Goal Preservation by Choreography-Driven Matchmaking*

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. In this work we give a formal background and identify the limits of applicability of local matching criteria (among which the well-known Zaremski and Wings's plugin-match) when they are used to automatically retrieve all the capabilities that are necessary to instantiate a given choreography. In doing this it is necessary to take into account and somehow merge two possibly conflicting perspectives: the local criterion for selecting single capabilities and the overall goal that we mean the composition to pursue. Formally, the problem is interpreted as the study of the preservation of the global properties of a choreography, when its roles are played by specific services.

1 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 investigated, in the literature, for instance in [16], 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. 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" [10].

The introduction of the concept of "choreography" (and of languages like WS-CDL [21]) has opened new perspectives on the way an abstract specification should be given, but the idea of using an abstract specification as a model for guiding the selection and composition of services is still embryonic. In this work, we study different kinds of match, that have been proposed for web service discovery, in the perspective of using them for retrieving software components that

* This research has partially been funded by the 6th Framework Programme project REWERSE number 506779 and by the Italian MIUR PRIN 2005. Claudio Schifanella is partially supported by "Fondazione CRT - Progetto Lagrange".

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 413–426, 2009.

are to be assembled in the context given by a choreography, assuming that all the parties share a common communication language. More precisely, the role description is used as a schema of the desired implementation, for producing a sort of *policy skeleton*; the adoption of the schema guarantees *a priori* the interoperability with the other parties. A similar approach has been adopted, in the past, for synthesizing agent behaviors from UML specifications [1, Section 8.4]. As we have shown in [3], roles in a choreography must specify not only the interaction schema, that intuitively defines the interoperability conditions for the potential role players, but also the capabilities requested for playing the role. Thus, it is important to enrich choreographies by specifying *capability requirements*. A capability requirement expresses an operation, that a peer should be able to perform at some specific point of the choreography[1]. Once defined such requirements, the policy skeleton produced from the choreography can be completed by substituting *capabilities* to capability requirements, so to make it executable. In other words, in order for the role to be "playable" by the peer, the peer must have the requested capabilities. For instance, it must have some means for producing or retrieving a piece of information to send to a partner. The specific implementation does not matter as long as the requirement is fulfilled. The selection of those capabilities of the candidate player that will substitute the capability requirements requires a matching process.

The task of retrieving capabilities that match given requirements is analogous to the task of service discovery. Therefore, it is straightforward to think to use the same techniques, like for instance [8,15,22]. In particular, in this work we focus on the matches proposed by Zaremski and Wing in their seminal work [22], where various kinds of relaxed match, that capture the notions of generalization, specialization, and substitutability of software components, are proposed. The specification is given in terms of pre- and post-conditions, written as predicates in first-order logic. These matches are at the basis of many proposals, both for software component match and for web service match [12,14,15,17]. The idea of working on a specification of software components (although not yet on pre- and post-conditions) can be found also in earlier works, such as [9], where a compositional method for the automatic analysis and verification in the area of "transition systems" is given, in which interface specifications are used to express context constraints between two processes.

Specifically, we consider the case when a role is adopted because it allows the achievement of a goal of interest. The *decision* is taken after a reasoning process applied to the specification. Intuitively, a role will be adopted if it allows an execution that leads to the goal [5,13]. If, on a hand, the decision is taken on the role description, on the other hand, the actual execution will involve the policy for the specific peer, i.e. an instantiation of the role in which capability requirements have been substituted by matching capabilities. The natural question is: "will the policy still allow to reach the goal?". If the selected capabilities match in an exact way with the requirements, the obvious answer is *yes*. However, it is unlikely to have capabilities that perfectly match the requirements and many kinds of match

[1] In [3] an extension of WS-CDL with capability requirements is presented.

that have been proposed in the literature are in some way partial. We show that none of the partial matches in [22] guarantee that the policy, obtained by performing the substitution, will allow to reach the goal of interest (Theorem 1, Section 4). The reason is that, by their nature, 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 (Theorem 2, Section 4).

The article is organized as follows. In Section 2 we recall the matches introduced in [22], and explain their relations with a choreography and with the goals. Section 3 introduces a simple declarative representation for services and choreographies. Section 4 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*. An example of a simple room reservation protocol is presented in Section 5. Conclusions and related works end the paper.

2 Capability Match: Local vs. Global Properties

We suppose, in the line of previous work [3], 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. This selection can be done by applying matching techniques that are analogous to those used for service discovery. Zaremski and Wing [22] propose a formal specification to describe the behavior of software components. Each software component has precondition S_{pre} and postcondition S_{post}. 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 [22], 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

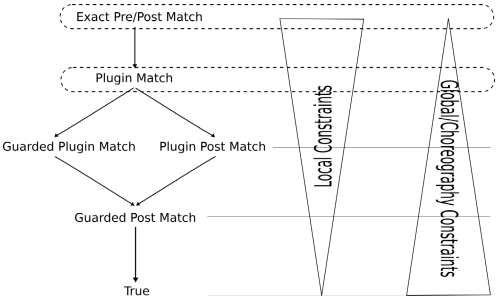

Fig. 1. The lattice of the different local matches: on top the strongest. Our claim is that the local and global constraints are related; the stronger the local match, the weaker the global constraints.

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 4, 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.

3 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 [5], 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 [2]. 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 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 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. DyLOG complex behaviors are specified by means of *procedures*, Prolog-like clauses built upon the other kinds of action. We represent the behavior of both *roles* and *policies* by DyLOG procedures [2]. Intuitively, a

[2] 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.

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. $\mathcal{S}_\mathcal{A}$ is a set of *speech acts*, represented as:

   $\mathsf{performative}(sender, receiver, l)$ **causes** $\{E_1, \ldots, E_n\}$
   $\mathsf{performative}(sender, receiver, l)$ **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 $\mathsf{receive\_act}$; 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.

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 previous 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.

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.

4 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 is the initial state, and G is 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. θ is a substitution $\mathcal{CR}_\sigma \rightarrow \mathcal{C}$, where $\mathcal{CR}_\sigma \subseteq \mathcal{CR}$, $\mathcal{CR}_\sigma = \{cr \in \mathcal{CR} \mid cr$ 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 2, 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 $\theta_{EM}, \theta_{PIM}, \theta_{POM}, \theta_{GPIM}, \theta_{GPOM}$, 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. Proof by counterexample. Let us consider a role description $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$, where $\mathcal{P} = \{p \ \textbf{is} \ cr_1, a\}$, $\mathcal{S}_\mathcal{A} = \{a\}$, $\mathcal{G}_\mathcal{A}$ is empty, and the capability requirement cr_1 in $\mathcal{CR}$ and the speech act a in $\mathcal{S}_\mathcal{A}$ are described by [3]:

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

Assuming as goal $G = \mathcal{B}l_2 \ \textbf{after} \ p$, 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 $\mathcal{C} = \{c_1\}$:

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

By applying the substitution $\theta = \{[c_1/cr_1]\}$ we obtain the new policy $\mathcal{P}\theta = \{p \ \textbf{is} \ c_1, a\}$. However, by using this policy, the query $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \vdash G$ 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, 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.

[3] For the sake of readability, we will omit the indexing of modal operator $\mathcal{B}$ when the entity that owns the belief is clear from the context.

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 $R_d = \langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{CR}, \mathcal{P} \rangle$ 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.[4] 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.

[4] In the following we focus on linear plans. Conditional plans can be tackled by considering each path separately.

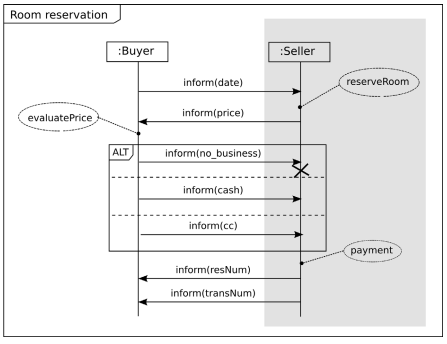

Fig. 2. The Room Reservation Protocol, represented by means of UML sequence diagrams, and enriched with capability requirements (oval elements)

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 [5]. Let us assume that $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{CR}, \mathcal{P} \rangle, S_0) \vdash G$ w.a. σ but $(\langle \mathcal{S_A}, \mathcal{G_A}, \mathcal{C}, \mathcal{P}\theta_{PIM} \rangle, S_0) \nvdash G$ 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 $(a_0, a_1, \ldots, a_{i-1})\theta_{PIM} \nvdash F$, 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 $j \leq i-1$, such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in \mathsf{Effs}(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_{PIM} \nvdash F$, that is $F \notin \mathsf{Effs}(a_j\theta_{PIM})$. This is absurd due to the hypothesis that θ_{PIM} is an uninfluential substitution. □

The verification that a substitution is uninfluential involves the derivation σ. 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 (which is an 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 competencies that are either supplied or used as prerequisites.

5 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 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 [5] $R_{seller} = \langle \mathcal{S}_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}}, \mathcal{CR}, \mathcal{P} \rangle$, 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** $\mathsf{receive\_date}(s, b, date), \mathsf{reserve\_room}_{\mathsf{CR}}, \mathsf{inform}(s, b, price),$
$\mathsf{receive\_evaluation}(s, b, [no\_business, cash, cc]), \mathsf{finalize\_reservation}$
$\mathsf{finalize\_reservation}$ **is** $\mathcal{B} no\_business?$
$\mathsf{finalize\_reservation}$ **is** $\mathsf{payment}_{\mathsf{CR}}, \mathsf{inform}(s, b, resNum), \mathsf{inform}(s, b, transNum)$

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

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

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 [2]):

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

Let us consider the goal $G = \{\mathcal{B} transNum, \mathcal{B} resNum\}$ **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. Among the two possible execution traces, successfully leading to a state where G holds, let us focus on the trace:

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

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

Let's 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** $\mathsf{payment_C}, \mathsf{inform}(s, b, resNum), \mathsf{inform}(s, b, transNum)$

However, by using the resulting policies, the query $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle, S_0) \vdash G$ 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 $(\langle \mathcal{S}_\mathcal{A}, \mathcal{G}_\mathcal{A}, \mathcal{C}, \mathcal{P}\theta'_{PIM}\rangle,\ S_0) \vdash G$. As introduced in the Section 4, θ''_{PIM} represents a *conservative substitution.*

6 Conclusions and Related Works

In this work we have studied the relation between matchmaking and goal achievement in a choreography, showing that local matches generally 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. The goals that we consider are *existential*, i.e. they allow to verify the existence of an execution trace with given properties. We do not consider *universal* properties, e.g. "never a server ends with an incorrect state". Examples of the use of DyLOG can be found in [2]. In [19,4] we show how the approach presented in this paper can be used in the process of selecting a service, which can play the role of interlocutor in a given choreography, preserving at the same time the satisfiability of a goal of interest.

The matches proposed in [22] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [17] 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) [8] 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 [12] an approach that is based on [22] and on [14], which, in turn, is based upon [17]. More recently, a WSMO matchmaker has been proposed in [11], which combines several aspects: type matching, relation matching, constraint matching, parameter matching, intentional matching. Last but not least, in [15] 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 [22]. 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 [7], 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 [20] a tool for service (activity in the paper) coordination and evaluation, based on the MetaFrame open tool coordination environment, is introduced. 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 [18,6] propose approaches for goal-driven service composition based on planning. However, this task is accomplished without reference to any choreography. In particular, in [18] 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.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer, Heidelberg (2004)
2. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Reasoning about interaction protocols for customizing web service selection and composition. JLAP, special issue on Web Services and Formal Methods 70(1), 53–73 (2007)
3. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Reasoning on choreographies and capability requirements. International Journal of Business Process Integration and Management 2(4) (2007) (in press)

4. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Service selection by choreography-driven matching. In: Proc. of the 2nd ECOWS Workshop WEWST 2007, January 2008. CEUR, vol. 313, pp. 1–17 (2008)
5. Baldoni, M., Giordano, L., Martelli, A., Patti, V.: Programming Rational Agents in a Modal Action Logic. AMAI 41(2-4), 207–257 (2004)
6. Bryson, J., Martin, D., McIlraith, S., Stein, L.A.: Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web. In: Web Intelligence. Springer, Heidelberg (2003)
7. Casati, F., Chien, M.C.: Dynamic and adaptive composition of e-services. Information Systems 26, 143–163 (2001)
8. Fensel, D., Polleres, A., Lausen, H., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, Heidelberg (2007)
9. Graf, S., Steffen, B.: Compositional minimization of finite state systems. In: Clarke, E., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 186–196. Springer, Heidelberg (1991)
10. Huget, M.-P., Koning, J.-L.: Interaction protocol engineering. In: Huget, M.-P. (ed.) Communication in Multiagent Systems. LNCS (LNAI), vol. 2650, pp. 179–193. Springer, Heidelberg (2003)
11. Kaufer, F., Klusch, M.: Wsmo-mx: A logic programming based hybrid service matchmaker. In: Proc. of ECOWS 2006, pp. 161–170. IEEE Comp. Soc., Los Alamitos (2006)
12. Keller, U., Laraand, R., Polleres, A., Toma, I., Kifer, M., Fensel, D.: D5.1 v0.1 WSMO web service discovery. Technical report, WSML deliverable (2004)
13. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for logic domains. JLP 31, 59–83 (1997)
14. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic technology. In: Proc. of WWW Conference. ACM Press, New York (2003)
15. Medjahed, B., Bouguettaya, A.: A multilevel composability model for semantic web services. IEEE Trans. on KDE 17(7), 954–968 (2005)
16. Örrens, B., Yang, J., Papazoglou, M.P.: Model driven service composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)
17. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: Proc. of ISWC 2002, pp. 333–347. Springer, Heidelberg (2002)
18. Pistore, M., Spalazzi, L., Traverso, P.: A minimalist approach to semantic annotations for web processes compositions. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 620–634. Springer, Heidelberg (2006)
19. Schifanella, C.: Reasoning on Web Services with Choreographies and Capabilities. PhD thesis, Dip. Informatica, Università degli Studi di Torino, Italy (2008)
20. Steffen, B., Margaria, T., Braun, V.: The electronic tool integration platform: Concepts and design. STTT 1(1-2), 9–30 (1997)
21. WS-CDL (2005), `http://www.w3.org/tr/ws-cdl-10/`
22. Moormann Zaremski, A., Wing, J.M.: Specification matching of software components. ACM Transactions on SEM 6(4), 333–369 (1997)