Abstract:
Web services are the next wave of Internet application. They are dramatically prevailing over the World Wide Web. How to discovery the right services is a big challenge. In this paper, a new approach is proposed for web services matching within the UDDI registry, based on the matching results a ranking process will significantly reduce the number of recommend services and increase the accuracy. This process is based on semantics in domain ontology. The whole solution provides a novel way to discover and utilize published web services. It is flexible and extensible to accomplish complex web service requests.

Keywords:
Web Service; Web Service Ranking; Ontology

1. Introduction

Information agents are ubiquitous on the World Wide Web. The next wave on Internet is service oriented computing, and Web services are becoming popular and widely accepted due to their accessibility and compatibility. How to let locate suitable web services to accomplish your information tasks is a new challenge. UDDI (Universal Description Discovery and Integration) is an industry initiative enabling businesses to define their Web Services, discover other Web Services, and share information about how they interact in a global registry.

There have been a number of efforts to add semantics to the discovery process. An early work in this area has been the creation of DAML-S which uses a DAML+OIL based ontology for describing Web Services [1]. The latest draft release [2] of DAML-S uses WSDL (Web Service Description Language) in conjunction with DAML-S for Web Service descriptions. Recently, Sivashanmugam et al. [3] have highlighted the complexity and the non-standard approach of implementing DAML-S and they have proposed another approach to add semantics to Web Services. Previously, we proposed an XML based language to describe the semantics of Web Services [4].

Web Services Description Language (WSDL) is an emerging communication standard used by Web Services. Management of the interaction between the broker and the service provider based on WSDL is difficult for non-trivial tasks [11]. An architecture is proposed for wrapping information agents into Web Services. These agents will interact with the service provider. Agent Communication Languages KQML (Knowledge Query and Manipulate Language) for the communication between agents will be translated into WSDL. One important function, we believe, of a broker is to rank the recommended services according to their semantic relevance to the request. Ranking web services is critical for the wide acceptance and practical applications. At this stage, web services ranking is still in its early days of research.

2. Web Services Description

Web services description serves an important role in discovering and ranking relevant web services. The overview of the proposed system is shown in Figure 1. The Web Services are published in UDDI by the service providers. The UDDI entry is complemented with semantics description. The semantics description of Web Services is described. The UDDI publisher API is used to publish the Web Services. The discovery process is initiated by the broker. The broker uses the UDDI inquiry API to obtain information about relevant Web Services. In the proposed technique, the broker will obtain semantics description of Web Services in addition to the basic information supplied by UDDI.

After the discovery process, the broker is able to create agents for invocation of Web Services. The broker models the agent according to the semantics description gathered during the discovery process.
To match to suitable web services, a language is needed to describe the request. In this section, a language for the semantics description of Web Services is presented. The syntax of the proposed language is based on factors required to represent knowledge in Web Services description. The following factors are identified to describe the semantics of Web Services [4].

- **Inputs**: This part specifies the objects that a Web Service takes as inputs;
- **Outputs**: This part of the representation specifies the objects that will be the outputs generated by this service;
- **Input Constraints**: This part defines the constraints that expected to hold before this capability can be applied, i.e. the preconditions of this capability;
- **Output Constraints**: This defines the postconditions after the capability has been performed;
- **Input-Output Constraints**: This part defines the conditions that should hold across the input and output;
- **Privacy**: Privacy should also be considered, since some service providers or consumers may not want their identities to be revealed to others, whom they know nothing or little about;
- **Quality**: Quality is always a concern of service consumers. Different service providers might provide the same service, but the qualities of their services may vary a lot. By specifying the quality of the required service, consumers have the choice of selecting the service according to their requirement.

The factors identified above are represented in an XML language which has the structure shown below. The root element is “sdlws”, which represents the “semantics description language for Web Services”. Ontologies form the basis for shared conceptualization of a domain [7]. The above language uses terms defined in a domain ontology, which specifies domain concepts and their relationships. The language is independent of any condition language, which enables developers to choose any suitable language to describe the constraints for the Web Services.

```
<acdl> ::= <sdlws>
    <condition-language>
    <name>
    </condition-language>
    <communication-language>
        <value term=""> <name> </value> +
    </communication-language>
    <input>
        <value term=""> <name> </value> +
    </input>
    <preconditions>
        <expression>
            <condition> </expression> +
        </preconditions>
    <output>
        <value term=""> <name> </value> +
    </output>
    <postconditions>
        <expression>
            <condition> </expression> +
        </postconditions>
    |
        <quality>
            <value term=""> <name> </value> +
        </quality>
    |
        <privacy>
            <value term=""> <name> </value> +
        </privacy>
    </sdlws>
    <name> ::= <identifier>
    <condition> ::= <expression in condition-language>
```

### 3. Web Service Match

Based on the previous service description language, the signature of a web service is defined as following. In this definition, the sequence of variables types, specified in domain ontology terms, taken and given by a web service decides its signature.

**Definition 1 Web Service Signature** The signature of a web services is the sequences of types of its input and output variables.

#### 3.1. Signature Match

**Definition 2 Signature Match** Let $C$ be a service description in ACDL (Agent Capability Description Language) containing: an input specification $I^C$ containing the variables $v_{1},..., v_{n}$ and output specification $O^C$. Let $T$ be
a service request in AC
D with input specification $I'$ containing variables $u_1, \ldots, u_m$, and output specification $O'$. $C$ is a signature match web service of $T$, if $I' \subseteq \Delta I'$ and $O' \subseteq \Delta O'$

where $I' \subseteq \Delta I'$ means for all $v_i$ in $I'$, there exists $u_i$ in $I'$, and for $i \neq k$, $u_i \subseteq \Delta v_i$, we have $u_i \subseteq \Delta v_i$, for $j \neq l$.

In the above definition, if we have $t_i \subseteq \Delta t_i$, we know that the variables of each type also have a subtype relation, which we denote as: $u \subseteq \Delta v$. This match strategy is the basic and necessary requirement in the process, since without signature compatibility a requested service's input variables can not be accepted by the provider.

3.1.1. An Example of Signature Match

Following an example is given to illustrate the above definitions and the signature algorithm. Considering the following agent capability description of SportsFinder [6], an information agent to extract sports results from the web: 

(capability
  :cap-id SportsFinder
  :constraint-language fopl
  :input ( (Team ?team) )
  :input-constraint ( (elt ?team TeamName) (CompeteIn ?team Sports) )
  :output ( (TeamScore ?score) )
  :output-constraint ( (HasScore ?team ?score) )
)

The above description shows that SportsFinder can find out the scores of a sports team. Suppose the mediator has already received the above service advertisement. Then some information agent sends the following service request to the mediator:

(capability
  :cap-id SoccerResult
  :constraint-language fopl
  :input ( (SoccerTeam ?soccer_team) )
  :input-constraint ( (elt ?soccer_team TeamName) (CompeteIn ?soccer_team Soccer) )
  :output ( (Score ?result) )
  :output-constraint (HasScore ?soccer_team ?result)))

When applying the signature match algorithm on these two service descriptions, we have input variable soccer_team as a subtype of team and output type TeamScore as a subtype of Score, thus SoccerResult is signature matched against SportsFinder. That means the service of SportsFinder can take the variables of the request as input, and its output is compatible with the variables' types of request. From this example it is easy to understand that signature match relationship is not commutative. For the above two descriptions, service SportsFinder is not signature matched against SoccerResult, though vice versa.

3.2. Constraint Match

Definition 3 Constraint Match Let $C$ be a capability description in AC
D with input constraints $C_i^f = C_{i1}^f, \ldots, C_{ik}^f$ and output constraints $C_o^f = C_{o1}^f, \ldots, C_{ok}^f$. Let $C_i^f = C_{i1}^f, \ldots, C_{ik}^f$ and $C_o^f = C_{o1}^f, \ldots, C_{ok}^f$ be the input and output constraints respectively of service $T$. We define $T$ is constraint matched with $C$ if

$C_i^f \subseteq C_i^o$ and $C_o^f \subseteq C_o^o$

where $\subseteq$ denotes the $O$-subsumption relation between constraints. For $C_i^f \subseteq C_i^o$ means for all $C_{ik}^o$ in $C_i^o$ there exists $C_{ik}^f$ in $C_i^f$ that $C_{ik}^f \subseteq C_{ik}^o$ and for $i \neq k$, $C_{ik}^f \subseteq C_{ik}^o$, and $C_{ik}^f \subseteq C_{ik}^o$, we have $i \neq l$.

In this definition a $O$-subsumption relation between constraints is introduced, the definition is given next.

3.2.1. $O$-subsumption between Constraints

Following we will explain the $O$-subsumption relation between constraints. Since all the constraints are given in constraint-language, the details of $O$-subsumption depends on the constraint-language. In first order predicate logic, which we used as the constraint-language in our examples, constraints are a set of clauses. In [13] the definition of clause $O$-subsumption is given as follow.

Definition 4 Clause $O$-subsumption A clause $C$ is $O$-subsumed by another clause $D$, denoted as $C \subseteq O D$, if there exists a substitution $\Theta$ such that $C$ is a subset of $\Theta(D)$.

For example, (CompeteIn Tigers NBL) $\subseteq$ (CompeteIn ?team NBL) which means when we substitute the variable ?team with a constant Tigers, a Melbourne Basketball team, the two clauses are equivalent.

$O$-subsumption between two constraints, each a set of clauses, is defined in terms of subsumption between single clauses. More specifically, let $C^f$ and $C^o$ be two constraints. Then we define $C^f$ $O$-subsumption $C^o$, denoted as $C^f \subseteq O C^o$, if every clause in $C^o$ is $O$-subsumed by a clause in $C^f$.

3.2.2. An Example of Constraint Match

It is straightforward to have an algorithm to check if two capability descriptions are constraint matched. Next an example is given for constraint match. Consider the
following capability description which has been registered to the mediator.

\begin{verbatim}
{capability
  :cap-id Golf
  :constant-language fopl
  :input ( (GolfPlayer ?player) (Tournament ?tour) )
  :input-constraint ( (elt ?player PlayerName) (CompeteIn ?player ?tour) )
  :output ( (Rank ?rank) )
  :output-constraint ( (HasRank ?player ?rank) (RankingIn ?rank ?tour) )
}
\end{verbatim}

The above advertisement describes an information agent with the capability to find out the position of a golf player in a tournament. Input variables are the player's name and the tournament that the player attended; output is the ranking of the golf player. Suppose an agent requests a service of finding a golf player's position in the PGA Golf Tournaments, then it will send a service request as following to the mediator:

\begin{verbatim}
{capability
  :cap-id GolfPGA
  :constant-language fopl
  :input ( (GolfPlayer ?player) )
  :input-constraint ( (elt ?player PlayerName) (CompeteIn ?player PGA) )
  :output ( (Rank ?rank) )
  :output-constraint ( (HasRank ?player ?rank) (RankingIn ?rank PGA) )
}
\end{verbatim}

According to the constraint definition, we know that if we apply a O-substitution: O: ?tour ← PGA to the constraints in Golf service description, the above two descriptions will have the same constraints. So the mediator can give that service request GolfPGA is constraint matched against Golf, that is the service of Golf is applicable to the request of GolfPGA. It is worth noting that we can use the same process as above to check the io-constraint fields of service descriptions.

### 3.3. Partial Match

If two services are not completely matched, but they have some functions in common, then we call this kind of match partial match.

**Definition 5** Partial Match Let C be a service description in our ACDL containing: an input specification \( f^C \) containing variables \( V_{11}^C, \ldots, V_{lC}^C \), and output specification \( O^C \) with variables \( V_{01}^C, \ldots, V_{0m}^C \), and Cs input constraints \( C_{i1}^C = C_{i1}^C, \ldots, C_{ik}^C \) and output constraints \( C_0^C = C_{01}^C, \ldots, C_{0k}^C \). Let \( T \) be another agent service with the correspondent description parts as: input \( f^T \) containing variables \( V_{11}^T, \ldots, V_{lT}^T \), and output specification \( O^T \) with variables \( V_{01}^T, \ldots, V_{0mT}^T \), and Ts input constraints \( C_{i1}^T = C_{i1}^T, \ldots, C_{ik}^T \) and output constraints \( C_0^T = C_{01}^T, \ldots, C_{0k}^T \). We define \( T \) is partially matched against \( C \) if there exist \( V_{i1}^T \) in \( f^T \) and \( V_{i1}^C \) in \( f^C \) that \( V_{i1}^T \leq V_{i1}^C \); there exist \( V_{01}^C \) in \( O^C \), and \( V_{01}^T \) in \( O^T \) that \( V_{01}^T \leq V_{01}^C \); there exist \( C_{i1}^T \) in \( C_i^T \) and \( C_{j1}^C \) in \( C_i^C \) that \( C_{i1}^T \leq C_{j1}^C \); and there exist \( C_{i1}^T \) in \( C_i^T \) and \( C_{j1}^C \) in \( C_i^C \) that \( C_{i1}^C \leq C_{j1}^T \).

The above definition means for two capability description, if some of their input, output variables have subtype relations, and there are constraint clauses in their input and output constraint specifications that are O-subsumptioned, these two services are partial matched. Semantically, in some circumstances, i.e. the unmatched variables and constraints are irrelevant; the partial matched service is applicable.

#### 3.3.1. Partial Match Algorithm

It is straightforward to give an algorithm to evaluate if two service descriptions are partial matched. Given two service descriptions \( C \) and \( T \) with their input, output and constraints specifications as in Definition 5, the algorithm checks both signature and constraint parts to decide if the two services are partially matched.

In the algorithm, we can also find out the matched variables and constraints in these two services. The matched variables and constraints will be used to sort discovered web services in the ranking process, which is described in next section.

### 4. Ranking Web Services

The above algorithms are designed to find relevant web services within the UDDI registry. With the dramatically growing numbers of available web services, the results from match algorithms are becoming beyond our acceptance. Most of the time, there are tens or even hundreds of services recommended as crucial to your service request. To solve this information over loading problem, a ranking algorithm is given to narrow down the range of web services.

The ranking process is going to decide which matched services are more semantically similar to the one requested. It calculates a semantics distance number, from 0 to 1, to present the similarity.

Consider the two service descriptions, \( C \) and \( T \), given
in Definition 2, where $C$ has an input specification $f^C$ containing the variables $v_i$, ..., $v_n$, and output specification $O^C$, and $T$ is a service request with input specification $f^T$ containing variables $u_1$, ..., $u_m$, and output specification $O^T$. In (1), function $F_2(C, T) \rightarrow [0, 1]$, is defined to calculate the similarity of input parts.

$$F_2(C, T) = \sum f_d(v_i, u_i) / |f^C|, i = 1, ..., |f^T|$$

where $f_d(v_i, u_i) \rightarrow [0, 1]$ is the type distance of $v_i$ and $u_i$ in domain ontology.

For output fields, function $F_3(C, T) \rightarrow [0, 1]$, can also be described based on the same idea to measure output type distance. To be concise, the details of the function are not given here. For input constraint parts, the function is defined in (2) as:

$$F_3(C, T) = (\sum f(C_{i,j}^C) + \sum f(C_{O,i}^O) / |C^C| + |C^O|)$$

where $C_{i,j}^C \leq 0$, $C_{O,i}^T \leq 0$, $C_{O,j}^C$, and $f(C_{i,j}^C, C_{O,j}^C) \rightarrow [0, 1]$, is the semantic distance of $C_{i,j}^C$ and $C_{O,j}^C$ in domain ontology.

For the final ranking the relevance value of $C$ and $T$ is given by $F(C, T) \rightarrow [0, 1]$ in (3):

$$F(C, T) = \sum F_2(C, T) / |C|, i = 1, ..., |C|$$

Applying the above functions in ranking process, the number of relevant web services is reduced, and we can sort the recommended services according to their values. This can significantly increase the accuracy of service discovery, which is vital to the practical application of web services.

5. Conclusion

This paper addresses the web service discovering problem. First I describe the problem and process of service discovering. An XML based language for web service description is presented. The constraint parts in this language are independent, which users can plug-in any constraint language, such as first order predicate logic language. This provides flexibility and compatibility. Multiple service match strategies are given according to different levels of requirements and various features of service advertisements and requests. In the match strategies, multiple attributes of the services and the features of service requirements and advertisements are considered. Multiple match methods give users more choices and they are flexible to deal with the changing environment. Based on domain ontology, a ranking process will reduce the burden of finding the most relevant web services. The ranking algorithm considers the semantics of each service, and it only recommends the most crucial ones to the requester. Combining service matching and ranking increase the accuracy of service discovery; and provides a practical solution for web services discovery.

References


