International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

Personalisation of SOA Registry Query Results: Implementation, Performance Analysis and Scalability Evaluation

Kee-Leong Tan, Karyn Wei-Ju Khoo, and Hui-Na Chua

***Abstract*—**Service discovery is a very important component of Service Oriented Architectures (SOA). This paper presents two alternative approaches to customise the query results of private service registry such as Universal Description, Discovery and Integration (UDDI). The customisation is performed based on some pre-defined and/or real-time changing parameters. This work identifies the requirements, designs and additional mechanisms that must be applied to UDDI in order to support this customisation capability. We also detail the implements of the approaches and examine its performance and scalability. Based on our experimental results, we conclude that both approaches can be used to customise registry query results, but by storing personalization parameters in external resource will yield better performance and but less scalable when size of query results increases. We believe these approaches when combined with semantics enabled service registry will enhance the service discovery methods within a private UDDI registry environment.

***Keywords*—**Service Oriented Architecture (SOA), Web service, Service discovery, 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 Figure 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.

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

The service discovery process forms a relationship between Service Requestor and Service Provider. It also defines a process for locating service providers and and its associated service description documents. 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. The service discovery process can be grouped into two main groups: static and dynamic [20]. Static discovery occurs during application development time where a developer uses a browser or other user interface to perform a find operation on the service registry. For dynamic service discovery, the service implementation details such as service interface location and network protocol to use are not defined at design time so that they can be determined at runtime. At runtime, the application will find for one or more services and based on certain required parameters in application logic. The application will choose a Web service to invoke from the find results of the find operation, extracts necessary information (service interface location, network protocol, etc) and finally invokes the Web service.

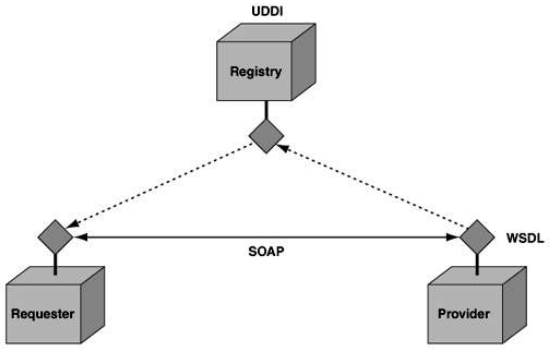

Fig. 1 Basic Web service architecture

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 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 fulfill, hence arise a need to customise the results of Web service discovery. One example scenario will be a business entity that owns private UDDI, may wish to have a control on the way UDDI query results are displayed to the service requestor. The control mechanism will be based on certain business criteria, which will be mapped to certain parameters.

Let us consider the following example which illustrates this scenario. We have a telecommunication service operator who uses a private UDDI registry to store and publish mobile services to its customers. Besides its in-house developed services, it also hosts some services offered by third service providers. The UDDI operator may use UDDI classification scheme like NAICS [21] and UNSPSC [22] to better describe and categorise the service functionality. For example, a mobile game service can use UNSPSC classifications scheme with value code of 43223208 to describe its functionality.

Now consider a static discovery scenario when a mobile consumer browses for available mobile game services, UDDI will perform a find inquiry for all services which has UNSPSC 43223208 value. The registry will return a list of services and the customer has to make the final decision to determine the service he intends to subscribe or purchase. However, since this private registry is owned and hosted by a business entity, the operator may wish to prioritise the list of services to be displayed. Example like to show only selected services or rank services according to pre-defined business rules such as vendor priority or service popularity. This process should be automated and its mechanism transparent to consumers. However, present UDDI inquiry API does not support complex ranking function, and is not able to support this requirement.

The requirements become more complicated for dynamic service discovery scenario as it is the application logic which has to determine which service to invoke upon receiving more than two matches in a service discovery result. In such scenario, there is a need for a unique and linear parameter which can be used as additional or final reference, to assist in the final decision making. Hence there is a need to have a separate mechanism to publish and retrieve these external parameters.

In this paper, we implement and evaluate two practical approaches to customise registry query results according to static and dynamic parameters values as proposed in [19]. However, we have generalized the proposed models into two broader approaches. The first approach, known as *Internal Parameters Approach* has all parameters stored as data records within the UDDI registry; while the second *External Parameters Approach*, has parameters stored in external resources, such as in text file, database, log files, etc. Both approaches adhere to UDDI version 2 standards, and customisation of service discovery results is handled by a proxy who acts as intermediary to intercept the UDDI query results, and manipulate the records before returning the final list to service requestor.

Our contributions are:

(1) We present the existence of a requirement to customize service registry query results in practical usage scenario. This happen when a service discovery return two or more matches and the results listing needs to be personalized according to certain business criteria.
(2) We propose two approaches that enable the personalization of registry query results by referencing to personalization parameters which could be accessed either within or outside the registry.
(3) The design, implementation and evaluation of the two approaches. Based on the performance analysis, we have demonstrated the approaches are feasible as complementary to semantic enabled service discovery for enhancing service discovery in a private UDDI environment.

The rest of the paper is organized as follows. Section III discusses the related works. We have developed a UDDI registry testbed to implement and evaluate both approaches, as discussed in section IV. Section V describes details of experimental setup and objectives. Section VI presents the performance experiment results and gives detailed analysis. We finally summarize the strength and weaknesses of each approach and conclude this paper with notes on future work in this research area.

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. [6][7]. 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 [8]. 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, [9] 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 [10] which introduced the use of agent to automatically establish ranking capabilities to web services and [11] 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 [12] suggested an approach to use dedicated server to collect feedback of users and predict the future performance of published services.

IV. TestBed Architecture And Implementation

A. Architecture

Both approaches described in Section II share some common architecture components as shown in Figure 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 business list or service list to *User Interface*.

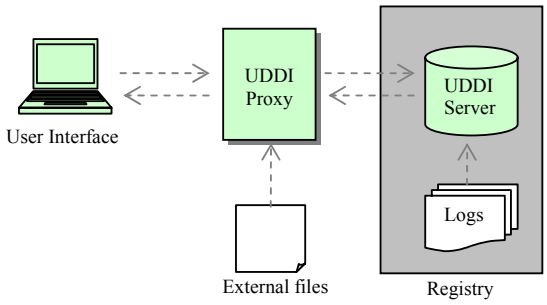

Fig. 2 Proposed model architecture

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 [13]. *User interface* allows a requester (or consumer) 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 *UDDI Proxy* acts as an intermediary between the *User Interface* and the *UDDI Server*. Its main function is to intercept the UDDI query result and rearrange the records based on certain pre-defined criteria. The criteria can be created either at design time or run time.

B. Static and Dynamic Parameters

This paper describes two alternative approaches to personalise UDDI query result based on criteria managed by the UDDI administrator. The business criteria will be mapped to certain variable parameters to manipulate the final query result. These parameters are generally grouped into two types: static and dynamic [19]. The above parameters usage according to business and service entities is summarised in Table 1.

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.

The *static parameter* will hold certain values which has been fixed and do not change during run-time. 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. One example 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 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.

One of the usages of UDDI *tModel* is to define a namespace used to identify entities or classify business services. The namespaces (*tModelKey*) are used in the *identifierBag* and *categoryBag* elements, which will be referenced by

keyedReference element for categorization and identification purposes. We take advantage of this classification scheme feature to define new schemes for static and dynamic parameters. Figure 3 shows the *tModel* definition to represent static and dynamic parameter.

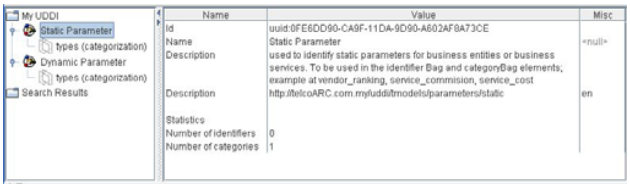

Fig. 3 (a) *tModel* Definition for Static Parameter

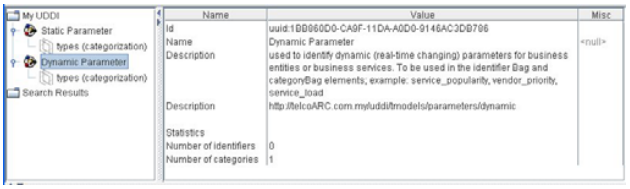

Fig. 3 (b) *tModel* Definition for Dynamic Parameter

C. Implementation

Our implementations for both approaches are based on the assumption that 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 approach are (1) the location on where the parameter values are stored and retrieved; and (2) ranking mechanism. Each approach's requirements, ranking algorithm and implementations are further elaborated in this section.

i. Internal Parameters Approach

This approach involves two main components: *UDDI Proxy* and *UDDI Server*, as shown in Figure 4, where the parameters will be saved inside the UDDI server itself.

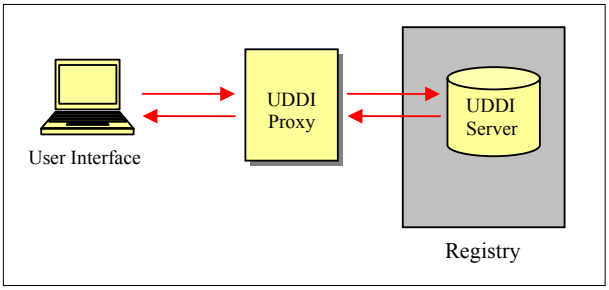

Fig. 4 Internal Parameters Approach proposes parameter values to be saved and retrieved from UDDI server

After the static and dynamic parameters are published as *tModel,* a *keyedReference* element which contains reference to the parameter *tModelKey* will be added to the *category* bag of *businessEntity* or *businessService* element. The term "bag" indicates a generic container of multiple values, and enables a company to register multiple business identifiers or categories.

To further illustrate the example above, Figure 5(a) shows a *BusinessEntity* record includes a *keyedReference* element called *Vendor_Ranking* which reference to a *static parameter tModel* in its category bag. Figure 5(b) shows similar usage scenario for *dynamic parameter*, which is referenced *Service_Popularity* parameter.

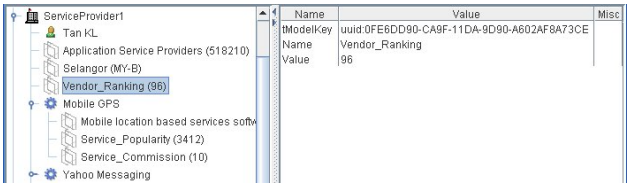

Fig. 5 (a) *Vendor_Popularity* referencing static parameter *tModelKey*

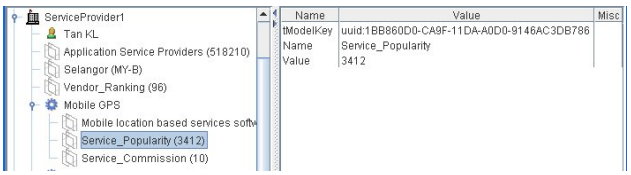

Fig. 5 (b) *Service_Popularity* referencing dynamic parameter *tModelKey*

During service discovery, 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 and retrieve the associated parameter values. Certain *Find Qualifiers* can also be used to enable more precise search criteria. If personalization of query is required, the *UDDI Proxy* will process the list accordingly, such as rank using the embedded parameters values retrieved from *UDDI server*. Once processing is done, the new ranked list will be sent to user interface, and all the parameters values will be discarded. The algorithm for this approach is presented in Figure 6.

1. After receiving UDDI query result, check if personalisation is required. If not, proceed to step 13.
2. Store query results in a dynamic array.
3. For personalisation of business query results, proceed to step 4, else for business service, proceed to step 8.
4. For each business record, invoke *getBusinessKey()* method to retrieve the business key.
5. Using business key, retrieve personalisation parameter values from identifier or category bag.
6. Store business name and personalisation parameter values in a temporary array.
7. Repeat Step 3 to 6 for all business records in dynamic array
8. For each service record, invoke *getServiceKey()* method to retrieve the service key.
9. Using service key, retrieve personalisation parameter values from category bag.
10. Store service name, its associated business name and personalisation parameter values in a temporary array.
11. Repeat Step 7 to 9 for all service records in dynamic array
12. Sort the temporary array records according to parameters value.
13. Send the list to *User Interface*.

Fig. 6 Algorithm to rank UDDI query result using Internal Parameters Approach

The main advantage of the first approach 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.

ii. External Parameters Approach

This second alternative approach is based on [19] where parameter values are stored and accessed outside UDDI registry, which could be external resources such as text file, database, logs, etc. As shown in Figure 7, the parameters should be accessible directly from the Proxy, outside the UDDI server. For external file, it can either be in pipe-delimited or even XML format. *File A* is used to store values for static parameters *File B* is used to store values for dynamic parameters. There is a need for a separate mechanism to publish and update the parameter values to the external resources; however this subject matter is beyond the scope of this paper.

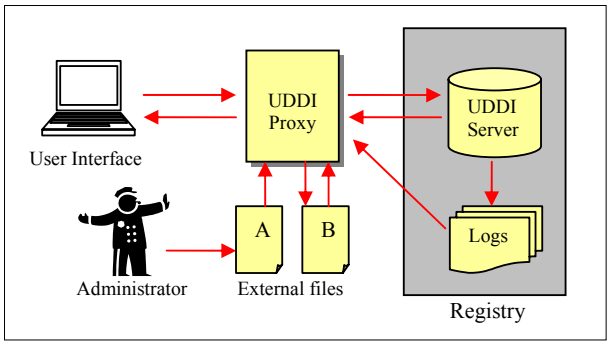

Fig. 7 Parameter values to be saved and retrieved from external resources such as text files or server logs

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 the 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 [14]. These logs data could provide useful semantic information for ranking criteria. Certain dynamic parameter values such as service or vendor popularity can be calculated based on client accessing pattern [23][24] or by extracting the data from log files of SOAP server, application server and *UDDI server* [17]. However this will require 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.

The algorithm in Figure 8 shows the necessary steps to be performed by the *UDDI Proxy* in order to retrieve both parameters values from external resources.

1. After receiving UDDI query result, check if personalisation is required. If not, proceed to step 9.
2. Store query results in a dynamic array.
3. For each record, retrieve the business/service key.
4. Group and store business/service names and keys into a temporary array A. Create a parameter value column for each business/service key record.
5. Open the external file which stores the personalisation parameters data.
6. Read all business/service keys and their values from the file and store into array B.
7. For each record in Array B, check if the business/service key matches the business/service key in array A.
8. If found match, retrieve the corresponding parameter value from array B and store into parameter column in array A.
9. Repeat Step 7 and 8 for all records in array A.
10. Sort array A according to required parameter values and make necessary formatting.
11. Send the list to *User Interface*.

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

This approach introduces distributed storage of the parameters data, it has the advantages of lowering the *UDDI Server* load, and provides a better control over the external files. However, with more control, the tradeoff is *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. Another advantage of this approach is that 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].

V. Experiment On Performance and Scalability

In this section we describe our assumptions, experimental setup, testing procedures and test cases used to implement the two approaches.

A. Experimental Assumptions

In evaluating the performance of customising query results, jUDDI registry was chosen from several other UDDI implementations surveyed. Since all registry originated from the same specification, we assume the underlying data modeling and its data accessing/retrieving API are similar. jUDDI is a Java-based implementation of UDDI that was developed to integrate effectively with Apache Tomcat

application server.

In this work, we only evaluate one standalone UDDI server which does not reflect on the benefit of multiple distributed registries which UDDI supports. We assume the test scenario is for a private registry which store and advertise mobile services to a specific group of customers, which would only connect to a limited set of registries that address a very specific group of services. Another assumption made is there will be only one query session carried out at any one time. In another word, no concurrent queries submitted to the same registry simultaneously.

B. Testbed Setup

Our testbed consists of one UDDI registry server which is accessed by a Web service client across a local area network. The Web service client is implemented with UDDI4J [20], an open-source Java class library that provides an API to interact with a UDDI registry. Table 2 shows a list of hardware and software of the server and client machines. Figure 9 details the testbed components setup. We also run UDDI Browser [18] that provides a friendly user interface to browse and manipulate content in UDDI registries for verification purposes.

TABLE II
HARDWARE AND SOFTWARE ENVIRONMENT

	Hardware	Software
UDDI Server	Dell Intel Pentium (M) 1.73GHz, 1GB Memory	jUDDI ver. 0.9rc4 Apache Tomcat ver. 5.0.28 Apache Axis ver. 1.2 JDBC ver. 2.0 MySQL ver. 5.0
Client	Dell Intel Pentium (M) 2GHz, 2GB Memory	UDDI4J ver. 2.0.3 Java SDK UDDI Browser

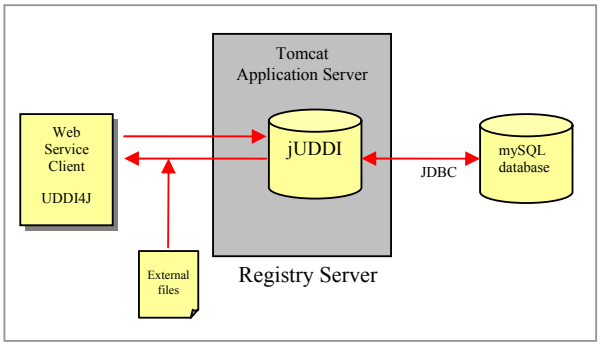

Fig. 9 Testbed Component Setup

C. Testing Procedures

The implemented UDDI registry emulates a Telecommunication Service Operator private registry, used to store and publish mobile services to its customers. A mobile consumer will request query and receive results via *UDDI Proxy*. The query results will be ranked depending on static or dynamic parameters as discussed in section IV.B. We have developed a Web service client using Java and UDDI4J, which is used to perform the testing procedures as follow:

1. Capture start time, *Start_time*.
2. Submit a UDDI query request and retrieve results.
3. Retrieve parameter values from query results, or from external file.
4. Customise the query results. Sort the results order according to certain parameter values.
5. Display the customised results. Capture end time, *End_time*.
6. Calculate *Response time = End_time – Start_time*

D. Test Cases

We have identified three test cases to evaluate the impact of personalization of service query results to performance (response time). Each test case will be executed for each of the followings: Normal condition (when no approach is applied), *Internal Parameters Approach*, *External Parameters Approach* and. The client application is programmed to capture response time (in second) for these test cases:

Test A - Query for all business records and rank according to *vendor_ranking* (static parameter). This is the simplest form of find query used to measure the performance impact for small/medium query data size. The query results obtained will be sorted according to *vendor_ranking* in ascending order. Example of the CLI output for Test A is shown in Figure 10.

```
          Find Business By Static Parameter
********** (Internal Parameters Approach) **********

Static Parameter: Vendor Ranking

Ranking        Business
1              ServiceProvider6
2              ServiceProvider24
3              ServiceProvider43
4              ServiceProvider34
...
98             ServiceProvider10
99             ServiceProvider32

Query Response Time (ms) = 11704
```

Fig. 10 Example of CLI Output for Test A

Test B - Query for all service records and rank according to *service_popularity* (dynamic parameter). Each business has five associated services, hence total number of *businessService* records are five times more than of *businessEntity*. This test case will query for all *businessService* records, and display the query results sort according to *service_popularity* in descending order. This test case measures the performance impact for large query data size. Figure 11 shows example of Test B CLI output.

```
        Find Business By Dynamic Parameter
********** (Internal Parameters Approach) **********

Dynamic Parameter: Service Popularity

Popularity      Service         Business
9989            Mobile GPS      ServiceProvider4
9853            Online Games    ServiceProvider19
9853            Online Games    ServiceProvider3
9853            Online Games    ServiceProvider34
...
263             Mobile GPS      ServiceProvider8
126             Mobile GPS      ServiceProvider48

Query Response Time (ms) = 106844
```

Fig. 11 Example of CLI Output for Test B

Test C - Query for service records limited to those categorized under UNSPSC taxonomy and ranked according to *service_commission* (static parameter). This is similar to Test B, but the service records are filtered to one specific category. The filtered records are then sort according to *service_commission* in descending order. This test case measures the performance impact for complex query scenario, and the sample CLI output is shown in Figure 12.

```
          Find Service By Taxonomy And Sort
            According to Static Parameter
********** (Internal Parameters Approach) *********

Taxonomy: UNSPSC (43233508, Mobile operator specific
application software)
Static Parameter: Service Commission

Commission      Service                 Business
70              SMS News        ServiceProvider19
70              SMS News        ServiceProvider3
70              SMS News        ServiceProvider34
70              SMS News        ServiceProvider40
...
20              SMS Stock Quotes    ServiceProvider6
20              SMS Stock Quotes    ServiceProvider9

Query Response Time (ms) = 71313
```

Fig. 12 Example of CLI Output for Test C

VI. Experimental Results

In order to evaluate our proposed approaches, we carried out two experiments on the testbed environment as discussed in previous section. In this section, we present the results of the experiments.

The first experiment analyses and compares the performance of our proposed approaches in Web services discovery. The UDDI inquiry function was executed 5 times sequentially and the average response time was recorded. The reason of taking average response times is to reduce the impact of response time inconsistency. We also compare the overheads of running the proposed approaches compare to the inquiry without retrieving additional parameters (the proposed personalized parameters i.e. vendor ranking, service popularity and service commission). The T-Test significant values are calculated based on the response times and overheads taken.

In the second experiment we evaluated the scalability of the two approaches by increasing UDDI query data size linearly.

i. Experiment I: Performance Analysis

In this experiment, we published 50 business entities (known as *Service Provider* in our registry), each with 5 different services into our registry. Total time taken for the *Web Service Client* to submit a query, plus the UDDI proxy to retrieve the query results from UDDI database server and to sort the results according to a given parameter values is recorded. For *Internal Parameters Approach*, static parameters, *vendor_ranking* and *service_commission* are assigned to each business and service respectively. Besides the static parameters, a dynamic parameter *service_popularity* is also assigned to each service.

Figure 13 shows snapshot of a business entity with its associated services. For *External Parameters Approach*, all the personalised parameters are stored in an external file.

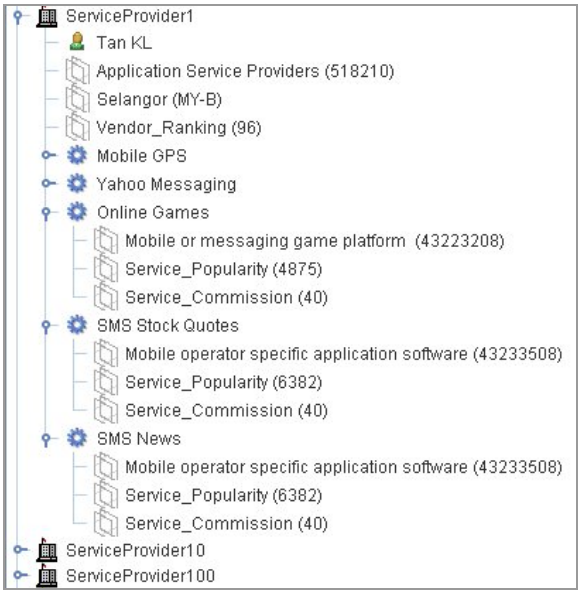

Fig. 13 Snapshot of a business entity Service Provider 1 with its associated services

Figure 14 shows the first experimental results for different test cases:

- Test case A: Search for all 50 busines records, sorted by *vendor_ranking*.
- Test case B: Search all 250 service records, sorted by *service_popularity*.
- Test case C: Search for specific 100 service records, sorted *service_commision* and filtered by taxonomy.

Based on the average response times taken, both test cases A and B showed that *External Approach* performs faster retrieval compares to *Internal Approach* with T-Test significant less than 0.01%. For test case C, the results

demonstrated that both approaches have insignificant difference (T-Test significant at the level of 44%) in response time performance.

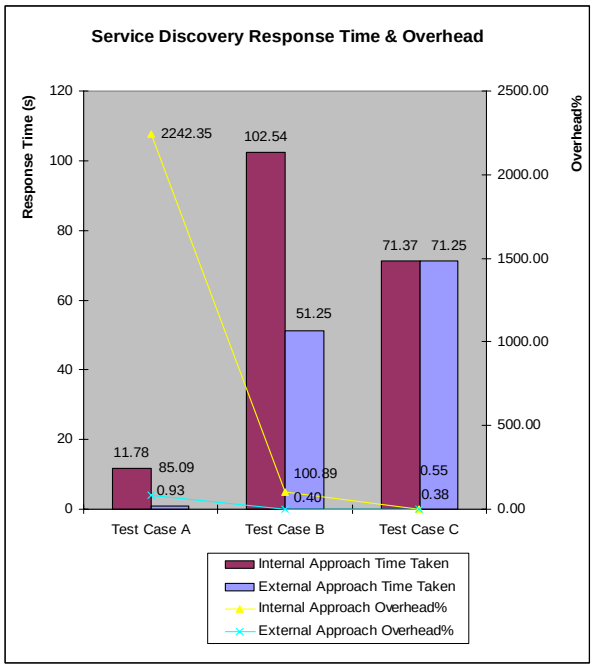

Fig. 14 Experiment Results for Performance Testing

Similar pattern showed in overhead consumption as to average time taken. In both Test Cases A and B, *External Approach* consumed less overhead compares to *Internal Approach* with significant at less than 5% level in all results of both cases. The overhead percentage differences between the two approaches for test cases A and B were 2235% and 100% respectively. This shows that the performance overhead caused by the customising query results is severe in small to medium data query size. For test case C, *Internal Approach* demonstrated that it consumes more overhead by 48% than *External Approach* with significant at the levels of 4% and 1% respectively.

To compare results across the three test cases, *External Approach* significantly demonstrated better performance in small to medium data size retrieving with results sorting. However, both approaches did not show obvious difference in performing retrieval with filtering condition.

ii. Experiment II: Scalability Evaluation & Comparisons

In the second experiment, we analysed the performance of both approaches when size of UDDI query results linearly increases. We published 20 business records, each with 5 associated services and measure the response time taken for UDDI proxy to submit a query to UDDI, retrieve the query results and systematically sort the results by *service_popularity* parameter.

Experiment result in Figure 15 shows the response time increases linearly with the number of records for the *Internal Parameters Approach*, but *External Parameters Approach* showed the tendency of reaching its threshold in retrieving huge record size (as shown in our experiment after 4000 records in Figure. 15). *Internal Parameters Approach* showed more stable and consistence in retrieval time growing pattern, which is mainly due to the use of more reliable and efficient database storage (mySQL) as compared to ASCII text file for *External Parameters Approach*.

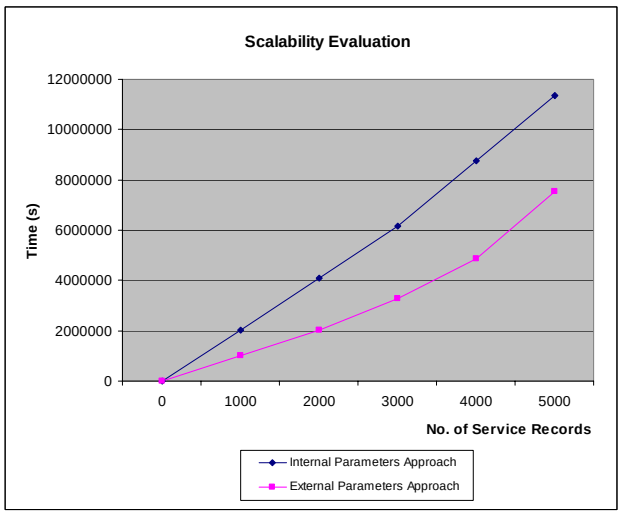

Fig. 15 Experiment Results for Scalability Testing

VII. Conclusion and Future Work

In this paper, we have presented two alternative approaches to customise private UDDI registry query results, using personalisation parameters which could be stored within or outside the service registry. Conceptually, the practical approaches can be applied to other SOA registry candidate such as ebXML. We have also measured the performance and scalability of each approach. Based on the experiment results, we conclude that storing the parameters in external resources is a more efficient approach as compared to keeping the parameters as *keyedReference* value within UDDI. A closer investigation shows the main cause of performance degradation is due to time taken by *UDDI proxy* to obtain various UUID keys before it could retrieve the parameter values from the *category bags*.

Table 3 summarises the main characteristics for each approach.

TABLE III
COMPARISONS OF TWO APPROACHES

	Internal Parameters Approach	External Parameters Approach
Parameters location	Within Registry	Outside Registry
Implementation Complexity	Easier	More Complex
Effect on registry performance	Lower	Higher
Scalability	Stable and consistent	Tendency to reach threshold in retrieving huge record size

The two proposed approaches are designed to suite different needs of private registry systems. These approaches implementation will 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 be focussing on the refinement and implementation of the two approaches. We are investigating the possibility of combining the *Internal Parameters Approach* with semantically enabled service discovery mechanism, as it will offer dynamic service discovery and invocation capabilities. We are also studying the usage scenarios that will potentially benefit from the combined approach. Another interesting area is to further investigate the possibility of retrieving and dynamically create the personalisation parameters from external resources/services, such as server logs, customer relationship management services, network monitoring services or Service Level Agreement contracts.

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.
[18] The UDDI Browser Project http://uddibrowser.org/
[19] Kee-Leong Tan, Cheng-Suan Lee, and Hui-Na Chua, Models to Customise Web Service Discovery Result Using Static and Dynamic Parameters. 2nd International Conference on Computer Science, Vienna, Austria, 2006. 198-204.
[20] UDDI4J - a Java Class Library to Interact with UDDI http://uddi4j.sourceforge.net/
[21] North American Industry Classification System (NAICS) http://www.census.gov/epcd/naics02/naicod02.htm#N51
[22] United Nations Standard Products and Services Code (UNSPSC) http://www.unspsc.org
[23] Dwi H. Widyantoro, Thomas R. Ioerger, John Yen, "An Adaptive Algorithm for Learning Changes in User Interests", Eighth International Conference on Information and Knowledge Management, 1999.
[24] C.R. Anderson, P. Domingos and D.S. Weld, "Personalizing Web Sites for Mobile Users", Proceedings of the 10th International WWW Conference, Hong Kong. May 1-5, 2001.