Towards Practical Semantic Web Service Discovery

Martin Junghans, Sudhir Agarwal, and Rudi Studer

Karlsruhe Institute of Technology (KIT)
Institute of Applied Informatics and Formal Description Methods (AIFB)
Karlsruhe Service Research Institute (KSRI)
Karlsruhe, Germany
{martin.junghans,sudhir.agarwal,rudi.studer}@kit.edu

Abstract. Service orientation is a promising paradigm for offering and consuming functionalities within and across organizations. Ever increasing acceptance of service oriented architectures in combination with the acceptance of the Web as a platform for carrying out electronic business triggers a need for automated methods to find appropriate Web services.

Various formalisms for discovery of semantically described services with varying expressivity and complexity have been proposed in the past. However, they are difficult to use since they apply the same formalisms to service descriptions and requests. Furthermore, an intersection-based matchmaking is insufficient to ensure applicability of Web services for a given request. In this paper we show that, although most of prior approaches provide a formal semantics, their pragmatics to describe requests is improper since it differs from the user intention. We introduce distinct formalisms to describe functionalities and service requests. We also provide the formal underpinning and implementation of a matching algorithm.

1 Introduction

Service-oriented computing is an interdisciplinary paradigm that revolutionizes the very fabric of distributed software development Applications that adopt service-oriented architectures (SOA) can evolve during their lifespan and adapt to changing or unpredictable environments more easily. When properly implemented, services can be discovered and invoked dynamically, while each service can still be implemented in a black-box manner. Despite these promises, service integrators, developers, and providers need to create techniques and tools to support cost-effective development, as well as the use of dependable services and service-oriented applications.

Brief Overview. Service discovery deals with finding appropriate Web services for the task at hand and is one of the central components needed for developing a SOA application. Universal Description, Discovery and Integration (UDDI) based service discovery is rather syntactical and requires a lot of manual effort for finding the right services [1]. For example, UDDI is not able to deal with synonyms or relations between terms that describe services. Since the advent of the Semantic Web, many semantic Web service discovery approaches have been proposed to deal with heterogeneity in the terminology used in different services. Some of them consider the functionality description of services, which allows for automated tasks like service composition. The common model to describe the functionality of a service is represented by inputs, outputs, preconditions, and effects, or shortly denoted by (I, O, ϕ, ψ). Inputs denote the set of user-provided message parts at Web service invocation. Outputs describe the set of values returned to the user after service execution. Preconditions and effects describe the information states of the world before and after service execution, resp., by logical formulas. Semantic Web service discovery approaches compute the match between a service offer that describes the functionality of the service and a service request.

OWL-S Matchmaker uses OWL-S profile for describing Web service offers as well as requests [2,3]. Even though OWL-S Profile has elements for preconditions and effects, the OWL-S matchmaker uses types of input and output parameters only. The approach presented in [4] models Web services as well as requests as description logic (DL) classes and bases the matchmaking on the intersection of service offer and request, which is computed by a DL reasoner. Such approaches fail to reason about the dynamics of Web services, since DL reasoners can not reason about changing knowledge bases. The approach in [5] deals with variables, but is limited to Web services that do not change the world and, thus, can be described by a query. Efficient semantic discovery approaches that can deal with functionality of Web services are presented in [6,7]. Efficiency is achieved by pre-computing a classification of services in a hierarchy of goal templates. However, the requirement of such a classification hierarchy hinders the usability of creating service descriptions and requests since it is not feasible to maintain a global hierarchy in a decentralized and open environment of the Web. Furthermore, [6,7] do not support matching of inputs and outputs nor do they deal with the possible inconsistency between functional description of Web services and their classification.

Problem of Using One Formalism. Apart from the problems mentioned above, one common problem of all existing approaches is that they apply the same formalism for describing service offers and requests. The use of the same formalism for both descriptions does not correspond with the intuition of the requester. Such mismatch between the semantics of formalisms and the intuitive interpretation of the requester makes these approaches hard to use in practice.

If the same formalism is used for offers and requests, then a service request corresponds to a service offer description from which a set of desired services is derived. In our view, this is an impractical and unintuitive approach as we will further justify in the subsequent section. Consequently, we propose to use two distinct formalisms for service descriptions and request. It is more intuitively that a service description formalizes the actual functionality of a Web service and a service request describes the set of services that provide a requested functionality.

Structure. In this paper we provide a semantic service discovery approach to overcome the above outlined problems. In Section 2, we elaborate on the problems of state of the art approaches and the motivation for our approach. Two different formalisms to describe services offers and requests are introduced in Sections 3 and 4, resp. Based on the semantics of the formalisms that we provide, the definition of a match between offers and requests is also presented in Section 4.4. We further present an implementation of our approach accompanied with performance tests in Section 5. In Section 6, we discuss the relation of our discovery approach to other approaches. Finally, we conclude in Section 7 by summarizing our results and giving an outlook.

2 Motivation

At first sight, a practical semantic service discovery should feature expressive description languages that are non-ambiguous, easy to use as well as support heterogeneous descriptions. We do not investigate usability aspects by means of a simple syntax, query language for expressing formulas or supportive user interfaces. To this extent, our approach distinguishes from goal driven approaches as in [8] by not focusing on an abstract and non-technical request (goal) description that a user creates. Such user goals have to be translated into machine understandable requests. The latter representation of a request is the starting point of our work.

The formalisms to describe Web service functionalities and requests must provide sufficient expressivity to allow users to formulate rich functionality descriptions and to precisely formulate constraining requirements in requests, resp. Users should be rather limited by their willingness to invest effort than by any technical limitation.

Although using a formal way to express a request, usability can be fostered by the provision of an unambiguous and thus comprehensible interpretation of service descriptions and requests. E.g., it should be obvious to users whether the requested inputs are interpreted as inputs that must or can be accepted by desired services.

Regarding the openness of the Web, semantic service descriptions are considered to be highly heterogeneous as different actors may use different vocabularies and ontologies. Also, a discovery solution must scale against a large number of available semantic Web service descriptions.

Requirements on Formalisms. A service request describes a class of services, namely the desired ones. If the description of a service request uses the same formalism as the one used for service descriptions, then there exists a mismatch between the interpretation of a service description $D = (I, O, \phi, \psi)$ and a request description $R = (I_R, O_R, \phi_R, \psi_R)$. As depicted in the left part of Figure 1, D and R are both interpreted as a set of execution runs. A match between them is given if there is an intersection between the two sets of runs. Degrees of matches, for instance plugin and subsume match, present different types of the intersection of both sets of runs. We refer to [3] for details on the degrees of matches.

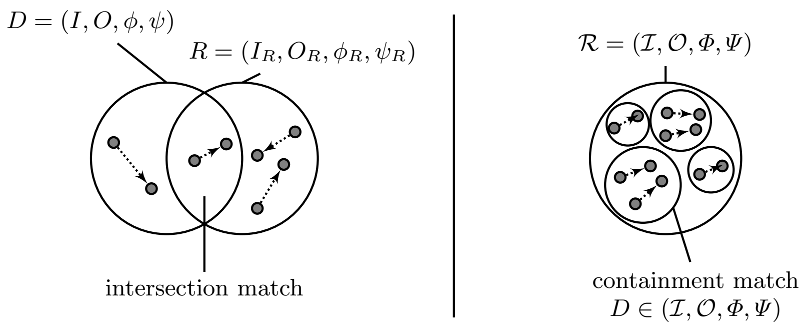
$D = (I, O, \phi, \psi)$
$R = (I_R, O_R, \phi_R, \psi_R)$
intersection match
$\mathcal{R} = (\mathcal{I}, \mathcal{O}, \Phi, \Psi)$
containment match
$D \in (\mathcal{I}, \mathcal{O}, \Phi, \Psi)$

Fig. 1. On the left, the same formalism is applied to Web service description D and request R. Using our different formalisms for offers and requests allows for the description of several run sets in a request $\mathcal{R}$ as shown in the right part.

Intersection-based approaches lack the ability to let requests exclude certain properties because not all requested properties need to be fulfilled by matching services. Furthermore, an intersection-based match cannot guarantee that a matching service can be successfully invoked as the execution run that the user wants to invoke may not comply to the requirements specified in the request. This can be since the desired run might not be member of the intersection between service offer and request. Thus, in order to guarantee applicability of a matching service, intersection-based approaches using the same formalism for offers and requests need to further check for applicability in a further step. Consequently, the freedom provided by the different matching degrees is not practicable for the purpose of service invocation. As an example that was already discussed in [9], a service offers to ship goods from a city in the UK to another city in Germany. A user requests for a shipping provider that operates between European cities. Using intersection based matchmaking will identify the mentioned service as a match. However, if the requester wants to ship an item from Berlin to Hamburg, then the matched service offer fails.

Another example explains why the service description formalism cannot be used for requests and vice versa. Consider the functional description of a book selling Web service that requires the invoker to provide an ISBN book number of the book to order as input. While inputs of the service descriptions are consentaneously interpreted as required for invocation of an atomic Web service, the inputs specified in a request can be interpreted differently. From the user perspective, a request for book selling Web services may contain different interpretations of inputs simultaneously. Either the user provides an ISBN number or alternatively author name and book title as inputs. This simple example leads to the observation that a request cannot simply specify a set of inputs. The same conceptual mismatch between service descriptions and requests occurs in preconditions and effects. Employing the same formalism for the description of a service and a set of desired services is not appropriate, because their interpretation and their intended use are different. Requests conceptually differ from service functionality descriptions. We believe that this mismatch makes (I, O, ϕ, ψ)-based formalisms difficult to use.

In order to develop a formalism that allows to practically describe requests that can be matched against (I, O, ϕ, ψ)-based service descriptions, we identified the requirements of requests descriptions that also effect the formalism of service descriptions. First, users should be able to describe required properties of a set of desired services in a request. As depicted in the right part of Figure 1 and in analogy to database queries, a request rather describes properties of the result set than a precise desired service functionality. Second, it needs to be clear to requesters how the request description effects the set of matching services. In contrast, it is not obvious which requested properties of a matching service justify the match if discovery approaches consider different degrees of matches. The reason is that intersection-based matchmaker cannot guarantee that matching services comply to all requested properties. And third, to provide a matchmaking algorithm, offers and requests have to be mapped to a common formal model where matches can be identified.

3 Service Descriptions

In this section we first introduce our formal model of Web services and then present the formalism to describe service functionality descriptions. The functionality of a Web service is described by a set of inputs I, set of outputs O, precondition ϕ, and effect ψ. We consider atomic Web services that may require user inputs at service invocation time and provide outputs at the end solely. There are no user interactions in between, which allows us to describe service functionalities by the states before and after execution without stating anything about the intermediate states.

3.1 Formal Model of Web Services

In our formal model, we consider a set of actors $\mathcal{A}$ identified by a unique identifier, e.g., their public key. For each actor $A \in \mathcal{A}$, we consider a knowledge base KB_A. Furthermore, each actor $A \in \mathcal{A}$ can provide a set of Web services. A Web service can use other Web services (of the same or different actors). That is, an execution of a Web service W provided by an actor $A \in \mathcal{A}$ can cause changes not only in the knowledge base KB_A of actor A, but also in the knowledge bases of (external) actors whose Web services are used by the Web service W. However, the execution of a Web service W can not cause any changes in the knowledge bases of the actors that are not involved in W. We model a state as the set of knowledge bases of all the actors. Formally, a state is $\{KB_A : A \in \mathcal{A}\}$. The execution of a Web service W is equivalent to a transition between states. The transition models changes in the knowledge bases within the resp. states.

3.2 Inputs and Outputs

The sets of inputs I and outputs O denote the set of inputs that are compulsory for service invocation and the set of outputs returned after successful execution, respectively. They assign the service's inputs and outputs to variable names that can be referenced in preconditions and effects and also allow us to distinguish inputs and outputs from instances that already exist in the provider's knowledge bases. We model a book selling Web service as a running example for illustration. The types of inputs and outputs are specified in preconditions and effects. The service with $I = \{\mathsf{u}, \mathsf{p}, \mathsf{a}, \mathsf{t}\}$ requires a user identification u, password p, author name a, and a title t for successful invocation. Further, $O = \{\mathsf{b}, \mathsf{i}\}$ describes that the service provides a book b and invoice i as outputs after execution to the invoker.

3.3 Preconditions and Effects

In order to provide a practical discovery approach, we now clearly specify the intention and the interpretation of preconditions and effects. By this we clarify the pragmatics and avoid ambiguities about what is modeled by a provider and in turn which conditions a requester may query in a request. The pragmatics of the logical formula that represents the precondition is restricted to the description of (i) requirements on inputs, like their types, relationships among them, conditions on the values of the inputs, and (ii) conditions that must hold in the resp. state from the perspective of service providers. A precondition describes the state from the perspective of the service provider(s) before a service can be successfully invoked. In contrast to [10], by preconditions and effect we do not intend to describe global states that model the knowledge bases of the entire world as perceived by an external observer which certainly causes several problems.

An effect formula is restricted to describe (i) constraints on returned outputs, (ii) the relation between inputs and outputs, and (iii) changes made by the service in the knowledge bases of service providers.

Below an example precondition ϕ and effect ψ description of the book selling service is partly shown.

$$\begin{aligned}
\phi \equiv\ & \mathsf{UserId}(\mathsf{u}) \wedge \mathsf{isRegistered}(\mathsf{u}) \wedge \mathsf{isAuthorized}(\mathsf{u},\mathsf{p}) \wedge \mathsf{Password}(\mathsf{p}) \wedge \mathsf{Book}(\mathsf{b}) \wedge \\
& \mathsf{hasAuthor}(\mathsf{b},\mathsf{a}) \wedge \mathsf{Author}(\mathsf{a}) \wedge \mathsf{hasTitle}(\mathsf{b},\mathsf{t}) \wedge \mathsf{Title}(\mathsf{t}) \wedge \mathsf{isAvailable}(\mathsf{b}) \wedge \ldots \\
\psi \equiv\ & \mathsf{Order}(\mathsf{o}) \wedge \mathsf{containsProduct}(\mathsf{o},\mathsf{b}) \wedge \mathsf{containsPrice}(\mathsf{o},\mathsf{p}) \wedge \mathsf{hasPrice}(\mathsf{b},\mathsf{p}) \wedge \\
& \mathsf{Invoice}(\mathsf{i}) \wedge \mathsf{containsProduct}(\mathsf{o},\mathsf{i}) \wedge \mathsf{hasAddress}(\mathsf{u},\mathsf{ad}) \wedge \mathsf{isShipped}(\mathsf{o},\mathsf{ad}) \wedge \ldots
\end{aligned}$$

The precondition ϕ states that the described service requires that the user with ID u is registered and authorized by its password p. Both, u and p are inputs. Furthermore, the service requires for a successful execution that the book b with author a and title t is available.

3.4 Semantics of Service Description

The semantics of service descriptions translates them into a formal model. We use a labeled transition system (LTS) as formal state-based model for both service descriptions and requests. This powerful model allows us to enhance description formalisms in future work. An LTS $L = (S, \mathcal{W}, \rightarrow)$ comprises a set S of states, a labeled transition relation $\rightarrow \subseteq S \times \mathcal{W} \times S$, and a set $\mathcal{W}$ of transition labels. A state is described by the knowledge bases of involved service providers.

The description (I, O, ϕ, ψ) of a service w is translated to an LTS $L = (S, \mathcal{W}, \rightarrow)$ such that the execution of w is modeled by two states $s \in S$ and $t \in S$ and a transition $(s, w, t) \in \rightarrow$ that is labeled with $w \in \mathcal{W}$. The state s is the one described by the precondition and inputs and the state t is the one described by the effect, inputs, and outputs. Consequently, s models the state before and t models the state after service execution. In summary, the LTS L that models a service description is defined as follows.

$$\begin{aligned} &L = (S, \mathcal{W}, \rightarrow) \\ &S = \{s, t\}, \mathcal{W} = \{w\}, \rightarrow = \{(s, w, t)\} \end{aligned}$$

4 Service Requests and Matchmaking

A service request aims at specifying constraints in order to restrict the set of available Web services to the set of desired Web services. Within our model, this can be done by specifying (i) constraints on inputs and outputs and (ii) constraints on preconditions and effects. In Section 4.1, we show how constraints on inputs and output can be specified. Then we show in Section 4.2 how desired preconditions and effects can be expressed. Analogously to the prior section, we then present the semantics of a service request description. At the end of this section, we define matches between requests and service descriptions.

Service requests use a formalism that is different from the one of service descriptions. It allows for the description of a request such that a set of matching services is characterized. Requests are denoted by $(\mathcal{I}, \mathcal{O}, \Phi, \Psi)$. The right part of Figure 1 depicts the motivation of a clear distinction between service descriptions $D = (I, O, \phi, \psi)$ and a service request $\mathcal{R} = (\mathcal{I}, \mathcal{O}, \Phi, \Psi)$ describing a set of desired Web services.

4.1 Constraints on Inputs and Outputs

The specification of desired inputs (outputs) of a request describes the set of desired sets of inputs (outputs). The description of the set of sets is expressed in a logic that allows us to express conjunctions, alternatives, and exclusions of inputs and outputs.

As an example, a library system that frequently places book orders requests for services with the following input specification. Disregarding the remaining request description, the input specification $\mathcal{I}$ corresponds to the set of services that either only require an ISBN i or at least author a and title t of the book to order but do not require a date of birth bday. Any coherencies between the inputs or outputs are expressed in the formulas that model requested preconditions and effects, resp.

$$\mathcal{I} \equiv \big(\exists \mathsf{isbn} \in I \wedge \forall \mathsf{j} \in I : \mathsf{isbn} = \mathsf{j}\big) \oplus \big(\exists \mathsf{author} \in I : \exists \mathsf{title} \in I : \nexists \mathsf{bday} \in I\big)$$
$$\mathcal{O} \equiv \exists \mathsf{book} \in O$$

The input set I is the set of inputs provided by the service description, which is matched against this query. The requested outputs $\mathcal{O}$ specify that desired services return a book book.

4.2 Constraints on Preconditions and Effects

Now we focus on formalizing the specification of requested preconditions $\varPhi$ and effects $\varPsi$. Similar to requesting for inputs and outputs, we aim at describing a set of desired services. Preconditions ϕ and effects ψ of a service describe the states before and after service execution, resp., and model the available knowledge. A knowledge base is described by facts that must hold in it. Preconditions $\varPhi$ and effects $\varPsi$ of a request are interpreted as queries against a repository of service descriptions. The queries retrieve the set of services that accept or provide the requested conditions in the states before and after service execution modeled by ϕ and ψ, resp.

A requested precondition description $\varPhi$ might be modeled as follows. The precondition request $\varPhi$ is a query against the preconditions ϕ of service descriptions and matches those that provide the requested the required facts.

$$\begin{aligned}\varPhi \equiv\ & \exists \mathsf{b} : \mathsf{Book}(\mathsf{book}) \wedge \mathsf{hasAuthor}(\mathsf{book}, \mathsf{author}) \wedge \mathsf{hasTitle}(\mathsf{book}, \mathsf{title}) \wedge \\ & \mathsf{Author}(\mathsf{author}) \wedge \mathsf{isAvailable}(\mathsf{book}) \wedge \neg\mathsf{isRegistered}(\mathsf{user}) \wedge \mathsf{Birthday}(\mathsf{d})...\end{aligned}$$

Using a logic like first-order logic to specify the set of requested preconditions and effects not only allows for precisely expressing which conditions are provided by service offers, but also for excluding services with undesired conditions. For example, the negation of the condition isRegistered(.) prevents matching Web services that require a user registration for the order of the specified book. Consequently, the example of $\varPhi$ prevents that it will match the Web service indicated with the example precondition ϕ in the previous section since the service required a registered user. The description of requested effects $\varPsi$ is conceptually similar and omitted due to space limitations.

4.3 Semantics of Service Request

A service request is translated to a set $\mathcal{L}$ of labeled transition systems. As a request describes a set of possible input sets, output sets, preconditions, and effects, the set $\mathcal{L}$ is used to formally model the set of possible service executions described by a request $(\mathcal{I}, \mathcal{O}, \varPhi, \varPsi)$. For each state $s \in S$ that fulfills the requested precondition $\varPhi$ and for each state $t \in S$ that fulfills the requested effect $\varPsi$, there exists one LTS $L \in \mathcal{L}$ with an unlabeled transition $(s, \epsilon, t) \in\rightarrow$ in L. That is, there exists an LTS in $\mathcal{L}$ for each combination of a start and a end state. A start state s represents a possible answer of the knowledge base query $\varPhi$. Analogously, each knowledge base that is an answer to the requested effect $\varPsi$ introduces one end state t.

In summary, an LTS $L = (S, \{\}, \rightarrow)$ that is member of the set $\mathcal{L}$ comprises one start state $s \in S$, one end state $t \in S$, and one transition $(s, \epsilon, t) \in \rightarrow$. The transition is labeled with an empty label ϵ. Since a request potentially describes many services, several potential service executions modeled by the LTS' are modeled in the set $\mathcal{L}$.

For example, considering the example request above, the desired sets of inputs specified by $\mathcal{I}$ embrace $\{\mathsf{isbn}\}$, $\{\mathsf{author}, \mathsf{title}\}$, and further supersets of them. Combined with each desired precondition, a set of start states is constructed and combined to each end state that represents a combination of desired inputs, outputs, and effect.

4.4 Matchmaking

The discovery of semantically described services identifies service descriptions from a repository that fulfill the requirements specified in a request. After both formalisms and their translation to a common formal model were introduced, the task of discovery adds up to the identification of a containment relation between an LTS L^d, which models the description (I, O, ϕ, ψ) of a service w, and a set of LTS' $\mathcal{L}$, which models a request $(\mathcal{I}, \mathcal{O}, \Phi, \Psi)$. Let the indexes d and r of variables indicate their belonging to the LTS L^d and $L^r \in \mathcal{L}$, respectively.

The description of service w matches a request if and only if $L^d \in \mathcal{L}$. That is, $\exists L^r \in \mathcal{L} : L^r \equiv L^d$. Latter equivalence holds if and only if there are transitions $(s^d, w, t^d) \in \rightarrow^d$ and $(s^r, w, t^r) \in \rightarrow^r$ with (i) equivalent states $s^d \in S^d$ and $s^r \in S^r$, and (ii) equivalent states $t^d \in S^d$ and $t^r \in S^r$.

We have described a match within the formal model and we further show how this match corresponds to a match in terms of the descriptions of a service and a request. Basically, the containment relation $L^d \in \mathcal{L}$ corresponds to question answering task of a reasoner to compute the match. Therefore, the reasoner identifies a model for the query by binding the variables of the request to individuals modeled in the knowledge base that describes the service description. To show the equivalence of the match in the formal model and the match identified by a reasoner, both directions of the implication between them are discussed. For simplicity, we only consider knowledge bases that model the states before execution.

If $L^d \in \mathcal{L}$, then $s^d = s^r$ and $t^d = t^r$ as defined above. Then the knowledge bases that describe s^d and s^r are equal and the knowledge bases that describe t^d and t^r are equal. Then, there obviously exists a variable binding to answer the query against the knowledge base KB^d that models the service description. In the other direction, if there exists a variable binding to answer the query, then the knowledge base KB^d is a model of the request and contains at least the information that has to be satisfied to fulfill the request. Due to the definition of the request semantics, there must be also an LTS with states that are described by knowledge bases that are equal to KB^d, because the model $\mathcal{L}$ of a request contains all the LTS' with all possible states and respective knowledge bases that are model of the request.

The same applies to knowledge bases representing the states after service execution. Consequently, a match with respect to preconditions and effect is given if query answering is used as the reasoning task. Matching inputs and outputs is guaranteed since they are part of the knowledge bases.

It was shown that in our discovery approach it is not necessary to distinguish different degrees of matches, which correspond to certain degrees of intersection of the set of service execution runs as depicted in Figure 1. Instead, we provided formalisms that allow users to clearly specify a query with unambiguous interpretation. The query is interpreted as the set of conditions that a matching service must at least provide. This implies, that a matching service may require further inputs, offer further outputs, require further preconditions, or generate further effects within the constraints the user could specify in a query using the expressivity we provide. If a query does not deliver any or the desired results, then query relaxation or manual query refinement can be applied. The former method can thus simulate a subsume or intersect match since a less restrictive query will be able to return services that would match a query by subsume or intersect match.

5 Implementation and Evaluation

We implemented the presented discovery approach that uses both formalisms for service descriptions and requests as well as the matching algorithm from Sections 3 and 4, resp. As we used a WSML reasoner[1], the syntax of descriptions and queries are bound to the syntax of the Web Service Modeling Language WSML [11]. Of course, our approach is not bound to this choice. It is possible to use a different syntax to create the query if another reasoner is used. The implementation is publicly available[2]. A simple user interface allows to enter a request. After submission, two reasoners compute the list of matching services out of a repository of randomly generated Web service descriptions. Services that were identified as match by both reasoners are results of the entire request and are thus displayed to the user.

In this section, we explain the creation of knowledge bases from service descriptions, the implementation of the matchmaking algorithm, and at the end we present some performance results.

Knowledge Base Construction. The description (I, O, ϕ, ψ) of a service w describes two states. Two knowledge bases KB_0, KB_e that model the state before and after execution, respectively, are constructed. The inputs from I are parsed and modeled as instances in both knowledge bases. Output variables from O are also modeled as instances but only in KB_e. The precondition ϕ is added as an axiom to the KB_0 that models the state before service execution and the effect ψ is added as an axiom to KB_e. The same procedure applies for adding further service descriptions to the knowledge bases.

[1] See `http://tools.sti-innsbruck.at/wsml2reasoner` for details on the reasoning process.

[2] `http://www.aifb.uni-karlsruhe.de/WBS/mju/soa4all-discovery` subject to change. Please contact authors if necessary.

Matchmaking. The discovery engine receives a request $(\mathcal{I}, \mathcal{O}, \Phi, \Psi)$ from the user interface and translates it into two queries q_0 and q_e expressed in the WSML query language syntax. The query q_0 is created from the requested inputs $\mathcal{I}$ and precondition Φ and is sent to the first reasoner instance that models KB_0. The query q_e is created from the requested inputs $\mathcal{I}$, outputs $\mathcal{O}$, and effect Ψ and is sent to the second reasoner instance that models KB_e. Both reasoner instances execute the respective queries on their knowledge bases, which may model several service descriptions. In order to answer the query q_0, the first reasoner determines for each service w modeled in KB_0, whether required inputs and the precondition ϕ is a model of the requested precondition Φ. That is, the reasoner checks whether the precondition Φ of a request is fulfilled by the facts in the A-Box that were introduced by the precondition ϕ of the service w. Query q_e is processed analogously by the second reasoner instance on KB_e.

Below, a fragment of an example query that is sent to the first reasoner instance is presented in WSML syntax. The query contains the specification of requested inputs, their types, and the precondition. sm# denotes the namespace of the ontology that formalizes the service model. The shown query asks for services that have two inputs ?a and ?t of type Author and Title of an example ontology denoted by the namespace ex#, respectively. The inputs describe a book ?b of type Book. The continuation of the example may specify further conditions on the book ?b et cetera.

```
?w memberOf sm#Service and ?w[sm#hasPrecondition hasValue ?p] and
?w[sm#hasInput hasValue ?t] and ?t memberOf ex#Title and
?w[sm#hasInput hasValue ?a] and ?a memberOf ex#Author and
?p[sm#hasVariable hasValue ?b] and ?b memberOf ex#Book and
?b[ex#hasTitle hasValue ?t] and ?b[ex#hasAuthor hasValue ?a] and ...
```

After sending both queries q_0 and q_e to the reasoners, the reasoners bind the variable ?w to references of service annotations that fulfill the queries q_0 and q_e, respectively. A service is a match for a given request, if the service is an answer to both queries q_0 and q_e, i.e., the service is identified as a match by both reasoners.

Performance Results. We generated a repository of randomly generated service descriptions. We used the Semantic Web for Research Community ontology [12] as background knowledge base. It provides classes and properties used to model types of individuals and to express conditions used in preconditions and effects. The generated service descriptions are also available at the supplementary Web page. We measured the reasoner's mean query answering time of 100 repetitive runs. Both queries q_0 and q_e were sent in parallel to the reasoners, which computed the answers on an ordinary laptop with dual core 2.4GHz CPU and 4GB of main memory. Both knowledge bases modeled 1000, 2000, 3000, 4000, and 5000 Web service descriptions. Queries of three different sizes were sent to each knowledge base. Small (S), medium (M), and large (L) queries with 1, 2, 3 instances and 2, 4, 6 properties on those instances were tested, respectively. Figure 2 shows the mean time in milliseconds for different knowledge base

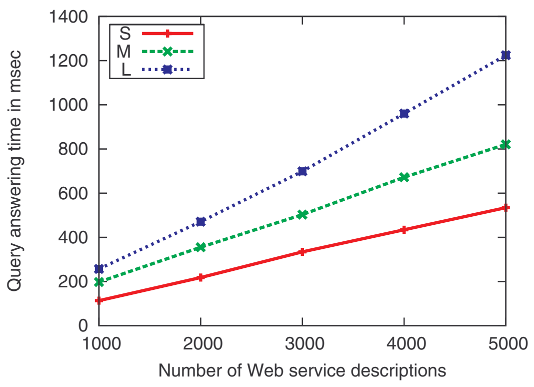

Fig. 2. Mean time to answer the two queries q_0 and q_e

and query size. It is not our intention to claim scalability based on these measurements. We rather want to show feasibility of the presented approach as this paper was mainly focusing on the underlying description formalisms.

6 Related Work

The description of the functionality of a software by preconditions and effects was introduced by [13]. In contrast to description and discovery approaches in the field of software specification, the assumption of a closed world does not hold for Web services. The ability to model side effects to the world and the consideration of background knowledge thus were not considered. Zaremski and Wing consider different match types based on the implication relations between preconditions and postconditions of software library components and a query [14].

OWL-S Profile introduced in [3] proposes to model Web services semantically with inputs, outputs, preconditions and effects. However, OWL-S Profile since being an OWL ontology can not capture the semantics of variables and thus the dynamics of Web services.

Recently, WSMO-Lite has been proposed for describing Web services semantically as the next evolutionary step after SAWSDL[3], filling SAWSDL annotations with concrete semantic service descriptions [7]. WSMO-Lite ontology is on one side lightweight and on the other side provides elements for modeling functionality of Web services. WSMO-Lite does not provide modeling of input and output parameters explicitly and relies on their derivation from the free variables in the formulas for precondition and effect. Note that such a derivation is not possible if a formula does not have any free variables but the Web service has inputs or outputs.

Description logic (DL) based approaches [15,2,16] for describing Web services propose to model inputs and outputs as concepts in description logics, while

[3] Semantic Annotations for WSDL `http://www.w3.org/2002/ws/sawsdl`

discovery, i.e., matchmaking is reduced to checking subsumption of input and output types.

Li et al. combine in [4] the use of description logics with DAML+OIL and DAML-S. Service description and request are similarly structured comprising inputs, outputs, preconditions, and effects. However, they also base their matchmaking on different matching degrees. Another problem is that DL based approaches lack the ability to describe changes in the world that are often caused by Web service executions. Consequently, more recent research activities concentrate on more detailed formalisms, for instance the state-based perspective on Web services that is discussed below. These models allow to model the dynamics of Web services.

Martin et al. presents a discovery approach in [17] that is based on OWL-S and describes services functionalities semantically by inputs, outputs, preconditions, and effects. This approach interprets preconditions as constraints that need to be satisfied for the service requester only and effects as side effects of the service execution on the world. In our approach we model conditions that hold at the service provider side since those conditions can be evaluated during service invocation and execution time.

The state-based service discovery approach [8,18] developed by Stollberg et al. uses an abstract state space as the underlying formal model of service descriptions [10]. The functionality of a Web service is formally described by the set of possible Web service executions while each normal execution of a Web service is determined by its start and end state. The discovery algorithm relies on the assumption that the precondition ϕ logically implies the effect ψ of a Web service execution. Modeling a transition as a logical implication can be problematic, e.g., in case of a Web service that deletes a certain fact, the existence of the fact would imply non existence of the fact, e.g., a user subscription would imply that the user is not subscribed anymore.

Goal-driven approaches like [8,19,20] do not explicitly specify inputs as parts of the goal. However, a goal needs to be mapped to a request for finding appropriate Web services. In such a request, constraints on inputs can be useful, in particular if a user wishes to exclude a particular input parameter. In goal based approaches, goals are mapped to predefined goal templates that are used to find appropriate Web services. However, the usability of one global hierarchy of goal templates is hardly feasible in an open environment like the Web. One major difference between our approach and the goal based approaches is that we interpret inputs, outputs, preconditions, and effects of descriptions and requests differently, namely the former as a pair of states the latter as a pair of queries.

In contrast to the state-based approaches, Hull et al. propose a matching technique for stateless Web services in [5]. They argue that reasoning for stateful service descriptions and expressive background ontologies becomes practically impossible. Thus, this attempt solely considers inputs, outputs, and their relationships. With the restriction to conjunctive queries, the query containment problem is decidable. In our approach, we deal with stateful services since our discovery approach is not based on query subsumption but on query answering.

7 Conclusion and Outlook

In this paper we presented a discovery approach that is more practical as it overcomes several problems of other approaches. We thoroughly investigated the problem of using the same formalism for service descriptions and requests. A formalism to describe service requests as a set of potentially matching services was introduced. Also the formalism to describe service functionalities semantically was renovated by clarifying its interpretation. We defined the matchmaking between descriptions and requests. Therefore, the semantics of both formalisms that translates the descriptions to the same formal model was provided. We presented a prototypical implementation of our semantic discovery technique, which is a part of the larger system developed under the EU funded project SOA4All.

The focus of the present paper was to provide appropriate formalisms in a first step. Since scalability and efficiency is crucial to enable semantic Web service discovery in a large scale on the Web, we will focus on improving the performance and scalability in the future. Among other options, by the introduction of indexing structures, materialization, and more computational resources we expect to handle larger sets of semantic Web service descriptions.

We furthermore plan to integrate non-functional properties to discovery since it is also valid to request for services including the specification of desired values of non-functional properties. In the settings of an open Web with distributed service providers and consequently decentralized knowledge bases, conditions specified in preconditions and effects cannot hold generally. The functionality description of a software artifact is usually described with respect to a local and closed world of the runtime environment [14]. However, Web service functionalities cannot be described in a closed and local context as service executions may involve further external services. Preconditions and effects must be capable to distinguish different actors of the Web. Conditions and changes must therefore explicitly state where they hold. We will therefore examine techniques to refer to the respective actor in order to identify the knowledge base in which a condition holds.

Acknowledgments. The authors gratefully acknowledge funding by the European project SOA4All (FP7-215219, `http://www.soa4all.eu`).

References

1. UDDI: UDDI Executive White Paper. Technical report, UDDI.org (2001)
2. Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, p. 333. Springer, Heidelberg (2002)
3. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services. Web Semantics: Science, Services and Agents on the World Wide Web 1(1), 27–46 (2003)
4. Li, L., Horrocks, I.: A Software Framework for Matchmaking Based on Semantic Web Technology. Int. J. Electron. Commerce 8(4), 39–60 (2004)

5. Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., Stevens, R.: Deciding Semantic Matching of Stateless Services. In: Proc. of 21st Nat. Conf. on Artificial intelligence (AAAI 2006), pp. 1319–1324. AAAI Press, Menlo Park (2006)
6. Stollberg, M., Martin Hepp, J.H.: A Caching Mechanism for Semantic Web Service Discovery. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 480–493. Springer, Heidelberg (2007)
7. Vitvar, T., Kopeck, J., Viskova, J., Fensel, D.: WSMO-Lite Annotations for Web Services. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 674–689. Springer, Heidelberg (2008)
8. Stollberg, M., Hepp, M., Hoffmann, J.: A Caching Mechanism for Semantic Web Service Discovery. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 480–493. Springer, Heidelberg (2007)
9. Grimm, S., Motik, B., Preist, C.: Variance in e-business service discovery. In: Proceedings of the ISWC Workshop on Semantic Web Services (2004)
10. Keller, U., Lausen, H., Stollberg, M.: On the Semantics of Functional Descriptions of Web Services. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 605–619. Springer, Heidelberg (2006)
11. de Bruijn, J., Fensel, D., Kerrigan, M., Keller, U., Lausen, H., Scicluna, J.: Modeling Semantic Web Services: The Web Service Modeling Language. Springer, Heidelberg (2008)
12. Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., Oberle, D.: The SWRC Ontology - Semantic Web for Research Communities. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 218–231. Springer, Heidelberg (2005)
13. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969)
14. Zaremski, A.M., Wing, J.M.: Specification matching of software components. ACM Trans. Softw. Eng. Methodol. 6(4), 333–369 (1997)
15. Benatallah, B., Hacid, M.S., Leger, A., Rey, C., Toumani, F.: On automating Web services discovery. The VLDB Journal 14(1), 84–96 (2005)
16. Gonzalez-castillo, J., Trastour, D., Bartolini, C.: Description Logics for Matchmaking of Services. In: KI 2001 Workshop on Applications of Description Logics (2001)
17. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
18. Stollberg, M., Keller, U., Lausen, H., Heymans, S.: Two-phase Web Service Discovery based on Rich Functional Descriptions. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 99–113. Springer, Heidelberg (2007)
19. Lara, R., Corella, M., Castells, P.: A Flexible Model for Locating Services on the Web. Int. J. Electron. Commerce 12(2), 11–40 (2008)
20. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic Location of Services. In: Proceedings of the 2nd European Semantic Web Symposium (ESWS 2005), Heraklion, Crete (2005)