# Public Deployment of Semantic Service Matchmaker with UDDI Business Registry

Takahiro Kawamura[1], Jacques-Albert De Blasio[1], Tetsuo Hasegawa[1], Massimo Paolucci[2], and Katia Sycara[2]

[1] Research and Development Center, Toshiba Corp.
[2] The Robotics Institute, Carnegie Mellon University

**Abstract.** Public deployment of the semantic service matchmaker to a UDDI registry for half a year is shown in this paper. UDDI is a standard registry for Web Services, but if we consider it a search engine, its functionality is limited to a keyword search. Therefore, the Matchmaker was developed to enhance UDDI by service capability matching. Then, we have deployed the Matchmaker to one of four official UDDI registries operated by NTT-Communications since September 2003. In this paper, we first introduce the Matchmaker with UDDI, and illustrate client tools which lower the threshold of use of semantics. Then, we evaluate this deployment by benchmarks in terms of performance and functionality. Finally, we discuss user requirements obtained by two ways: questionnaire and observation of search behaviour.

## 1 Introduction

The recent spread of web services reveals the needs of sophisticated discovery mechanism. Although UDDI[1] is emerging as a de-facto standard registry for web services, the search functionality provided by UDDI is the keyword search only on names, comments and keys of businesses and services descriptions. Unfortunately, the keyword search fails to recognize the similarities and differences between the capabilities provided by web services. To address this problem, we developed the Matchmaker in collaboration with Carnegie Mellon University[2], which enhances the search functionality of UDDI by making use of semantic information. Then, we initiated an experiment on a publicly available UDDI maintained by NTT-Communications, one of four official UDDI operators[3]. Since started from September 2003, we have evaluated the scalability and feasibility of our approach with performance and functional comparison and collected user requirements through half a year.

In this paper, we first introduce the Matchmaker in section 2, and the one deployed with UDDI in section 3. After illustrating supporting tools that provide user interfaces to the Matchmaker in section 4, we will present the evaluation in section 5.

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 752–766, 2004.

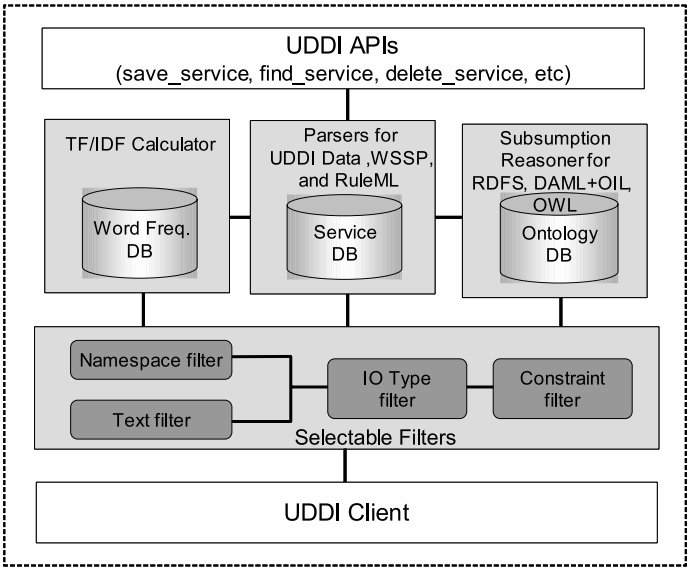


**Fig. 1.** Matchmaker

## 2 Semantic Service Matchmaker

Figure 1 shows the architecture of the Matchmaker.

### 2.1 Service Description

First of all, we mention Web Service Semantic Profile (WSSP), a way to encode semantic information with WSDL. WSDL specifies the programming interface of a service to be used at invocation time. To this account it specifies the name of the function to invoke and the types of data that the service expects as inputs and outputs. The data types describe how the information is formatted, but because there are arbitrarily many ways of encoding the same information in data types, they fail to express the semantics of the information, which is needed to express the capability of the service.

The aim of WSSP is to enrich WSDL with the basic semantic information to represent the services capabilities. As in WSDL we can define input and output parameters, the ontologies of those parameters are defined in WSSP. Further, WSSP offers the possibility to add constraints to those parameters to represent conditions for the service invocation more accurately. Since automatic service composition is out of our experiment for now, preconditions and effect (postcondition) will be our next priorities. An example of the WSSP is given below, in which the ontology for the parameter in RDFS[4], DAML+OIL[5] or OWL[6], facts and rules in RDF-RuleML[7] as the constraints, and the original WSDL parameter are encoded.

```
<profile:input>
  <profile:message rdf:ID="1st_INPUTMessage">

    <profile:parameter>
      <profile:ParameterDescription rdf:ID="1st_INPUTMessage_Param_1">
        <profile:parameterName>Param_1</profile:parameterName>
        <!-- Ontology -->
        <profile:restrictedTo rdf:resource="http://ont.com/Onto.owl#class1"/>
        <profile:wsdlParam>xPointer to wsdl parameter</profile:wsdlParam>
        <profile:datatype>XMLSchema.xsd#String</profile:datatype>
      </profile:ParameterDescription>
    </profile:parameter>

    <!-- Constraint -->
    <profile:constrainedBy rdf:resource="http://rule.com/rdfrules.rdf#fact1"/>
    <profile:constrainedBy rdf:resource="http://rule.com/rdfrules.rdf#rule1"/>

    <profile:wsdlMessage>xPointer to wsdl message</profile:wsdlMessage>

  </profile:message>
</profile:input>
```

We should note that WSSP has been inspired by DAML-S/OWL-S Service Profile[8]. Although DAML-S is very powerful to represent all the details about the service by combining Service Profile, Service Model and Service Grounding, it is difficult to use only the Service Profile without adopting the Process Model and the Grounding because of relationship among those three modules. Further, we would have redundancy if we use both the Service Profile and WSDL. Thus, we adopted the basic ideas of the Service Profile, and imported them into WSSP. Due to the highest priority on the standards, if any standard language will allow to express the same information as WSSP near future, we will be pleased to adopt it.

### 2.2 Matchmaking Mechanism

In brief we describe how the matching of capabilities is performed. Ideally, when the requester looks for a service, the matchmaker will retrieve a service that matches exactly the service that the requester expects. But, in practice it is very unlikely that such a service is available, instead the matchmaker will retrieve a service whose capabilities are *similar* to the capabilities expected by the requester. To address this problem, the matchmaker identifies the three levels of matching.

**Exact match** is the highest degree of matching, it results when the two descriptions are equivalent.

**Plug-in match** results when the service provided is more general than the service requested, but in practice it can be used in place of the ideal system that the requester would like to use. To this extent the result can be "plugged in" place of the correct match.

**Relaxed match.** The relaxed match has a weakest semantic interpretation. It is used to indicate the degree of similarity between the advertisement and the request.

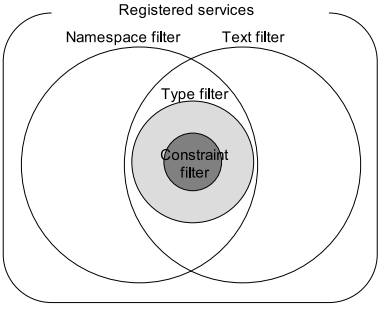


**Fig. 2.** Four filters

The second aspect of our matchmaker is to provide a series of four filters (Fig. 2), which are independent from each other, and narrow the set of registered services with respect to a given criterion. User may select any combination of these filters at the search time, considering the trade-off between accuracy and speed of the search. We briefly illustrate each filter as follows. Further details are provided in Sycara et al.[9]. The degree of the match are calculated as the sum of scores at the passed filters.

1. Namespace Filter

Pre-checking process which determines whether or not the requested service and the registered ones have at least one shared namespace (a url of an ontology file). The intersection of namespaces can be considered shared knowledge between the request and the advertisement. Therefore, only the registered services which have at least one shared namespace go into the next filter. Namespaces of default like rdf, rdfs, xsd, etc. are not considered the intersection. Of course, there is a case that the relation between two nodes of different ontology files does exist, although the distance would be relatively long. This filter and the next filter are meant for the reduction of the computation time of the last two filters.

2. Text Filter

Pre-checking process for human-readable service explanation parts such as comment and text descriptions. It utilizes the well-known information retrieval (IR) technique called Term Frequency Inverse Document Frequency (TF/IDF) method. This filter help to minimize the risk to miss the services which have any relation to the requested one.

3. I/O Type Filter

Type Filter checks to see if the ontologies of the input and output match. A set of subtype inferencing rules mainly based on structural algorithm are used to determine in this filter. Such a match is determined if: (Fig. 3)

- the types and number of input/output parameters exactly matches, OR
- outputs of the registered service can be subsumed by outputs of the requested service, and the number of outputs of the registered service is greater than the number of outputs of the requested service, AND/OR

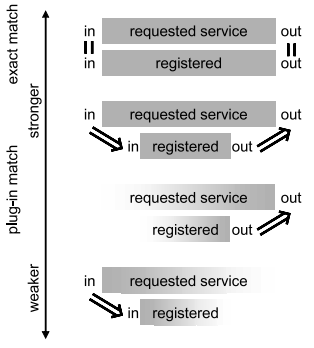


**Fig. 3.** Levels of plug-in match

– inputs of the requested service can be subsumed by inputs of the registered service, and the number of inputs of the requested service is greater than the number of inputs of the registered service.

If there is a mismatch in the number of parameters, then the filter attempts to pair up parameters in the registered one with those in the request, by seeking the registered one's parameters that are sub-types of the requested one's parameters.

Further, the request may not have a model of what inputs may be required, and may need to obtain this information from the returned service. To support this, inputs and inputs' constraints in the next section also match when those of the request are empty.

4. Constraint Filter

The responsibility of this filter is to verify whether the subsumption relationship for each of the constraints are logically valid. The constraints filter compares the constraints to determine if the registered service is less constrained than the request. The Matchmaker computes the logical implication among constraints by using polynomial $\theta$-subsumption checking for Horn clauses. Matching is achieved by performing conjunctive pair-wise comparisons for the constraints.

The constraints for inputs and outputs are defined for the request ($R_I$ and $R_O$) and for each registered service ($A_I$ and $A_O$). The constraints for inputs $R_I$ is compared with $A_I$, and a match is determined if $A_I$ subsumes $R_I$, i.e.

$$match(R_I, A_I) \Leftarrow (\forall j, \exists i : (i \in R_I) \wedge (j \in A_I) \wedge subs(j, i)) \vee R_I = \emptyset$$

where $subs(j, i)$ is true when $j$ subsumes $i$. The constraints for outputs match when all the elements in $R_O$ subsumes elements in $A_O$, i.e.

$$match(R_O, A_O) \Leftarrow \forall i, \exists j : (i \in R_O) \wedge (j \in A_O) \wedge subs(i, j)$$

## 3 Matchmaker Deployed with UDDI

At the design phase of this development, we have collected VOCs (Voice Of the Customer) from system integrators and user companies of web services. The

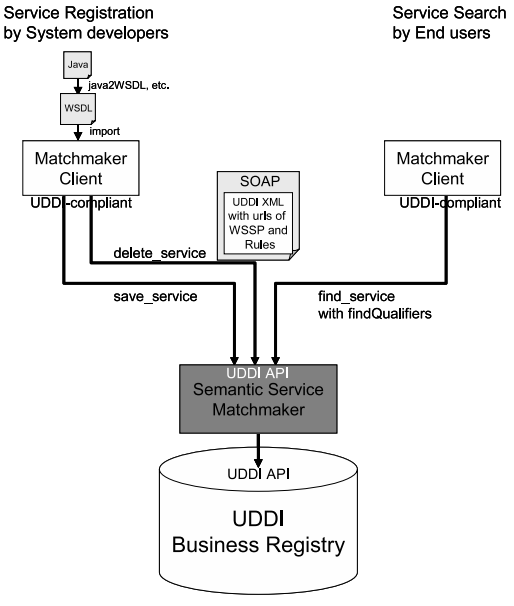


**Fig. 4.** UDDI and Matchmaker

first request from both of the integrators and the users was the reduction of the cost of development and administration. Then, the second requirement was interoperability with the current systems running or selling in their companies. The third ones were a track record and security in general. As a consequence, we have decided to obey the standards as much as possible. This means not only the improvement of the interoperability, but also the reduction of the cost because it will give users and developers freedom of choice at the selection of the private UDDI registry product and the client tools. Besides, we hope this public deployment contributes to the track record.

The architecture of the Matchmaker with UDDI is shown in Fig. 4. It strives to be compliant with the current standards of web services such as SOAP and WSDL. Also, the Matchmaker API is equivalent to UDDI API to facilitate the users seamless connection. Furthermore, the format of the results returned adopts the same format of the results returned by UDDI. So that we hope the Matchmaker can be added as a part of the common e-Business platform.

## 4 Supporting Tools

The above VOCs also showed the most important thing in a service search engine was ease of use and search speed. Although using the semantics surely has higher value on the service capability matching than the keyword search, it needs much expense in time and effort. Therefore, we introduce our client tools, which support to lower the threshold of use of semantics.

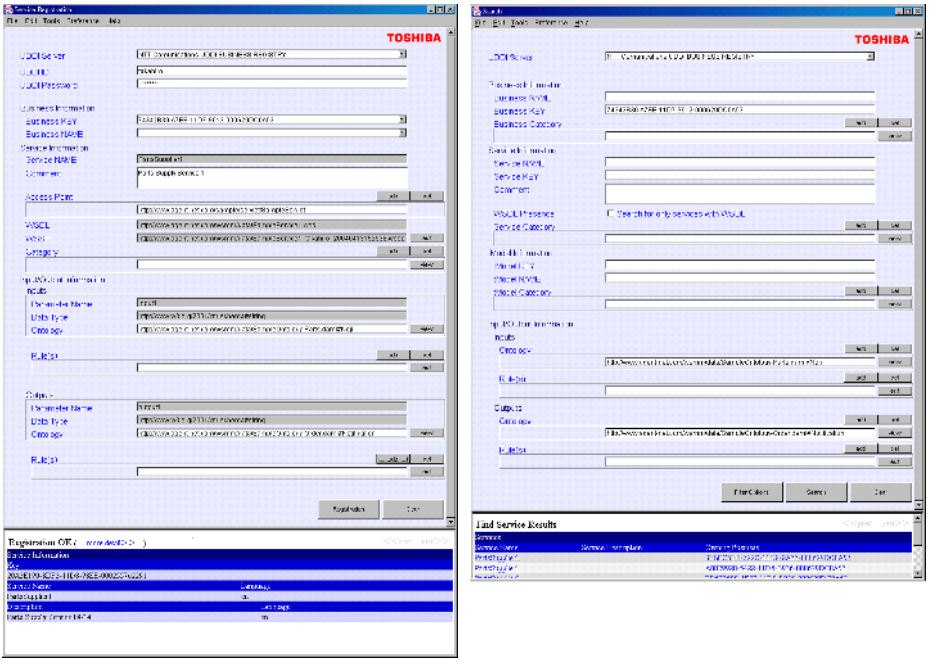


**Fig. 5.** Matchmaker Client

### 4.1 Registration of Business Service

Registration of *Business Service* (UDDI terminology) are shown in the left side of Fig. 5 (field names are corresponding to UDDI Data Structure). Firstly, the user can import a WSDL file, which may have been generated automatically from Java source code to *Matchmaker Client*. The Client will automatically complete some of the fields (gray ones in the figure) from the information in the WSDL.

Then, the user can annotate the input and output with the ontology using *Ontology Viewer*(Fig. 6), which parses ontology files written by RDFS, DAML+OIL, or OWL, then show them as graphical trees. The only necessary to annotate each parameter is to click a node in the trees. Also, the user can add constraints on each parameter using *Rule Editor* described later.

Next, the user can register the service to the Matchmaker and the UDDI as a consequence. The semantic service description, WSSP is automatically generated by the Client, and uploaded to the Matchmaker. Upon receiving the registration, the Matchmaker extracts the semantic annotation, and stores it after loading all the ontologies it refers to. Then, the Matchmaker registers the service to the UDDI.

The user may decide not to annotate any parameter with the ontology or constraints. If this is the case, the Client registers the service just to the UDDI. Note that the Matchmaker has the same API as the UDDI, so the Client can be used as a UDDI client for any UDDI registry.

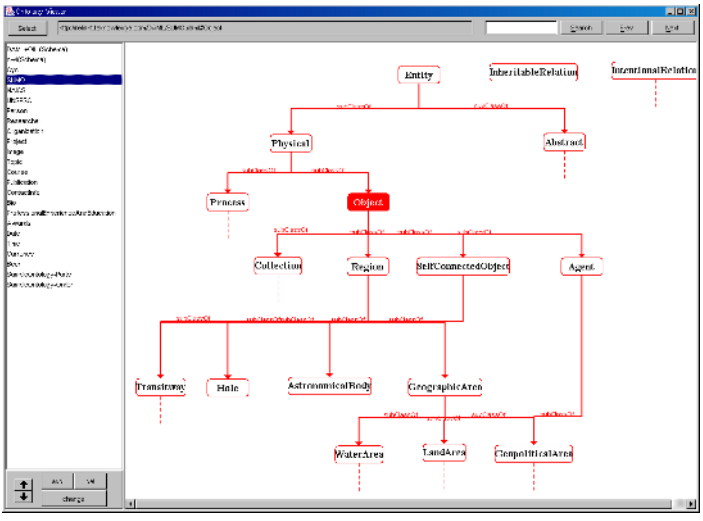


**Fig. 6.** Ontology Viewer

### 4.2 Search of Business Service

Search of *Business Service* are shown in the right side of Fig. 5. In addition to the ordinary UDDI search conditions like keywords for service names, the Client allow to specify the capability of the desired service, such as the number of inputs and outputs, the meaning of each parameter, that is, the ontology, and the constraints if necessary. Then, the Client sends a search request to the Matchmaker.

Upon receiving the request, the Matchmaker searches for services that is similar enough to the requested from the registered services. After making the matching results, the Matchmaker retrieves the detailed information of those results from the UDDI, and return them to the Client.

The user may also decide to restrict his search to the UDDI keyword search. If so, the Client searches just in the UDDI.

### 4.3 Rule Editor

Writing rules precisely is a difficult and wordy task even for logic experts. To address this problem, we developed Rule Editor (Fig. 7) which allows anyone to easily write facts and rules, as if they were writing natural sentences. The user can just select any predicate for fact or rule as constraint on a parameter. The parameter corresponds to variable in the fact or rule. After creating the constraints, they appear in “Rule” field the Client, and RDF-RuleML files will be automatically generated.

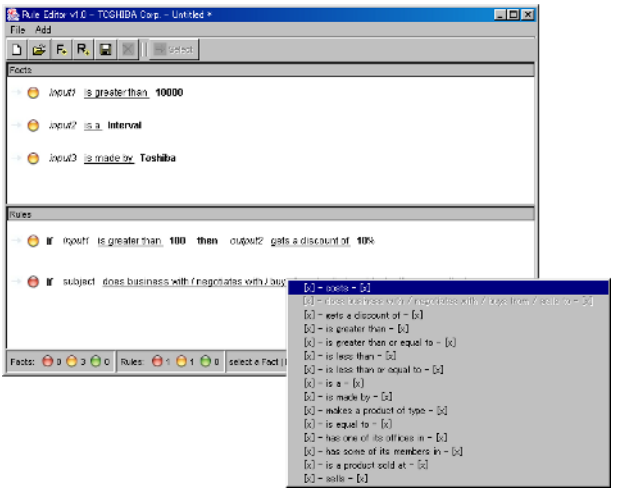


**Fig. 7.** Rule Editor

## 5 Evaluation

We have publicly deployed our Matchmaker to the official NTT UDDI registry for half a year since September 2003. In this section, we evaluate the Matchmaker in terms of performance with UDDI and functionality with other matchmakers.

### 5.1 Performance Comparison

In order to compare the performance of UDDI with UDDI with the Matchmaker, firstly we made a number of datasets and measure the average search time of each filter for a query, which always returns 10% of each dataset as matched. The dataset has a certain number of registered services, each of which has 3 input parameters and 2 outputs parameters, and each parameter has 1 constraint rule. Further, we used one of the largest ontology Cyc[10], and randomly put a node in the Cyc to each input and output parameter. The result is shown in Fig. 8

Although some items are difficult to see in the figure, all the search time are almost linearly increasing along with the size of the dataset even in UDDI. However, the rate of increase for Type and Constraint filter is relatively higher than UDDI. This was caused by the fact that UDDI is running at the very high-performance server machine, although the Matchmaker is sitting at just a common PC server with RedHat 7.3 on Intel Xeon 2.7GHz, and the keyword search costs much lower than our logical computation. But we can say at least that our Matchmaker succeeded to avoid the exponential increase of the logical computation time.

Secondarily, we constructed a search model for the users and measured the total search time for a query. In this model four filters are sequentially connected,

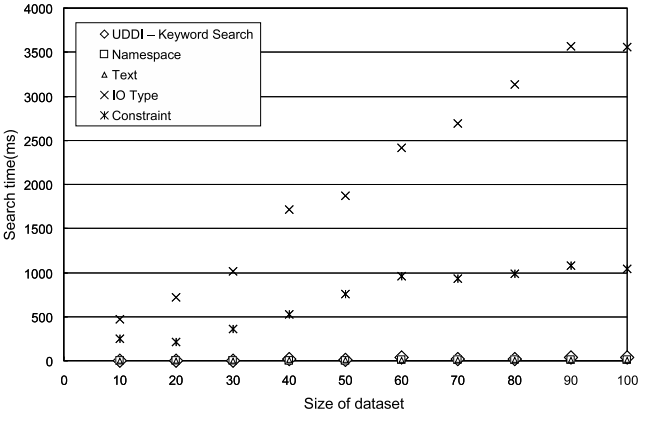


**Fig. 8.** Search time at each filter

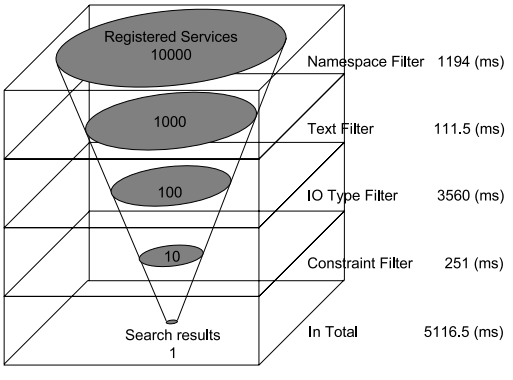


**Fig. 9.** A search model

and every filter outputs 10% of the services inputted from the previous one. Figure 9 illustrates this model, and the search time for each and in total.

UDDI currently stores at most 10,000 services in its database, so that it's enough realistic to have 10,000 services as the initial dataset. Further, we annotated each input and output parameter by the Cyc. We believe this setting is one of the largest cases in scale which can be taken at the present. Therefore, the search time, 5 seconds in total proves that our Matchmaker can be deployed in the practical business use.

We also note that the above search does not include the loading time of any new ontology, because as a way of performance improvement all the ontology that the Matchmaker encounters at the service registration will be immediately loaded and parsed in a fast-accessible internal form. Thus we assumed that in most cases the ontologies are already loaded before the service search. Therefore,

| Ontology | Num. of Nodes | Loading time (s) |
|---|---|---|
| WordNet | 6 | 0.690 |
| SUMO | 631 | 4.246 |
| Cyc | 1740 | 49.461 |

**Fig. 10.** Loading time

| System Name | UDDI | Web Serivce Matchmaker | InfoSleuth | Semantic Web based matchmaking prototype | n/a | WebSphere Matchmaking Environment |
|---|---|---|---|---|---|---|
| Developer / Organizaion | OASIS | Toshiba | MCC | HP Laboratories Bristol | University of Manchester | IBM Zurich Research Laboratory |
| Syntactical matching | | | | | | |
| Keyword Search | ○ | | | | | ○ |
| IR / Text mining | | ○ | | | | ○ |
| Semantic matching | | | | | | |
| Ontological Subsumption | | ○ | ○ | ○ | ◎ | |
| Constraints / Rules | | ○ | ◎ | ○ | ○ | ○ |
| Compliance with Web Services Standards | | | | | | |
| UDDI | ○ | ○ | | | | |
| SOAP | ○ | ○ | | | | ○ |
| WSDL | ○ | ○ | | | | ○ |
| Compliance with Semantic Web | | | | | | |
| DAML+OIL / OWL | | ○ | | ○ | ○ | |
| DAML-S / OWL-S | | | | ○ | ○ | |

**Fig. 11.** Functional comparison

we measured the ontology loading time separately. Figure 10 shows the loading time for some of the well-known ontologies.

### 5.2 Functional Comparison

We picked up the following matchmakers from academic and industrial activities, and summarised the comparison in Fig. 11 and the following list, in which criteria were syntactical matching like keyword search, semantic matching like ontology, and compliance with the standards for web services and semantic web.

- UDDI, OASIS
  - A standard of Web Services registry, linked to WSDL and SOAP.
  - Search functionality is limited to the keyword search.
- InfoSleuth, MCC[11]
  - Based on constraint-based reasoning, a broker of information gathering agents.
  - The original service capability and constraint description, LDL++.
  - No explicit relationship to Web Services standards.
- Semantic Web based matchmaking prototype, HP Lab.[12]

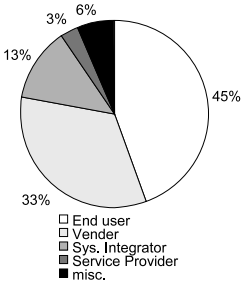


**Fig. 12.** Target users

- Based on ontological subsumption and rules similar to ours.
- The original service description and rule language.
- No integration to UDDI.

– WebSphere Matchmaking Environment, IBM Zurich Research Lab.[13]
  - A add-on for IBM Web application server WebSphere.
  - Search conditions can be written in a script, which is sent to the matchmaker.
  - Semantics seems to be not considered.
– Ian Horrocks et al., University of Manchester[14]
  - Based on highly functional DL (Description Logics) reasoner Racer with DAML-S.
  - Semantic matchmaker without any IR techniques like TF/IDF and prefiltering for the reduction of computation time.
  - No compliance to Web Service standards.

## 6 Discussion

This section shows user requirements for the Matchmaker explicitly obtained by questionnaires at the download of the Matchmaker Client, and implicitly recognised by our observation of users' search behaviour.

### 6.1 Explicit Requirements

The targets of this questionnaire are users who intentionally accessed our Matchmaker web page (www.agent-net.com) and downloaded the Client. That is, the targets are people keeping their eyes on the trend of web services. Figure 12 shows the breakdown of the targets, which are selected as valid votes. We got future improvements by adapting DFACE (Define–Focus–Analyze–Create–Evaluate) methodology, which is a version of "Design for Six Sigma"[15] developed by Stanford University, to user requirements described in the questionnaire. Figure 13 shows Cost–Worth analysis for the conceivable improvements, in which

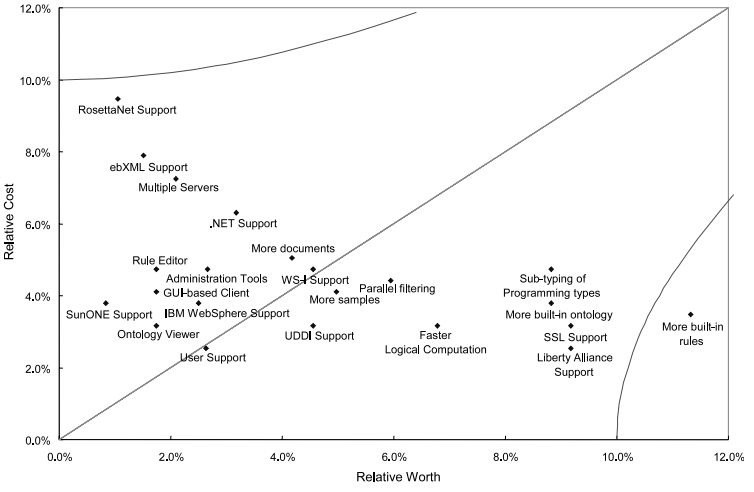


**Fig. 13.** Cost-Worth analysis

you will see for example, compliance to web services standards is so much important, and automatic service invocation should be discussed after the service search. We couldn't describe the process to get this analysis in detail due to the page limitation, but basically it was calculated by proportional distribution from user requirements considering CSM (Customer Satisfaction Measurement) and workload.

### 6.2 Implicit Requirements

The following lists are part of users' indirect suggestions gotten from the log files which record every action in the Matchmaker.

- IO Type filter: we found that some amount of users specify the ontology only to an output. This means that search conditions for the human users are somewhat vague, and they are willing to manually choose the most desirable one from the results. This behaviour inevitably reduces the accuracy of logical matching due to the lack of semantic information for the request. Therefore, matchmakers relying only on the ontology and rules would need to incorporate other aspects like conventional IR techniques to be practically deployable. However, for automatic service search and composition by machines in future ambiguous requests are not allowed, so logical matching has more importance.
- IO Type filter: some users tend to give just a keyword as the ontology. It seems that they are thinking the ontology can be automatically specified from an arbitrary word such that a specific category is determined by any keyword in Yahoo!. Therefore, pre-process to reason about a specific ontology node from a keyword would be useful for them.

- Constraint filter: it seems that the most familiar predicates for the users are numerical ones like equals, greaterThan, lessThan, etc. This is because the meanings and effects of those rules are easily imaginable for the users. On the other hand, putting facts and rules with generic predicate seems to be difficult enough for the users. This means that our Rule Editor which now automatically generates the rules from the user's simple selection of predicates need to be improved to make the user understand what happens when you choose it. The same things would be said to precondition and effect in future.
- Text filter: most of registered services do not have a certain length of comments. Therefore, adopting IR techniques only to comment parts has some limitation. So we are trying to take other parts of service description like parameter names and related web pages of the service provider referring to syntactic approaches of service search like Wang et al.[16].
- Namespace filter: since this filter is the first one, it compares a request to all the registered services. To reduce the burden of this filter, and to increase the accuracy even in the above-mentioned case that the user specifies the ontology only to an output, we are planning to take Category field for Business and Services in UDDI Data Structure specified by NAICS, UNSPSC, etc. into account before this filter. In case that the number of results is less than the user wants, the services in other categories can be checked by the filters. This also means more tightly integration to UDDI.
- In general: as you see Link Popularity has one of the biggest roles in Google, we will have to consider the heuristics even for the logical service matching. We are now examining some of the ideas on the analogy of web search engines. We hope to show their effectiveness and efficiency in near future.

## 7 Conclusion

In this paper, we evaluated the deployment of the semantic service matchmaker to the UDDI in public, and discussed scalability in performance, feasibility in functionality, and future works from users' explicit and implicit requirements. According to the future works, we will make improvement on the Matchmaker and client tools, and keep our deployment to get further feedbacks near future.

## References

1. Universal Description, Discovery and Integration, http://www.uddi.org/.
2. M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, "Semantic Matching of Web Services Capabilities", Proceedings of First International Semantic Web Conference (ISWC 2002), IEEE, pp. 333-347, 2002.
3. T. Kawamura, J. D. Blasio, T. Hasegawa, M. Paolucci, K. Sycara, "Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry", Proceedings of First International Conference on Service Oriented Computing (ICSOC 2003), pp. 208-224, 2003.

4. Resource Description Framework, http://www.w3.org/RDF/.
5. DAML Language, http://www.daml.org/language/.
6. Web-Ontology Working Group, http://www.w3.org/2001/sw/WebOnt/.
7. The Rule Markup Initiative, http://www.dfki.uni-kl.de/ruleml/.
8. DAML Services, http://www.daml.org/services/.
9. K. Sycara, S. Widoff, M. Klusch, J. Lu, "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace". In Autonomous Agents and Multi-Agent Systems, Vol.5, pp.173-203, 2002.
10. Cyc, http://www.cyc.com/2002/04/08/cyc.daml.
11. M. H. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh, "Active information gathering in infosleuth", International Journal of Cooperative Information Systems, pp. 3-28, 2000.
12. J. G. Castillo, D. Trastour, and C. Bartolini, "Description Logics for Matchmaking of Services", Proceedings of Workshop on Application of Description Logics, 2001.
13. WebSphere Matchmaking Environment, http://www.zurich.ibm.com/wme/.
14. L. Li, I. Horrocks, "A software framework for matchmaking based on semantic web technology", Proceedings of the Twelfth International World Wide Web Conference (WWW 2003), pp. 331-339, 2003.
15. Six Sigma Academy, http://www.6-sigma.com/.
16. Y. Wang, E. Stroulia, "Semantic Structure Matching for Assessing Web-Service Similarity", Proceedings of First International Conference on Service Oriented Computing (ICSOC 2003), pp. 194-207, 2003.