Towards Extending Service Discovery with Automated Composition Capabilities*

Roman Vaculín¹, Roman Neruda¹, Katia Sycara²

¹ Institute of Computer Science, Academy of Sciences of the Czech Republic
(\{vaculin,roman\}@cs.cas.cz)
² The Robotics Institute, Carnegie Mellon University (katia@cs.cmu.edu)

Abstract

In most of research, service composition and discovery are treated separately. In the composition, it is assumed that primitive services are known to the composition component in advance. At the same time, in matchmaking algorithms, usually only one service is considered as a suitable candidate satisfying a query while service combinations are not allowed. The problem is that in realistic composition or mediation scenarios in dynamic environments some specific gap can be bridged only by using a combination of several services which are not known and need to be discovered. We consider such a situation and we propose an approach using a combined match and a composed match. We give a characterization of these two matching types and we develop appropriate matching algorithms.

1 Introduction

An automated discovery and composition of web services are two active research fields which both aim at a similar goal, which is to enable a higher level of flexibility of WS-based systems operating in dynamic changing environments. Even when sharing the same goal, the constraints of and requirements for composition and discovery algorithms are somewhat different. Therefore, traditionally, discovery and composition are treated separately. In composition algorithms [11, 14, 1, 9], it is assumed that primitive services are known to the composition component in advance. At the same time, in matchmaking algorithms [10, 12, 6, 2] used in discovery components, typically only one service is considered as a suitable candidate satisfying a service request while service combinations are not allowed. We argue that while the basic division of concerns between composition and discovery components makes a perfect sense for obvious reasons, in open environments the strict assumptions on both sides lead to serious problems.

In many scenarios, especially in closed or semi-closed enterprise networks, the composition component can be assumed to know all needed services. However, this is clearly not the case in open environments, such as internet, in which the services need to be discovered before or during the composition process. In situations like that, services need to be discovered based on needs identified by the composition component. For example, during the composition a "gap" might be identified [7] which cannot be bridged by using any of the known services. To bridge such a gap, new services need to be found. This introduces a problem, because oftentimes some specific gap cannot be bridged by any single service and a combination of several services must be used. However, as noted before, the vast majority of discovery services implement a matching algorithm that does not allow service combinations to be matched against a service request.

To deal with the problem the matching assumptions need to be relaxed to allow a combination of several services as an acceptable match for a given service request. This has to be done carefully though, since allowing combinations of services can lead to efficiency problems, as identified in [3] where Benatallah et. al. show that finding an optimal combination of services covering the request is NP-hard. In this paper, we explore a similar direction by allowing the combination of services satisfying the request to be returned as a relevant match — we call it a combined match. In combined match we do not strictly insist on optimality in order to prevent hard computations. We prefer the coverage instead, since we assume that, if needed, the composition algorithms can find the optimal combination in next steps after discovery is done. Thus, the main goal of a composed match is to support service combinations to be matched with as little time impairment as possible. We also suggest a composed match as a more computationally intensive alternative. In essence, while the combined match finds a set of services...
that can produce requested outputs and effects without allowing chaining of services, the composed match supports chaining which allows proper service compositions to be found. We give characterizations of both the combined and the composed match building mainly on OWL-S concepts of inputs, outputs, preconditions and effects. We extend the basic matching algorithm and the indexing schema introduced in [10] to support combined and composed match and we show that under usual circumstances the combined match can be computed in the about the same time as the basic single service match. Finally, we devise an iterative deepening algorithm which starts with the combined match and, depending on the available time, it either returns an incomplete composed match in which some inputs and preconditions might be missing, or it eventually returns a complete composed match. The presented algorithm computes the best possible match in a given time constraint.

The paper is structured as follows. In Section 2, we motivate the problem by an example from the process mediation domain. In Section 3, we introduce the basic terminology and definitions. In Section 4, we discuss the matching conditions of individual services. In Section 5, we extend the characterization to the combined and composed match. Section 6 introduces the matching algorithms. In Section 7, we present preliminary results, and in Sections 8 and 9 we discuss the related work and conclusions.

2 Motivating Example

We demonstrate the motivation for matching sets of services on the problem of process mediation where it arises very naturally. In the process mediation, the goal is to achieve interoperability of two or more possibly incompatible process models. Ideally, this should be done in an automated or semi-automated fashion. Figure 1 presents an example of the mediation problem between a hypothetical requester and provider from the flights booking domain. Specifically, Figure 1a depicts a fragment of the process model of the requester while the provider’s process model fragment depicted in Figure 1b represents a more elaborate scenario that allows the requester to book either the whole itinerary or to pick the departure and return flights separately.

The requester’s process model starts with the Login atomic process that has two inputs, ?userId which is an instance of the UserID class and ?password of Password type, one output ?logResult of boolean type and a conditional effect expressing that the predicate LoggedIn(?userId) will become true if the value of ?logResult equals to true. Similarly the process continues by executing other atomic processes. Input and output types used in process models refer to a simple ontology showed in Figure 2.

Dashed arrows between parts (a) and (b) of Figure 1 represent symbolically possible mappings between requester’s and provider’s process models that can be found by the process mediation algorithm (see [15] for details about the process mediation). Sometimes, the mapping can be achieved without the use of any help of other services, such as in the case of requester’s Login atomic process. However, often the identified data incompatibilities or missing pieces of information require external services to be used in order to construct a meaningful mapping between process models. Consider, for example, the requester’s SearchFlight atomic process and the provider’s SearchFlightOne process. The requester has the inputs \( I_{\text{SearchFlight}} = \{ (?\text{from}, \text{FromCity}), (?\text{to}, \text{ToCity}), (?\text{depTime}, \text{USDepTime}), (?\text{retTime}, \text{USRetTime}) \} \), while the provider expects the inputs \( I_{\text{SearchFlightOne}} = \{ (?\text{from}, \text{AirportFromCode}), (?\text{to}, \text{AirportToCode}), (?\text{depTime}, \text{ISODepTime}), (?\text{retTime}, \text{ISORetTime}) \} \). The mapping cannot be constructed directly, since the input types of processes do not match. Differences between \( I_{\text{SearchFlight}} \) and \( I_{\text{SearchFlightOne}} \) define an information gap that can be used to construct a query for the discovery service. An ideal service which would bridge the identified gap has to consume \( I_{\text{SearchFlight}} \) as its inputs and produce \( I_{\text{SearchFlightOne}} \) as its outputs. Clearly, it is extremely unlikely that there would ever exist one single service satisfying such a requirement. However, if combinations of services are allowed to be matched, the chances of a successful match are much higher. In our particular case, a combination of external services AirportCity-ToCode and USTimeToISO can be used as a match bridging the gap as shown in Figure 1. After the discovery service returns such a set of services as a valid match, the process mediation component uses a composition algorithm to construct the sought mapping by employing the newly discovered services.

An important fact to notice is that even in our relatively simple example the need for some kind of match allowing service combinations arises. We believe that such a need is universal for almost any composition scenario which has to be realized in open changing environments.

3 Preliminaries

For characterization of services we use OWL-S [13] which is a semantic web services description language. OWL-S covers three areas: the Service Profile describes what the service does in terms of its capabilities and is used for discovery purposes; the Process Model specifies ways of how clients can interact with the service; the Grounding links the process model to the specific execution infrastructure (e.g., maps processes to WSDL operations and allows for sending messages in SOAP). The elementary unit of the Process Model is an atomic process, which represents one
indivisible operation that the client can perform by sending a particular message to the service and receiving a corresponding response. Processes are specified by means of their inputs, outputs, preconditions, and effects (IOPEs).

The Service Profile supports description of several perspectives of the service which are relevant for the discovery purposes. Most importantly, it describes the service from the functional perspective by means of its IOPEs in a similar fashion as processes are defined. The Service Profile is used as an advertisement when publishing the service capabilities to the discovery service. On the other hand, a service request describes required capabilities of a searched service.

For purposes of this paper we consider only inputs, outputs, preconditions and effects in service advertisements. A service advertisement $A$ is a tuple $A = \langle I, O, P, E \rangle$, where $I$ and $O$ are sets of typed input and output parameters of the advertised service, i.e., $I, O = \{ (?v, T) \mid ?v \in Var, T \in Types \}$, and $P$ and $E$ are sets of preconditions and effects respectively. $Var$ is a set of input and output names (we assume each input and output name to be unique) and $Types$ is a set of types (either primitive XSD types or OWL classes defined in some ontology). We use $Var_I = \{ ?v \mid (?v, .) \in I \}$ and $Var_O = \{ ?v \mid (?v, .) \in O \}$ to denote the set of input names and output names respectively. $Var_{Free}$ is used to denote free variables which are not used in $Var_I$ or $Var_O$, i.e. $Var_{Free} \cap (Var_I \cup Var_O) = \emptyset$.

Similarly, we define a basic service request $R$ as a tuple $R = \langle I, O, E \rangle$, where $I$ and $O$ are sets of typed input and output parameters of the requested service, $E$ is a set of required effects. The set of inputs $I$ contains parameters the requester has readily available and wants them to be used by the requested service in order to produce requested outputs $O$ and effects $E$.

We represent preconditions and effects as expressions in the form of conjunction of description logic atoms enriched with OWL datatypes. An expression is a conjunction of description logic atoms enriched with OWL datatypes. An atom can be one of the following expressions: $C(s)$ (concept atom), $Po(s,t)$ (object property atom), $Pd(s,d)$ (datatype property atom), where $C$ is an OWL class name, $Po$ is an OWL object property, $Pd$ is an OWL datatype property, $s$ and $t$ are variables or OWL individuals and $d$ is a variable or an OWL data value. In preconditions variables from $Var_I \cup Var_{Free}$ can be used, while in effects variables from $Var_I \cup Var_O \cup Var_{Free}$ are allowed.

To better support matchmaking in contexts such as com-
position and process mediation, we introduce a notion of the requester’s state in the service request. The purpose of the requester’s state is to provide more information to the matchmaker about the state of the world as seen by the requester. Specifically, the requester’s state might contain set of valid facts about the world in the time when the request is made, and the set of additional available data which might be possibly used by the requested service. Thus, the requester’s state is a tuple \( S = (F,D) \), where \( F \) is a set of valid facts as seen by the requester, and \( D \) is a set of additional typed data that the requester has available.

We define a state enhanced service request by adding the requester’s state to the basic service request, i.e., the state enhanced service request is a tuple \( R = (I,O,E,S) \), where \( S \) is the requester’s state.

**Example:** Considering the situation of the requester in Figure 1 for the SearchFlight atomic process, the requester’s state could look like as follows:

\[
S_{\text{SearchFlight}} = \{(\text{LoggedIn}(\text{userId}))\}
\]

\[
F = \{\text{(?userId,\text{sessionID}), (?password, Password)}, \text{(?logResult, boolean), (Session\_ID, Session\_ID)}\}
\]

The requester’s state \( S_{\text{SearchFlight}} \) captures that, as a result of some previous actions, the fact \( \text{LoggedIn}(\text{userId}) \) is valid, and that some more data, such as \( \text{(?sessionID,Session\_ID)} \), is available to the requester. Such an information can be employed by the matchmaker to make a more informed match.

## 4 Individual Services Matching

The matchmaking problem for a single service can be formulated as follows: given the service request \( R \) and a set of published service advertisements \( \text{Advertisements} \), find the set \( \text{Match} \) such that each \( A \in \text{Match} \) represents a service capable of satisfying the request \( R \). The matching algorithm should be flexible, efficient and should minimize false positives and false negatives. In [10] the ability of service to satisfy a request is defined by means of relations between input and output types of advertisements and requests. A service can satisfy a request, if it produces at least all requested outputs and does not need any other inputs than those provided by the requester. The following definition gives a precise characterization of match for a basic service request and the OWL-S service advertisement.

**Definition 1 (IOs Matching Conditions for Individual Services):** For a basic request \( R = (I_R,O_R,E_R) \) and a service advertisement \( A = (I_A,O_A,P_A,E_A) \) the following conditions need to hold for \( A \) to satisfy \( R \):

1. \( \forall (\text{?}o_r,T_o) \in O_R \exists (\text{?}a, T_{o_a}) \in O_A \text{ such that } T_{o_a} \approx T_{o} \)
2. \( \forall e_r \in E_R \exists e_a \in E_A \text{ such that } S_R \models e_a \Rightarrow e_r \)
3. \( \forall (\text{?}i_r,T_i) \in I_R \exists (\text{?}i_a,T_{i_a}) \in I_A \text{ such that } T_{i_a} \approx T_i \)
4. \( \forall p_a \in P_A \models S_R \models p_a \)

Preconditions and effects evaluation in the IOs matching conditions relies on knowing the requester’s state \( S_R \). In particular, preconditions cannot be properly evaluated without knowing \( S_R \) (only tautologies would satisfy the condition 4 without knowing \( S_R \), which is not very useful). For effects, the situation is quite different. In case of not knowing \( S_R \), we can assume \( S_R \) to be empty in which case the condition 2 transforms into the form

Relation \( \approx \) defines a match between two types. We say that there is a match between two types \( T_{o_a} \) and \( T_{o_a} \) if one of the following conditions hold: (1) \( T_{o_a} = T_{o_a} \) or \( T_{o_a} \text{ subclassOf } T_{o_a} \) (exact match); (2) \( T_{o_a} \text{ subsume } T_{o_a} \) (plug in match); (3) \( T_{o_a} \text{ subsume } T_{o_a} \) (subsume match). We say that there is no match (or match failed) if none of the conditions holds for two types.

The three possible conditions of the \( \approx \) relation can be seen as degrees of match with the decreasing quality. An exact match is the most preferable, followed by the plug in match and subsume match. In [10] the characterization of match from Definition 2 is used for computing the overall degree of match between the request \( R \) and an advertisement \( A \) and also for computing the set \( \text{Match} \) ordered according to the degree of match.

The original work in [10] does not take effects and preconditions into account. Adding effects is actually quite easy. Naturally, in terms of effects, a service is able to satisfy a given request if it can produce such a set of effects that implies at least all requested effects.

Speaking of preconditions, the advertised service should be matched against the request, only when the requester is able to satisfy all preconditions of this services. Otherwise, it would not make a sense to match a service since the requester would not be able to execute it because of failed preconditions. The problem with preconditions evaluation in the matchmaker is that in order to evaluate the preconditions, the provider has to provide the matchmaker with its state (the requester’s state). This might be a problematic issue in many scenarios which maybe explains why service preconditions are very often completely ignored during the discovery.

The following definition summarizes our short discussion and gives the matching conditions for an individual service including preconditions and effects.

**Definition 2 (IOPEs Matching Conditions for Individual Services):** For a state enhanced request \( R = (I_R,O_R,E_R,S_R) \), and a service advertisement \( A = (I_A,O_A,P_A,E_A) \) the following conditions need to hold for \( A \) to satisfy \( R \):

1. \( \forall (\text{?}o_r,T_{o_a}) \in O_R \exists (\text{?}a, T_{o_a}) \in O_A \text{ such that } T_{o_a} \approx T_{o} \)
2. \( \forall e_r \in E_R \exists e_a \in E_A \text{ such that } S_R \models e_a \Rightarrow e_r \)
3. \( \forall (\text{?}i_r,T_i) \in I_R \exists (\text{?}i_a,T_{i_a}) \in I_A \text{ such that } T_{i_a} \approx T_i \)
4. \( \forall p_a \in P_A \models S_R \models p_a \)

Preconditions and effects evaluation in the IOPEs matching conditions relies on knowing the requester’s state \( S_R \). In particular, preconditions cannot be properly evaluated without knowing \( S_R \) (only tautologies would satisfy the condition 4 without knowing \( S_R \), which is not very useful). For effects, the situation is quite different. In case of not knowing \( S_R \), we can assume \( S_R \) to be empty in which case the condition 2 transforms into the form
2'. \( \forall e_r \in E_R \ \exists e_a \in E_A \) such that \( e_a \Rightarrow e_r \). This form allows us to derive useful conclusions even without knowing the requester’s state as we show in Section 6. For example, consider the Login atomic service and the LoginStep2 in Figure 1. Login requires the LoggedIn(?userID) effect and the LoginStep2 produces LoggedIn(?user). After variables unification condition 2' holds for these two services.

Knowing the \( S_R \) in the matchmaker also allows a slight modification of the condition 3 for inputs compared to the Definition 1 (condition 2). Specifically, the inputs required by the advertised service can either be taken from inputs specified in the \( I_R \) of the request (which is a preferable solutions) or to be taken from the set of additional data \( D_R \).

In the following text we will often refer to the match defined in Definition 2 as to the individual service match.

5 Combined and Composed Match

In this section we give precise definitions of the combined and the composed match. The idea is to extend matching conditions for a single service as defined in the previous section so that sets of services can be considered as a valid match. Basically, a set of advertisements satisfies a service request, if together all advertisements from the set are able to produce outputs and effects required by the requester, while using only inputs specified in the request and while the preconditions of all services hold. By allowing sets of advertisements, new issues need to be considered.

First of all, the efficiency is a big concern. Discovery components generally, should avoid heavy computations so that a timely response to the requester can be guaranteed. To solve the problem of finding the right balance between the response time and the quality of results, we define the combined match (a less computationally expensive match) and the composed match in such a way which allows the matching algorithm to start with the cheaper match and possibly improve it if enough time is available. Thus, the requester can influence the quality of result by specifying the response time.

Next, the problem of duplicate effects might arise. By this we mean a situation when two or more advertised services in one matched set claim to produce the same effect. Such a situation needs to be avoided, since an effect means a change in the real world and we assume that the requester wants to achieve a given effect only once. As an example, imagine a situation when a flight ticket would be booked twice with two different providers. For outputs, this is generally not the problem, since outputs present only new information a the duplicity should not matter. Of course there might be issues such as price for getting the information, but we believe that this is rather a problem that should be resolved by the requester and not by the matchmaker.  

In the following definitions, we assume that the requester’s state is available to the matchmaker. Such an assumption might be either unrealistic or undesirable in many scenarios. We consider also situation when the requester’s state is available, however, the definitions with the requester’s state assumption and preconditions evaluation included are more general and can be easily modified for situations when only a basic service request is considered.

We start with a combined match definition. To simplify the notation we first define the notion of an effect implied by an advertisement.

**Definition 3 (Implied Effect):** Let \( S_R \) be a requester’s state, \( e \) an effect and \( A = \langle I_A, O_A, P_A, E_A \rangle \) an advertisement. We say that \( e \) is implied by an advertisement \( A \) and write \( S_R \models A \Rightarrow e \) if \( \exists e_a \in E_A \) such that \( S_R \models e_a \Rightarrow e \).

In essence, the combined match is just a set of service advertisements which are able to produce required effects and outputs. In the combined match no service chaining is allowed. Thus, every advertisement in the combined match can use only inputs provided by the service requester. The same holds for preconditions.

**Definition 4 (Combined Match):** Let \( R = \langle I_R, O_R, E_R, S_R \rangle \) be a state enhanced request, \( S_R = \langle F_R, D_R \rangle \). We call a set of service advertisements \( M = \{ A \mid A = \langle I_A, O_A, P_A,E_A \rangle \} \) a combined match satisfying \( R \) if the following conditions hold:

1. \( \forall (i_o, T_o) \in O_R \ \exists A \in M \ \exists (o_a, T_a) \in O_A \) such that \( T_a \approx T_o \)
2. \( \forall e_r \in E_R \ \exists A \in M \) such that \( S_R \models A \Rightarrow e_r \)
3. \( \forall e_r \in E_R \ \forall i_A, A_j \in M \) \( (S_R \models A_i \Rightarrow e_r) \land (S_R \models A_j \Rightarrow e_r) \) \( A_i \approx A_j \)
4. \( \forall M \in M \exists i_A, T_i \in I_A \exists (i_r, T_r) \in (I_R \cup D_R) \) such that \( T_r \approx T_i \)
5. \( \forall A \in M \ \forall p_a \in P_A \ \exists p_a \) \( S_R \models p_a \)

With an exception of the condition 3, all other conditions are basically reformulations of conditions from Definition 2 in the context of the combined match. The purpose of condition 3 is to avoid duplicate effects.

Figure 3 depicts a symbolic example of service advertisements forming the combined match. Circles labeled \( A, B \) and \( C \) in the bottom part stand for inputs specified in the request \( R \). Circles labeled \( X \) and \( Y \) in the top part stand for requested outputs and effects. The combined match in Figure 3 consists of two advertisements, \( A_0 \) consuming \( A, B \) and producing \( X \), and \( A_1 \) consuming \( C \) and producing \( Y \).

The following definition introduces a composed match as a natural extension of the combined match by allowing

---

1Similarly we also do not consider the problem of unwanted effects, i.e. those effects produced by the matched services that the requester did not request. We think it is again a problem that can be better resolved by the requester than by the matchmaker.
chaining in the matched advertisement set. The idea of the composed match is to organize the matched advertisements set into layers with each layer producing outputs and effects for higher layers while depending only on lower layers or inputs provided by the requester. As a result of such a definition, the composed match allows generic service compositions without loops. We number the layers in the composed match with numbers from 0 to n, with 0 standing for the highest layer and n for the lowest layer.

Definition 5 (Composed Match): Let \( R = \langle I_R, O_R, E_R, S_R \rangle \) be a state enhanced request with \( S_R = \langle F_R, D_R \rangle \). Let \( CM = \{M_0,M_1,\ldots,M_n\}, M_i = \{A \mid A = \langle I_A, O_A, P_A, E_A \rangle \} \) set of advertisements, \( i = 1,\ldots,n \). We call CM a composed match satisfying \( R \) if the following conditions hold:

1. \( \forall (\text{?}o_r, T_o) \in O_R \ \exists A \in M_0 \ \exists (\text{?}o_a, T_o) \in O_A \) such that \( T_o \approx T_o \)
2. \( \forall e_r \in E_R \ \exists A \in M_0 \) such that \( S_R \models A \Rightarrow e_r \)
3. \( \forall e_r \in E_R \ \forall A_i \in M_k \ \forall A_j \in M_l \ \left( (S_R \models A_i \Rightarrow e_r) \land (S_R \models A_j \Rightarrow e_r) \right) \Rightarrow A_i \Rightarrow A_j \)
4. \( \forall A \in M_i \ \forall (\text{?}i_r, T_o) \in I_A \ \exists (\text{?}i_r, T_o) \in (I_R \cup D_R \cup O_A) \) such that \( T_o \approx T_o \), where \( A_j \in M_j, A_j = \langle I_A, O_A, P_A, E_A \rangle, j > i \)
5. \( \forall A \in M_i \ \forall p_a \in P_A \) either
   (a) \( S_R \models p_a \)
   (b) \( \exists A_j \in M_j, A_j = \langle I_A, O_A, P_A, E_A \rangle, j > i \) such that \( S_R \models e_j \Rightarrow p_a \)

Conditions 1 and 2 in Definition 5 express that outputs and effects of the request must be produced by the highest most layer (\( M_0 \)). Condition 3 avoids effects duplication. Condition 4 specifies that inputs of each advertisement must originate either from the requester or must be produced by advertisements in lower layers (i.e., layers with higher index). Finally, condition 5 expresses the same for preconditions.

Figure 4 presents a symbolic example of service advertisements forming the composed match with 3 layers. As can be seen, it fulfills all the constraints of the combined match since every layer depends only on lower layers or inputs from the request.

Let us mention relations between the combined and the composed match and the individual service match. The composed match can be seen as a special case of the composed match with one layer only. Also, each layer in the composed match can be seen as a combined match which corresponds to an imaginary request defined by the needs of higher layers and inputs and preconditions provided by lower layers. Next, the individual service match can be seen as a special case of the combined match with one advertisement only. This close relations allow us to reuse the theory and algorithms of the individual service match to compute the combined and the composed match.

Finally, we introduce the notion of an incomplete combined or composed match.

Definition 6 (Incomplete Combined Match): Let \( R = \langle I_R, O_R, E_R, S_R \rangle \) be a state enhanced request with \( S_R = \langle F_R, D_R \rangle \). We call a set of service advertisements \( M = \{A \mid A = \langle I_A, O_A, P_A, E_A \rangle \} \) an incomplete combined match if all conditions of Definition 4 hold except for conditions 4 and 5, i.e., some inputs required by advertisements in \( M \) are not provided in the service request \( R \) or some preconditions are not guaranteed to hold. We define the incompleteness degree of \( M \) as the number of inputs and preconditions that are not provided or not valid.

Definition 7 (Incomplete Composed Match): Let \( R = \langle I_R, O_R, E_R, S_R \rangle \) be a state enhanced request with \( S_R = \langle F_R, D_R \rangle \). Let \( CM = \{M_0,M_1,\ldots,M_n\}, M_i = \{A \mid A = \langle I_A, O_A, P_A, E_A \rangle \} \) set of advertisements, \( i = 1,\ldots,n \). We call CM an incomplete composed match if all conditions of Definition 5 hold except for conditions 4 and 5, which do not hold for the lowest most layer only (i.e., \( M_0 \)). This means that some inputs required by advertisements in \( M_n \) are not provided in the service request \( R \) or some preconditions are not guaranteed to hold. We define the incompleteness degree of \( CM \) as the number of inputs and preconditions of \( M_n \) that are not provided or not valid.

We use definitions of incomplete matches in matching algorithms in the next sections. The incompleteness degree will be used in the heuristic during the composed match search. Oftentimes even an incomplete match might be of use for the requester. This might be the case when the re-
requester does not want to disclose its state. In such a situation it might be possible that the requester has more information available than those explicitly specified in the request and thus even an incomplete match could be of use for the requester.

6 Matching Algorithms

Definitions of combined and composed match from the previous section present conditions that service advertisements need to fulfill in order to match a given request. In this section we use these conditions to define the matchmaking algorithms which are used in the discovery service. We defined the combined and the composed match as a conservative extension of the match for single service which allows us to extend the original algorithm presented in [10] relatively easily. We extended the original algorithm in two aspects. First, we added the support for preconditions and effects evaluation, and second, we added the support for composed and combined match.

The matchmaking algorithm works in two phases. In the registration phase, a service advertisement is registered with the matchmaker. During this phase the advertisement is saved in the main data structure, Advertisements, which is basically a look-up table (an inverted index) in which advertisements together with some auxiliary data structures are stored. Stored advertisements are indexed by classes (types) to support a fast retrieval of advertisements which produce or consume a given class. In addition to input and output types, the advertisements can be also retrieved by using atoms appearing in advertisements effects and preconditions. Advertisements are stored together with precomputed degree of match for each relevant class. In the look-up phase, the Advertisements structure is used for finding the set Match for a given service request \( R \). Algorithm 1 is used for finding the combined match and Algorithm 2 for finding the composed match. Since the main computation burden is carried out during the registration phase, the retrieval of matching individual advertisements can be done relatively efficiently.

When answering a combined match query (Algorithm 1), the discovery service first finds a set of services that together produce the required outputs (step 1) and effects (step 2) (i.e., any service producing some of required outputs or effects is a good candidate). In the next step, out of these candidates, those advertisements that constitute a complete single service match (as defined in Definition 2) are added to the final results set Match (step 3.1). In steps 3.2 and 3.3, the remaining advertisements are possibly labeled as incomplete if some of their inputs or preconditions are missing. As the next step (step 4), all possible combined matches are generated out of the matched candidate advertisements (PreMatch) by employing the generateCombined procedure. The generateCombined procedure is basically quite a simple procedure which uses a brute force algorithm to generate all possible advertisement combinations compliant with the combined match definition. Finally, the resulting matches are sorted according to their degree of match.

Since the combined match is a conservative extension of the single service match, the degree of match is computed in the same fashion as in the original algorithm for matching single services (see [10] for details). The only main difference is implied by the fact that the combinedMatch can return also incomplete matches. An incomplete match is considered worse than an exact, plugin and subsume match and is penalized accordingly. Still, the incomplete match can be useful. Either it can be used directly by the requester who might try to get required pieces of information from some other source, or it can serve as a starting point to compute the composed match, as we show in the next paragraph.

One interesting thing to mention, is the fact that implementation of the combinedMatch procedure required relatively minimal modifications to the single service matching procedure. The most visible modification was addition of the generateCombined procedure, which also has the possible biggest impact on the time complexity. We will discuss this issue in more detail in the next section.

---

**Algorithm 1 combinedMatch** \( R = \langle I_R, O_R, E_R, S_R \rangle \)

1. for each \( (?o_a, T_o) \in O_R \)
   1.1 retrieve all \( A = \langle I_A, O_A, P_A, E_A \rangle \) from Advertisements for which \( \exists (?o_a, T_o) \in O_A \) such that \( T_o \approx T_o \) (Def. 4, C. 1)
   1.2 add all such \( A \) to PreMatch
2. for each \( e_r \in E_R \)
   2.1 retrieve all \( A = \langle I_A, O_A, P_A, E_A \rangle \) from Advertisements for which \( S_R \models A \Rightarrow e_r \) (Def. 4, C. 2)
   2.2 add all such \( A \) to PreMatch
3. for each \( A \in \text{PreMatch} \)
   3.1 if \( A \) is a complete single service match (Def. 2, C. 1–4) then add \( A \) to Match; delete \( A \) from PreMatch; continue;
   3.2 if \( \exists (?i_a, T_i) \in I_A \ \forall (?i_r, T_r) \in (I_R \cup D_R) \text{ such that } T_i \neq T_o \) (Def. 2, C. 3 fails) then mark \( A \) as incomplete
   3.3 if \( \exists p_a \in P_A \ S_R \not\models p_a \) (Def. 2, C. 4 fails) then mark \( A \) as incomplete
4. add to Match results of generateCombined(PreMatch)
5. return sort(Match)
Clearly, the main problem of the combined match is the fact, that it does not consider proper service compositions. Thus there might be situations when there exists a composition of advertisements able to satisfy some service request, however, the combinedMatch procedure does not discover it since it does not allow service chaining. The composed match overcomes this problem by allowing proper service composition. But the costs in terms of time complexity might be high (service composition is NP-hard). With this problem in mind we designed the procedure for answering composed match queries as an iterative deepening algorithm. It starts with finding an initial imperfect solution as fast as possible (by using the combinedMatch procedure), and if some more time is remaining, it tries to improve the discovered solution(s) (i.e., incomplete matches) by exploring the search space defined by registered service advertisements. We employed a simple heuristic in the algorithm. Every time there are more possible incomplete composed or combined matches which need to be improved the algorithm picks the one with the smallest incompleteness degree. The heuristic is used in a hope that there is a higher chance of completing the match with the lowest incompleteness degree than for the other incomplete matches.

In Algorithm 2, we define the composedMatch procedure realizing the above described ideas. Without loss of generality we assume that the Match set contains composed matches only (we explained that both the combined and the single service match can be seen as a special case of the composed match). The composedMatch procedure starts with computing an initial set of combined matches as a starting point (step 1). Complete matches are left in the Match set (we are done with these), while the incomplete matches are prepared for the next stage (step 2). In the next stage (step 3), the backward chaining iterative deepening algorithm tries to complete as many incomplete matches in the given time as possible. In step 3.1, the heuristic is used to pick one incomplete match (CM). In step 3.2, the missing inputs end effects of CM are transformed into a new “fictive” service request $R_{bh}$. If $R_{bh}$ could be answered, the result would provide all missing inputs and preconditions of CM and thus allowing to make CM a complete match. To answer $R_{M_n}$, the combinedMatch procedure is used (step 3.3). Each result of step 3.3 can be seen as new layer that can be attached to the composed match CM. In step 3.4, a new composed match is created for each combination of returned combined match with the initial composed match CM. The resulting new composed match $CM_{new}$ is either added to the Match or to the IncompleteMatch set depending on its completeness. After that, a new iteration is started with the new most promising incomplete match. The main loop finishes when time runs out or when all choices are exhausted. Finally, the resulting set of matches is returned sorted by their matching degree.

**Algorithm 2**

composedMatch($R = \langle I_R, O_R, E_R, S_R \rangle, time$)

1. $Match \leftarrow combinedMatch(R)$
2. foreach incomplete $CM \in Match$ do remove CM from $Match$; add CM to IncompleteMatch
3. while currentTime() < time and $IncompleteMatch \neq \emptyset$
   3.1 from $IncompleteMatch$ remove $CM$ where $\{M_0, M_1, \ldots, M_n\}$ with the smallest incompleteness degree
   3.2 create new request $R_{M_n} = \langle I_{M_n}, P_{M_n}, S_{R} \rangle$ where
      - $I_{M_n}$ set of inputs required by advertisements in $M_n$ not provided in the service request $R$
      - $P_{M_n}$ set of preconditions required by advertisements in $M_n$ not guaranteed to hold in $S_R$
   3.3 NewMatch $\leftarrow combinedMatch(R_{M_n})$
   3.4 for each $M \in NewMatch$ do
      - create new combined match $CM_{new}$ by adding $M$ as a new layer $M_{n+1}$ to $CM$
      - if $CM_{new}$ incomplete then add $CM_{new}$ to $IncompleteMatch$
      else add $CM_{new}$ to $Match$
4. return sort($Match$)

7 Discussion and Preliminary Evaluation

One perspective of how one can think about matchmaking and composition problem is in terms of the worst case properties. Apparently, from this point of view, this paper does not have much new to tell. A combined match retrieval is designed to be fast, but in the worst case its time complexity will be exponential in number of effects and outputs of the service request because of the generateCombined procedure. Additionally, it does not consider compositions and thus does not always guarantee the result to be found. A composed match retrieval takes service compositions into account, but it is NP-Complete because of the very nature of OWL-S and planning based service composition. So clearly, that is not the point of this paper.

Our motivation stems from several important observations. First, a typical web service has only a rather small number of inputs, outputs, preconditions and effects. This is also the case for service requests. For example, services in an OWL-S service retrieval test collection\(^2\) typically do not have more than 5 to 10 outputs (as a matter of fact an average number is as small as 2 or 3). Next, reasonable compositions that make a sense tend to be small and shallow. This is especially the case in the context of process

---

\(^2\)Available at http://projects.semwebcentral.org/projects/owls-tc/
mediation, in which we often deal mainly with the problem of conversions or data incompatibilities. Finally, even when the service registries might contain many registered advertisements, we believe that since the registries content will be fragmented by domains, the initial selection of candidate advertisements for a given request will be very small compared to the initial registry size.

These observations motivated our design choices. We were hoping that the combined match will indeed be cheap in terms of time in a normal case, and that the burden of the exponential \texttt{generateCombined} procedure will not be important because of a small number of outputs and effects in a usual advertisement and request. As a result, the provider will get an added value in terms of much higher chances of finding a working match for its request for about the same price (in terms of time).

In our preliminary test scenario we were focusing on the matching time of the combined match which is used also in the composed match. Our preliminary tests performed with a testing set of advertisements seem to confirm our hypothesis. For our testing scenario we randomly generated 50 – 100 advertisements with 1 – 10 IOPEs in each group. IOPEs were annotated with concepts form an ontology containing 100 distinct concepts. We randomly generated service requests with up to 20 IOPEs each and queried the matchmaker with these requests. Figure 5 summarizes average measured request times over several experiments with different sets of services registered in the registry. As it can be seen, for small number of effects and outputs, the difference in retrieval time between the single service match and the combined match are rather insignificant. Unsurprisingly, with the growing number of outputs and effects in the request the impact of the combination increases exponentially. Figure 6 illustrates the number of returned matches for a combined match. Again, with growing number of effects and outputs in the request we can observe an exponential growth of potential matching candidates. Several strategies can be applied to deal with the exponential growth of retrieval time and number of matches. One approach is to consider only several best matches according to the matching degree (i.e., modifying the \texttt{generateCombined} procedure to realize a top-k retrieval). Another solution is to shift the task of combining the discovered service to the requester. In other words, this would mean that the matchmaker discovers only services covering the service request and returns this set to the requester without trying to find possible combinations.

To overcome the problem of the worst case time complexity of the composed match retrieval, we design the algorithm so that it allows to accommodate to a time constraint provided by the requester. In the given time, the algorithm tries to maximize the chance, that a complete match will be return. We are currently working on testing the composed match retrieval.

8 Related Work

Traditionally, the web services discovery field does not primarily focus on matching sets of advertisements. We extend the work of Paolucci et. al. [10], which was recently extended in other directions as well. For example, Klusch et. al. [6] address the problem of flexible matchmaking by using a hybrid matchmaker which combines the semantic approach with approaches from the information retrieval field. Most recently, [2] analyzes the correctness of [10] and suggests an improved algorithm based on bipartite graph matching.

Benatallah et. al. in [3] propose an approach based on request rewriting that allows a combination of several services to satisfy the service request. The hypergraph theory is used in order to find a combination of web services that best match the given request. The optimality criteria is derived from the notion of covering as much as possible the
outputs of the request and requiring as little as possible of inputs that are not provided in the description of the request.
The approach of Benatallah et. al. is different from our approach in the sense that in the combination match in [3] authors insist on the optimality, while we do not. On the other hand, we added a composed match which goes beyond the combination of services.

An approach similar to ours was developed by Küster et. al. in [8]. In [8] the authors propose an integrated approach to service matchmaking and composition in the context of the DIANE Service Description language. In principle, the authors extend the matching work of [4, 5] so that combinations of services are possible in order to satisfy requests with multiple effects. There are several differences between our work and [8]. First of all, the DIANE language is quite different from OWL-S, and also the basic matching algorithm differs substantially. For example, the DIANE matching is based on graphs matching and fuzzy sets comparisons while, OWL-S relies mainly on DL reasoning. In terms of combination based matching, our notion of a combined match is very similar in nature to the concept of multiple effects matching introduced in [8]. In our understanding, nothing similar to composed match is supported in [8].

9 Conclusions and Further Work

In this paper we described mechanisms for matchmaking and discovery of sets of services. We defined a combined and a composed match, which present an extension to the single service based matching. Such an extension finds its use mostly in dynamic open environments in the contexts such as service composition or process mediation, in which it is not realistic to assume the perfect knowledge of the environment. Therefore, discovery services need to be used as part of the composition or mediation processes and be able to respond to needs identified within these processes. We argued that introducing matches derived from service combinations increases substantially the likelihood of a successful match. The reason why we introduced two new different types of matches stems from our belief that depending on the application context, different constraints will apply and as a result different types of service discovery will be useful. We devised matchmaking algorithms for both types of matches. On the one hand, one goal of the design was to allow the discovery of service sets with minimal overhead compared the single service discovery. On the other hand, we also wanted to give completeness guarantees, given enough time. Therefore, we developed an iterative matching algorithm which improves found solutions with time. In the near future, more experiments has to be performed. We also plan to employ the developed algorithms in cooperation with the process mediation agent.

References