione e analisi formale di sistemi complessi*, F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, Eds. Camerino, Italy: Pitagora Editrice Bologna, november 2005.

[18] L. Padgham and P. Lambrix, "Agent capabilities: Extending BDI theory," in *AAAI/IAAI*, 2000, pp. 68–73. [Online]. Available: citeseer.ist.psu.edu/625805.html

[19] P. Busetta, N. Howden, R. Ronquist, and A. Hodgson, "Structuring bdi agents in functional clusters," in *Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL99)*, 1999.

[20] V. Padmanabhan, G. Governatori, and A. Sattar, "Actions made explicit in BDI," in *Advances in Artificial Intelligence*, ser. LNCS, no. 2256. Springer, 2001, pp. 390–401.

[21] U. Keller, R. L. A. Polleres, I. Toma, M. Kifer, and D. Fensel, "D5.1 v0.1 wsmo web service discovery," WSML deliverable, Tech. Rep., 2004.

[22] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic technology," in *Proc. of WWW Conference*. ACM Press, 2003.

[23] F. Kaufer and M. Klusch, "WSMO-MX: A logic programming based hybrid service matchmaker," in *Proc. of ECOWS'06*. IEEE Computer Society, 2006, pp. 161–170.

[24] F. Casati and M. Chien, "Dynamic and adaptive composition of e-services," *Information Systems*, vol. 26, pp. 143–163, 2001.

[25] B. Steffen, T. Margaria, and V. Braun, "The electronic tool integration platform: Concepts and design." *STTT*, vol. 1, no. 1-2, pp. 9–30, 1997.

[26] M. Pistore, L. Spalazzi, and P. Traverso, "A minimalist approach to semantic annotations for web processes compositions." in *ESWC*, 2006, pp. 620–634.

[27] 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," in *Web Intelligence*. Springer, 2003.