Automatic Matchmaking of Web Services

Sudhir Agarwal and Rudi Studer
Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe (TH), Germany.
{agarwal, studer}@aifb.uni-karlsruhe.de

Abstract

Web services help in achieving increased automation across organizational boundaries. In this paper, we present an approach for annotating WSDL documents with semantically rich descriptions. We also present an algorithm that considers such annotations in addition to just the types of input and output parameters. Our matchmaking algorithm not only returns match/no-match answers but in case of a match a set of conditions under which a web service offers the desired functionality.

1. Introduction

Web services have primarily been designed for providing interoperability between business applications. Current technologies assume a large amount of human interaction, for integrating two applications. This is primarily due to the fact that business process integration requires understanding of data and functions of the involved entities. Semantic Web technologies, powered by description logic based languages like OWL [12], aim to add meaning to Web content, by annotating the data with ontologies. This allows agents to get an understanding of users' Web content and greatly reduces human interaction for meaningful Web searches. A similar approach can be used for adding meaning to Web service descriptions, which will in turn, allow more automation, by reducing human involvement for understanding the data and functions of the services.

WSDL [17] operations however offer minimal semantics (types of input and output parameters) of the functionality of a web service. Since the actual functionality of a web service is described in natural language within the WSDL `documentation` tags, a lot of human effort for discovering and using web services is needed. In order to achieve more automation in web service usage life cycle, at least the relationship between inputs and outputs need to be modeled. This leads to various *semantic web services* efforts.

Web service discovery is one of the most important tasks, that needs more automation. Existing matchmaking approaches based on semantically rich descriptions hardly consider more information than just the types of input and output parameters, although the underlying language or the ontology typically allows to model much more properties of a web service than just the input and output types. That is, existing matchmaking approaches do not exploit the expressiveness of the semantic web service modeling languages.

In this paper, we introduce a novel approach for modeling and reasoning about web services. We model semantic, temporal and security constraints in a unified way and present a matchmaking algorithm that considers all three types of information than just the types input and output parameters of a web service. Further, our matchmaking algorithm not only returns a yes/no answer but a set of conditions under which a web service offers the desired functionality.

The paper is structured as follows. In section 2, we show how web services can be modeled semantically. In section 3, we show how different types of requests can be specified. In section 4, we present a matchmaking algorithm that given a request specified in our request specification formalism, automatically finds web services descriptions that offer the desired functionality. For each match, the algorithm identifies the conditions under which the match offers the desired functionality. In section 5, we give a brief introduction of our prototypical implementation. After discussing some related work in section 6, we conclude in section 7.

2. Specification of Web Services

If the web is seen as a huge database, web services are like stored procedures. Like stored procedures, web service work on a schema and data. Hence, in order to enable automation in the usage of web services, schema and data (static aspects) as well as the behavior of a web service (dynamic aspects) must be modeled formally.

While the static aspects can be modeled with set description languages like description logics[5], the dynamic aspects of web services can be modeled with the help of an appropriate process algebra, like π-calculus [13, 15] or Petri nets to name the most common. While set description languages cannot capture the semantics of the dynamics aspects, process algebras typically abstract from the semantics of the data objects that are communicated among processes.

Therefore, in order to specify web services semantically, we need to combine a process description language with a set description language. Considering our analogy with stored procedures, we need to have something like formal PL/SQL for the web. In our case, we choose description logics as set description language and π-calculus as process description language.

In this section, we give short introductions to description logics and π calculus. Then, we present our approach for modeling web services.

2.1. Introduction to Description Logics with Concrete Domains

Description logics are a family of knowledge representation formalisms closely related to first-order and modal logic. They are useful in various application fields, such as reasoning about database conceptual models, as the logical underpinning of ontology languages, for schema representation in information integration systems or for metadata management [5].

One of the drawback of the description logic $\mathcal{ALC}$ is that all the terminological knowledge has to be defined on an abstract logical level. In many applications, one would like to be able to refer to concrete domains and predicates on these domains while defining concepts. For example the domain of integers or real numbers with predicates like equality, inequality etc. In [6] a scheme for integrating such concrete domains into description logics is presented. The scheme introduces partial functions that map objects of the abstract domain to values of the concrete domain, and can be used for building complex concepts. Further research on decidability, computational complexity, and reasoning algorithms for different DLs with concrete domains has influenced the design of the Ontology Web Language (OWL), which supports a basic form of concrete domains, so-called datatypes.

In the following, we give the basic definitions regarding description logics with concrete domains and refer to [6] for more details.

Definition 1 *A concrete domain* $\mathcal{D} = (\mathsf{dom}(\mathcal{D}), \mathsf{pred}(\mathcal{D}))$ *consists of a set* $\mathsf{dom}(\mathcal{D})$ *(the domain) and a set of predicate symbols* $\mathsf{pred}(\mathcal{D})$*. Each predicate symbol* $P \in \mathsf{pred}(\mathcal{D})$ *is associated with an arity* n *and an* n*-ary relation* $P^{\mathcal{D}} \subseteq \mathsf{dom}(\mathcal{D})^{\mathsf{n}}$.

Definition 2 *In addition to the concepts expressions in* $\mathcal{ALC}$*,* $\mathcal{ALC}(\mathcal{D})$ *allows expressions of the form* $P(u_1, \ldots, u_n)$ *as concept descriptions, where* $P \in \mathsf{pred}(\mathcal{D})$ *is an* n*-ary predicate and* $u_1, \ldots u_n$ *are feature chains. The semantics of such concepts expressions is defined as follows:*

$$P(u_1, \ldots, u_n)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | (u_1^{\mathcal{I}}(d), \ldots u_n^{\mathcal{I}}(d)) \in P^{\mathcal{D}}\},$$

where for $u = f_1 \ldots f_m$ *a feature chain,*

$$u^{\mathcal{I}}(a) = f_m^{\mathcal{I}}(f_{m-1}^{\mathcal{I}}(\ldots(f_1^{\mathcal{I}}(a)\ldots).$$

Definition 3 *A concrete Domain* $\mathcal{D}$ *is called admissible iff (1) the set of its predicate names,* $\mathsf{pred}(\mathcal{D})$ *is closed under negation, (2)* $\mathsf{pred}(\mathcal{D})$ *contains a unary predicate* $\top_{\mathcal{D}}$ *for* $\mathsf{dom}(\mathcal{D})$*, and (3) the satisfiability problem for finite conjunctions over* $\mathsf{pred}(\mathcal{D})$ *is decidable.*

Subsumption and satisfiability of $\mathcal{ALC}(\mathcal{D})$ is decidable iff the concrete domain $\mathcal{D}$ is admissible [6].

2.2. Introduction to Pi-Calculus

In this section, we give a short introduction to π-calculus and refer to [13, 15] for details. The syntax of an agent can be summarized as follows:

$$\begin{array}{rcl} P & ::= & 0 \\ & | & P_1 + P_2 \\ & | & \overline{y}x.P \mid y(x).P \mid \tau.P \\ & | & P_1 \parallel P_2 \\ & | & [x = y]P \\ & | & A(y_1, \ldots, y_n) \end{array}$$

Summation $\sum_{i \in I} P_i$, where the index set I is finite. This agent behaves like one or other of the P_i. We write 0 for the empty summation, and call it inaction; this is the agent which can do nothing. Henceforward, in defining the calculus, we confine ourselves just to 0 and binary summation, written $P_1 + P_2$.

Prefix form $\overline{y}x.P$**,** $y(x)$ **or** $\tau.P$. $\overline{y}x$ is called a negative prefix. $\overline{y}$ may be thought of as an output port of an agent which contains it; $\overline{y}x.P$ outputs the name x at port y and then behaves like P. $y(x)$ is called a positive prefix. A name y may be thought of as an input port of an agent; $y(x).P$ inputs an arbitrary name z at port y and then behaves like $P\{z/x\}$. The name x is bound by the positive prefix '$y(x)$'. (Note that a negative prefix does not bind a name). τ is called a silent prefix. $\tau.P$ performs the silent action τ and then behaves like P.

Composition $P_1 \parallel P_2$. This agent consists of P_1 and P_2 acting in parallel. The components may act independently; also, an output action of P_1 (resp. P_2) at any output port x may synchronize with an input action of P_2 (resp. P_1) at x, to create a silent (τ) action of the composite agent $P_1 \parallel P_2$.

Match $[x = y]P$. This agent behaves like P if the names x and y are identical, and otherwise like 0.

Defined agent $A(y_1, \dots, y_n)$. For any agent identifier A (with arity n) used thus, there must be a unique defining equation $A(x_1, \dots, x_n) \overset{\text{def}}{=} P$, where the names $x, \dots, x_n$ are distinct and are the only names which may occur free in P. Then $A(y_1, \dots, y_n)$ behaves like $P\{y_1/x_1, \dots, y_n/x_n\}$ (see below for the definition of substitution). Defining equations provide recursion, since P may contain any agent identifier, even A itself.

Definition 4 *A substitution is a function σ from $\mathcal{N}$ to $\mathcal{N}$ which is almost everywhere identity. If $x_i\sigma = y_i$ for all i with $1 \le i \le n$ (and $x\sigma = x$ for all other names x).*

Definition 5 *Processes P and Q are α-convertible, $P \equiv Q$, if Q can be obtained from P by a finite number of changes of bound names. α-convertibility can be seen as syntactic identity between processes.*

Let $P\sigma$ denote the process obtained by simultaneously substituting $z\sigma$ for each free occurrence of z in P for each z, with change of bound names to avoid captures. In particular the following hold, when the bound name y is replaced by the name y'.

$$\begin{array}{rcl} (x(y).P)\sigma & \equiv & x\sigma(y').P\{y'/y\}\sigma \\ ((y)P)\sigma & \equiv & (y')P\{y'/y\}\sigma \end{array}$$

2.3. Modeling Web Services

π-calculus has a similar drawback as the description logic $\mathcal{ALC}$. Namely, it requires to specify all the process knowledge at the abstract logical level. In practical settings however specification of well known operations like "add", "subtract" etc. does not bring much added value. Rather, it makes the specification of processes very tedious. In order to overcome this problem, we assume a set of agents the functionality of which does not need to be defined further. For example, $\mathsf{add}, \mathsf{subtract}, \mathsf{select}, \mathsf{insert}, \mathsf{update}$ etc., and propose to use the simulation relation for such agent analogous to subsumption relation of concrete domain predicates in $\mathcal{ALC}(\mathcal{D})$. For example, $\overline{\mathsf{service_1}}\mathsf{B.P}$ simulates $\overline{\mathsf{service_2}}\mathsf{A.P}$ if $\mathsf{A} \sqsubseteq \mathsf{B}$, in case the services $\mathsf{service_1}$ and $\mathsf{service_2}$ are query answering services. Note, that such simulation relationships also provide simple process mediation.

Another important issue that we need to resolve before we can specify web services as π-calculus processes, is the connection between domain ontologies, semantics of data in the messages and process description. For this purpose, we use the following restrictions:

- Inside "[]", we write DL concrete domain predicates [6] to model conditions semantically. For example, the expression $\mathsf{instanceOf(x, C)}$ inside "[]" specifies the condition "x is an instance of C.
- We assume that names that are communicated between processes are description logic expressions. We use a special symbol $\mathsf{valueOf}$ inside a DL expression to refer to the value of a bound name (variable).

The concrete agents mentioned above and other web services run concurrent to a web service process. To keep the things a little simpler, we assume that a web service has a communication channel with an agent a. We denote the channel also by a. Further, we assume a process user also running concurrent to the web service process to denote the user. To keep the specifications short and better readable, we will only consider the specification of the web service process and not all the concurrent running processes, since they are often obvious from the context.

In the following, we discuss some examples to illustrate the strength of our formalism.

2.3.1. Only Output The simplest kinds of web services are web services that provide some information and do not require any inputs.

Example 1 *Consider a web service that returns the list of professors working in a university in Germany. The web service does not require any inputs.*

$$\overline{\mathsf{select}}\, \text{“}\mathsf{Professor} \sqcap \exists \mathsf{worksIn.(University} \sqcap \exists \mathsf{locatedIn.\{Germany\})}''.\mathsf{select(x)}.\overline{\mathsf{user}}\, \mathsf{x}$$

Example 2 *Consider a web service that returns the list of (Professor, University) pairs, where the professor P works in the university U, if the pair (P, U) belongs to the output. Again, the web service does not require any inputs.*

To model this web service, the web service provider needs to model a new concept, say $\mathsf{Affiliation} \sqsubseteq \exists \mathsf{employee.Professor} \sqcap \exists \mathsf{institution.University}$. The web service can now be modeled as:

$$\overline{\mathsf{select}}\ \text{“}\mathsf{Affiliation}\text{''}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\,\mathsf{x}$$

2.3.2. Output dependent on input This type of information providing web services provide some information that is dependent on the input. The input must be provided by the user.

Example 3 *Consider a web service that returns the list of all professors working in a certain university. The user has to provide the value of the university, for which he wishes to have the list of professors working in the university.*

$$\mathsf{user(u)}.\{[\mathsf{valueOf(u)\ instanceOf\ University}]$$
$$\overline{\mathsf{select}}\ \text{“}\mathsf{Prof} \sqcap \exists \mathsf{worksIn}.\{\mathsf{valueOf(u)}\}\text{''}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\,\mathsf{x}\}$$

Example 4 *Consider a web service that returns the list of all (professor, university) pairs, such that if professor P works in university U, then the pair (P, U) is in the output. The web service requires the list of universities as input.*

$$\mathsf{user(u)}.\overline{\mathsf{select}}\ \text{“}\mathsf{Affiliation} \sqcap \exists \mathsf{employee.Professor} \sqcap$$
$$\exists \mathsf{institution}.\{\mathsf{valueOf(u)}\}\text{''}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\,\mathsf{x}$$

2.3.3. Conditional Outputs In some cases, a web service may offer different types of output depending on the value of the input.

Example 5 *Consider two disjoint concepts* C_1 *and* C_2 *and a web service that expects an input u and depending on if the value of u is less than* 100 *or not the web service returns either an instance of* C_1 *or an instance of* C_2.

$$\mathsf{user(u)}.[\mathsf{P}_<(\mathsf{valueOf(u)}, 100)]\{\overline{\mathsf{select}}\ \text{”}\mathsf{C}_1\text{”}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\ \mathsf{x}\}$$
$$+[\mathsf{P}_\geq(\mathsf{valueOf(u)}, 100)]\{\overline{\mathsf{select}}\ \text{”}\mathsf{C}_2\text{”}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\ \mathsf{x}\}$$

2.3.4. Services with inputs at runtime Until now, we had examples, where the inputs of a web service are provided at the beginning. In some cases, a web service may not require all the inputs at the beginning but only at run time, depending on the execution path it takes. With our approach, it is straightforward to model this artifact.

Example 6 *Consider a web service that requires an input parameter u in the beginning and then depending on the condition c_1 (may be dependent on the value of u) either asks for another input v and then does the activities in P or if the condition c_1 is not satisfied it just performs the activities in Q.*

$$\mathsf{user(u)}.[\mathsf{c}_1]\mathsf{user(v)}.\mathsf{P} + [\neg \mathsf{c}_1]\mathsf{Q}$$

2.3.5. Web Services with Access Control Semantic web services promise automation in dynamic business that can be offered and carried out in the Web. In such an open market, access control, which means the users must fulfill certain conditions in order to access certain functionality plays an important role.

Many existing approaches view security related properties of a web service and non-functional properties and treat them separately from (typically on top of) functional properties. However, often it is not possible to separate them as the following example shows: *Consider a library web service that sends a book via surface mail and a confirmation as the output of the web service, if the user is member of the library. Otherwise, the web service sends a failure message to the user.* Now, in order to prove his eligibility to gain access to the web service, a user has to show a set of certificates. The functionality of the web service can not be modeled independent from the access conditions.

In [3], we have shown, how access control policies can be modeled with ontologies in order to achieve better interoperability between certification authorities, web service providers and end users. In [2], we have shown how access control policies can be composed.

We foresee an input parameter for the set of certificates a user has to show in order to prove his eligibility to access a web service. Further, we use a special predicate $\mathsf{CCD}(\mathsf{C}, \mathsf{P})$, that is true iff the set C of certificates fulfills the access control policy P according to the certificate chain discovery algorithm [7].

Example 7 *Consider the web service from example 1 with the slight modification that only the employees of a university have access to it.*

$$\mathsf{user(C)}.[\mathsf{CCD}(\mathsf{valueOf(C)}, \mathsf{UniEmployee})]$$
$$\overline{\mathsf{select}}\ \text{“}\mathsf{Professor} \sqcap \exists \mathsf{worksIn}.(\mathsf{University} \sqcap$$
$$\exists \mathsf{locatedIn}.\{\mathsf{Germany}\})\text{''}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\,\mathsf{x}$$

In this section, we have seen how various for practice useful aspects of web services can be specified with our formalism.

3. Request Specification

We now turn our attention to request specification. With a request a user specifies conditions a web service must fulfill in order to be considered as a match.

Conditions on Output A requester specifies with

$$\overline{\mathsf{select}}\ \mathsf{query}.\mathsf{select}(\mathsf{x}).\overline{\mathsf{user}}\mathsf{x}$$

conditions on the output x.

Conditions on Input Depending on what a user exactly wants, inputs of a web service can sometimes mean restriction, sometimes flexibility. To understand this, consider the web services from example 2 and example 4. The only difference between the two web services is that the latter requires a list of universities as input. The service in example 2 returns a list of (Professor, University) pairs. If a user does not want to have all (Professor, University) pairs but only those for certain universities that he wishes to determine, he should be able to specify this wish. Considering such a constraint, the service in example 2 should not be detected as a match. If a requester wishes to provide an input, the request can be defined as $\mathsf{user(u)}.\overline{\mathsf{select}}\ \mathsf{query.select(x)}.\overline{\mathsf{user}}\mathsf{x}$, where query may be dependent on the value of u.

Conditions on the sequence of operations Sometimes, a requester wishes to define constraints on the sequence of certain actions. For example, credit card should not be charged before the order has been placed. However, a web service may or may not perform some tasks after the order placement and before the action for charging the credit card. To model such constraints, we use a symbol $\Box$ to denote a set of *don't-care activities.*

If a user wants that the credit card should be charged immediately after the order has been placed, the request can be defined as follows[1]:

$$\mathsf{user(Order).user(ChargingInfo).}\\ \overline{\mathsf{placeOrder}}\ \mathsf{Order}.\overline{\mathsf{chargeCC}}\ \mathsf{ChargingInfo}$$

In case, a user only wants that the credit card is charged some time after but not necessarily immediately after the order has been placed, the request can be defined as follows:

$$\mathsf{user(Order).user(ChargingInfo).}\\ \overline{\mathsf{placeOrder}}\ \mathsf{Order}.\Box.\overline{\mathsf{chargeCC}}\ \mathsf{ChargingInfo}$$

Conditions on multiple outputs As shown in one of the examples, there are web services that offer more than one output. If a user wishes to have more than one output, he may or may not be interested in a particular sequence the outputs are delivered. While the desired sequence of outputs can be described as shown above, we need a mechanism to describe the request in which the sequence of the outputs is not important.

To model such conditions on the behavior of a web service, we use the symbol Σ. $\Sigma(a_1, \ldots, a_n)$ with $a_1, \ldots, a_n$ being sets of activities means that the sets of activities $a_1, \ldots, a_n$ may be performed in any order; however, the activities in a particular set a_i must be performed in the defined order.

1 Note, that the agents $\mathsf{placeOrder}$ and $\mathsf{chargeCC}$ do not have to be necessarily the same agents as described as components of some web service, but may have simulation relationships with the components of a potential match.

Suppose a user needs a web service that places a laptop order, charges the credit card and sends the confirmation to the user. In addition to this, the web service should also close an insurance policy for the maintenance of the laptop after the order has been placed successfully. However, it does not matter for the user whether the credit card is charged (for the laptop purchase) before or after closing the insurance policy.

$$\mathsf{user(Order).user(ChargingInfo).}\\ \overline{\mathsf{placeOrder}}\ \mathsf{Order}.\Sigma(\Box.\overline{\mathsf{chargeCC}}\ \mathsf{ChargingInfo},\\ \overline{\mathsf{closePolicy}}\ \mathsf{PolicyInfo}, \overline{\mathsf{user}}\ \mathsf{Confirmation})$$

4. Matchmaking

In the first step, a requester defines a request describing conditions on outputs, e.g. type constraints and conditions on the structure of the web service. or wish to provide an input. The matchmaking algorithm returns the set of matches together with the conditions for each match under which the match can offer the required functionality.

4.1. Matchmaking Algorithm

Our matchmaking algorithm not only yields yes/no answers, but in case of yes answers, it also yields conditions a user has to fulfill in order to achieve the desired functionality with a match.

(1) Since the request may contain Σ expressions to describe that the order of the execution of activity sets is not important, we need to preprocess the request. Consider a request R that contains k Σ expressions. For a $\Sigma(a_1, \ldots, a_n)$, we generate $n!$ new requests $r_1, \ldots, r_{n!}$ for each possible execution sequence of the activities $a_1, \ldots, a_n$. Now, each of the r_is contains $k-1$ Σ expressions. This step is performed for each r_i to obtain the set of requests that have only $k-2$ Σ expression and so on until all the Σ expressions have been treated yielding a set of Σ expression free requests $r_1 \ldots, r_\Omega$. Let us call this set R_Ω.

(2) The request R may contain variables, i.e. bound names. In this step, we calculate the set of variables in the request R and denote it by $V(R)$. (3) Consider a web service description W. (4) Calculate the set of bound names of W and denote it by $V(W)$.

(5) Calculate substitution functions $\sigma_1, \ldots, \sigma_k$ from $V(R)$ and $V(W)$ for each possible renaming of the bound names in $V(W)$ by bound names $V(R)$.

(6) For each substitution function $\sigma \in \{\sigma_1, \ldots, \sigma_k\}$ calculate with α-conversion W_i from W by renaming the bound names in W according to σ. This step delivers a set of descriptions W_i that are syntactically equivalent to the W except for variable renaming. While changing a bound name x by y, we replace each occurrence of $\mathsf{valueOf(x)}$ by $\mathsf{valueOf(y)}$. It has been shown in [8] that α-conversion is decidable.

(7) Each description W_i is a sequential process with conditions. For each process description W_i, calculate the set of sequential process descriptions W_{ij} without conditions and remember the conditions in $C(W_{ij})$. That is, from $\mathsf{[A]P}$ infer P and add A to the set of conditions.

(8) For every $r \in R_\Omega$ and every W_{ij}, check which activities in r are syntactically same to which activities of W_{ij}. While doing so, do not consider don't care ($\square$) activities in r. Further, if there are any expressions denoting predefined agents, e.g. select, then add the conditions under which the agent simulates the corresponding expression in r to $C(W_{ij})$.

(9) Consider those $r \in R_\Omega$ for which there exists a W_{ij} such that all the non-don't-care activities in r have a syntactical equivalent activity in W_{ij}. If there is no such r, the web service W is not a match for the request R.

(10) Check whether W_{ij} can become syntactically equal to the r by considering possible values for $\square$ activities. If a possibility is found then the web service W is a match for the request R. Add W to the result set along with the corresponding set of conditions $C(W_{ij})$.

4.1.1. Examples Consider the request $R =$

$$\overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.(University} \sqcap \exists\mathsf{locatedIn.\{Germany\})}''.\mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}$$

that asks for all professors that work in a university in Germany and the web services from our examples. Since, the request does not contain any Σ expressions, step 1 will yield $R_\Omega = \{R\}$. Step 2 yields $V(R) = \{\mathsf{answer}\}$. Consider the web service from example 1 as W. The set of bound names $V(W)$ of the web service W is $\{x\}$. Step 5 yields only one possible substitution $\sigma_1 = \{(\mathsf{answer}, \mathsf{x})\}$. Step 6 yields

$$W_1 = \overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.(University} \sqcap \exists\mathsf{locatedIn.\{Germany\})}''.\mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}$$

Steps 7 yields $W_{11} = W_1$. Steps 8 and 9 do not change W_{11} any further. Steps 10 detects that W_{11} and the request are syntactically same. Hence W is a match.

Performing the same steps with the web service from example 2 will not lead to syntactically equal expressions, since the web service has a different query than the request. Also, the web service from example 3 will not be detected as a match, since it requires an input variable u. Now, consider a slightly different request

$$R = \square.\overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.University}''.\mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}$$

Steps 1 to 5 are easy to follow. Suppose, the algorithm is considering the web service from example 3. Step 6 yields

$$\begin{aligned}
W_1 &= \mathsf{user(answer)}.\{[\mathsf{valueOf(answer)\ instanceOf\ University}]\\
&\quad \overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.\{valueOf(answer)\}}''.\\
&\quad \mathsf{select(x)}.\overline{\mathsf{user}}\,\mathsf{x}\}\\
W_2 &= \mathsf{user(u)}.\{[\mathsf{valueOf(u)\ instanceOf\ University}]\\
&\quad \overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn}.\\
&\quad \mathsf{\{valueOf(u)\}}''.\mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}\}\\
W_3 &= \mathsf{user(answer)}.\{[\mathsf{valueOf(answer)\ instanceOf\ University}]\\
&\quad \overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.\{valueOf(answer)\}}''.\\
&\quad \mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}\}
\end{aligned}$$

In step 7, let us consider W_2. $W_{21} =$

$$\mathsf{user(u)}.\{\overline{\mathsf{select}}\,\text{“}\mathsf{Professor} \sqcap \exists\mathsf{worksIn.University}''.\mathsf{select(answer)}.\overline{\mathsf{user}}\,\mathsf{answer}\}$$

with $C(W_{21}) = \{\text{”}\mathsf{instanceOf(valueOf(u), University)}\text{”}\}$

Step 8 does not modify the set of conditions $C(W_{21})$. Step 9 produces a positive answer, since all the non-don't-care activities of the request have an equivalent in W_{21}. Finally, step 10 detects that setting the activity $\mathsf{user(u)}$ at the place of $\square$ will make W_{21} syntactically same as the request. So, W, that is the web service from example 3 is a match with conditions $C(W_{21})$. It is easy to see that W_1 and W_2 will not be detected as match since they fail the test in step 9.

The result of the matchmaking algorithm is a set of web service descriptions along with the conditions for each match under which it offers the required functionality. On receiving such a result, a user has to check for each match, whether and how he can fulfill the corresponding conditions. We believe that automatic composition techniques can be useful for performing this task by considering the conditions returned by the matchmaking algorithms as goal for an automatic composition algorithm.

Consider the last example from the previous section that illustrates the matchmaking algorithm. The algorithm yields that the web service from example 3 is a match if the condition ”$\mathsf{instanceOf(valueOf(u), University)}$” is fulfilled. This condition can be automatically converted to the following request to find web services that can fulfill the condition: $\overline{\mathsf{select}}$”$\mathsf{University}$”$.\mathsf{select(x)}.\overline{\mathsf{user}}\mathsf{x}$.

5. Implementation

We have a prototypical implementation of a server that maintains a large number of web services descriptions described with our formalism. Our matchmaking algorithm is implemented in Java and uses the KAON2 reasoner[2] for reasoning with DL ontologies.

A client sided tool allows a web service annotator to load a WSDL document, annotate it with our formalism and upload it to the server. Roughly, when a WSDL file is loaded, the tool converts the information inside WSDL `types` tags into a DL ontology and WSDL operations to process descriptions. The client sided tool is connected with the server and allows an annotator to relate and edit/correct already existing ontologies on the server.

The client sided tool also allows users to specify their requests and send them to the server. The server performs the matchmaking and sends the matches to the client, which shows the matches together with the conditions to the user. The reason of offering this functionality in the client sided tool too, is that we plan to extend the client sided tool for automatic checking of the fulfillment of the conditions by using simple automatic composition algorithm, e.g. the one presented in [1]. Note, that in order to evaluate how the conditions can be satisfied, access to user's local knowledge is required. Hence, it is more practical and realistic to perform such evaluations at the client side.

6. Related Work

Recently, automatic matchmaking of web services has gained tremendous importance and new approaches are introduced frequently. In the following, we will discuss some of the most widely known approaches.

Consider the text "The service advertised is a car selling service which given a price reports which cars can be bought for that price" in section 3.2 of [14]. The reason for describing the meaning of the web service functionality in natural language is that OWL-S still does not mandate a logic for describing pre- and post-conditions though OWL-S suggest SWRL as possible candidates. As a consequence, OWL-S based matchmaking approaches can not consider the relationships between inputs and outputs [14, 16] but only the types of input and output parameters. Further, note that matchmaking algorithms that consider only the types of input and output parameters do not actually require OWL-S profile, since the required information is available in a WSDL document.

WSDL-S proposes to extend WSDL by adding pre- and post conditions for operations [4]. However, similar to OWL-S, WSDL-S also does not fix the logic for describing such conditions. As a consequence, it is not possible to develop matchmaking algorithms that can consider such conditions. Note, that the decidability and complexity of the matchmaking algorithms depends on the logic that is used for describing the conditions.

The WSMO initiative addresses the issue of goal definition and web service discovery in much more detail than the above mentioned approaches [11]. The WSMO web service discovery approach differentiates between service and web service discovery. WSMO also addresses the issue of heterogeneity of descriptions of requesters and providers. Similar to OWL-S matchmaker, WSMO differentiates between different types of matches. For example, exact-match, subsumes-match, plugin-match and intersection match.

In [10], it is argued that subsumption based matchmaking can be too strong and thus may not find some matches. The authors suggest *entailment of concept non-disjointness* instead of subsumption to overcome this shortcoming. Section 4.3 of [10] says "The requester accepts shipping from Plymouth or Dublin and the provider accepts shipping from UK cities. Since Dublin is not in the UK, neither of the two associated sets of service instances fully contains the other in every possible world.". That is, matching based on subsumption will not detect the service as a match, but matching based on entailment of concept non-disjointness will. Provided with only a positive boolean answer a requester may believe that the service also accepts shipping from Dublin.

Major difference of our approach to all other currently existing approaches is that the existing approaches produce only yes/no answers. That is, they either do not consider the conditions while performing matchmaking or do not provide the client with conditions under which a web service offers the desired functionality. Note, that our conditions regarding the functionality of a web service should not be confused with the result of OWL-S based matchmaker in case of partial matches [14, 16], since the latter only lets the user know which inputs (or outputs) did not match and as we discussed above input and output types do not actually describe the functionality of a web service.

Another difference between our approach and existing approaches is that our request specification technique is more expressive than the existing approaches. With our request specification formalism, a user can

2 `http://kaon2.semanticweb.org`

specify semantic constraints, security constraints and temporal constraints in a unified way. The matchmaking approaches discussed above consider only semantic constraints. [9] presents an approach for checking security constraints. To the best of our knowledge, there is no existing approach that can handle (specify and reason over) all the three types of constraints in a unified way.

7. Conclusion and Outlook

In this paper, we have introduced a simple and powerful approach to describe web services semantically. We have identified different types of web services and showed how their semantic, security and temporal properties can be described with our approach. We have also identified different types of requests a user may wish to specify and showed how they can be specified.

Unlike most of the existing approaches that may provide expressive formalism for describing web services but do not provide matchmaking algorithms that make use of the expressivity, our matchmaking approach can use all the available semantic information. The main feature of our matchmaking algorithm is not only to provide a match/no-match answer as it is the case with most of the existing approaches, but a set of conditions, that a user has to fulfill in order to obtain the required functionality from a web service. Further, we have implemented our approach prototypically and tools are available for public use soon.

The presented matchmaking algorithm is exponential in the number of condition-expressions. In future, we intend to investigate whether incremental construction of only relevant models in step (7) and optimization techniques known from "Ordered Binary Decision Diagrams" can be useful to optimize the algorithm.

Acknowledgments This work was supported by the Federal Ministry of Education and Research (BMBF) under the Internetökonomie-SESAM project and the European Commission under contract IST-2003-506826 SEKT.

References

[1] S. Agarwal, S. Handschuh, and S. Staab. Annotation, Composition and Invocation of Semantic Web Services. *Journal of Web Semantics*, 2(1):1–24, 2005.

[2] S. Agarwal and B. Sprick. Access Control for Semantic Web Services. In *2nd International Conference on Web Services*, 2004.

[3] S. Agarwal and B. Sprick. Specification of Access Control and Certification Policies for Semantic Web Services. In *6th International Conference on Electronic Commerce and Web Technologies*, August 2005.

[4] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S, " A joint UGA-IBM Technical Note, version 1.0,. Technical report, April 2005.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory Implemenation and Applications.* Cambridge University Press, 2003.

[6] F. Baader and P. Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.

[7] D. E. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9:285–322, 2001.

[8] M. Dam. On the Decidability of Process Equivalences for the pi-Calculus. *Theor. Comput. Sci.*, 183(2):215–228, 1997.

[9] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security For DAML Web Services: Annotation and Matchmaking. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *2nd International Semantic Web Conference*, volume 2870 of *LNCS*, pages 335–350, Sandial Island, Fl, USA, October 2003. Springer.

[10] S. Grimm, B. Motik, and C. Preist. Matching semantic service descriptions with local closed-world reasoning. In *In 3rd Annual European Semantic Web Conference. Springer, Budva, Montenegro, June 2006. (to appear)*, 2006.

[11] U. Keller, R. Lara, A. Pollers, I. Toma, M. Kifer, and D. Fensel. WSMO Web Service Discovery, November 2004. http://www.wsmo.org/TR/d5/d5.1/v0.1.

[12] D. L. McGuinness and F. van Harmelen (eds.). OWL Web Ontology Language. Technical report, W3C, March 2003.

[13] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I+II. *Journal of Information and Computation*, pages 1–87, September 1992.

[14] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In I. Horrocks and J. A. Hendler, editors, *Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science (LNCS)*, Sardinia, Italy, 2002. Springer.

[15] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes.* Cambridge University Press, New York, NY, USA, 2001.

[16] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1):27–46, December 2003.

[17] W3C. Web Service Description Language (WSDL) Version 1.2, March 2003. http://www.w3.org/TR/wsdl.