Vol. 46 No. 3 IPSJ Journal Mar. 2005

Regular Paper

A Multi-faceted Approach for Searching Web Applications

Sasiporn Usanavasin,[†] Takahiro Nakamori,[†] Shingo Takada[†] and Norihisa Doi[†,††]

Today the WWW contains not only a tremendous amount of information but also a variety of Web applications, such as online shopping. Searching for such applications can be difficult and time consuming because current keyword-based approaches do not always deliver results that match the user's intention. The key underlying problem is that keywords do not capture the semantics of the user's query and the functional capabilities of Web applications. This paper presents a multi-faceted approach for searching Web applications. In our approach, the search process is based on various facets, such as service functionality, item (product), result type, inputs/outputs, and the detailed information of items. It is based on matching queries and Web application profiles that are described in DAML-S. The matching process is augmented with the use of ontologies. The result of applying our approach to a set of Web applications resulted in a precision of 93% and a recall of 99%.

1. Introduction

Querying the Web can be difficult and time consuming because of the tremendous amount and variety of information being stored. Keyword-based search engines such as Google[3] are commonly used to search the Web. However, when a user performs the search, he/she may not be only looking for Web pages that contain just information but for some applications or services such as online shopping. In this paper, we call such applications as *Web applications* (*WA's*), i.e., Web services that provide user interface via Web browser. For example, when a user enters a query such as *I want to buy a book titled "Harry Potter"*, the search engine should return an online service that sells books titled "Harry Potter", and not a list of Web pages that contain information about books or book stores, or pages that contain "Harry Potter".

Keyword-based approaches are prone to low precision and recall. Irrelevant or incorrect results are returned to the user because the search results may contain keywords that appear in the user's query, leading to low precision. In addition, these keyword-based search engines do not recognize synonyms and other types of relationships between terms such as "*textbook* is a kind of *book*". Therefore, some relevant results can not be located, which lead to a decrease in recall. The key underlying problem is that keywords alone are insufficient for capturing the semantics or concepts of a query, especially when a query involves searching for something more than just information, such as Web applications.

The faceted classification approach[9] is a popular concept used in many work to retrieve software components for reuse[2],[10]. In this approach, software components are retrieved based on various aspects (or "facets") that have been defined in advance. Queries are made based on these facets. This approach can also be applied to the discovery of Web applications, where the matching of Web applications could be based on facets that express the essential information about the functional capabilities of the applications.

We have thus developed a semantic search engine that takes the concept of the faceted-based approach, and augmented it with Semantic Web technologies to maximize both the precision and recall of Web application retrieval.

Our approach exploits Semantic Web[1],[14] technologies such as DAML-S[13] to realize the semantics embedded in the user's queries and in WA profiles (which correspond to DAML-S *serviceprofile* ontology). We propose a multi-faceted approach for searching Web applications, where the search process is based on various facets, such as service functionality, item (product), result type, inputs/outputs, and the detailed information of items. It is based on matching queries and WA profiles that are described in DAML-S. The matching process is

† Graduate School of Science and Technology, Keio University
†† Faculty of Science and Engineering, Chuo University

augmented with the use of ontologies.

The contributions of this research are as follows. First, we extend the DAML-S *serviceprofile* ontology with the facets of *function type*, *item*, and *result type*, which can be used to describe more precisely the functional capability and characteristics of a WA. With such information, the system can retrieve WA's that are similar to the desired application that is specified by the user. Second, we provide two levels of matching processes, called basic matching and detailed matching, so that our system not only delivers WA's based on the desired functional capability and characteristics (performed by the basic matching process) but is also capable of helping the user to find a specific product or item by matching the user's given *detailed information* about the item or product, such as book title or hotel name, with product information provided by the discovered WA's (executed during the detailed matching process).

The paper is organized as follows. Section 2 gives an overview of our approach and then Section 3 describes the implementation. Section 4 evaluates our approach. Some related works are described in Section 5. Section 6 makes concluding remarks.

2. Our Approach: Multi-faceted Matching of Web Applications

To be able to locate WA's that most satisfy the user's needs, the search engine must recognize the essential semantic concepts or knowledge (facets) of WA's. Thus, our objective can be framed as being able to apply Semantic Web technologies to the search process in order to enable the comprehension of concepts or knowledge embedded in the queries and in WA profiles.

2.1 DAML-S Expressions of Multiple Facets

Our approach relies on the use of ontologies to semantically describe queries and Web applications. Similar to the area of knowledge sharing, we use the term "ontology" to refer to *a specification of a conceptualization* 4). An ontology contains a set of classes that represent concepts or knowledge. Each class has an associated set of properties. Each property has a range indicating a restriction on the values the property can take. An ontology relates more specific concepts to more general ones (from which generic information can be inherited). Such links have been variously named "is a," "subset of," "member of," "sub-concept of," "superconcept," etc. Such links are used to organize concepts into a hierarchy called a "taxonomy."

Table 1 Multiple facets based on DAML-S Profile properties.

Facets	Properties of DAML-S profile
function type	profile:serviceCategory
item (product)	profile:serviceParameter
result type	profile:serviceParameter
inputs	profile:input
outputs	profile:output

DAML-S is an ontology of services and based on this, a service is characterized by three types of information: a *serviceprofile* (or *Profile*), *processmodel*, and *servicegrounding*. The *serviceprofile* describes "what the service does." The *processmodel* describes "how the service works" and the *servicegrounding* specifies the details of "how a service can be accessed." Generally speaking, the *Profile* provides the information needed for service discovery. In this study we only incorporate the use of the *serviceprofile* to support our *multi-faceted matching*, therefore, we will omit the discussions of the *processmodel* and the *servicegrounding*.

Our *multi-faceted matching* is based on matching user's queries and WA's based on multiple facets. To support the *multi-faceted matching*, we use the DAML-S *Profile* ontology to describe a user's query and a WA, and utilize an appropriate `property` of a class *profile* to express each facet. We now describe each of the facets used in our approach. **Table 1** shows the *profile* properties that are used to describe these facets.

Function type facet: The *function type* facet identifies the type of functionality that a WA provides to users. This is used to differentiate between various types of WA. To support the semantic search based on the *function type* facet, we have developed a *functionType* ontology that currently contains seven types of service functionalities that are often encountered in today's applications: *Buy_Function*, *Sell_Function*, *Quote_Function*, *Search_Function, BuyByAuction_Function*, *SellByAuction_Function*, and *Reserve_Function*. We have reviewed more than 50 leading Web applications such as amazon.com, expedia.com and yahoo.com. All of their offered services can be classified as one of the function types specified in our ontology. We do not claim that these seven types are sufficient, but new types can be added as necessary to accommodate new unclassifiable services without making any major changes to our approach.

In DAML-S *Profile*, the `serviceCategory` property is used to refer to an ontology of services that may be offered[13]. This ontology could include classification based on service functionality. Therefore, we use the `serviceCategory` property to describe the *function type* facet.

Item facet: *Item* is an object or product that is handled by a service or function of a WA. If the WA provides an auction service, the object to be auctioned is considered as an *item*. This facet is important for identifying what product the user is looking for. An *item* is described through a concept (a class) that is defined in ontologies according to application domains, e.g., *book* is a concept that is defined in the *Publication* ontology and can be used as an *item* for online book selling applications, while *car* or *truck* which are defined in the *Vehicle* ontology can be used as an *item* for car selling or quoting applications. We have developed several ontologies for this purpose.

The `serviceParameter` property of the DAML-S *Profile* represents an expandable list of information that may accompany a profile description of a service[13]. Currently, there are no range restrictions placed on the `serviceParameter`. Therefore, we expand the `serviceParameter` property to describe the *item* facet.

Result type facet: The *result type* facet describes the form in which the user expects to obtain the item. The final result of executing a WA can be in various forms, such as downloadable data, displayed information, or a physical product that is sent via postal mail. Information about the result type can greatly aid the search engine to find the correct type of WA. Similar to the item facet, the result type facet is also described by the `serviceParameter`. We have defined five result types:

(1) *Display_information* — The final result is information that is displayed on the Web browser.

(2) *Download_file/document/data* — The final result is a file, document, or data that is downloaded.

(3) *Physical_item/document/data* — The final result is some type of physical item such as a book or catalogue that is physically sent to the user.

(4) *Inform* — The final result is some type of information that the user receives via e-mail.

(5) *Get_rights* — The final result is a granted right, for example, access to a database.

Input facet: Information that is needed to execute the WA is described as *inputs* to the WA. Inputs correspond to attributes of the item, which are described by properties that are associated with the class that represents the item. For example, properties of a class *Book* such as *title*, *author*, *code* (*ISBN*), *subject*, *publisher*, and *language* in the *Publication* ontology are considered as attributes of the item *Book*. Thus, a book selling service may require the user to enter the title and the names of the authors as inputs. As shown in Table 1, *input* facet is described through the `input` property of the DAML-S *Profile* within requests and WA profiles.

Output facet: An *output* facet describes the outcome of a WA after execution. An output can be an attribute of an item or something else that is declared by the WA providers. For example, the outputs of a book selling service could be the price of the book, the shipping order number, and an online receipt. Output is declared with the `output` property. The value of the output facet takes a secondary role depending on the value of the result type facet: specifically, when the result type facet value is either *Download_file/document/data*, *Physical_item/document/data*, or *Get_rights*, the most important result is the item which is either downloaded, sent physically, or whose right is obtained. In these cases, the output facet takes the role of providing some kind of supporting information to the user.

Moreover, we consider not only retrieving relevant applications/services for the user but also finding the specific item or product that the user requests by performing further matching of ***detailed information*** about the item, such as matching the title of book. This can further aid the user in performing the desired task, such as auctioning or buying this particular book. *Detailed information* is considered as a value to the attributes of the item in question. For example, "Harry Potter" can be considered as the *detailed information* to the attribute *title* of the item *book*.

Suppose a user gives the query *I want to buy a book titled "Harry Potter"*. Our system interprets this query and recognizes that

```
<profile:input>                                                          (1)
  <profile:ParameterDescription  rdf:ID="bookTitle">                      (2)
    <profile:ParameterName>bookTitle</profile:ParameterName>              (3)
    <profile:restrictedTo  rdf:resource="&publication;#Title"/>           (4)
    <profile:refersTo   rdf:resource="&process;#title"/>                  (5)
  </profile:ParameterDescription>                                         (6)
</profile:input>                                                          (7)
<bookTitle> Harry Potter </bookTitle>                                     (8)
```

Fig. 1 DAML-S description of input and detailed information.

the user wants to find a book-selling service that takes `title` as an `input` to its service, where the value of the "*detailed information*" for this `title` is *Harry Potter*. This results in the DAML-S descriptions of the input and the detailed information to be generated (**Fig. 1**).

Each input parameter of a Web service is described within `profile:input` (lines 1–7). `profile:ParameterDescription` gives an ID (line 2) of the input and associates the input name ("bookTitle" in line 3) to the value of the parameter (line 4). `profile:restrictedTo` (line 4) restricts the range of the parameter to the class *Title* in *Publication* ontology. `profile:refersTo` (line 5) is a reference to the input parameter in the *processmodel*. The *detailed information* is described by an instance of the input parameter (line 8).

2.2 Matching Methodology

WA's can be expressed based on the facets using DAML-S as described in the previous section. A user query can similarly be expressed with the facets.

Our system searches for WA's by matching facets, but this is divided into two parts: (1) basic matching (based on *function type*, *item*, *result type*, and *input/output* facets), and (2) detailed matching (based on *detailed information* of the item).

The main reason that we present two levels of matching process is to distinguish our work from other Web service search engines in a way that our system is not only able to deliver the requested WA's to the user but is also capable of facilitating the user to find some particular item or product that is offered or handled by those matched WA's as well. This way the user does not have to waste time with the WA's that do not provide such desired information or products.

We thus set the objectives of each of the matching processes as follows. First, the objective of the basic matching process is to discover WA's that match the functional capabilities and characteristics with the description given in the user request. Then in the second part, if the user is also looking for some specific information or items, the detailed matching process is then carried out further to help the user to find such information or items that are offered by those discovered WA's. By this division, we can eliminate the search time that may be wasted in matching the detailed information on those irrelevant Web services. The use of two levels will be more efficient than using just one level search based on Web service matching or detailed information matching alone. For example, if a user wants to find some hotel reservation services, based on the request the basic matching process will retrieve WA's that offer reservation as its function type and hotel as its handled item. Moreover, if the user is looking for some specific hotel by giving the hotel name then the detailed matching process will use this specific hotel name on the discovered hotel reservation services and return the result to the user.

2.2.1 Basic Matching Process

The basic matching process is based on calculating similarity scores between corresponding facets in a query and WA, and filtering out irrelevant WA's.

The basic matching process consists of five sub-matching processes: (1) Function type matching, (2) Result type matching, (3) Item matching, (4) Output matching, and (5) Input matching. The first two sub-matching processes conduct exact matchings, i.e., the WA profile either exactly matches the query, or it doesn't match resulting in the WA profile to be filtered out. The next three sub-matching processes are based on calculating similarity scores. The calculations of these similarity scores are explained in detail in Section 3.

The basic matching process ensures that the delivered results conform (at least to a certain level) to the user's requests, by fulfilling the following three conditions:

(1) the matched WA handles the type of service that the user wants,

(2) the matched WA can produce results that satisfy the user's need, and
(3) the user can provide information that is needed to execute the matched WA.

2.2.2 Detailed Matching Process

The goal of this process is to perform further matching of WA's based on *detailed information* about the item that is given in the query. The *detailed information* is the value (s) to the inputs of the WA profile. For example, given a query *I want to purchase a book about "Semantic Web"*, the information within the double quotes (" ") can be considered to be the *detailed information* about the desired item.

The detailed matching process is carried out after the system completes the basic matching process, but only when *detailed information* is present in the query. Generally, such *detailed information* about items or products is not provided within the WA profile; indeed they may actually be part of the input that is used to execute the WA. Therefore, we send a request to the WA, and analyze the results returned from the WA to check for its validity. To make the detailed matching feasible, we require that the WA provides an API for interacting with our engine and its returned message from the WA is described based on XML based knowledge markup languages. Details of this analysis and the similarity calculation of the *detailed information* are given in Section 3.2.3.

3. Multi-faceted Web Application Search Engine

This section gives the details of the implementation of our Multi-Faceted Web Application Search Engine, focusing on the two main modules: (1) the User Interface and Query Construction Module and (2) the Matching Engine (**Fig. 2**).

3.1 User Interface and Query Construction Module

We use a web browser as the user interface for the user to input a simple natural language based query. Our query construction module then takes this natural language query and converts it into a DAML-S based query. The query construction module takes a simple approach by taking the input query string and extracting four types of information (*function type*, *item*, *inputs*, and *detailed information about the item*) through keywords.

To extract the information from the natural language query and construct a DAML-S query,

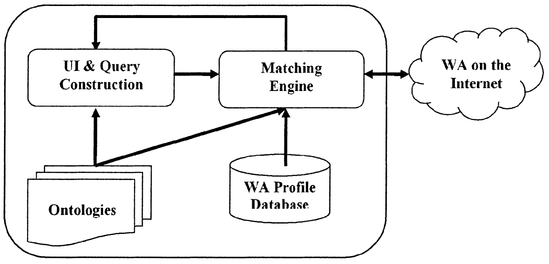

Fig. 2 An overview of the multi-faceted Web application search engine.

our system breaks the query into words (tokens) as shown in **Fig. 3**.

Once the query string is broken up into words, our system will read each word and compare it with two lists of keywords: *Service Functionality Lookup List* and *Item Lookup List.*

The *Service Functionality Lookup List* contains a list of words that may represent a service function and the *Item Lookup List* is composed of a list of words that may represent items or specify the item attributes. **Table 2** and **Table 3** show parts of our lookup lists that are supported by our current prototype system.

The procedure for extracting information from the natural language query is as follows:

(1) The system searches for a word that represents a function type by comparing each word in the query with words in the *Service Functionality Lookup List.* In the example given in Fig. 3, the word "purchase" tells the system that the user wants to find a WA that provides a "sell" function☆.
(2) The system scans for words that may represent an item or product by comparing words that follow the word "purchase" in the query with words in the *Item Lookup List.* In our example, the word that may represent the item is "book".
(3) After the system recognizes that the item is "book", it continues to scan for words that identify attributes of the book. In our example, the user gives two pieces of information about the desired book, i.e., the book title and the author name. The information that is given within double quotes right after the words "titled" and "written by" are the detailed informa-

☆ The function type is determined from the viewpoint of service providers, that is the word "buy" or "purchase" represents Sell_Function while the word "sell" specifies "Buy_Function".

Purchase | the | book | titled |"| Harry | Potter |"|,| written | by |"| J.K. Rowling |"|.|

Fig. 3 A natural language query string is broken up into words.

Table 2 Service functionality lookup list.

funtionType	**Lookup Words**
Buy_Function	sell, selling
Sell_Function	buy, buying, purchase, purchasing
Search_Function	search, searching, retrieve, retrieving, find, finding
Quote_Function	quote, quoting, check the price
Reserve_Function	reserve, reservation
SellByAuction_Function	buy by auction
BuyByAuction_Function	sell by auction

Table 3 Item lookup list.

Ontology	**Item**	**Attributes**	**Item Attributes Lookup Words**
Publication	Book	Title	title, name, named, titled, by title
		Author	write by, written by, by author
		Language	write in, written in, by language
		Subject	about, subject, by subject
		ISBN	ISBN, code, by ISBN
	Magazine	Title	title, name, named, titled, by title
		Subject	about, subject, by subject
		ISSN	ISSN, code, by ISBN
		Issue	issue on, by issue
Movie	DVD	Title	title, name, named, titled, by title
		Director	direct by, directed by, by director
		Actor	act by, acted by, by actor
		Genre	genre
		Region	for region

tion of the *Title* and *Author* attributes respectively.

(4) After the system finishes scanning the query and has all relevant information about a requested WA, it then constructs a DAML-S query based on the extracted information.

Figure 4 shows the DAML-S query that was constructed for our example and **Table 4** shows it in a tabular form.

3.2 Matching Engine

The Matching Engine conducts the basic matching process between the DAML-S based query and WA profiles as well as the detailed matching process, which were described conceptually in Sections 2.2.1 and 2.2.2. The function type matching and result type matching in the basic matching process are exact matches, so we omit their discussion. We focus on the calculation of the similarity scores for the item matching, output matching, input matching, and detailed matching.

The reason that we perform exact matches on *function type* and *result type* facets is due to the clear and distinctive definitions that are used to describe each type of function and result type. For *function type*, when the user inputs a query in natural language, the system will scan for words in the query that represent type of function and map it to one of the *function type* concepts that are defined in our *functionType* ontology. For example, if the user uses the word "purchase" or "buy", the system will map it to *Sell_Function*, which is a distinctive type that has no similarity with other functions defined in the ontology. Therefore, we can use exact matching for *function type*. Even if we use similarity matching at this step, the result would be the same but it will take more time for the search. The same argument can be said for the result type matching. The user selects the type of the result he/she wishes to obtain and the system will map it to the corresponding *result type* concept that is defined in the ontology. Since there are no overlaps in the definitions used to describe *result type*, exact matching would give the best result.

However, as for the item, input, output, and detailed matching, the terms that are used to describe each facet are different by WA

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!-- Name Space Declarations go here -->
 ...
 <service:ServiceProfile rdf:ID="anonymous">
         <profile:serviceName> anonymous </profile:serviceName>
         <profile:serviceCategory rdf:resource="&category;#Sell_function"/>
         <profile:serviceParameter rdf:resource="&category;#Physical_item_document_data"/>
         <profile:serviceParameter rdf:resource="&publication;#Book"/>
         <profile:textDescription>
          Purchase the book titled "Harry Potter", written by "J.K. Rowling".
         </profile:textDescription>
           <profile:input>
            <profile:ParameterDescription rdf:ID="bookTitle">
                  <profile:parameterName> bookTitle </profile:parameterName>
                  <profile:restrictedTo rdf:resource="&publication;#Title"/>
                  <profile:refersTo rdf:resource="&process;#title"/>
            </profile:ParameterDescription>
           </profile:input>
           <bookTitle> Harry Potter </bookTitle>
           <profile:input>
            <profile:ParameterDescription rdf:ID="bookAuthor">
                  <profile:parameterName> bookAuthor </profile:parameterName>
                  <profile:restrictedTo rdf:resource="&publication;#Author"/>
                  <profile:refersTo rdf:resource="&process;#author"/>
            </profile:ParameterDescription>
          </profile:input>
          <bookAuthor> J.K. Rowling </bookAuthor>
          <profile:output>
            <profile:ParameterDescription rdf:ID="EReceipt">
                  <profile:parameterName>EReceipt</profile:parameterName>
                  <profile:restrictedTo rdf:resource="&concepts;#EReceipt"/>
                  <profile:refersTo rdf:resource="&process;#EReceipt"/>
            </profile:ParameterDescription>
          </profile:output>
          <profile:output>
            <profile:ParameterDescription rdf:ID="ShippingOrder">
                  <profile:parameterName>ShippingOrder</profile:parameterName>
                  <profile:restrictedTo rdf:resource="&concepts;#ShippingOrder"/>
                  <profile:refersTo rdf:resource="&process;#ShippingOrder"/>
            </profile:ParameterDescription>
          </profile:output>
           ...
 </service:ServiceProfile>
</rdf:RDF>
```

Fig. 4 Sample DAML-S query.

Table 4 Extracted semantics from the user's query.

DAML-S Profile Properties	Facets	Info. extracted from query
profile:serviceCategory	function type	Sell_Function
profile:serviceParameter	item (product)	book
profile:serviceParameter	result type	Physical_item/document/data
profile:input	inputs	title, author

	detailed information	title = Harry Potter
		author = J.K. Rowling

providers and users. For example, *book* may be defined as *magazine*, *textbook*, *comic book*, or many other words that exist to represent a written article that can be defined as a "book". If we use exact match for *item*, when the user is looking for a textbook while the WA provider defined their product as just "book", then the system will not be able to retrieve those WA's, even though they may be relevant to what the user wants. The same argument can be said for input, output and the detailed matching.

3.2.1 Item Matching

Item matching is carried out by calculating the similarity scores between the *item* facet of the query and the *item* facet of the (remaining) WA profiles, and then filtering out the WA

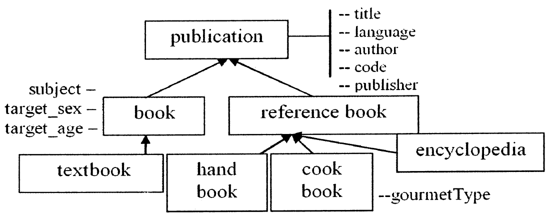

Fig. 5 Ontology example: Publication.

profiles that have similarity scores less than a specified threshold value. We are currently using 50% as the threshold, where 100% would be exact match. This value is based on several trial experiments. Note that the threshold may be changed according to the user's needs. With the similarity matching between two items, WA's that do not exactly match the request but have high similarity scores are also left as candidate results. The similarity score for item matching is calculated based on two viewpoints: *similarity of the concepts based on ontology hierarchy* and *similarity of the attributes that are associated with a concept.*

We determine the similarity of concepts based on the number of superclasses that are shared between the concepts within an ontology. **Figure 5** shows part of the *Publication* Ontology we have developed. In Fig. 5, the concepts *book* and *reference book* are defined as kinds of the concept *publication*, *textbook* is defined as a kind of *book*, and *cookbook* is defined as a kind of *reference book.* If the *item* in the query is a *textbook*, based on the viewpoint of the number of shared superclasses in the ontology hierarchy, the semantic similarity of *book* is higher than that of *cookbook.* Therefore, the WA's that have *book* as the value for the *item* facet are more relevant to the user's need than those that publish their *item* as *cookbook* or *reference book.*

More precisely, the concept similarity (CS) value is calculated based on Concept Match and Upwards Cotopy, proposed by Maedche, et al[6]. We have made a subtle extension to their definition in order to be able to compute between concepts, as Maedche, et al defined the equations for computing similarity between what they call "instances". CS is calculated using Eqs. (1)–(3). Details of the original equations are found in Ref. 6).

$$CS(c_1,c_2) = \begin{cases} 1 & if\ c_1 = c_2 \\ \frac{CM(c_1,c_2)}{2} & \text{otherwise} \end{cases} \quad (1)$$

$$CM(c_1,c_2) = \frac{|(UC(c_1,H^c)) \cap (UC(c_2,H^c))|}{|(UC(c_1,H^c)) \cup (UC(c_2,H^c))|} \quad (2)$$

$$UC(c_i,H^c) = \{c_j \in c | H^c(c_i,c_j) \vee c_j = c_i\} \quad (3)$$

where

- c_i is a concept defined in the ontology.
- $CM(c_1,c_2)$ is the concept match between the concepts c_1 and c_2.
- H^c is the concept hierarchy showing directed, transitive relations between concepts. Figure 5 is an example concept hierarchy for the *Publication* ontology.
- $H^c(c_1,c_2)$ is a predicate whose value is true if c_1 is a sub-concept of c_2.
- UC (Upwards Cotopy) is a set that contains c_i and its super-concepts within a given H^c.

We also calculate the semantic similarity of items based on the number of the same attributes and different attributes associated with the concepts that are defined in the ontology. For example, *textbook* shares more attributes with *book* than *cookbook.* Therefore, *textbook* is considered to be more similar to *book* than to *cookbook.* This is expressed in the following Eqs. (4) and (5), which are based on the work by Rodriguez, et al[11], and readers can refer to their work for details of the equation.

$$AS(c_1,c_2) = \frac{|C_1 \cap C_2|}{|C_1 \cap C_2| + \alpha(c_1,c_2)\cdot|C_1 - C_2| + (1-\alpha(c_1,c_2))\cdot|C_2 - C_1|} \quad (4)$$

$$\alpha(c_1,c_2) = \begin{cases} \frac{d(c_1,l.u.b)}{d(c_1,c_2)} & \text{where} \\ d(c_1,l.u.b) \le d(c_2,l.u.b) & \\ 1 - \frac{d(c_1,l.u.b)}{d(c_1,c_2)} & \text{otherwise} \end{cases} \quad (5)$$

where

- c_i is the concept defined in ontology.
- C_i is a set of attributes associated to concept c_i.
- $\alpha(c_1,c_2)$ is determined by the distance between the two concepts, $d(c_1,c_2)$.
- $d(c_1,c_2)$ is the distance between two concepts in the concept hierarchy, which is the number of steps it takes to get from concept c_1 to concept c_2. In our example, $d(textbook, publication) = 2$.
- $l.u.b.$ (least upper bound) is the immediate superclass that subsumes c_1 and c_2. When one concept is the superclass of the other,

the former is also considered to be the immediate superclass (*l.u.b.*) between them. For example (see Fig. 5), *l.u.b.* of concepts *textbook* and *encyclopedia* is *publication*.

The final similarity score (semantic similarity of item; SOI) is obtained by summing the concept similarity (CS) and the attribute similarity (AS) scores as shown in Eq. (6). The value of SOI ranges from 0 to 2, where 0 represents non-match and 2 represents exact match between the two items. Since we use 50% as the threshold, the SOI threshold is 1.

$$SOI = CS + AS \tag{6}$$

3.2.2 Output Matching and Input Matching

The similarity scores that are calculated during output matching are determined by the number of same *output* facet values expressed in the query and in the WA profiles (Eq. (7)). The calculation is done regardless of *result type* facet values; however the resulting similarity score will be used for filtering out irrelevant WA's only when the *result type* facet value is *Display_information* or *Inform*. This is because the *output* facet value is not the most important result that is wanted by the user when the *result type* facet takes other values. For example, when the *result type* facet value is *Download_file/document/data*, the most important result is the item (or file) that is downloaded; the output value in this case normally has some other supporting information. This is also the case for *Physical_item/document/data* and *Get_rights*. The value of output similarity score ranges from 0 to 1, where the value 1 means all outputs of the WA are the same as all outputs in the query. The threshold value that we used in output matching is 0.5.

$$\text{Output\_similarity} = \frac{\text{numberOfMatchedOutput}}{\text{numberOfOutputsInQuery}} \tag{7}$$

The similarity scores for input matching are similarly obtained (Eq. (8)). Note that the denominators for the two similarity scores are different. For output matching, the denominator uses the query, because the output is what the user wants. For input matching, the denominator is the WA profile, because the input is what the WA needs for it to be executed. Similar to the output similarity score, the value of input similarity score ranges from 0 to 1, where 1 means all inputs of the WA are the same as all inputs in the query. The threshold value that we used in input matching is also 0.5.

$$\text{Input\_similarity} = \frac{\text{numberOfMatchedInput}}{\text{numberOfInputsInWAProfile}} \tag{8}$$

3.2.3 Detailed Matching

An overview of the detailed matching process was given in Section 2.2.2. We describe how the WA is accessed through an example, using the DAML-S based profile of a bookstore, which was created based on our perception of the well known amazon.com online bookstore. DAML-S *servicegrounding* describes how to access the service. However, at the time of development of our system, the specification of *servicegrounding* was under the proposal stage. Therefore, in this study, we exploit the `communicationThru` property of DAML-S profile to describe the information of where to communicate with the WA. In future work when the *servicegrounding* specification is finalized, we can exploit it to improve the WA accessing method. **Figure 6** shows part of the amazon.com WA profile that advertises its service as a selling service for books. According to the information of where to communicate with this service (line 3) and the information about inputs (lines 4–17), we can construct a request message to this service and analyze its result in terms of the appropriateness for a user search query.

From the example query given in Section 3.1 (see Fig. 3), our system conducts the basic matching process. Suppose that the amazon.com profile is one of the remaining WA's. Our system sends a request to the service at *http://xml.amazon.com/onca/xml2* (line 3) with the supplied input values of title and author name as *ProductName = Harry Potter* and *Author = J.K. Rowling* respectively. The following shows part of the result returned from the service:

```
<Details>
  <Asin>0807281956</Asin>
  <ProductName>Harry Potter and the
     Sorcerer's Stone</ProductName>
  <Catalog>Book</Catalog>
  <Author>J.K. Rowling</Author>
     ...
</Details>
```

The ID of the input properties (lines 5 and 12) tells the system where to look for the detailed information of that particular `input`. In this case, the ID of the author name is *Author*

```
<service:ServiceProfile rdf:ID="Profile_Amazon_ServiceAgent">                (1)
...
<profile:webURL>http://www.amazon.com</profile:webURL>                      (2)
<profile:communicationThru>http://xml.amazon.com/onca/xml2                  (3)
</profile:communicationThru>
<profile:input>                                                             (4)
  <profile:ParameterDescription rdf:ID="Author">                            (5)
      <profile:parameterName>AuthorSearch</profile:parameterName>           (6)
      <profile:restrictedTo rdf:resource="&publication;#Author"/>           (7)
      <profile:refersTo rdf:resource="&Amazon_process;#bookAuthor"/>        (8)
  </profile:ParameterDescription>                                           (9)
</profile:input>                                                            (10)
<profile:input>                                                             (11)
  <profile:ParameterDescription rdf:ID="ProductName">                       (12)
      <profile:parameterName>ProductNameSearch</profile:parameterName>(13)
      <profile:restrictedTo rdf:resource="&publication;#Title"/>            (14)
      <profile:refersTo rdf:resource="&Amazon_process;#bookName"/>          (15)
  </profile:ParameterDescription>                                           (16)
</profile:input>                                                            (17)
```

Fig. 6 Sample WA profile of a bookstore.

and the ID of the book title is *ProductName*. With this knowledge, we can match these detailed information with those given in the query. The detailed matching score can now be calculated according to Eq. (9):

$$\text{Detailed\_Matching\_Score} = \sum \text{each\_detailed\_information\_score} \quad (9)$$

The detailed information score for each of the details are scored as follows: Exact Match = 1, Partial Match = 0.5, and No Match = 0.

3.3 An Example

To illustrate how our approach works, let's use the example in Fig. 4 as a request to our system. For simplicity, suppose there are eight WA profiles in the registry as shown in **Table 5**.

Based on our approach, our system firstly performs the function type matching between the query and each profile in Table 5. Since the *function type* facet that is specified in the query is *Sell_Function* and the function type matching is exact matching, the system filters out the profiles that have different types of function. The results that are obtained from the function type matching are profiles A, B, C, E, and J.

The system then continues performing result type matching, which is also exact matching. At this step, profile E whose *result type* is not *Physical_item/document/data* is filtered out while profiles A, B, C, and J are still left as candidate results.

The next step is to perform item matching. The system compares item *Book* (*item* facet of the query) with items of the candidate profiles and calculates the similarity scores. From the calculations, we obtain the item similarity scores of profiles A, B, C, and J as 2, 1.67, 0, and 0.76 respectively. Based on these similarity scores, the system then filters out profiles C and J whose similarity scores are less than the threshold (which is 1). Profiles A and B remain as candidate results and are passed to the next two steps, output and input matching.

The similarity scores of *output* of profiles A and B are 0.67 and 1 respectively. Based on our output matching technique as described in Section 3.2.2 and since the *result type* of profiles A and B are *Physical_item/document/data*, profiles A and B remain as the candidate results. The similarity score of the input matching of profiles A and B are 1 and 0.67 respectively. Also, these scores are greater than the threshold; therefore, profiles A and B are still left as candidate results. The last step is to perform the detailed matching. The system matches the *title* and *author name* of the request book whose values are "Harry Potter" and "J.K. Rowling" respectively with the information returned from Web applications A and B. If we assume that only profile A can return information about books that have similar title and author name to those of the request, profile A would be returned as the final result to the user.

4. Evaluation

We have performed an evaluation of our multi-faceted search engine using the following scheme. We created 160 DAML-S based WA profiles, each of which belong to one of three domain ontologies: *Publication*, *Vehicle*, and *Movie/Music*. The evaluation was carried out

Table 5 Sample WA profiles with their facets.

WA	functionType	resultType	Item	Output	Input
A	Sell_Function	Physical_item /document/data	Book	Price, EReceipt ShippingOrder	Title, Author
B	Sell_Function	Physical_item /document/data	Text Book	EReceipt, ShippingOrder	Title, Author, Subject
C	Sell_Function	Physical_item /document/data	DVD	EReceipt, ShippingOrder	Title, Actor
D	Quote_Function	Display_infor-mation	Car	CarDetail Price	Brand, Model
E	Sell_Function	Download_file /document/data	Book	EReceipt	Title, Author,
F	Buy_Function	Physical_item /document/data	Book	NoticeOfPayment	Title, ISBN
I	SellByAuction_Function	Physical_item /document/data	Book	NoticeOfPayment	Title, Author, ISBN
J	Sell_Function	Physical_item /document/data	Cook Book	EReceipt, ShippingOrder	Author, gourmet-Type

based on a set of 62 valid queries such as *search books by publisher*, *quote car price by brand and model*, *I want to buy a dictionary*, and *find DVD titled "Spiderman"*. By "valid", we mean that the queries belong to one of the three domains.

In this section, we divide the discussion into two parts. First, we discuss the evaluation method and results of our approach. Second, we compare our method with the keyword-based search using Namazu system[7].

4.1 Evaluation Method and Results

We evaluate our multi-faceted based search engine based on calculations of precision and recall of the returned results for 62 queries. The definitions we used for precision and recall are shown in Eqs. (10) and (11) respectively.

$$\text{Precision} = \frac{\text{NumberOfMatchedWA's}}{\text{TotalNumberOfRetrievedWA's}} \quad (10)$$

$$\text{Recall} = \frac{\text{NumberOfMatchedWA's}}{\text{NumberOfRelevantWA'sInRegistry}} \quad (11)$$

In Eqs. (10) and (11), "matched WA's" are those whose profiles represent the functional capability and characteristics that are similar (all facets match well with the query) to what is described in the query. This is done by manually checking the result of the retrieved WA's by comparing each retrieved profile with the user query.

For example, if the user queries a book selling application and our system retrieves 10 WA's but only 8 WA's services that sell books, then the precision is 0.8 or 80%. If there were 12 book selling WA's in the registry then the recall is 8/12 = 0.67 or 67%.

To find out what sorts of knowledge (facets) could be used to increase precision and recall of the searched results, we performed several experiments based on different combination of facets. Each combination was used to retrieve WA's based on a set 62 queries. **Table 6** gives the results.

The evaluation results shows that with the ability to recognize more knowledge (facets) about WA's, the search engine can return the WA's that are highly correct or relevant to the user's request. As we can see from Table 6, the average precision of the searched results increased as we added more facets to the search process.

4.2 Comparison with the Keyword-based Search

To verify that our approach is more effective than the keyword-based search approach, we compared our system with the keyword-based search engine of Namazu system. We performed an evaluation on the Namazu search engine based on the same set of 62 queries that were used in our system, resulting in average precision and recall of 54.2% and 91.6% respec-

Table 6 Precisions and recalls of multi-faceted matching approach.

Matching Approach	Facets	Precision (%)	Recall (%)
Multi-faceted search	I+O	25.6	97.1
	I+O+RT	28.7	97.1
	I+O+RT+item	31.9	98.2
	I+O+RT+item+FT	89.5	99.6
	I+O+RT+item+FT+DT	93.2	99.6

(I: input, O: output, RT: result type, FT: function type, DT: detailed information)

Table 7 Statistical evaluation of multi-faceted based vs. keyword-based matching approach.

	Namazu Precision	Namazu Recall	Our system Precision	Our system Recall
Average (%)	54.2	91.6	93.2	99.6
Standard Deviation	22.9	21.6	13.5	2.2

Table 8 Statistical variables for precision and recall based on ANOVA.

	Grand Mean	MSTr	MSE	f value
Precision	73.7	55534.5	707.7	78.5
Recall	95.6	2003.9	236.6	8.5

tively, which are lower than our average precision and recall of 93.2% and 99.6% respectively (see **Table 7**). This suggests that our multi-faceted based search engine is more effective in locating the correct WA's than the keyword-based search engine.

Moreover, to ensure the statistical significance of our proposed approach and to verify that our system performance actually outperformed the simple keyword based search such as Namazu system, we also performed statistical tests based on ANOVA (analysis of variance) technique. The statistical variables that are calculated based on ANOVA technique for both precision and recall are shown in Table 7 and **Table 8**.

In performing the ANOVA test, the samples' f values must be calculated and compared with the critical value of F distribution with confidence level α. The f value of our test set is calculated according to the following equation.

$$f = \frac{MeanSquareforTreatments(MSTr)}{MeanSquareforError(MSE)}$$

where,

$$MSTr = \frac{J}{I-1}\sum_i (X_i - X)^2$$

$$MSE = \frac{S_1^2 + S_2^2 + ... + S_i^2}{I}$$

S_i = Standard Deviation of sample set i

X = Grand Mean, which is an overall mean of all samples

X_i = Sample Mean

I = the number of methods being compared

J = number of queries in the test set

As shown in Table 7, the standard deviation of our system precision was 13.5, while that of Namazu was 22.9. The value of *MSTr* for precision was 55534.4, while *MSE* was 707.7. Thus the f value for precision was 78.46 which is much greater than the F value at 99% level of confidence under $F_{\alpha,I-1,I(J-1)}$ as $\mathrm{F}_{0.01,1,122} =$ 6.9. Based on this, we can state that the precision of our system is greater than that of Namazu system at 99% level of significance.

As for recall, the standard deviation of our system was 2.2, while that of Namazu was 21.6. The *MSTr* for recall was 2003.9, while the *MSE* was 236.6. The f value for recall was 8.5, which is also greater than $\mathrm{F}_{0.01,1,122}$ (6.9), thus we can state that the recall of our system is greater than that of Namazu system at 99% level of significance.

From these evaluations, we can conclude that with suitable facets, the multi-faceted based approach can return WA's that are more suitable to the user's needs than a keyword-based approach. However, we should note that we cannot conclude that the facets we have employed in this paper is enough, nor that there are not other useful facets. This is because, as the results show, the combination of facets affects the performance. We needed five facets (I, O, RT, item, and FT) before our approach was able to surpass Namazu.

The major factor that causes the difference in the results of our multi-faceted approach and the keyword-based approach is the lack of semantic concepts being used in Namazu. Namazu retrieves all WA profiles that contain the exact words presented in the query without understanding what each word refers to, e.g., function type, item, etc. For example, one of the test queries was *purchase book by ISBN*. Our system interpreted this to be the user looking for a WA that sells books and takes ISBN as input. But Namazu did not interpret in this way resulting in all WA profiles that contain the words "purchase", "book", and "ISBN", even though some of the retrieved WA's do not take ISBN as their input but return ISBN as an output. Due to this lack of semantic understanding, the precision of the search result from Namazu for this particular query was 75% while our system achieved 100% precision.

In terms of average response time, our system has an average response time of about 2 minutes while Namazu is about 5 seconds. The reason that our system takes longer is because our system performs many semantic comparisons for the six facets between the query and the WA profiles. Even though Namazu returns the results faster, based on the low precision of the keyword-based search, the user has to manually check and select the WA's that are relevant to his/her request. So, if we take the time that the user has to spend into account, the differences between our response times and those of Namazu will diminish. However, improving the response time is one of the issues for future work.

We should also note that comparisons between our system and other semantic based search systems, such as Refs. 12) and 8), are not possible at the moment due to many differences such as ontology, service description languages, the set of example WA profiles, and more importantly those systems are not stable as they are still under improvement. However, comparison may become possible in the future if those systems become stable and there are more available example sets of DAML-S profiles, which are compatible with our system and other similar systems.

5. Related Works

Much research has been devoted to the area of matchmaking of Web services 5),8),12). One of them is an ontology based approach that exploits the use of process model representations of service semantics and process ontologies to improve service retrieval 5). However, users are normally not concerned with how a retrieval process is carried out, thus limiting this approach.

Another approach is based on the matching of input/output descriptions between service advertisement profiles and requests, which are described in DAML-S, with the use of ontologies 8).

A third approach is based on a matchmaking process using an agent capability description language, LARKS (Language for Advertisement and Request for Knowledge Sharing) 12). The matching engine contains five different filters: Context matching, Profile comparison, Similarity matching, Signature matching, and Constraint matching. Different degrees of partial matching can result from utilizing different combinations of these filters. The context filter decides if two specifications are in the same semantic domain of the service by computing the semantic distances between words and the subsumption relations between the attached concepts of the pairs of most similar words. The profile filter applies TF-IDF (term frequency-inverse document frequency) technique to compare the two specifications. The similarity filter computes the distances of pairs of input and output declaration and their constraints. Signature matching uses a set of subtype inference rules and concept subsumption to test similarity of the input and output. The similarity of individual words in the description is taken into account in the similarity filter.

When we think of a Web application, we are not just thinking of inputs and outputs of the application but also its functionalities. Therefore, retrieving applications based on only input and output as done in Refs. 8) and 12) is not enough. Though the filters (which could be considered as "facets") in Ref. 12) perform semantic matches by determining a semantic distance between co-existent terms within shared ontologies, they still do not capture other important characteristics (facets) of WA's such as functionalities and their handled items. Due to the variety of applications available, different kinds of applications may have similar input and output requirements. Therefore, other knowledge (facets) about the WA could be used to increase the semantic understanding of the type of WA that the search engine should look for. Example knowledge are function type, item (product), result type, and the detailed information of the item.

6. Conclusion

This paper proposed a multi-faceted matching approach for WA's. To find the WA that is semantically similar to the user's query, we performed two matching processes, the basic matching process and the detailed matching process. These processes use information that are extracted from the query and are available in the WA profiles in the form of facets: *function type*, *item*, *result type*, *input*, *output*, as well as *detailed information* of the item. The information is described in DAML-S, and ontologies are used during the matching process. An evaluation of 62 queries on 160 Web application profiles resulted in an average precision of 93% and an average recall of 99%. We confirmed that this result was better than a simple keyword-based search.

For future work, we plan to enhance the UI by incorporating sophisticated natural language processing techniques and also utilize DAML-S *servicegrounding* to improve the WA accessing method. Finally, we also plan on improving our system to achieve better performance in terms of response time.

References

1) Berners-Lee, T., Hendler, J. and Lassila, O.: *The Semantic Web*, Scientific American, Vol.284, No.5, pp.34–43 (2001).
2) Eichmann, D.: A Hybrid Approach to Software Repository Retrieval: Blending Faceted Classification and Type Signatures, *3rd International Conference on Software Engineering and Knowledge Engineering*, Skokie, IL, pp.236–240 (1991).
3) *Google Search Engine*, http://www.google.com (2002).
4) Gruber, T.: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, Vol.5, No.2, pp.199–220 (1993).
5) Klein, M. and Bernstein, A.: Searching for Services on the Semantic Web Using Process Ontologies, *The Emerging Semantic Web-Selected papers from the first Semantic Web Working Symposium*, Cruz, I., Decker, S., Euzenat, J. and McGuinness, D. (Eds.), Amsterdam, IOS press, pp.159–172 (2002).
6) Maedche, A. and Zacharias, V.: Clustering Ontology-based Metadata in the Semantic Web, *Proc. Joint Conferences 13th European Conference on Machine Learning* (*ECML'02*) and *6th European Conference on Principles and Practice of Knowledge Discovery in Databases* (*PKDD'02*) (2002).
7) *Namazu Information Search*, http://search.namazu.org (2002).
8) Paolucci, M., Kawamura, T., Payne, T. and Sycara, K.: Semantic Matching of Web Services Capabilities, *International Semantic Web Conference* (*ISWC*), Sardinia, Italia (2002).
9) Prieto-Diaz, R.: Implementing Faceted Classification for Software Reuse, *Comm. ACM*, Vol.34, No.5, pp.88–97 (1991).
10) Prieto-Diaz, R. and Freeman, P.: Classifying Software for Reusability, *IEEE Software*, pp.6–16 (1987).
11) Rodriguez, M.A., Egenhofer, M.J. and Rugg, R.: Assessing Semantic Similarities among Geospatial Feature Class Definitions, *2nd International Conference on Interoperating Geographic Information Systems*, Vol.1580, pp.189–202, Springer-Verlag (1999).
12) Sycara, K., Lu, J., Klusch, M. and Widoff, S.: Dynamic Service Matching among Agents in Open Information Environments, *Journal ACM SIGMOD Record*, Vol.28, No.1, pp.47–53 (1999).
13) The DAML Services Coalition: *DAML-S: Semantic Markup for Web Services*, http://www.daml.org/services/daml-s/0.7/daml-s.html (2001).
14) *W3C: Semantic Web*, http://www.w3.org/2001/sw/ (2001).

(Received August 8, 2003)
(Accepted December 1, 2004)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.166–180.)

Sasiporn Usanavasin received her B.Sc. in Information Technology from Sirindhorn International Institute of Technology (SIIT), Thammasat University, Thailand in 1999. She recieved her M.E. in Information and Computer Science from Keio University, Japan in 2003. Currently, she is working towards the Ph.D. in Computer Science at Keio University. Her research interests include Semantic Web and Software Engineering.

Takahiro Nakamori received the B.E. in information and computer science from Keio University, Japan in 2003. Currently, he is working towards the M.E. in Computer Science at Keio University. His research interests include software engineering.

Shingo Takada received the B.E. in electrical engineering, and M.E. and Ph.D. in computer science from Keio University, Japan in 1990, 1992, and 1995, respectively. He is currently an assistant professor in the Department of Information and Computer Science, Faculty of Science and Technology, Keio University. His research interests include software engineering and information retrieval. He is a member of IPSJ, IEICE, JSSST, JSAI, ACM, and IEEE CS.

Norihisa Doi received the B.E., M.E., and Ph.D. in computer science from Keio University, Japan in 1964, 1966, and 1975, respectively. He is currently a professor of the Faculty of Science and Engineering, Chuo University, and also professor emeritus of Keio University. He is also a member of the Council for Science and Technology (MEXT), a member of the Council for Information and Communication (Ministry of Public Management, Home Affairs, Posts and Telecommunications), President of the Non-Profit Organization Japan Information Security Audit Association, and Chair of the Japan Chapter of the Association for Computing Machinery (ACM). In the past, he has been a member of the Science Council of Japan (1994–2003), Chair of the National Committee on Informatics, etc. His research interests include software and security. He is a member of various societies including IPSJ, IEICE, JSSST, JSAI, IEEE, and ACM.