International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:1, No:12, 2007

Models to Customise Web Service Discovery Result using Static and Dynamic Parameters

Kee-Leong Tan, Cheng-Suan Lee, and Hui-Na Chua

***Abstract*—**This paper presents three models which enable the customisation of Universal Description, Discovery and Integration (UDDI) query results, based on some pre-defined and/or real-time changing parameters. These proposed models detail the requirements, design and techniques which make ranking of Web service discovery results from a service registry possible. Our contribution is two fold: First, we present an extension to the UDDI inquiry capabilities. This enables a private UDDI registry owner to customise or rank the query results, based on its business requirements. Second, our proposal utilises existing technologies and standards which require minimal changes to existing UDDI interfaces or its data structures. We believe these models will serve as valuable reference for enhancing the service discovery methods within a private UDDI registry environment.

***Keywords*—**Web service, discovery, semantic, SOA, registry, UDDI.

I. Introduction

SERVICE-LEVEL discoverability is one of the primary principles within a Service Oriented Architecture (SOA). Due to the convergence of key technologies and popularity of Web service, most service-oriented enterprises are taking advantage of Web services capabilities to improve corporate agility, time-to-market for new products or services, reduce IT costs and improve operational efficiency. Among the major benefits of Web services are features such as pervasive, simple and platform-neutral. [1]

Implementing discoverability on SOA level basically requires the use of registry or directory technologies such as UDDI [2]. The interaction between UDDI and other components within web services architecture is shown in Fig. 1. Web services architecture consists of specifications such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and UDDI. All these components support the interaction of a service requester with a service provider and the potential discovery of the Web service description. The provider typically publishes a WSDL description of its Web service, and the requester accesses the description using a UDDI or other type of registry, and requests the execution of the provider's service by sending a SOAP message to it.

All authors are members of Mobile Web Service Team, Asian Research Centre, British Telecommunications Group, 63000 Cyberjaya, Selangor, Malaysia (e-mails: keeleong.tan@bt.com, chengsuan.lee@bt.com, hui.chua@bt.com).

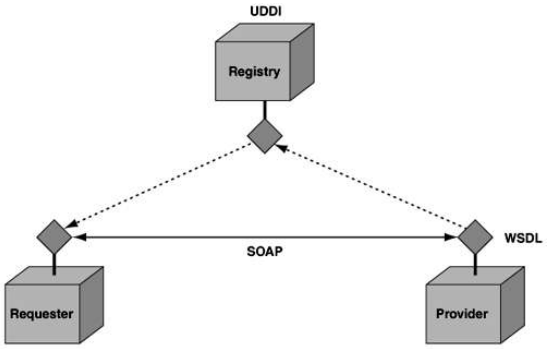

Fig. 1 Basic Web service architecture

However, present UDDI specification still has limitations, particularly on semantics information retrieval. Hence, unlike WSDL and SOAP, UDDI has not yet attained industry-wide acceptance, and remains an optional extension to SOA. For example, the present UDDI standard does not provide a built-in mechanism to personalise or rank its query results, and its search capabilities are unable to extend beyond the keyword-based matches [3]. To address some of these limitations, there are many on going research and standardisation activities within the SOA and semantics web communities which result in the introduction of semantic service markup language such as DAML-S and OWL-S [4]. Besides that, XML based languages for business process are also expanding, such as WSFL, ebXML, BPML, RuleML, and BPEL4WS.

Despite the limitations mentioned above, and the slow adoption of public UDDI implementation, private UDDI has gained success within inside-the-enterprise technology and support from major vendors such as Oracle, Microsoft and IBM. Based on this, UDDI will be the most popular candidate for SOA registry implementation. One recent announcement by Oracle to include UDDI-based registry as part of their latest Oracle Application Server 10g Release 3 further support this future trend. [5]

II. Problem Description

As UDDI has gained support from enterprises and major vendors, it's usage will be not be limited to business to business (B2B) scenario, but also into the area of business to customers (B2C) and peer to peer interaction. Within the B2C context, a business entity owns or implements private or semi-

private UDDI registries. The business entity will have certain business rules or interests to follow, to customise the results of Web service discovery. For example a business entity who own private UDDI, may wish to have a control on the way UDDI query results are displayed to the service requestor.

Let us consider the following example which illustrates this scenario. We have a telecommunication service operator who owns a private UDDI registry to store details of mobile services it offers. Besides its in-house developed services, it also hosts some third party or vendor developed services. Now consider a case when a mobile consumer browses the service category he intend to subscribe or purchase, result shows there are more than one available service under the chosen category. The registry will display a list of services and the final choice will be made by the customer.

However, the operator may wish to prioritise the list of services to be displayed. Some scenarios such as showing only selected services or rank all services according to pre-defined business rules such as preferred vendor or service popularity. Assuming this private registry is owned and hosted by a business entity, this requirement should be automated and its mechanism should be transparent to the consumers. With the present UDDIv3 capabilities [6], there is no direct approach to achieve this, and this limitation formed the motivation of our research problem description.

III. Related Works

Most efforts to customise Web service discovery results focused on creating semantic extensions to UDDI, pioneered by K.Sivashanmugam, et al. [7] and Paolucci, et al. [7][8]. It took advantage of DAML ontology to implement a matching algorithm used to enhance UDDI registries with additional semantic layer; this also allowed metadata pattern based matching. The work carried out also described how service capabilities within DAML-S can be mapped into UDDI records, which lead to a new technique to record semantic information within UDDI records. To achieve more accurate matching results, an algorithm was proposed to rank the level of matching for DAML-S description, where the result was an aggregation of several pre-defined individual verification and matching stages [9]. These approaches however are not suitable for private registry environment as effort to customise registry to support additional ontology languages like DAML-S will require too much modification effort and amplified system complexity.

Rama, et al. [3] questioned the effectiveness of these semantic extensions and argued a better approach would be to extend the UDDI API schema to enable a service requestor to specify the semantic properties. This approach will require new parameters to be added to UDDI API. For discovery, selection and combination of services according to the special preferences of an individual user, [10] introduced an algorithm for selection of appropriate service using cooperative databases and collaborative filtering techniques. However, we foresee these approaches will not gain wide industry acceptance as changes to existing UDDI API and data structures will add to the complexity of existing system and they do not conform to existing standards.

With regards to customisation by ranking of web services, there were several proposals such as [11] which introduced the use of agent to automatically establish ranking capabilities to web services and [12] described a framework for ontology-based discovery of semantic web services and allowed user to specify personalised ranking criteria as part of query result based on ontology. In [14], taxonomy for non-functional attributes namely QoS was proposed. The UX architecture [13] suggested an approach to use dedicated server to collect feedback of users and predict the future performance of published services.

In this paper, we propose three practical approaches to customise the registry query result according to certain criteria. We use static and dynamic parameters values to formulate the criteria to achieve the customisation of UDDI query results. Our approaches are represented by three alternative models which adhere to present UDDI standard. Further details of each model will be discussed in the next sections.

IV. Model Architecture

We propose three models to achieve the customisation of UDDI query results. All three share some common architecture components as shown in Fig. 2. They are: UDDI server, UDDI Proxy and User Interface. These components will interact with other external components. In this paper, we assume the customisation criteria required is the ranking of list of business or service list to User Interface. Load balancing also can be improved by keeping the User Interface and UDDI Proxy on separate servers.

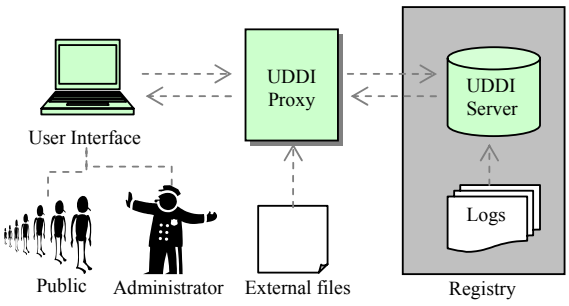

Fig. 1 Proposed model architecture

UDDI server is a server-side application that fully supports the UDDI API specification. Examples are Microsoft Enterprise UDDI Services, IBM Websphere UDDI Registry, Oracle AS UDDI Registry, webMethods GLUE and jUDDI [14]. *User interface* allows a requester (or comsumer) to manually locate and select a service description that meets his desired functional and criteria. It could be a web browser or standalone application accessed via mobile devices or desktop computers. The User Interface support two types of user access: (a) public access - for registry service browsing, accessible by everyone; and (b) administrator access - for

UDDI Proxy configuration and parameters settings, restricted user access. The *UDDI Proxy* acts as an intermediary between the *User Interface* and the *UDDI Server*. It supports two important features to make the customisation ranking of UDDI query results possible. First, if there is more than one record within the list, before returning the result to *User Interface*, it rearranges the records based on certain pre-defined criteria. The criteria can be created either at design time or run time. Secondly, *UDDI Proxy* will provide a mechanism (more details for each model to be explained in section V) to automate the update of parameter values which will be used to form the criteria.

V. Proposed Models

The most basic feature of UDDI is to allow businesses to publish their services in a directory and enable other business representatives to locate partners and to form business relationships based on the web services they provide. In this section, we propose three alternative models to extend its basic feature to allow personalisation of UDDI query result based on criteria managed by the UDDI administrator. The criteria consist of certain parameters which will determine its outcome. Here, we introduce two types of parameter: static and dynamic.

The static parameter will hold certain values which has been fixed and do not change during run-time. Only Administrator access can modify its values. Examples of static parameter are vendor ranking (for business), cost per transaction and advertisement priority (for service). Vendor ranking refers to priority values assigned for different vendors, based on certain business requirements. For example the most preferred vendor will be given value of 1, second be given value of 2, and so on.

Unlike static, the dynamic parameter will be used to store value which is real-time changing and gets updated during run-time. The updating frequency will depend on mechanism defined within the criteria. One usage of dynamic parameter is to keep track of service or business popularity, where it stores the total number of request to invoke or subscribe a specific service. The mechanism to determine the frequency of this update is described in more details in section 5A. The function is similar to webpage 'hits counter'. Usage described here can also be extended to track business or vendor popularity – to know how popular a vendor compared to others. Another example of dynamic parameter usage within a registry is to monitor service traffic load, where it can store data containing total number of concurrent users accessing a specific service at any point of time. The parameter examples described above are summarised in Table I.

TABLE I
Example of Parameters Categorized According to Business and Service Entity

	Static Parameter	Dynamic Parameter
Business	Vendor ranking	Vendor popularity
Service	Service cost, advertisement	Service popularity, service load.

In our proposed models, we assume the private registry is owned by a business entity that has control over the service discovery results. The criteria used to customise the UDDI query results will be represented by static and dynamic parameters. The key differences between each model are (1) the location on where the parameter values are stored and retrieved; and (2) ranking mechanism. Each model's requirements, ranking mechanisms, advantages and disadvantages will be further elaborated in the following sections.

A. Model Where Parameters are Saved and Retrieved from UDDI Server

In this first model, we propose the use of only UDDI Proxy and UDDI Server components (Fig. 3), where the parameters will be saved inside the UDDI server itself. This will require a new tModel definition to describe the parameters information. Each business entity and service will then contain a reference to this tModel in their record. For example, the reference can be inserted as one of the element inside the identifier bag of *businessEntity*, or its service tModel *identifier* bag. The term "bag" indicates a generic container of multiple values, and enables a company to register multiple business identifiers.

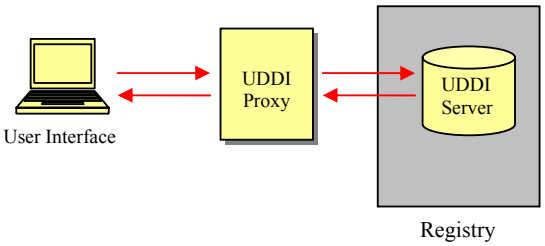

Fig. 3 Model 1 proposes parameter values to be saved and retrieved from UDDI server

i. tModel Definition

Here we show an example of the proposed tModel definition to represent static vendor ranking parameter for the *businessEntity* record. A new tModel will have to be created for each parameter type.

Name:	uddi-org:types
Description:	Business parameters: static: vendor ranking. To store vendor priority level data. Highest=1 and lowest=5.
UDDI Key (V3):	uddi:uddi.org:categorization:types
Evolved V1,V2 format key:	**uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4**
Categorisation:	Categorisation
Checked:	Yes

ii. tModel V3 Structure

This vendor ranking tModel is represented with the following structure:

```
<tModel
tModelKey="uddi:uddi.org:ubr:categorization:static:v
endorranking:2005">
<name>vendor ranking</name>
<description xml:lang="en">Business Taxonomy: Static
Vendor Ranking Parameter </description>
<overviewDoc>
<overviewURL useType="text">
http://bull.dot.com/uddi/taxonomy/parameters/static.
htm#vendor_ranking
</overviewURL>
</overviewDoc>
...
</tModel>
```

iii. Example of Use

From the tModel definition above, a *businessEntity* record includes a reference to vendor ranking tModel in its identifier bag, as below.

```
<businessEntity
businessKey="xxx-yyy-zzz-ac09-9955cff462a3"
operator="UDDI Department,Company Bull dot Com"
authorizedName="Tan KL">
<name>Bull dot Com</name>
...
<identifierBag>

<keyedReference
tModelKey="uuid:8609c81e-ee1f-4d5a-b202-
3eb13ad01823"
keyName="D-U-N-S" keyValue="22-333-444" />

<keyedReference
tModelKey="uddi:uddi.org:ubr:categorization:static:v
endorranking:2005" keyName="vendor ranking"
keyValue="3" />

</identifierBag>
...
</businessEntity>
```

iv. Retrieving Parameters Values

In this model, all the parameter values are stored using XML schema inside the UDDI server. Whenever a request is made by consumer to get a list of services, the *UDDI Proxy* will invoke the UDDI *Find* functions of the inquiry API. Certain *Find Qualifiers* can also be used to enable more precise search criteria. Let us take an example of mobile user who requests for online stock quote service. If there are 4 registered vendors providing the service, the *UDDI Proxy* will receive a list as shown in Table II.

TABLE II
INITIAL LIST RECEIVED AT *UDDI PROXY*

Service Name	Vendor Ranking	Service Popularity	Traffic Load
Ant Quotes	5	4	12
Bull Stock2U	3	2	30
Cat Stock	1	3	34
Deer online stock	2	1	22

As shown above, all static and dynamic parameters related to the services (as discussed in section V) are embedded in the list. This is very important as the *UDDI Proxy* will use some of this parameter values as ranking criteria. Based on the criteria preferences defined by administrator, if the ranking feature is enabled, the *UDDI Proxy* will further process the list accordingly, using the embedded parameters values.

Once processing is done, the new list which contains ranked and sorted services will be sent to user interface, all the parameters values will be discarded. This process is presented in Fig. 4.

1. After receive UDDI query result, check if ranking criteria is required. If not, proceed to step 8.
2. Store the all result (except for parameter values) into one temporary array column. Each parameter value is stored in one column.
3. Determine which ranking criteria to be used.
4. For each record in the array, perform a sort function based on ranking criteria determined in step 3.
5. If parameter value for current record is higher than previous, move current record up the list.
6. Start over with step 4 until reach last record.
7. Make necessary formatting on the query list.
8. Send the list to User Interface.

Fig. 4 Algorithm to rank UDDI query result

In this example, if the registry owner wish to default the ranking criteria based on 'vendor ranking', hence the processed list sent to User Interface will be as shown in Table III.

TABLE III
PROCESSED LIST SENT TO USER INTERFACE

No	Service Name
1	Cat Stock
2	Deer online stock
3	Bull Stock2U
4	Ant Quotes

v. Saving Parameters Values

Saving of parameters values to *UDDI Server* will be handled by the *UDDI Proxy* using the *Save* functions of the UDDI publishing API.

For static parameters, its values can be edited only by the administrator. This can be achieved by having *UDDI Proxy* to display and save the parameter values directly to *UDDI server*. The save frequency is solely depending on the registry administrator. As for the dynamic parameters, its values will be updated each time the Proxy detect a request has been made to access the respective business or service links. If the dynamic parameter is used to store an incremental number such as vendor ranking or popularity, first the *UDDI Proxy* is required to read the current parameter value, increment the value by 1 before it invoke the save function.

The main advantage of the first model is the criteria data are stored and bind with its associated business or service entity. This will be beneficial for private registry operator who wishes to extend UDDI capabilities to support ranking with minimal changes to his present system architecture. However, there might be certain performance issue if the Proxy accesses launch too many queries, too frequently to the UDDI server.

B. Model Where Parameters are retrieved from Server Logs

A private registry system normally consists of several application and server components. A typical UDDI server is often hosted together with application server (JBOSS, Apache Tomcat) and SOAP server (Apache Axis) or being part of a integrated solution package (Microsoft Enterprise Server, GLUE). As with the UDDI server, these servers do provide cross-language logging services for purposes of application debugging and auditing. Web service log data could provide information such as Web service usage, supporting information concerning business transaction and quality of service [15]. These logs data could provide useful semantic information for ranking criteria.

Fig. 5 shows the components and data flow of this second model. Note this model does not support the retrieving or saving of static parameters.

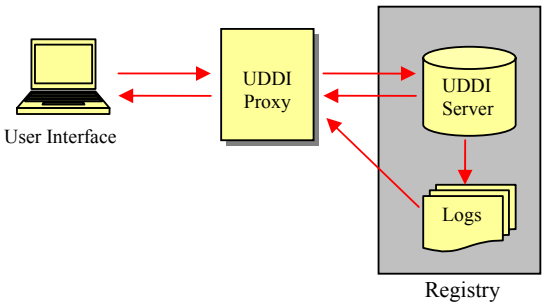

Fig. 5 Model 2 proposes dynamic parameter values to be retrieved from logs data

i. Retrieving Dynamic Parameter Values

In this second model, we propose the creating of dynamic parameter values by extracting and processing the data from log files of SOAP server, application server and UDDI server. A function to search, match and count for each parameter type is required within the *UDDI Proxy*. Examples of unique identifications are *businesskey* and *servicekey*, both assigned by UDDI. Our algorithm (Fig. 6) shows the necessary steps to be performed by *UDDI Proxy* in order to achieve this.

1. After receive UDDI query result, check if ranking criteria is required. If not, proceed to step 8.
2. Perform keyword search to identify all the businesses or services received in the list.
3. Group and store business/service unique identification (UID) into a temporary array. Create parameter value column(s) for each UID.
4. For each of this UID, perform a search through the log files, and count the UID match based on its ranking criteria.
5. For each match, store and increment the parameter value in the array.
6. Start over with step 4 until all UIDs have been updated.
7. Sort the query list based on the final parameter values in the array and make necessary formatting.
8. Send the list to User Interface.

Fig. 6 Algorithm to rank list using logs data

ii. Saving Dynamic Parameter Values

Since dynamic parameters values are extracted from the log files and the log processing is handled by the respective server logging services, there will be no saving mechanism introduced here. The only important requirement is to ensure all the servers logging service are turned on, or to the minimum level where UUID will be created within the logs.

The main advantage of the second model is the criteria data can be automatically generated from the server logs. This will simplify implementation procedures and ensure data received are the most recent. Registry administrator who does not require static parameters for their criteria will find this model suitable for their need. Besides, this model can be further extended to monitor the health of registry servers as described in [17].

C. Model Where Parameters Are Saved And Retrieved from External File

In this model, we propose keeping both parameter values in external files, one file for each parameter type. As shown in Fig. 7, the files should be accessible directly from the Proxy, outside the UDDI server. The flat ASCII file can either be in pipe-delimited or even XML format. *File A* is used to store values for static parameters and it can be modified by administrator only. *File B* is used to store values for dynamic parameters and gets updated by certain functions within the *UDDI Proxy*, without the intervention of administrator.

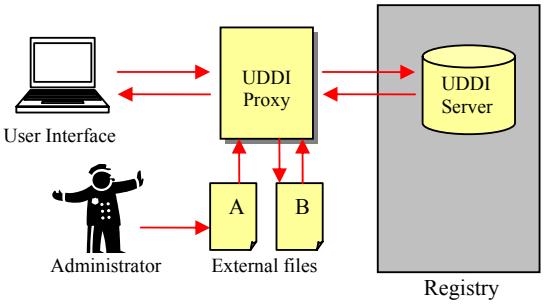

Fig. 7 Model 3 proposes parameter values to be saved and retrieved from external files (File A = Static, File B = Dynamic)

i. Retrieving Parameters Values

Parameters values for business and service ranking criteria will be retrieved from an external file. Functions to open file, search, match and count for a specific parameter are required within the *UDDI Proxy* design. The algorithm in Fig. 8 shows the necessary steps to be performed by the *UDDI Proxy* in order retrieve both parameters values from external files.

1. After receive UDDI query result, check if ranking criteria is required. If not, proceed to step 11.
2. Perform keyword search to identify all the businesses or services received in the list.
3. Group and store business/service unique identification (UID) into a temporary array. Create parameter value column(s) for each UID.
4. Check if ranking criteria is based on static or dynamic parameter. If dynamic, proceed to step 7.
5. If static parameter, open File A. For each UID element in step 3, perform a search through the File A, and count the UID match based on its ranking criteria. Else, if
6. For each match in File A, store and increment the parameter value in the array.
7. Repeat step 5 and 6 until all UIDs have been updated.
8. Proceed to step 10.
9. For dynamic parameter, repeat step 5 to 7 by referring to File B.
10. Sort the query list based on the final parameter values in the array and make necessary formatting.
11. Send the list to User Interface.

Fig. 8 Algorithm to rank list using parameters stored in external files

ii. Saving Parameters Values

Unlike the first model where saving of parameter values will be added to existing UDDI record based on XML schema, this model will have its own data structure to store business/service parameters values. An example of the data structure is as follow:

UID	*serviceKey* of the service
Name	Service name
Parent UID	*businessKey* of company who develop/own the service
Popularity	Real-time value of service access
Load	Real-time value of concurrent user

To reduce complexity, we propose the data to be stored in pipe-delimited or XML format. For static parameters (*File A*), its values can be edited and saved from the administrator interface. As for the dynamic parameters (*File B*), its values will be updated each time the Proxy detects a request has been made to access the respective business or service links. If the dynamic parameter is used to store an incremental number such as service load or service popularity, the Proxy first read the current value, increment the value by one, before it updates File B.

The third model introduces distributed storage of the parameters data, it has the advantages of lowering the *UDDI Server* load, and gives administrator more control over the external files. However, with more control available at the administrator interface, the *UDDI Proxy* will have to provide more complex functions to support these requirements and file handling processing. This model will best suite registry operator who has long list of criteria parameters, require full control of the parameters data, and has to generate complex criteria on the registry query results.

VI. Comparison of Three Models

Table IV summarises the main characteristics for each models. Each model has been designed to address specific usage scenarios and requirements for manual service discovery using a private UDDI. The discussion on the usage scenarios is beyond the scope of this paper.

TABLE IV
Models Main Characteristics Comparison

	Model 1	Model 2	Model 3
Parameters supported	Static & Dynamic	Dynamic only	Static & Dynamic
Model Complexity	Medium	Low	High
Effect on registry performance	High	Medium	Low
Parameters location	Within Registry	Within Registry	Outside Registry

VII. Conclusion and Future Work

In this paper, we have presented three alternative models to customise private UDDI registry query results based on business requirements such as ranking of service list.

All the models proposed are designed to suite different practical needs of private registry systems. These models serve as valuable reference for registry administrators to further enhance the service discovery process within their private UDDI registry environments.

Aiming to achieve complete service delivery assurance for a private SOA system, our future work will focus on the refinement and implementation of proposed models. Each proposed model will be further tested on their complexity, performance and suitability, to support a reliable service discovery mechanism for occasionally connected computing environment.

References

[1] Eric Newcomer, Greg Lomow, *Understanding SOA with Web Services* (Upper Saddle River, NJ: Addison Wesley Professional, 2004).
[2] Thomas Erl, Service-Oriented Architecture: Concepts, Technology, and Design (Upper Saddle River, NJ: Prentice Hall, 2005)
[3] Rama Akkiraju, Richard Goodwin, Prashant Doshi, Sascha Roeder, A method for semantically enhancing the service discovery capabilities of UDDI. Proc. Workshop on Information Integration on the Web, Acapulco, Mexico, 2003. 87–92
[4] Anupriya Ankolekar, Mark Burstein, Jerry Hobbs J, DAML-S: Web service description for the semantic web. Proc. First Int'l Semantic Web Conf. (ISWC02), Sardinia, Italy, 2002.
[5] Oracle Unveils Oracle(R) Application Server 10g Release 3. 19 September 2005. http://biz.yahoo.com/prnews/050919/sfm087.html?.v=24
[6] OASIS. Introduction to UDDI: Important Features and Functional Concepts. October 2004. http://lists.oasis-open.org/archives/uddi-spec/200410/pdf00001.pdf
[7] K. Sivashanmugam, K. Verma, A. Sheth, J. Miller, Adding Semantics to Web Services Standards, Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada, June 2003, 395 - 401.

[8] OASIS. UDDI Version 3 Features List http://uddi.org/pubs/uddi_v3_features.htm
[9] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara, Semantic Matching of Web Services Capabilities. The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.
[10] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara, Importing the Semantic Web in UDDI. In Web Services, E-Business and Semantic Web Workshop, 2002.
[11] Luc Moreau, Simon Miles, Juri Papay, Keith Decker, Terry Payne, Publishing Semantic Descriptions of Services, Semantic Grid Workshop, Chicago, 2003, 48-54.
[12] Wolf-Tilo Balke, Matthias Wagner, Towards Personalized Selection of Web Services, 12th International World Wide Web Conference, Budapest, Hungary, 2003.
[13] Abdelmounaam Rezgui, Athman Bouguettaya, Privacy Ranking of Web Service, ACM International Conference On Service Oriented Computing, New York, NY, 2004.
[14] Jyotishman Pathak, Neeraj Koul, Doina Caragea, Vasant G Honavar, A Framework for Semantic Web Service Discovery, ACM International Workshop on Web Information and Data Management, Bremen, Germany, 2005.
[15] Z.Chen, C.Liang-Tien, B.Silverajan, L.Bu-Sung, UX – An Architecture Providing QoS-Aware and Federated Support for UDDI, Proc of International Conference on Web Services, Las Vegas, Nevada, USA, 2003. CSREA Press 2003, ISBN 1-892512-49-1.
[16] OASIS. UDDI solutions: UDDI Products and Components. http://www.uddi.org/solutions.html
[17] Serra da Cruz Serra da Cruz, Maria Luiza M. Campos, Paulo F. Pires, Linair Maria Campos, Monitoring E-Business Web Services Usage through a Log Based Architecture. IEEE International Conference on Web Services, San Diego, CA, 2004, 61-69.