Interoperability among Service Registry Implementations: Is UDDI Standard Enough?

Alexander Mintchev
SAP LABS Bulgaria
a.mintchev@sap.com

Abstract

The aim of this paper is to reveal some intrinsic disadvantages of the current version of UDDI standard, which create problems in using it as a standard for private, in-house storage of enterprise services. Examples include: access control mechanisms in UDDI, limited rich queries capability, inappropriate mapping of web-service artifacts into UDDI entities, impossibility of managing classification system values, etc. For each disadvantage we consider, we give an illustrative example of its impact in an enterprise service environment. To overcome these disadvantages, some service registry implementations, based on UDDI introduce proprietary extensions to the standard, or embed additional programmatic logic in their client modules of UDDI, which decreases interoperability between them.

1. Introduction

The current work is inspired by our experience as SAP Service Registry developers. Some of the challenges we successfully overcome when developing the SAP Services Registry gave us the impetus to create this article. It reveals some problems in using UDDI standard as a base for private, in-house storage of web-services and is intended for IT managers, enterprise architects, and software developers.

We aim at drawing industrial and standards' body community attention towards the need of re-thinking the usage of the UDDI standard in service registry implementations taking into account the issues addressed here.

2. Related work

Recently there has been an increasing interest towards interoperability among service registry standards [1], and service registry implementations [2], [3], [4], [5], [6]. While Masri and Mahmod [1] claim that *"the standards' distinctive approaches to discovery make it unclear whether the provisional specifications will eventually merge or coexist"*, the Gartner report [2] ascertains that UDDI standard is used (directly or in supplementing modules) by all SOA vendors it investigates (IBM, Microsoft, Oracle, SAP). In an earlier research [3], we suggested an implementation of a tool that transfers data between any two commercially available UDDI implementations, eventually filling in an interoperability gap among them.

3. Service Registry vs. Service Repository

A service registry is defined as *"a specific type of repository that allows companies to catalog and reference the resources required to support the deployment and execution of services"* [20]. While a registry is a storage containing references to resources, a repository is storage, containing the resources themselves [20 p.3]. These two storages may be physically separated, or could be one and the same, depending on the implementation.

We define a service registry, respectively a service repository, as storage of (web)-service descriptions according to the WSDL 1.1 specification [7].

In this paper, we limit our considerations to service registries.

4. UDDI is the common standard of service registry implementations

There are two common registry standards – UDDI [8] and ebXML. For each of them there exists an accompanying technical note [9], [10] specifying how to store service descriptions according to WSDL specification in a UDDI registry and in an ebXML registry.

Major software vendors, such as BEA, HP, IBM, Microsoft, Oracle, SAP, Software AG, and Sun ship a service registry product implementation [2], [4], [6], [14], [15], [16], [17], [18]. All of these vendors, with the exception of IBM, emphasize that their service registry product does support the UDDI standard. Although IBM's Service Registry product, called WSRR *"is not using the UDDI standard"* [4], IBM provides a UDDI synchronization module that synchronizes the content of their WSRR product with any UDDI v3 registry.

Regarding ebXML, Software AG [6] and Sun [14] service registry implementations claim to support it. While Sun *"clearly prefers ebXML over UDDI"* [6],

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.70

other vendors do not provide indications to support ebXML in their service registry products.

Therefore, the UDDI is the only common standard for storing service descriptions according to WSDL specification in all major service registry implementation products. Hence, if we are looking for interoperability between them, it must be achieved based on this standard.

5. Architecture and Requirements

In this section, we present an architectural view of a service registry and define the major requirements towards any service registry. The view is based on the SAP Service Registry [17]. Other service registry views can be found in [4] for IBM WSRR and [15] for HP Systinet Registry.

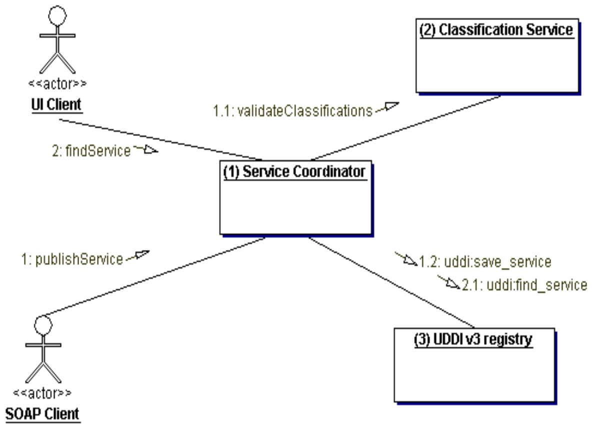


Fig. 1 An architectural view of a service registry using UDDI v3 as the back-end, source [17]

The SAP Service Registry [17] consists of a service coordinator module (1) that accepts SOAP/ EJB calls, processes them and calls the SAP classification service (2) to save taxonomies or validate values and any UDDI v3 registry to publish / inquiry WSDL entities. Performing similar tasks, HP Systinet Service Registry [15] uses a UDDI v3 registry as a back end to publish / inquire WSDL entities, which also exposes additional, custom APIs for saving of taxonomies and validating values. IBM Service Registry [4] does not use a UDDI registry as a back end, but instead a proprietary persistence layer. If presented in terms of Fig. 1, IBM Service Registry would lack the UDDI registry module (3). Instead it will publish / find WSDL entities still in module (1). However, as IBM WSRR offers an option of synchronizing its content with any UDDI registry, (3) could be considered an optional module to it.

In [20] and [25] the following requirements towards a service registry are defined:

5.1) Discovery (Categorization, Rich query capability, Dynamic location and binding)

5.2) Governance (Approval workflow, Compliance validation, Model, and manage dependencies)

5.3) Lifecycle Management (Versioning, Subscription, Impact analysis, Support for lifecycle stages and roles)

Requirements 5.2 and 5.3 are not directly supported by the UDDI standard (apart from the optional Subscription API). However, they could be enforced in UDDI v3 registries using standard categorization of entities.

Two of the requirements from 5.1 Discovery - Categorization and Dynamic location and binding are directly supported by the UDDI v3 standard. However, *rich query capability* is definitely not an inherent feature of the standard. Instead it is one of its major weaknesses.

6. Interoperability defined

Let us have two service registry implementations – SR1 and SR2 from two different vendors – V1 and V2.

6.1) Interoperability is defined as the ability of SR1 to exchange data with SR2 and to work with the exchanged data.

6.2) In terms of registry architecture of section 5, interoperability is defined as the ability of a Service Coordinator Module (1) of a SR1 to work with the UDDI v3 registry module (3) of SR2 and vice versa (see fig.1). Under this definition, there is no data exchange, but an ability to switch UDDI v3 registry modules between different vendors, while preserving the original Service Registry functionality.

7. Challenges with UDDI as a standard for enterprise storage of web-services

7.1 Inappropriate mapping of web-service artifacts into UDDI

Impact: Low performance of UDDI-based service registries or broken interoperability among them

Proof

Originally, UDDI was not designed as dedicated storage of WSDL entities (web-service descriptor references), but rather as a general registry for storage of businesses and services offered by them (including services offered by phone, fax, e-mail).

WSDL entities are mapped into UDDI artifacts according to an OASIS Technical Note [9]. According to this note, a wsdl:portType is mapped into a UDDI tModel, so that the name of tModel is equal to the name of the wsdl:portType. Consequently, the UDDI response to a find_tModel request – an array of tModelInfo elements - contains information, which is insufficient, if not irrelevant in the world of WSDL. Of all the items returned in a tModelInfo element - name, description and UDDI key of the tModel, only the name bears meaningful information. The namespace of the wsdl:portType, which is mapped in the categoryBag of a tModel, is simply not included in the response. Consequently, if the response contains multiple tModelInfo elements with equal name, there is no way to distinguish which one is the actual portType sought, without additional requests to UDDI registry. This poses a performance problem on any service registry based on UDDI. A similar problem occurs if inquiring UDDI for *"all portTypes having name1 OR name2 OR name3… OR nameN*". As there is no way to include multiple name elements in a UDDI find_tModel request, the above statement would require ***N*** separate individual requests to UDDI, where ***N*** is the number of ***OR*** operators.

We outline the following solutions to the problem:

1.) Develop a custom extension to a UDDI that returns the *catogoryBag* of a *tModel* as a response to the *find_tModel* request. However, relying on this custom extension may break interoperability in terms of definition 6.2.

2.) Extending the mapping of a wsdl:portType into uddi:tModel [9] by repeating the portType namespace information into the description of the tModelInfo entity. This solution may also break the interoperability in terms of definition 6.2, especially if a UDDI client expects particular information in each tModelInfo description.

3.) Develop programmatic logic in UDDI client tools that performs additional requests to UDDI to obtain the namespace of a wsdl Port type for each find_tModel request. This would preserve interoperability between service registries, however at the cost of decreasing performance.

7.2 Interoperability between UDDI implementations

In terms of definition 6.1, no interoperability between service registry implementations based on the UDDI standard currently exists. Once having bought a UDDI implementation of a particular vendor and having stored their data into it, customers are not able to transfer their data into any other UDDI v3 implementation, as neither there exist a commercial implementation of the UDDI standard that supports the optional UDDI v3 replication API [3], nor there exist a commercial tool that is capable of transferring user data between any two UDDI v3 compliant registries. Indeed, [3] offers a design of such a tool, yet no commercial implementation of it is available on the market.

7.3 Access control mechanisms in UDDI

The UDDI standard does not specify any limitations about the visibility of entities stored in a UDDI registry - every user of the registry can inquire, and hence, see any piece of information there. In contrast to the time, when UDDI was predominantly used in UBR (UDDI Business Registry), this is unacceptable in times, when UDDI is used as enterprise storage of web-services. Consequently, different approaches were suggested [12], [13] that introduce access control mechanisms in UDDI, but none of them is officially endorsed by the UDDI standard. We can then summarize that any extension to a UDDI registry that introduces some kind of an access control mechanism would make the service registry product that uses it not interoperable in terms of definition 5.2.

7.4 UDDI's rich queries capability

Section 5 outlined rich query capability as one of the major requirements towards Service Registries. In this section, we reveal some major disadvantages of UDDI in this respect.

7.4.1 Challenge: No Logical Not operator in UDDI

Impact: Inability of UDDI to solve fundamental service registry problems

Proof:

Let us refer to an overview of the SAP Services Registry found in [17], p. 28. One of the purposes of the SAP Service Registry is to answer the question "Which services are modeled or implemented in the landscape?" as opposed to the question "Which services can be invoked in the landscape? The first question is meant to represent those interfaces and implementation of services, which cannot be invoked yet, as they do not have callable endpoints. The second question, in turn, is meant to represent only those service definitions, which can be invoked, i.e. have callable endpoints. We show that because UDDI lacks a logical not operator, it is not capable of answering the first question.

In UDDI terms, the first question would be interpreted as *"find all tModels that are of type portType, for which there does not exist a binding template"*, while the second question would be interpreted as *"find all tModels of type portType, for which there exist at least one binding template"*

The first question cannot be represented in UDDI, as it does not define a logical NOT operator.

The second question could be represented by the following UDDI *find_binding* request:

Table 1: Finding all service end-points in a landscape

```
<find_binding          xmlns="urn:uddi-org:api_v3"
maxRows="X"  listHead="Y">
     <findQualifiers>
       <findQualifier>orAllKeys</findQualifier>
     </findQualifiers>
     <find_tModel>
       <categoryBag>
         <keyedReference
tModelKey="uddi:uddi.org:wsdl:types"
```

```
keyValue="portType"/>
       </categoryBag>
     </find_tModel>
   </find_binding>
```

The result of the above request would be an array of uddi binding template elements - each one containing the UDDI key of the portType referred, as well as the callable endpoint for this portType. To answer the first question, one would have first to answer the second question – i.e. to find all portTypes, for which there exist a callable endpoint. Then, one would have to find the set of all possible portTypes in UDDI using the following request:

Table 2: Finding all portTypes a landscape

```
<find_tModel xmlns="urn:uddi-org:api_v3">
       <categoryBag>
         <keyedReference
tModelKey="uddi:uddi.org:wsdl:types"
keyValue="portType"/>
       </categoryBag>
</find_tModel>
```

Finally, the answer to the first question is found as the difference of sets between the results of Table (2) and the results in Table (1). All portType tModel keys that are met in the response to the query in Table (2), but are not met in the response to query in table (1) are the ones, for which there is no endpoint defined.

The outlined solution relies on the existing capabilities of UDDI and has a bad performance, as it requires iterating over all possible portTypes in UDDI, without any built-in paging functionality. Introducing any cashing mechanisms in UDDI clients as an alternative approach has the cost of losing the real-time availability of the information. On the other hand, introducing additional, custom defined attributes in those portType tModels, for which there exist at least one binding template in UDDI could break interoperability in terms of definition 6.2.

7.4.2 Challenge: Limited adoption of nested find_tModel queries

Impact: Future SOA Governance capabilities not flexible

Proof:

Nowadays SOA Governance has become a buzzword [2], [6], having a broad scope of definition. In UDDI terms, SOA governance comes down to managing visibility, lifecycle and policy of WSDL entities. The ability to assign a particular policy to a WSDL entity and to find WSDL entities based on a particular policy is an essential part of SOA governance According to the rules for registering policies in UDDI [11], reusable policy expressions are mapped into separate tModel structures in UDDI. Consequently, if we have multiple wsdl:portTypes that refer one and the same policy expression tModel, it is not possible to find all wsdl:portTypes that refer the same policy expression in UDDI using a single UDDI request. This is due to the fact that UDDI does not have a standard means to embed a find_tModel query into another find_tModel query. It seems that the near future will impose new requirements in the SOA governance area, thus also requiring this ability. At the best case currently, we need two separate find_tModel requests to UDDI.

The first request to UDDI to find the uddi keys of the policy expression files is the following one:

Table 3: Finding all Policy Expression whose URI starts with http://www.example.com/, source [11]

```
<find_tModel xmlns="urn:uddi-org:api_v3">
   <findQualifiers>
<findQualifier>approximateMatch</findQualifier>
   </findQualifiers>
   <categoryBag>
      <keyedReference
keyValue="http://www.example.com/%"
tModelKey="uddi:schemas.xmlsoap.org:remotepoli
cyreference:2003_03"/>
   </categoryBag>
</find_tModel>
```

The second request to UDDI to find all portTypes that refer the uddi key of the policy file found as a result of the request in Table (3) is the following one:

Table 4: Finding all portTypes implementing a particular policy tModel, source [11]

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
       <categoryBag>
         <keyedReference
tModelKey="uddi:uddi.org:wsdl:types"
keyValue="portType"/>
         <keyedReference
tModelKey="uddi:schemas.xmlsoap.org:localpolic
yreference:2003_03" keyValue="uddi:abe……"/>
       </categoryBag>
 </find_tModel>
```

The inability to embed a *find_tModel* request into another *find_tModel* request in UDDI requires at least two separate requests to UDDI registry. If many policy expression files match the first query; additional *get_tModel* requests will be needed to locate the tModel of the policy expression sought. Consequently, there is no way to propagate paging mechanisms between the first and the second UDDI requests. As a result, additional programmatic logic must be built in UDDI client tools, which decreases performance of service registries using UDDI as a back-end. If service registries rely on custom extensions to UDDI to overcome the problem, this could break interoperability between them in terms of definition 6.2

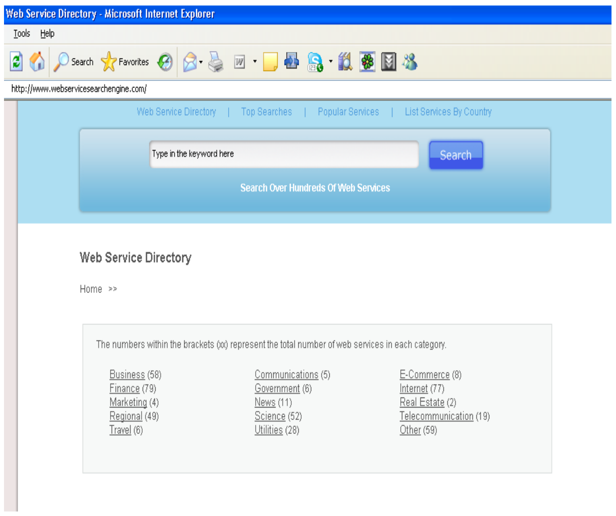
7.4.3 Challenge: No Group By Operator in UDDI

Impact: No real-time browse-by-category pattern

Proof:

We define browse by category pattern as the ability to browse entities in a UDDI registry based on the values of a particular classification system, with which these entities are classified.

A complex example of this pattern could be found in the public site of the SAP Services Registry [26]. However, for simplicity, we will consider the user interface of Wsoogle (Fig 2) - a public, global online directory of web services, available on the internet at [21].


Web Service Directory - Microsoft Internet Explorer
Tools
Help
Search
Favorites
http://www.webservicesearchengine.com/
Web Service Directory | Top Searches | Popular Services | List Services By Country
Type in the keyword here
Search
Search Over Hundreds Of Web Services
Web Service Directory
Home >>

The numbers within the brackets (xx) represent the total number of web services in each category.

Business (58)	Communications (5)	E-Commerce (8)
Finance (79)	Government (6)	Internet (77)
Marketing (4)	News (11)	Real Estate (2)
Regional (49)	Science (52)	Telecommunication (19)
Travel (6)	Utilities (28)	Other (59)

Fig. 2 Browse by Category Pattern –an exempt of a UI of an online web-service directory UI, source [21]

The main page shows different categories of web-services, each grouping services in a particular industrial activity - business, finance, marketing, etc.

Let us further assume that Wsoogle represents a service registry having an architecture defined in Fig.1. Let us also assume that industrial activities are represented in UDDI using the category system NAICS, as described in [24]. This means that web-service definitions (wsdl:portTypes) are represented in UDDI as tModels, categorized with a keyed reference using the key *uddi:uddi.org:ubr:categorization:naics:1997.*

Let us find out how would a service registry using UDDI as a back-end provide a browse-by-category functionality to a UI front-end using solely UDDI standard requests.

First, the service registry quires the underlying UDDI registry to find out all uddi:tModels (wsdl:portTypes) classified with the category system with key *uddi:uddi.org:ubr:categorization:naics:1997.*

The UDDI query looks as follows:

Table 5 finding all portTypes classified with the values of a given classification system

```
<find_tModel        xmlns="urn:uddi-org:api_v3"
maxRows="X" listHead="Y">
  <findQualifiers>
<findQualifier>approximateMatch</findQualifier>
    </findQualifiers>
    <categoryBag>
      <keyedReference
tModelKey="uddi:uddi.org:wsdl:types"
keyValue="portType"/>
   <keyedReference    tModelKey="uddi:uddi.org:
ubr:categorization:naics:1997" keyValue="%"/>
    </categoryBag>
</find_tModel>
```

If the UDDI registry contains tens of thousands of entities, then, due to performance reasons, the response to the above *find_tModel* request would be determinable in chunks, using a simple paging mechanism with the help of the attributes *listHead, includeCount,* and *actualCount*, as described in UDDI specification [8]. The result of the above request would then be an array of *tModelInfo* structures, each one representing a portType *tModel*, classified with any value of the classification system *uddi:uddi.org:ubr:categorization:naics:1997.* The problem is that there is no way to request that the UDDI registry returns these *tModelInfo* structures grouped by the values of the classification system, with which the corresponding *tModels* are classified with. There is even no way to determine the number of different classification values (the number of different groups), with which there exist at least one portType classified. This would have been possible, if UDDI contained an operator that is equivalent to the SQL Group By operator.

Our careful reader might oppose us that since the duty of UDDI is discovering and integrating, introducing of 'Group By' operator into UDDI would make the standard too huge and may lead to related changes in other SOA standards. This is true. But if we do not introduce 'Group By' operator in the UDDI standard itself, we must build the corresponding logic in UDDI client tools.

This would mean that a UDDI client must iterate over all tModels that match the request in Table 5, inspect their *categoryBag* child element and count the number of different values of the given classification system. This means developing a programmatic logic that performs the above *find_tModel* request N times, with increasing values of the *listHead* attribute until an empty response is returned. For each separate chunk response to the above *find_tModel* request, an additional *get_tModel* request is send to the UDDI registry to determine the details of the *tModels* contained in the *tModelInfos* structure. The *categoryBag* of each returned *tModel* is then inspected, the corresponding classification system value is located and the number of times it is met in the response is increased. Consequently, if there are ten thousand entities that match the above request, and the *includeCount* parameter in the response of the UDDI registry is limited to 100, then 100 *find_tModel* requests will be needed to obtain all matching *tModelInfo* structures. Additional *get_tModel* requests will be needed to obtain the complete *tModel* structures from the UDDI registry.

Unfortunately, alternative solutions would either break interoperability in terms of definition 6.2, if they rely on a proprietary extension to UDDI that allows the usage of group by operator, or will loose the real-time browsing if they are implemented as a UDDI client tool, working in a background mode, performing the above steps using standard UDDI requests and cashing the result.

7.4.4 Challenge: No arbitrary combining of logical AND and logical OR operators

Impact: No flexible run-time discovery of services

Proof

In SOA world, run-time discovery of service means discovering the callable endpoints of a service, as opposed to design-time discovery, which usually denotes discovering the service portTypes. Since a web-service endpoint is represented by a wsdl:port structure [7], which, in turn, is mapped into a UDDI binding template element [9], the run-time discovery of web-services comes down to the content of the UDDI binding template element and the corresponding UDDI *find_binding* request. Indeed, UDDI version three introduces major improvements in both these structures in comparison to version two. In version three the binding template element contains a *categoryBag* element, which allows categorizing each service end point with some categorizations criteria. On the other hand, the UDDI v3 *find_binding* request allows searching for binding templates based on its classifications. This allows UDDI client tools to pick-up the run-time provider of a given protType based on its endpoint classifications. Although a major improvement in comparison to previous version, this might yet not be enough for enterprise storage of web-services.

To illustrate, let us have several versions of a given portType, represented in UDDI as separate tModels, all of which sharing semantically the same set of operations (the same interface). Let each portType has a couple of endpoints in different geographical locations. Let it be equal to a client which actual portType implementation will be used, but very important to chose the run-time provider (the endpoint) with the nearest geographical location to the client, provided that it is callable using HTTP as transport and SOAP as protocol. These requirements would be represented in UDDI terms as "*find all binding templates, implementing portType1 OR portType2… OR portTypeN AND having a catgoryBag representing the nearest geographical location AND referring a tModel that represents a wsdl:binding with HTTP as a transport and SOAP as s protocol"*. Unfortunately, this request cannot be expressed in a single UDDI query, because UDDI does allow combining logical AND and logical OR operators in an arbitrary way.

7.5 Challenge: No Repository functionality Impact: Difficulties in managing complex SOA deployments

Proof

A recent white paper [14] concludes that *"services registries based solely on UDDI lack many important capabilities, including repository functions, which are necessary for governing and managing complex SOA deployments"*.

Indeed, UDDI does not offer repository functionality. The storage of resources themselves is outside the UDDI registry. Consequently, there is no standard way to search within the entire content of a given WSDL document and important parts of it, such as operation name, operation type, import declarations, are not searchable in UDDI.

In this respect most vendors either ship a single product that consists of a service repository and a service registry bundled together [4], [14] or ship two separate products –a registry and repository, but advertise them as two parts of a single product [15], [17].

7.6 Challenge: what is the "provider of services"? Impact: different mappings in UDDI entities

Proof

WSDL specification [7] defines the service as an element that "groups a set of related ports together", but remains silent about what the provider of a given service is. A wsdl:service is mapped into a UDDI business service element [9]. However, the UDDI standard requires that each business service is wrapped in a parent business entity element. What information does this business entity element include?

In UDDI terms, a business entity is meant to represent an entire organization (a legal entity). But private, in-house service registries are often owned by a single organization (single legal entity), in which service providers are separate organizational units (such as HR, Finance, etc.) or dedicated application servers, depending on the level of abstraction each service registry vendor applies.

In the absence of standards or specifications that govern what is the provider of services and how it is mapped in UDDI, it is not guaranteed that each service registry vendor would interpret "the provider of services" in the same way. Simply complying with the UDDI standard does not guarantee interoperability between different service registry implementations even if each vendor maps a service provider as a uddi:business entity element, because each service vendor might decide to store different attributes of the respective service provider in the uddi:business entity.

7.7 Challenge: Management of classification systems and their values

Impact: proprietary extensions to UDDI and custom developed software modules

Proof

Classifications of entities in UDDI are provided by checked and unchecked value sets. A values set is represented by a tModel, in which users could store meta-information about the value set as a whole, such as name, description, purpose. However, there is no standard means for users of UDDI registry to add values to a checked value set (classification system).

The SAP approach to this issue is to provide a custom developed module, called classification service (Fig.1), which is tightly coupled in the SAP service registry and gives users the ability to create, update, and delete classification systems, as well as values associated with these systems.

The HP Systinet approach is to provide a proprietary extension to UDDI – called Taxonomy API, which provides a high-level view of taxonomies and makes them easy to manage and query. Users can also plug a validation service to their HP Systinet UDDI implementation, as described by [22].

IBM's WSRR approach is to represent a classification system as ontology, which can be described in an OWL file, according to [23]. Entities stored into WSRR can be managed and sought for based on these OWL classifications.

The approaches of IBM, SAP and HP are similar, yet different. A client tool working with any of these service registries cannot expect the same behavior. In addition, as classification system values are stored in a proprietary way, it could be hard to achieve interoperability in terms of either of definitions 6.1 or 6.2.

8. Conclusion

The rapid growth of enterprise SOA solutions on the market seems to occur without always having the right standards to back up their performance, interoperability and extensibility. One such example is the market for service registries. While most of the solutions currently available are based on the UDDI v.3, this standardization does not always seem to offer them high performance, interoperability or extensibility. In the current paper we have revealed some disadvantages of using the UDDI v3 as a standard for enterprise storage of services.

We firmly believe that the future belongs to standards, and hope that whatever comes after UDDI v3 would be an open standard (extension), jointly contributed to by all parties interested.

9. References

[1] E. Al-Masri, Q. H. Mahmoud, "Interoperability among Service Registry Standards", *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, May 2007 pp 74-77.

[2] L. Frank Kenney, Daryl C. Plummer, "Evaluating IBM, Microsoft, Oracle and SAP Commitment to SOA Governance", *Gartner RAS Core Research Note G00150869*, October 2007

[3] A. Mintchev "Transferring Data between UDDI v3-Compliant Repositories", in *Expanding the Knowledge Economy: Issues, Applications, Case Studies*, IOS Press, Amsterdam, October 2007

[4] IBM WebSphere WSRR to SAP ESR interoperability methodologies: http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wsapiw/wsapiw/5.0/Next_gen_tech/P22_WSRR_ESR_Part2/player.html

[5] S. Rothaugh, "Interoperability Options with the Services Registry". *SAP TechEd, Munich,* October 2007, http://www.sapteched.com/emea/edu_sessions/session.htm?id=732

[6] L. Frank Kenney, Daryl C. Plummer, "Magic Quadrat for Integrated SOA Governance Technology Sets", *Gartner RAS Core Research Note G00153858*, December 2007

[7] Web Services Description Language (WSDL) Specification 1.1 – a W3C Note: http://www.w3.org/TR/wsdl

[8] The UDDI v3 specification: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[9] Using WSDL in a UDDI Registry, Version 2.0.2, OASIS Technical Note http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm

[10] Registering Web Services in an ebXML Registry, Version 1.0, OASIS Technical Note http://www.oasis-open.org/committees/download.php/11907/regrep-webservices-tn-10.pdf

[11] Registering Policies in UDDI Version 3 – in Web Services Policy 1.2 – Attachment specification http://www.w3.org/Submission/WS-PolicyAttachment/#RegisteringPoliciesUDDIVersion3

[12] J. Dai, R.Steele "UDDI access control", Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume2, *IEEE Computer Society*, Year of Publication: 2005, pp. 778-783.

[13] R. Peterkin, J. Solomon, D. Ionescu, "Role Based Access Control for UDDI Inquiries", Second International Conference on Communication Systems Software and Middleware (COMSWARE 2007), *IEEE,* January 2007, p.1-6

[14] "Effective SOA Deployment using a SOA Registry Repository", A Practical Guide, *Sun Microsystems,* 2005 http://www.sun.com/products/soa/registry/soa_registry_wp.pdf

[15] HP SOA Systinet software, Data sheet: https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-130-27%5E1461_4000_100__

[16] BEA AquaLogic Registry Repository, Product Data Sheet http://www.bea.com/framework.jsp?CNT=index.jsp&FP=/content/products/aqualogic/registry_repository/

[17] Enterprise Services Repository and Registry - An Overview, *SAP NetWeaver Product Management*, 2007 https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/903a4127-5403-2a10-0a96-e9452c3ab1de

[18] Oracle Service Registry: http://www.oracle.com/technology/tech/soa/uddi/index.html

[19] Software AG CentraSite UDDI environement: http://documentation.softwareag.com/webmethods/inm/pdf/uddi/overview.pdf

[20] Kenney, L. F., et al. "SOA Registries and Policy Enforcement Bolster SOA Governance and Consumption", *Gartner Research,* 2005

[21] Wsoogle.com -a global online directory of web services, available at: http://www.webservicesearchengine.com/

[22] Providing a Value Set For Use In UDDI Version 3, OASIS Technical Note http://oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-valuesetprovider-20030212.htm

[23] OWL Web Ontology Language, W3C Recommenda-tion, February 2004, available at: http://www.w3.org/TR/owl-features/

[24] North American Industry Classification System (NAICS) 1997 Release http://www.uddi.org/taxonomies/UDDI_Taxonomy_tModels.htm#NAICS

[25] Mannes, A. T.: "Web Services Registry: The Foundation for SOA Governance", *Burton Group,* 2005

[26] The SAP Service Registry public site: http://sr.esworkplace.sap.com