

Allowing the use of Multiple Ontologies for Discovery of Web Services in Federated Registry Environment

Kunal Verma¹, Amit Sheth², Swapna Oundhakar³, Kaarthik Sivashanmugam⁴, John Miller³

¹*Accenture Technology Labs, Palo Alto, CA*

²*kno.e.sis Center, Department of Computer Science and Engineering, Wright State University, Dayton, OH*

³*LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA*

⁴*Microsoft, Seattle, WA*

k.verma@accenture.com, amit.sheth@wright.edu, jam@cs.uga.edu

Abstract

The potential of a large-scale growth of private and semi-private registries is creating the need for an infrastructure, which can support discovery and publication over a group of autonomous registries. Recent versions of UDDI have made changes to accommodate interactions between distributed registries. In this chapter, we discuss an ontology based Web Service Discovery Infrastructure (METEOR-S Web Service Discovery Infrastructure), to provide access to registries that are divided based on business domains and grouped into federations. In addition, we discuss how Web service discovery is carried out within a federation. We provide a discovery algorithm, which addresses semantic heterogeneity with respect to multiple ontologies from the same domain. We also show through our empirical evaluation, that even when services are annotated with different ontologies, our algorithm is able to find good matches and eliminate false matches by considering the context and the coverage information of the annotated concepts.

Keywords: Web Services Discovery Infrastructure, P2P Web Service discovery infrastructure, Decentralized UDDI, UDDI registries, Federated UDDI, Semantic Match of Web Services, Ontology-based matching of Web Services, Multi-ontology Web Service Matching, Multi-ontology Web Service Discovery Environment, Web Service Semantic Similarity

1. Introduction

There has been a significant change in focus of the vision of UDDI [UDDI, 2002] since its inception. This was evident in the release of version 3 [UDDI, 2003], which has several new features to augment the centralized paradigm of UBR to facilitate interaction between the UBR (Universal Business Registries) and private and semi-private registries. The current search facilities offered by the latest version of UDDI do have any special features for finding Web service registries. As a result, it is assumed that Web service clients have prior knowledge of the location of the registries. In this paper, we present our implementation of a peer-to-peer network of private semi-private and public UDDI registries, which allows transparent access to other registries based on registry federations or domains. We use an ontology-based approach to classify registries and locate them based on the users' requirements. We also provide a discovery algorithm, which uses ontology-based descriptions of Web services for computing the semantic similarity between the users request and the advertised services.

Let us consider the following scenario, which illustrates the benefits of private registries having the ability to interact with other private registries. We can imagine a manufacturer that maintains a private registry to maintain details about its suppliers and other partners. Now consider a case when its suppliers are unable to meet its demands either due to adverse circumstances or large orders, and the manufacturer has to locate other suppliers. Due to trust issues, the manufacturer may not want to search the UBR and find just any supplier. He may however want to request his partners or competitors for references of trusted suppliers or he may want to contact a marketplace to find similar services. Assuming that partners maintain similar private registries, this process can be automated by forming registry federations, where the registry owners give only members of the federation access to their registries. Forming a federation of registries will allow businesses to share their data while maintaining their privacy. Considering the above example, let us say that a manufacturer is registered with two such federations and each federation has a number of suppliers. Now the manufacturer has to choose the right supplier based on his requirement. This process of discovering the right supplier is handled by a discovery algorithm. Since, Web services from different federations might be associated with different ontologies, the discovery algorithm needs to understand the context of the requirement and the context of the candidate Web services in order to find correct matches.

Several papers [Paolucci et al., 2002] [Gonzales et.al. 2001] [Colgrave et.al. 2004] [Sivashanmugam et al., 2003] have discussed semantic discovery, when the advertisement and request use terms from the same ontology. An approach based on using a single ontology is not practical as it is highly unlikely that every service provider and requestor will adhere to the same ontology. Very few approaches [Cardoso and Sheth, 2003] consider the case, when the advertisement and request belong to different ontologies from the same domain. In this paper, we present an extension of the algorithm in [Cardoso and Sheth, 2003], which uses property and syntactic similarity. We introduce two new measures context and coverage similarity,

which try to capture additional semantic information about the concepts to be matched by looking at other concepts in their vicinity. This additional semantic information is used to find implied relationships between concepts, which tell us if the concepts are semantically similar or semantically disjoint and the match scores are improved accordingly. We present the algorithms for this approach and show the results of our preliminary testing, which suggests that our approach helps in reducing false matches.

In this paper, we leverage the METEOR-S Web Service Discovery Infrastructure (MWSDI) [Verma et al., 2004] for providing transparent access to private and public Web service registries. The focus of this paper is the creation of registry federations and the semantic discovery of Web services adhering to different ontologies. We present a discussion of registry federations and characterize them in the dimensions of distribution, autonomy and heterogeneity. The main contributions of this paper are -

- Creation of the Extended Registries Ontology (XTRO). In order to provide efficient access to the registries we store semantic metadata of Web service registry community
- A unique Service Discovery algorithm based on functionality for matching services when the advertisement and the request are from different ontologies with the introduction of two new measures for matching two ontological concepts
 - Context Similarity and
 - Coverage Similarity

We also discuss the different kinds of querying possible using our infrastructure. In this paper, we present evaluation of our algorithm using our prototype implementation of a peer-to-peer network of Web service registries. Our preliminary results show that using additional measures like context and coverage similarity can lead to higher precision than other approaches which do not consider them. This work was done as part of the [METEOR-S project](#), which aims at creating a comprehensive infrastructure for the complete lifecycle management of Semantic Web processes.

We briefly describe the need for creating a distributed system of UDDI registries in section . The technical details of MWSDI are described in section . The classes and relationships of the XTRO are shown in section . In section , we discuss and analyze Registry Federations. In section , we have discussed implementation details in distributed registries with the help of tModels. Service publication and discovery have been discussed in sections 7 and 8. Section discusses the concept matching algorithm and section describes the Evaluation and Empirical Testing. Section discusses related work. Conclusions and future work are mentioned in section .

2. Need for Decentralization

One of the main reasons that the UDDI specifications decided to acknowledge replication model for data in registries instead of distribution is to enable inspection of data in UDDI along multiple perspectives. There can be potentially several facets to distribute data in UDDI, such as those related to geographical location, nature of registered services, business functionality, technical specifications and so on. The data partitioning on the other dimension could be hierarchical or non-hierarchical. With such a distributed architecture, it should be possible not only to locate appropriate registry (by processing all kinds of data distribution facets and data partitioning criteria) but also to aggregate search results from different candidate registries. Replication was chosen in UDDI because creating a scalable model for distribution of data is inherently difficult. However, in version 3 of the UDDI specifications, the need for data partitioning and affiliation among registries have been acknowledged. In MWSDI, we advocate data distribution (as opposed to data replication) and support any kind of data distribution among multiple registries¹. The data partitioning criteria is stored in the *Extended Registries Ontology* (XTRO) in MWSDI.

3. MWSDI: METEOR-S Web Service Discovery Infrastructure

In this section, we briefly describe the conceptual foundations and implementation of MWSDI, which provides the infrastructure for the work presented in this paper. Detailed descriptions of the architecture, implementation and protocols can be found in [Verma et al., 2004]. The aim of MWSDI is to provide clients with an efficient publication and discovery mechanism in a multi registry environment. We use semantic metadata stored in the XTRO to identify appropriate registries and direct the queries to them. Each time, a new registry is added to MWSDI, the XTRO is updated with the relevant details of the new registry. We show the interaction of MWSDI, registries and clients in Figure 1.

¹ Our prototype implementation is tested with non-hierarchical distribution of data. Data replication is also supported using the “replicateOf” relationship in XTRO.

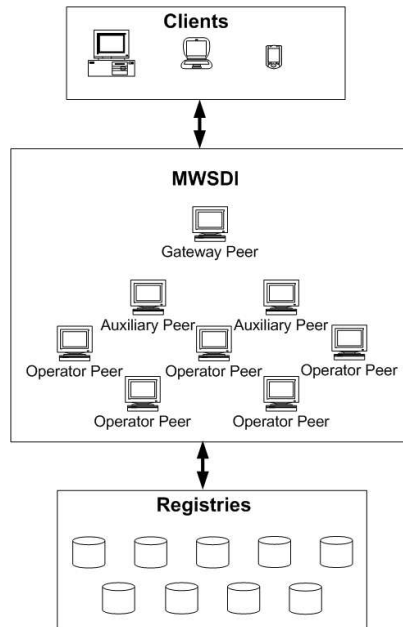


Figure 1. Interaction of MWSDI with clients and registries

We have implemented MWSDI as a peer-to-peer network. Based on the different kinds of functionality, we have implemented different peer types. They are the following.

Gateway Peer: Gateway Peer acts as an entry point for registries to join MWSDI. It is responsible for updating the XTRO when new registries join the network. Gateway Peer is the only peer that can update the XTRO or initiate new peers. It is also responsible for propagating any updates in the XTRO to all the other peers.

Operator Peer: The role of the Operator Peer is to operate a UDDI registry and to provide Operator Services for its registry. Operator Services are the value added services like semantic discovery and publication of Web services, provided by the registry operators. The Operator Peer also acts as a provider for the XTRO to all other peers who need it.

Auxiliary Peer: Auxiliary Peers act as providers of the XTRO to make it highly available, which is critical to the performance of the infrastructure. In event of failure of the Gateway Peer, one of the auxiliary peers starts to act as the Gateway peer.

Client Peer: The Client Peers are transient members of the peer-to-peer network, as they are instantiated only to allow users to utilize the capabilities of the MWSDI.

MWSDI has been implemented on a cluster of SUN workstations as peer-to-peer network using the JXTA [JXTA] framework. Any peer can be a JXTA peer if it implements one or more JXTA protocols. While there are a number of such protocols, we have used the Peer Discovery Protocol and the Pipe Binding Protocol. In addition to the protocols available in JXTA framework we have implemented two MWSDI specific protocols. They are:

Operator Peer Initiation Protocol: It defines the protocol involved in adding a new registry to the MWSDI system. It involves creating a new registry instance in XTRO. In some cases it may involve creating new federation or a domain.

Client Peer Interaction Protocol: It defines the protocol for accessing registries for publication and discovery. Details of these protocols are available in [Verma et al., 2004].

4. XTRO: Extended Registries Ontology

In our initial, naïve implementation registries could only be categorized based on business domains. We did not create constructs for creating registry federations. The *Registries Ontology* in our initial implementation is extended and called XTRO in this work that supports complex classification as well as Registry Federations. XTRO, represented in OWL, is a comprehensive ontology containing details of Domains, Registries, Ontologies and Registry Federation and network of relationships among them. All the classes and few important object properties in XTRO are shown in Figure 2.

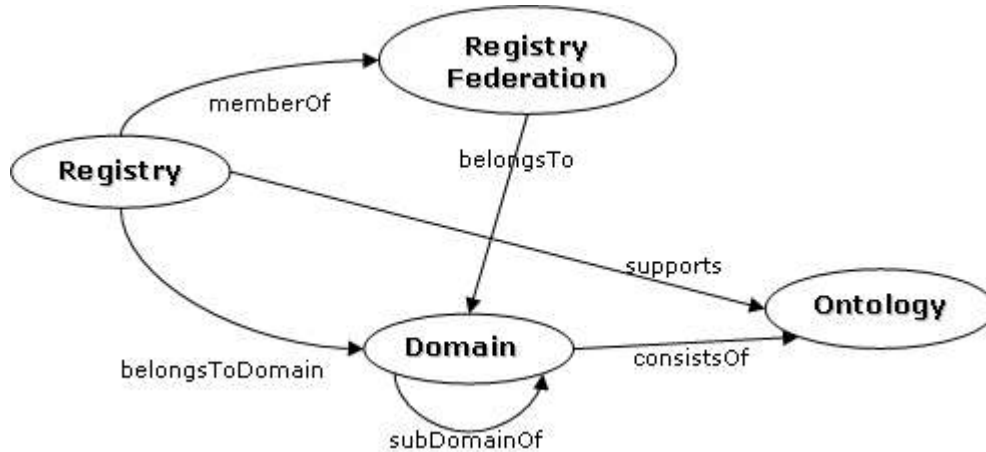


Figure 2. Classes and their Relationships in XTRO

Let us briefly examine the classes and their relationships shown in Figure 2. Each registry in the network is an instance of the class *Registry*. Different business domains are represented as instances of the class *Domain*. Ontologies available in MWSOI are instances of class *Ontology* and registry federations in MWSOI are instances of the class *RegistryFederation*. Instances of *Registry* and *RegistryFederation* can belong to one or more domains. A *Domain* instance has a number of *Ontology* instances to describe it. A *Registry* instance can support one or more domain ontologies and a *Domain* is usually a specialization of a more generic *Domain*. Each class has a number of attributes, which give us more information about their instances. We have not included them in this paper for brevity. This simple set of classes and relationships provides a simplified but exhaustive view of the Web service registry community, which can be used for efficient and accurate selection of registries. The XTRO allows us to maintain information and provide answers for the following type of queries to identify one or a group of registries:

- What is the access URL, available data model or type of the registry *R*?
- Does the registry *R* support the ontology *O*?
- Which are the registries available under the business domain *B*?
- Is the registry *X* a member of the registry federation *Y*?
- Which registries pertain to the domains that support the ontologies *O1* and *O2*?
- Get all the registry federations that belong to the domain *D*?
- Find all the registries that are categorized under the node *N* in the taxonomy (or ontology) *C*?
- What are the available relationships between the registry *R* and registry *S*?

With the use of such queries and their combination thereof to construct more complex queries, multi-perspective and intelligent querying can be carried out to locate registries for publishing or discovering Web services. XTRO can be semantically enriched to utilize more complex queries. To construct queries (discussed in section 8), it is helpful to have a UI for viewing XTRO. One such visualization of the XTRO is shown in Figure 3.

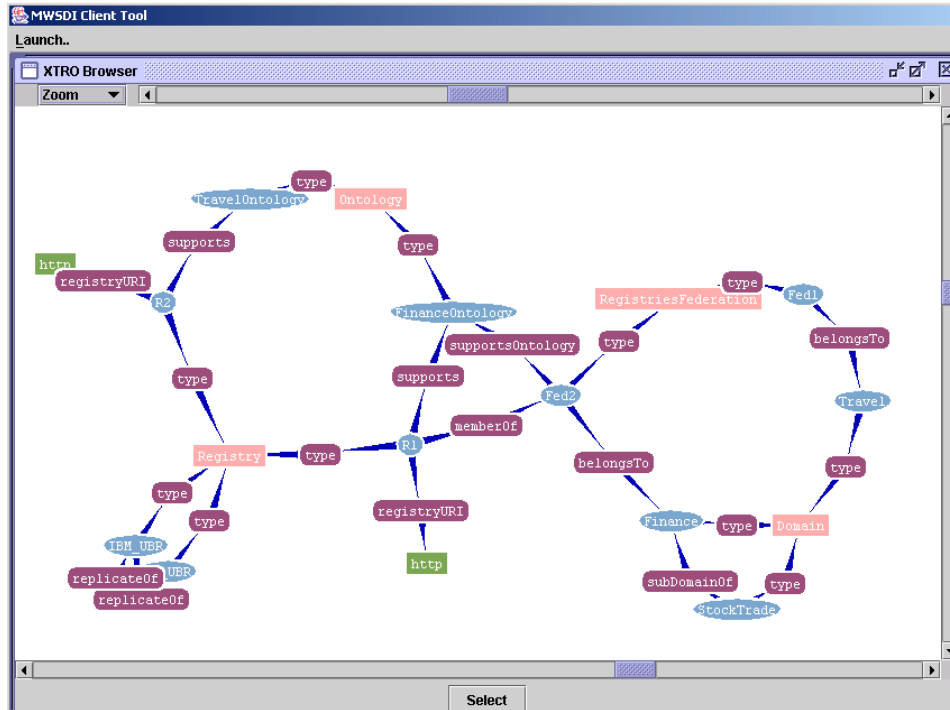


Figure 3. Visualization of XTRO

5. Registry Federation

We define a Registry Federation to be a collection of autonomous but cooperating Web service registries. The goal of a federation can be forming a registry community serving either a business domain or forming a market place of registries with similar but competing services. A federation can also aim to form an association of registries with interdependent services bound by trust and common network identity for confident collaboration. It can be a collection of private registries, public registries or even electronic marketplaces. The members of the registry federation can be heterogeneous and can have different data models and access APIs. A registry can participate in more than one federation. In the following sections, we explore the benefits of federations and characterize them with respect to the dimensions of distribution, heterogeneity and autonomy. We identify the issues associated with registry federations and discuss some of them in this paper.

- Comparison of the different federation models.
- Proposition of architecture to establish interoperation between data models of heterogeneous Web service registries like UDDI and ebXML.
- Analysis or implementation of typical features needed in a federation (like identity management, trust/contract issues and business aspects).

5.1. Need for Creating Registry Federations

Federation of registries can provide several advantages over individual registries. The benefits of a typical federated system include, but are not limited to the following:

- Managing distributed information infrastructure (sharing data, establishing inter-system data references/dependencies) without integrating the information into a single system.
- Accessing and integrating information from disparate systems without having to hop around these information repositories and manually integrate the information from each. It also helps in achieving data model transparency.
- Providing a scalable approach in accommodating new information sources to the existing applications independent of the technology, data structure, API or version of the information source.
- Providing value added services (like common authentication/authorization methods, QoS provisioning, interfaces to specify query and information integration rules).

5.2. Characteristics of Registry Federations

The dimensions of distribution, heterogeneity and autonomy used for characterizing different types of federations of database systems were discussed in [Sheth and Larson, 1990]. The same dimensions can be used to discuss characteristics of federation of registries. This work is not an attempt to discuss the intricate details of these characteristics. For the sake of clarity and completion, we have provided a brief description.

5.2.1. Data Distribution

Data may be distributed across several UDDI registries either in hierarchical or non-hierarchical models. This means that the entries in a UDDI need not be duplicated in all other registries. Current UDDI standard conforms to replication model, where all registries are equal and each registry is an exact and complete replica of each other. Hence executing a query in a registry would return the same result as that of executing the same query in another registry. Several researchers [Thaden et al., 2003, Schmidt and Parashkar, 2003] have argued that this kind of replication is not scalable. These research works also argue that distributing data among multiple registries based on vertical or horizontal partitions would provide increased availability and reliability.

5.2.2. Structural and Semantic Heterogeneity

The registries in a federation may display two types of heterogeneity, namely structural and semantic heterogeneities. Structural heterogeneity includes difference in data model. For example, an ebXML or a UDDI registry can be used as a Web service registry each with different data model. Within the same registry specification, registries could expose differences in terms of the data model and/or API versions they support. Semantic heterogeneity includes semantic differences in the elements of the data model. A typical federation allows the coexistence of the registries in a mutually beneficial and harmonious fashion in spite of the exhibited heterogeneities.

In addition, registries also contain heterogeneous content, based on the semantics of the services which are published in the registries. The recently evolving field of the Semantic Web has proposed ontologies for semantic normalization. While using ontologies can help in explicating the semantics of the service provider or service requestor, it is often the case that there exists more than one ontology in the same domain. In section , we present algorithms to handle discovery based on more than one ontology in the same domain.

5.2.3. Autonomy

The registries in the federation will typically be autonomous. As a result, a registry operator may have a separate and independent control over his registry. Though autonomy could be described for various criteria there are two basic types of autonomy based on which registries could be defined. They are:

- Design autonomy: This includes selection of data model, API selection and different types of access, different algorithms for semantic publication and discovery.
- Execution autonomy: A registry may be able to support publication and discovery of Web services independent of other registries in the federation. Hence the publication and discovery mechanisms in a registry may be unaffected by whether or not a registry is a part of a federation.

We characterize the federations supported by MWSDI in the following manner. Our implementation allows data distribution with the help of XTRO. We support design autonomy as different registries can have different algorithms for semantic publication and discovery. We also support execution autonomy as registries in MWSDI can be accessed in a standalone manner without using any MWSDI components. We support limited form of semantic heterogeneity in the federation with the help of tModel directories discussed in the next section. However, we do not support structural heterogeneity, as we only support UDDI registries.

6. TModels, UDDI Registry and Federation of UDDI Registries

TModels are reusable metadata constructs in UDDI data structure that are used to characterize and categorize businesses and their services. TModels provide the ability to describe compliance with a specification, a concept or a shared understanding. One of the main uses of a tModel is to define abstract namespace references. This means that a tModel can act as a reference that represents a relationship between keyed name-value pair and a namespace where the name-value pair has a meaning. With this characteristic, tModels are useful to annotate or attach categorization and identification information to the UDDI data. MWSDI utilizes this feature of tModels to add semantic annotations to the UDDI entries. Once tModels are registered

to represent an idea or shared meaning and the annotations are added to the UDDI data using the registered tModels, the reference (tModel keys) to the tModels can be used to discover information in UDDI that are associated with the tModels. However, to perform matching based on keyed references, the tModel keys have to be specified. As a result, a query directed to UDDI registry cannot be used to query another registry. This is because, even if the same idea or shared specification is registered in two different registries as tModels, the tModel keys could (and in most cases it will) be different. Hence, to run a query across multiple UDDI registries, some kind of query parameter translation has to be done to ensure uniform semantic interpretation of the query across all the registries. Query parameter translation in MWSDI involves translating the tModel keys used in the query. If the query sent to registry R1 references to a tModel TM1 (with the key TK1) with a value V1 (in the name-value pair), representing a concept C1 in a taxonomy, ontology or some other shared specification, then to run the same query in another registry R2 without altering the query semantics, we have to replace the key TK1 in the query with a corresponding key TK2 which is an identifier (key) for the tModel TM2 (in R2) representing the same concept C1. The value V1 should not be changed to retain the semantics of the original query. Our implementation of federation of registries is based on the idea of maintaining mapping between tModel keys of tModels with same semantics across multiple registries and to use it for federated search.

In MWSDI, each federation will have a tModel directory. The tModel directory is a simple special purpose UDDI registry that registers only tModel related data. For every unique tModel across the registries in the federation, there will be a representative tModel registered in the tModel directory. Each of the representative tModel stores mapping between registries and the corresponding tModel keys. For example, if there are three registries in the federation namely R1, R2 and R3 and if there is a tModel representing an ontology is published across these registries with three different keys TK1, TK2 and TK3 respectively, then the tModel directory will have a representative namespace tModel that stores the mapping R1-TK1, R2-TK2 and R3-TK3. If there is a query that is sent to the registry R2 referring to the tModel key TK2, then to run the same query in the registry R1, the key TK2 in the query is replaced with TK1 and to run the query in the registry R3, TK2 is replaced with TK3. Figure 4 shows the structure of a sample representative tModel. It is categorized as a namespace tModel. It also stores generic name-value pairs to store mappings between registries and the keys of the tModels (in these registries) that are linked by the representative tModel.

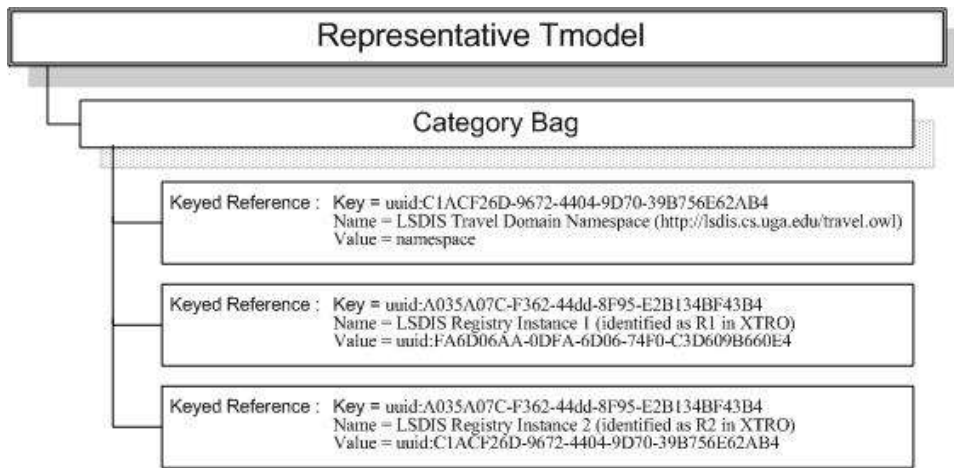


Figure 4. Representation of TModel Structure

6.1. Implementation of TModel Directory

Whenever a tModel is registered in the tModel directory as a representative of a tModel appearing in different registries with different keys, it will be categorized as a namespace tModel using the taxonomy value “namespace” in relation with one of the built-in core UDDI tModels which has the name “uddi-org:types”, tModel key “uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4”. A keyed reference is constructed with this tModel key and the name-value pair is used for this categorization. The purpose of this categorization is to have the namespace functionality and to find all the representative tModels in the tModel directory that belong to a particular namespace. The representative tModel will also be categorized using another core built-in tModel with the name “uddi-org:general_keywords”, tModel key “uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4” that is used to store generic name value pairs. The name value pair represents the registry name and the key of the tModel in the registry that is associated with the representative tModel.

6.2. Administration of the Federation

Federation of Registries can take place in two different ways. One way is to form the federation first, allowing only empty (non-populated or un-used) registries to join the federation wherein data in the registries are populated later and the other way is to form the federation that allows both used and un-used registries to join the federation. Version 3 specification of UDDI supports the former to establish affiliation between multiple registries with appropriate policies to allow controlled copying of data structures among them. These registries share a common namespace for entity keys for the purpose of entity promotion and hence if a tModel has to be published in more than one registry it can be published with the same tModel key in all the affiliated registries. This kind of affiliation makes it practically difficult for a registry to be a part of more than one independent affiliation because namespace consistency for the keys is difficult to enforce across independent affiliations. In MWSDI, we support both ways of establishing registries federation. In addition our implementation can support building federation using any version of UDDI registry as the implementation uses core UDDI data structure (tModels mainly) and nothing outside of it. Also, it empowers a registry to take part in more than one federation as the membership of a registry in a federation does not affect its other operations including key assignment for its data. The unique feature about our federation is that a registry can arbitrarily join or leave the federation without affecting other registries or without affecting the existing applications that are dependent on keys of the UDDI data.

In MWSDI, the registries join federation one by one. When the first registry (say R1) which probably will be the initiator of the federation joins the federation, all the tModels in the registry are treated unique (i.e. unique overviewURLs are assumed). For every tModel in the registry that joins the federation, a representative tModel is registered in the tModel directory of the federation. Each of the representative tModel is categorized as namespace tModel. As the overview URL of each tModel (in R1) is unique, the namespace keyed reference in the category bag of the corresponding representative tModel can be used to hold this URL in the name-value pair. Each of the representative tModel is also categorized using a keyed reference (in conjunction with “uddi-org:general_keywords” taxonomy) representing the mapping between R1 and the key of the corresponding tModel in this registry. Hence a tModel (in R1) identified by TK1 will have a representative in the tModel directory that is categorized using the mapping R1-TK1. It should be noted that the structure of the tModels are not copied from the federation member to the tModel directory. The mapping in the directory acts as a pointer to the tModel in R1. When another registry (say R2) joins the federation, overview URL of each tModel is compared against the value (in name-value pair) of all the namespace keyed references in the directory. If, for a tModel (in R2) with key TK2, there is any existing namespace keyed reference for a representative tModel (in tModel directory), then a keyed reference representing the mapping between R2 and TK2 is added to the existing categorization of the representative tModel. Consider a representative tModel that already has the mapping between the registry R1 and the tModel with key TK1. After the federation figures out that the equivalent tModel in R2 has the key TK2, the system will add another keyed reference to represent the mapping between the registry R2 and the tModel with key TK2. In this way a tModel registered across multiple registries are linked using one representative tModel that is characterized using the name-value pair Rx-TKx where Rx represents a registry that hosts the tModel with a key TKx. After the federation is formed, whenever a tModel is published in one of the member registries, a similar step is undertaken to ensure updated mapping details between registries and tModel keys. Deleting a tModel in a registry removes a corresponding keyed reference in the category bag of the representative tModel in the directory. It should be noted that the mapping between tModels are derived automatically using overviewURLs which are considered unique for tModels. If the overviewURLs are not unique within a registry, the publishers of tModels that share overviewURLs can be asked assign unique URLs to remove redundancy and enable reuse of the tModels across registries. We can also assume that if two registries refer to a same concept using different tModels in each, they share the same overviewURLs, and the tModels equivalence can be identified using the exact match of overviewURLs.

7. Service Publication in a MWSDI-2G

Service publication in MWSDI involves the following steps.

Creating Service Advertisements

[Sivashanmugam et al., 2003] described how to use extensibility elements of WSDL to provide hooks for semantically annotating various elements of the services. A more mature version of this work was published as WSDL-S [WSDL-S, 2004], leading to a W3C member submission [WSDL-S, 2005]. This is now being pursued by a W3C charter called Semantic

Annotation of Web Services (SAWSDL) [SAWSDL, 2006], which used WSDL-S as the primary input. Users can publish annotated or standard SAWSDL files using the client peers.

Registry Selection

In case of standard WSDL files, the user is required to manually specify the requirements for registry selection. For semantic publication of Web services, the users can specify either the federation names or business domains. We have added the functionality in MWSDI to automatically determine the registries on the basis of this information and ontologies used for annotation. As federation supports multiple domains and each domain in turn supports multiple Ontologies, a registry selection query or template can be expressed as RT, where

$RT = \langle f, d, o, r \rangle$ and f corresponds to the names of the registry federations, d corresponds to the business domains, o corresponds to set of ontologies, and r corresponds to a set of registry relationships. Let us assume that a particular query has the following values

$rt = \langle \{F1, F2\}, \{D2, D3\}, \{O1, O2, O3, O4\}, \{\} \rangle$

It should be noted here that when a Web service is published in multiple federations, based on which ontologies are supported by that federation, a service can be annotated with different ontologies. Hence, while discovering such Web services we face the problem of semantic heterogeneity.

The results of such a query are shown in Figure 5. The registries are shown categorized in Federations F1, F2 and F3 as well as domains D1, D2 and D3. The shaded registries are results of the query. Some registries in the appropriate domains and federations are not selected as they do not support the ontologies in the query. Even though registry relationships have not been shown in the sample query, relationships of registries like *belongsTo* and *partnerOf* can be used to select more registries. We use the inference capabilities provided by SNOBASE [Lee et al., 2003] for the handling the queries.

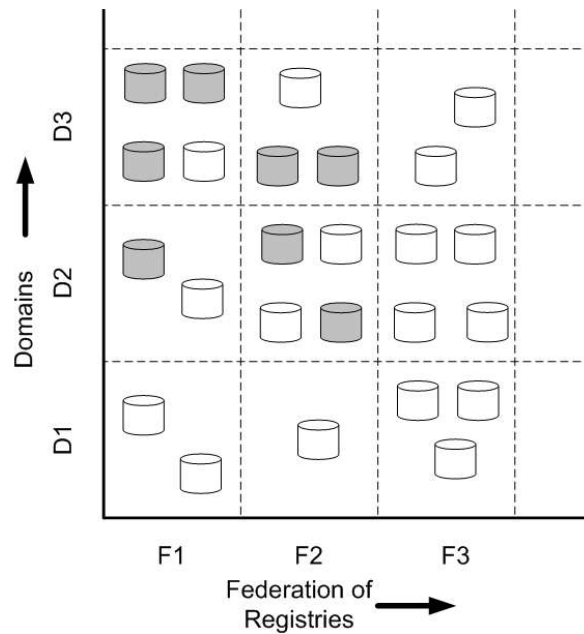


Figure 5. Selection of Registries

Service Publication

The services are the published by sending the advertisements to the operator peers of the selected registries.

8. Service discovery in MWSDI-2G

Discovery using same ontologies has been discussed in [Paolucci et al., 2002] [Gonzales et.al. 2001]. In this section we concentrate on discovery of Web services when they are annotated with different ontologies. MWSDI supports both semantic and syntactic discovery of services. Service discovery in MWSDI-2G involves following steps:

8.1. Creating Search Templates

Users can either use the search mechanism provided by UDDI or create semantic templates. We have discussed semantic templates in [Sivashanmugam et. al., 2003; Sivashanmugam et. al., 2003a].

A Search Template depicts the user's intent behind searching a Web service, which allows a user to specify the operations, inputs and outputs of the desired Web service and hence does not have a concrete implementation. Formally a Search Template can be defined as -

$$ST = \langle RT_{ST}, N_{ST}, D_{ST}, OPS_{ST} \langle NOP, D_{OP}, OS_{ST}, IS_{ST} \rangle \rangle$$

Where,

- RT_{ST} Registry Template
- N_{ST} - Name of the Web service to be found
- D_{ST} - Textual description of the Web service
- OPS_{ST} - Set of operations of the required Web service
The operations in turn are specified as

$$OPS_{ST} \langle NOP, D_{OP}, OS_{ST}, IS_{ST} \rangle$$

Where,

- N_{OP} - Name of the operation
- D_{OP} - Description of the operation
- OS_{ST} - Outputs of the operation
- IS_{ST} - Inputs of the operation

As an example we will consider a Search Template which depicts a simple Web service that takes a stock symbol as input and returns the stock quote as shown in Table 1.

	Search Template	Ontological Concept
Service Name	StockService	
Operation	getStockQuote	
Input	Ticker	StockTicker
Output	StockQuote	StockQuote
Operation	getCompanyNews	
Input	CompanyName	companyName
Output	CompanyNews	News

Table 1. Search Template

8.2. Finding Set of Candidate Web services

Registry Selection: Registry selection is the same as discussed in the previous section on publication. It based on the parameters of the registry template RT.

Query Execution: The Search Templates are then sent to the operator peers of the selected registries. We have provided an option of creating a "federated query". In case of the federated query, the TModel directory is used to translate and propagate the query to other registries in the federation.

When a registry is selected, the Search Template is matched against the services in the selected registry using the discovery algorithm detailed in section 9. These services against which the ST is matched are called Candidate Services (CS). A CS is described in the same fashion as the Search Template and is formally defined as -

$$CS = \langle N_{CS}, D_{CS}, OP_{CS} \langle N_{OP}, D_{OP}, OS_{CS}, IS_{CS} \rangle \rangle$$

Where, the terms correspond to the terms in the Search Template except for RT

Result aggregation: The operator peers return all the results to the client peer as a set of ranked candidate Web services.

9. Matching Candidate Web service to Search Template

The Search Template described above is matched to the set of candidate Web services returned by the operator peer. Match Scores are calculated for each (ST, CS) pair. These match scores are normalized over the interval of 0 to 1. Finally the pairs are ranked in the descending order of the match score and presented to the user. The user can then select his requirement from this set of ranked services. In the following few sections we will explain our matching algorithm with the help of the following candidate Web service.

	Candidate Service	Ontological Concept
Service Name	XQuotes	
Operation	getQuickQuote	
Input	quickQuoteInput	MultipleQuote
Output	quickQuoteResponse	QuickQuote
Operation	getQuote	
Input	quoteInput	MultipleQuote
Output	quoteResponse	StockQuote
Operation	getFundQuote	
Input	fundQuoteInput	FundQuoteQuery
Output	fundQuoteResponse	fundQuote

Table 2. Candidate Service

In this example both the ST and CS are annotated with different Ontologies. ST is mapped to the StocksOnt.owl² ontology and CS is mapped with xIgniteStocks.owl³ ontology.

MWSDI matches the Search Template (ST) to the candidate Web service based on the required functionality. This is achieved by matching the operations of the ST and CS rather than the inputs and outputs [Cardoso and Sheth, 2003] [Paolucci et al., 2002]. The match function matchServices (ST, CS) compares the ST and CS and returns a similarity value in the range of 0 and 1. This similarity value has two dimensions – Syntactic Similarity and Functional Similarity.

Syntactic Similarity – This computation is based only on syntactic considerations and no semantic information is taken into account. It relies on the name and the description of ST and CS.

Functional Similarity – Considering only the inputs and outputs for matching is not sufficient as they can be grouped differently to give different functionality. MWSDI uses operations as the matching unit while matching ST and CS. These operations are in turn matched based on their inputs and outputs.

The overall service similarity is the weighted average of Syntactic and Functional similarities and is calculated as shown in Equation 1.

$$MS(ST, CS) = \frac{w1 * SyntacticSim(ST, CS) + w2 * FunctionalSim(ST, CS)}{w1 + w2} \in [0..1]$$

Equation 1. Service Similarity Calculation

If the Search Template does not have a name or the user does not want to use the name and description similarity while matching services then w1 can be reduced to 0 and the MS will then consist of only the Functional similarity.

² StocksOnt.owl can be found at <http://lsdis.cs.uga.edu/Projects/METEOR-S/MWSAF/Ontologies/Stock/StocksOnt.owl>

³ xIgniteStocks.owl can be found at <http://lsdis.cs.uga.edu/Projects/METEOR-S/MWSAF/Ontologies/Stock/xIgniteStocks.owl>

9.1. Syntactic Similarity

The similarity function *SyntacticSim* measures the similarity between the names and descriptions of the Search Template and the Candidate Service. The result of this function i.e. Syntactic Similarity of ST and CS is normalized on a scale of 0 to 1. Equation 2 summarizes the calculation for syntactic similarity.

$$SyntacticSim(ST, CS) = \begin{cases} \frac{w3 * NameMatch(ST, CS) + w4 * DescrMatch(ST, CS)}{w3 + w4} \in [0 \dots 1] & Descr[ST] \neq \phi \text{ or } Descr[CS] \neq \phi \\ NameMatch(ST, CS) & Descr[ST] = \phi, Descr[CS] = \phi \end{cases}$$

Equation 2. Calculation of Syntactic Similarity

It is calculated as the weighted average of the Name similarity (*NameMatch* (*ST*, *CS*)) and the Description Similarity (*DescrMatch* (*ST*, *CS*)). The weights ω_3 and ω_4 are also real values between 0 and 1, and indicate the degree of certainty that a user has of the service name and service description supplied during a request. High weight values indicate that the user is confident that the information he supplied to search a Candidate Service is correct. If both ST and CS do not have a description then the syntactic similarity is the same as name similarity.

The **name similarity** is calculated using various name and string matching algorithms like NGram, synonym matching, abbreviation expansion, stemming, tokenization etc. The *NGram* algorithm calculates the similarity by considering the number of qgrams [Angell et. al., 1983] [Salton, 1988] [Zamora E., 1981] that the names of two concepts have in common. The *CheckSynonym* algorithm uses WordNet [Miller, 1990] to find synonyms whereas, a custom abbreviation dictionary is used by the *CheckAbbreviations* algorithm. The *TokenMatcher* uses the Porter Stemmer [Porter, 1980] algorithm, tokenization, stop-words removal, and substrings matching techniques to find the similarity. It first tokenizes the string based on punctuation and capitalization. Then it removes unnecessary words from the list of tokens, using a stop-word list. If it can not match these individual token then it stems them using porter stemmer algorithm and tries to match them using the NGram technique. If any of these algorithms return a full match, i.e., 1 on scale of 0 to 1, then a match score of 1 for name similarity is returned. If all the match algorithms give a match value of zero, then the name similarity of those concepts is 0. If on the other hand, none of the match algorithms give a match score of 1, i.e., an exact match, then the average of all non-zero match scores is taken.

The **description similarity** on the other hand uses only the n-gram algorithm.

9.2. Functional Similarity

MWSDI uses functional similarity as a measure while calculating similarity. The functional similarity (FS) is calculated as the average Match Score of the operations of ST and CS and is represented by Equation 3.

$$FS = \frac{\sum_{i=0}^n fs_i}{n} \in [0 \dots 1]$$

where, fs_i = best functional similarity of an Operation of ST
 n = number of Operations of ST

Equation 3. Functional Similarity for Service

The mappings between operations of ST and CS are calculated. A function *getFM*(*ST*, *CS*) returns a best mapping for each ST operation and can be described as in Equation 4.

$$getFM(OP_{st}, OP_{cs}) = best(getfm(OP_{st}, OPi_{cs}))$$

where, OPi_{cs} represents individual operation of CS

Equation 4. Finding the best mapping for individual operation

In the above equation $getfm$ calculates the functional similarity (fs) for an individual operation pair. Each operation of ST is mapped against each operation of CS and the best matching CS operation is selected for every ST operation. In our example, $getStockQuote$ operation of ST is mapped to all three operations of CS, and the best matching operation is considered for calculating the final service match.

Operations of ST and CS are annotated with ontological concepts, and we consider those concepts too while calculating similarity between these operations. Thus fs comprises of Syntactic similarity, conceptual similarity and IO similarity and is given by Equation 5.

$$fs = getfm(OP_{st}, OP_{cs}) = \frac{w5 * SynSim(OP_{st}, OP_{cs}) + w6 * ConceptSim(OP_{st}, OP_{cs}) + w7 * IOSim(OP_{st}, OP_{cs})}{w5 + w6 + w7}$$

Equation 5. Matching operations

Syntactic Similarity of Operations ($SynSim$) – It is the similarity of the names and descriptions of the operations to be matched and is calculated in the same manner as the Syntactic Similarity of services.

Conceptual Similarity of Operations ($ConceptSim$) – It is the similarity of the ontological concepts to which the Web service operations of the ST and CS are annotated. In the next section we will see in more details how these concepts are actually matched.

IO Similarity of Operations ($IOSim$) – This is the similarity between the inputs and outputs of operations and is calculated in terms of the similarity of inputs i.e. $InputSim$ and similarity of outputs i.e. $OutputSim$. The definition of the $IOSim$ (Equation 6) captures three distinct cases based on the inputs and outputs of ST and CS

$$IOSim(OP_{st}, OP_{cs}) = \begin{cases} \sqrt{InputSim(OP_{st}.Is_{st}, OP_{cs}.Is_{cs}) * OutputSim(OP_{st}.Os_{st}, OP_{cs}.Os_{cs})} & OP_{st}.Is_{st} \neq \phi, OP_{st}.Os_{st} \neq \phi \\ InputSim(OP_{st}.Is_{st}, OP_{cs}.Is_{cs}) & OP_{st}.Is_{st} \neq \phi, OP_{st}.Os_{st} = \phi \\ OutputSim(OP_{st}.Os_{st}, OP_{cs}.Os_{cs}) & OP_{st}.Is_{st} = \phi, OP_{st}.Os_{st} \neq \phi \end{cases}$$

Equation 6. IO similarity for Operations

- If an operation of ST has both a set of inputs and a set of outputs the similarity is calculated as the arithmetic mean of $InputSim$ and $OutputSim$
- If an operation of ST only specifies a set of outputs and no inputs, then the function only computes the similarity between the outputs of the Search Template and the outputs of the Candidate Service
- The same concept will be applied if the operation of ST includes inputs but not outputs

The function $InputSim(OP_{st}.Is_{st}, OP_{cs}.Is_{cs})$, computes the best set of mappings that can be obtained from matching the inputs of operation of ST with the inputs of operation of CS. This is done using the dynamic programming and can be represented as a recursive relation as shown in Equation 7

$$InputSim(OP_{st}.Is_{st}, OP_{cs}.Is_{cs}) = Max(InputSim(OP_{st}.Is_{st} - OP_{st}.I_{st}, OP_{cs}.Is_{cs} - OP_{cs}.I_{cs}) + iSim(OP_{st}.I_{st}, OP_{cs}.I_{cs}))$$

Equation 7. Calculating the best set of mappings for Inputs

For each potential mapping, the function $iSim(OP_{st}.I_{st}, OP_{cs}.I_{cs})$ is employed to evaluate the similarity of a single input of a Search Template operation with a single input of Candidate Service operation. As these inputs are annotated with the ontological concepts, $iSim$ actually matches two ontological concepts. In the next section we will see in more details how two ontological concepts are mapped. Output similarity is also calculated in the same way as input similarity.

9.3. Concept Similarity

There has been a significant amount of research in the area of ontology matching. Some approaches [Cardoso and Sheth, 2003], use properties as a measure of finding similarity between two ontological concepts, but these approaches are plagued with the problem of false matching. This is because properties are matched based on their names and ranges, and two entirely different concepts can still have properties with the same names. In our example, the $FundQuote$ concept from $xIgniteStocks$ ontology has property names similar to the $StockQuote$ concept of the $StockOnt$ ontology. Some approaches,

consider [Paolucci et al., 2002] [Gonzales et.al. 2001] subsumption relationships to find the similarity, but these are also limited as they match concepts from same ontology.

As explained in earlier sections, inputs, outputs and functions of the Search Template and Candidate Service are annotated with ontological concepts, and hence it is important to match these concepts while finding the similarity between them. Functional ontologies are an area of active research. Currently we consider taxonomies with no properties. We will explain concept matching for all ontological concepts in general and then mention concept matching for the case and when the concepts are from the same ontology.

In our approach we try to eliminate the limitations faced by other approaches by introducing two new measures, Coverage Similarity and Context Similarity. We also make the property matching more efficient by adding a penalty for unmatched properties. We will explain all of these in more details further.

In general there are four different parts of the similarity calculation of two ontological concepts as shown in Equation 8.

$$conceptSim(C_{st}, C_{cs}) = \frac{w8 * synSim + w9 * propSim + w10 * cvrgSim + w11 * ctxtSim}{w8 + w9 + w10 + w11}$$

Equation 8. Calculating Similarity between two Ontological Concepts

9.3.1. Syntactic Similarity (*synSim*)

This is the measure of syntactic similarity of the concepts. This includes matching the name and description of the concepts. The names are matched using techniques described earlier in section about String Matching algorithms, whereas descriptions are matched using n-gram technique.

9.3.2. Property Similarity (*propSim*)

Since a concept is defined using its properties, it is important that these properties should be matched while matching two concepts. These properties are matched as one to one mappings and the mappings are calculated in such a way that the average property match is maximized. Equation 9 shows how the property similarity is calculated.

$$PS = \frac{1}{n} \sum_{i=0}^n ps_i \in [0, \dots, 1]$$

$$propSim(C_{st}, C_{cs}) = Max(propSim(C.P_{st} - p_{st}, C.P_{cs} - p_{cs}) + propMatch(p_{st}, p_{cs}))$$

Equation 9. Finding property Similarity

propMatch calculates similarity (*ps*) between two individual properties of the concepts. While matching two properties it is important to remember that the values the properties can take are characterized by their ranges and hence range similarity is an important part of the similarity function. Since cardinality provides the information about how many range values a property can take at a time, it is also considered while calculating the *propSim*. The final part of the *propSim* is the syntactic information of the property i.e. name and description. We also penalize the property similarity for unmatched properties with a penalty of 0.05 for each unmatched property.

Equation 10 and Table 4 summarize this information and shows the calculations for *propSim*.

$$propSim = c * \sqrt[3]{rangeSim * cardinalitySim * syntacticSim} - 0.05 * |unmatched\ properties|$$

Equation 10. Property Similarity is the Geometric Mean of Range, Cardinality and Syntactic Similarity

propSim also has another contributor *c* which is a constant decided based on Equation 11.

$$c = \begin{cases} 1 & p_{st}, p_{cs} \text{ are inverse functional properties} \\ 1 & p_{st}, p_{cs} \text{ are not inverse functional properties} \\ 1 & \text{cardinality}(p_{st}) \geq 1 \text{ and } p_{cs} \text{ is inverse functional property} \\ 0.8 & \text{cardinality}(p_{cs}) \geq 1 \text{ and } p_{st} \text{ is inverse functional property} \end{cases}$$

Equation 11. Calculating the Constant factor of Property Similarity

This constant is calculated based on if the properties being mapped are inverse functional properties or not. Inverse Functional property is an OWL construct which tells us that if a property is inverse functional then that property value is unique for every instance of that concept. Consider the following example for an inverse functional property. SSN is unique for a person and Stock Symbol is unique for every Stock. No two stocks can have same stock symbol and no two persons can have same SSN. This information gives us some more insight into the real world entity that we are trying to capture with our concept and hence it is an integral part of our calculations. Thus, when the requirement is an inverse functional property and the candidate is not an inverse functional property, then although the requirement can be satisfied, the candidate property does not have the same strict restrictions on its values. Hence, it is necessary to reduce the match score by a minimal amount. With some experimentation we decided to use a value of 0.8. This value can be changed based on the requirements of the domain. For non-owl ontologies we consider the second case of Equation 11.

Syntactic Match (*syntacticMatch*) – This provides the syntactic similarity (*syntacticSim*) component of the *propSim*. It is calculated as the similarity of names and descriptions of the properties. This again uses the same algorithms mentioned in the section and is the weighted average of name and description similarity.

Range Match (*rangeMatch*) – The similarity component provided by the range match is called *rangeSim*. Range of a property can either be a primitive data type or another ontological concept. Hence based on the ranges of the properties to be matched we have three different cases.

The first case is when both the properties have primitive data types as their ranges. In such a case data type conversion is required and some times information can be lost, e.g. if the template requirement is a long and the candidate provides an integer, then it cannot effectively satisfy all the values that the template expects. Based on the information loss involved in such conversions, we present the match scores for different data types in the Table 3.

Source Datatype	Target Datatype	Match Score
Decimal	String	1
Decimal	Float	1
Decimal	Double	1
Long	Decimal	3/4
Long	Integer	1
Integer	String	1
Integer	Long	2/3
Integer	Decimal	1/3
Integer	Float	1/3
Integer	Double	1/3
String	Integer	1/2
String	Decimal	1/2

Table 3. Datatype conversion values for Range Match

The second case is that both the property ranges are Ontological concepts. In such case a **Shallow Concept Match** is done to match these range concepts. This shallow concept match involves matching names and descriptions of these concepts and matching the properties of these concepts syntactically as in Equation 12.

$$rangeSim = rangeMatch(p_{st}, p_{cs}) = \frac{w12 * synSim + w13 * propSynSim}{w12 + w13}$$

Equation 12. Range similarity when both ranges are Ontological Concepts – Shallow Concept Match

$propSynSim$ is calculated as the fraction of properties that the range concept for ST property has in common with the range concept of CS property. We call it **Shallow Property Match** and it is show using Equation 13.

$$propSynSim(p.Range_{ST}, p.Range_{CS}) = \frac{|p(p.Range_{ST}) \cap p(p.Range_{CS})|}{|p(p.Range_{ST})|}$$

Equation 13. Shallow Property Match

The third and the simplest case is when one property has a primitive data type as range and other has range of an ontological concepts. In such case as the ranges are incompatible the range match is 0. However making the range similarity 0 will also make the property similarity 0. This is not desirable, as ontologies model concepts with different levels of details. In order to avoid the property similarity to become entirely 0, we check for some more conditions and set the range similarity to a default value. For example, if both the properties are inverse-functional properties⁴ and their syntactic similarity is more than 0.5 and their cardinality is also the same then we assign a default value of 0.5 to the range similarity.

Cardinality Match (*cardinalityMatch*) – Cardinality also plays an important part when two properties are matched. For example, when the property from ST is expecting more than one value and the CS property has only one value, the match would be less as the ST requirement is not satisfied completely. . Similarly, when ST requirement has a smaller cardinality value than the CS, it may give some values which are not acceptable by the ST. Hence the match score needs to be reduced minimally. The actual values used can vary based on the domain of implementation. We have decided upon values 0.7 and 0.9 after some experimentation for the two cases described above. Equation 14 summarizes different cases for cardinalities. The value returned by this function contributes as $crdnSim$ for the $propSim$ calculations.

$$crdnSim(p_{ST}, p_{CS}) = \begin{cases} 1 & cardinality(p_{ST}) = cardinality(p_{CS}) \\ 1 & p_{ST} \text{ and } p_{CS} \text{ are functional properties} \\ 0.9 & cardinality(p_{ST}) < cardinality(p_{CS}) \\ 0.7 & cardinality(p_{ST}) > cardinality(p_{CS}) \end{cases}$$

Equation 14. Cardinality Similarity for properties

Property Penalty

When we match properties of two concepts, we can have different scenarios based on the number of properties both the concepts have and the actual number of properties matched. The candidate concept can have the equal, more or less number of properties as the requirement concept.

For the cases where we have same or more properties for candidate concept, we can find a one to one mapping for all the properties of our requirement concept. But when we have less number of properties in the candidate concept we can not satisfy all property requirements of ST concept and may have a many to one mapping. In such cases we penalize the candidate concept for not satisfying all our requirements by reducing the match score that is proportional to unmatched properties. With experiments, we have concluded that a reduction of 0.05 for every unmatched property is fair enough for getting proper match scores.

Example of PropSim

Table 4 shows an example of the property similarity for two *StockQuote* concepts from different ontologies. We can see that for *tickerSymbol* and *Symbol* pair has range similarity as 0.5. The reason is, *StockOnt* models range of *tickerSymbol* as concept *StockTicker* whereas, *xIgniteStocks* models range for *symbol* as a string. By human perception we can deduce that they both represent the Stock Symbol for the *StockQuote* concept but the range match will be zero as one is a data-type property while other is an object property. To avoid this, we consider that syntactic similarity is more than 0.5, the cardinality is also the same ($crdnSim = 1$) and that both the properties are inverse-functional properties. Hence we assign a value of 0.5 to range similarity.

fiftyTwoWeekLow and *fiftyTwoWeekHigh* are modeled as inverse-functional properties in *StockOnt* ontology but their counterparts in the *xIgniteStocks* are not. However, they have a cardinality of 1 giving us the value for $c = 0.8$. Similarly,

⁴ Inverse-functional property has a unique value for each instance of the Concept, e.g. StockSymbol is unique for every stock

bestBidSize from StockOnt has cardinality of 1 whereas, Bid-Size from xIgniteStocks has cardinality greater than 1 and hence we get a value of 0.9 instead of 1.

Property Pair		Syntactic Similarity	Range Similarity	Cardinality Similarity	c	Total Similarity
StockOnt	xIgniteStocks					
bestBidSize	Bid_Size	0.72	1.00	0.90	1.00	0.87
bestAsk	Ask	0.66	1.00	0.90	1.00	0.84
bestBid	Bid	0.53	1.00	0.90	1.00	0.78
bestAskSize	Ask_Size	0.70	1.00	0.90	1.00	0.86
lastSale	Last	0.79	1.00	0.70	1.00	0.82
Open	Open	1.00	1.00	1.00	1.00	1.00
dayLow	Low	0.54	1.00	1.00	1.00	0.81
fiftyTwoWeekLow	Low_52_Weeks	0.54	1.00	0.90	0.80	0.63
previousClose	Previous_Close	0.16	1.00	1.00	1.00	0.54
quoteTime	Time	0.56	0.50	1.00	1.00	0.65
netChange	Change	0.80	1.00	1.00	1.00	0.93
dayHigh	High	0.66	1.00	1.00	1.00	0.87
Date	Date	1.00	0.50	1.00	1.00	0.79
tradingVolume	Volume	0.56	1.00	1.00	1.00	0.82
tickerSymbol	Symbol	0.60	0.50	1.00	1.00	0.67
fiftyTwoWeekHigh	High_52_Weeks	0.56	1.00	0.90	0.80	0.64

Table 4. Example for property Similarity

9.3.3. Coverage Similarity (*cvrgSim*)

This is another dimension of the concept similarity which measures the extent of knowledge that concept covers. A concept in ontology is the abstract representation of a real world entity and it has sub-concepts which specialize this concept to even more specialized real world entities. For example, an ontological concept Automobile can have sub-concepts as TwoWheeler and FourWheeler. Hence, when our requirement is the concept automobile we can also satisfy it with either of TwoWheeler or FourWheeler. Considering this, MWSDI adds coverage similarity while measuring concept similarity. We will encounter different cases in this scenario as shown in Equation 15.

$$cvrgSim(C_{sr}, C_{cs}) = \begin{cases} 1 & MS(C_{sr}, C_{cs}) \geq 0.8 \\ 1 - 0.1 * 2^{x-1} & MS(C_{sr}, parent_x(C_{cs})) \geq 0.8 \text{ and } x \text{ is the level of parent above ST concept} \\ 1 - 0.05 * y & MS(C_{sr}, child_y(C_{cs})) \geq 0.8 \text{ and } x \text{ is the level of child below ST concept} \\ 0 & otherwise \end{cases}$$

Equation 15. Coverage Similarity

We reduce the coverage similarity exponentially for super-concepts based on the level of ancestry. For example, if we have an immediate parent then we will reduce the coverage similarity by 0.1, for a grandparent by 0.2 and so on till it reduces to zero. Similarly, we reduce the coverage similarity by a multiple of 0.05 if the candidate concept is a sub-concept, where the multiples are the levels we have to go below the required concept to reach the candidate concept. The reason behind a lesser penalty for the sub-concept is that the sub-concept satisfies the properties of our requirements completely, but we still need to distinguish it from a complete match.

We match concepts with a shallow concept match while calculating Coverage Similarity. The shallow concept match differs from the one described in section in terms of how it calculates the shallow property match. In this case we need to look for the number of properties of the candidate also and hence we change the shallow property match to the one in Equation 16.

$$\text{propSim}(\text{Concept}_{ST}, \text{Concept}_{CS}) = \frac{|p(\text{Concept}_{ST}) \cap p(\text{Concept}_{CS})|}{|p(\text{Concept}_{ST}) \cup p(\text{Concept}_{CS})|}$$

Equation 16. Shallow property match for Coverage Similarity

A detailed example of the coverage similarity with our Search Template is explained in the testing section.

9.3.4. Context Similarity (*ctxtSim*)

Most of the approaches for matching ontological concepts consider only the information that describes those concepts i.e. name, description and properties of these concepts. But many times this information is insufficient and confusing, and leads to false matches.

Considering this shortcoming, MWSDI adds another dimension to the concept similarity, called Context Similarity. In this we try to understand more about a concept by considering the semantic similarity and semantic disjointness of the concepts in the advertisement and the requirement. For example, a Bond means a Fixed Income Fund and not Stocks; Bond is also a different concept from the concept Investment Company. MWSDI tries to capture such information in two different sets.

The first set i.e. SameConceptSet is the set of the concepts which describes the same real world entities as described by the concept in question. The second set i.e. DifferentConceptSet is the set of concepts which are semantically disjoint from the concept in question. Considering the above example for the concept Bond the two sets will be as follows

SameConceptSet(Bond) = {FixedIncomeFund, Bond}

DifferentConceptSet(Bond) = {Stocks, InvestmentCompany}

Sets are created as follows

- SameConceptSet
 - Concepts are added to SameConceptSet, if an ontology specifies them as same or equivalent concepts e.g. in OWL we have sameClassAs or equivalentClass relationship
 - The SameConceptSet also contains the member concepts of the main concept i.e. concepts which are used to define main concept e.g. OWL has collection classes, which are described in terms of other classes
 - The concept itself is also added in the SameConceptSet
- DifferentConceptSet
 - Concepts which are explicitly modeled as disjoint concepts of the main concept e.g. in OWL concepts which are related with disjointWith or complementOf relationships
 - Concepts appearing as ranges of properties of main concept except if the range is concept itself e.g. in our stocks ontology, Company concept has a property with Stocks concept as range and hence Company and Stocks do not represent same the concept and Stocks will go in DifferentConceptSet of Company
 - Concepts which has properties with main concept as range e.g. Stocks appears as range of investsIn property of MutualFund concept and hence MutualFund goes in the DifferentConceptSet of Stocks
 - Siblings of main concept are also added to the DifferentConceptSet as they depict an entirely different specialization than the main concept e.g. EquityFund and FixedIncomeFund are siblings and can not replace each other in a request

Once these sets are established, the context match can be calculated. The context match is given by the following equation.

$$\text{contextSim} = \begin{cases} 1.0 & \exists SC_{ST}(\text{shallowConceptMatch}(SC_{ST}, SC_{CS}) > 0.8) \\ & \text{where, } SC_{ST} \in \text{sameConceptSet}(C_{ST}), SC_{CS} \in \text{sameConceptSet}(C_{CS}) \\ -1.0 & \exists SC_{ST}(\text{shallowConceptMatch}(SC_{ST}, DC_{CS}) = 1) \\ & \text{where, } SC_{ST} \in \text{sameConceptSet}(C_{ST}), DC_{CS} \in \text{differentConceptSet}(C_{CS}) \\ -1.0 & \exists DC_{ST}(\text{shallowConceptMatch}(DC_{ST}, SC_{CS}) = 1) \\ & \text{where, } DC_{ST} \in \text{differentConceptSet}(C_{ST}), SC_{CS} \in \text{sameConceptSet}(C_{CS}) \\ -1.0 & \text{avgConceptMatch}(SC_{ST}, SC_{CS}) < 0.5 \text{ and} \\ & \text{avgConceptMatch}(SC_{ST}, DC_{CS}) > 0.8 \text{ or } \text{avgConceptMatch}(DC_{ST}, SC_{CS}) > 0.8 \\ & \sqrt{ms_1 * ms_2} \text{ where, } ms_1 = \text{avgConceptMatch}(SC_{ST}, SC_{CS}) \\ & \text{and } ms_2 = \text{avgConceptMatch}(DC_{ST}, DC_{CS}) \end{cases}$$

Equation 17. Concept Match

The equation represents the following steps –

- The SameConceptSet and DifferentConceptSet of the ST concept are matched against SameConceptSet and DifferentConceptSet of the CS concept. This match is a shallow concept match, as we have seen in range match section in Equation 12. Each of these four combinations gives a set of mappings.
- Once we find these mappings we check for following conditions
 - If any concept mapping from (SameConceptSet_{ST}, SameConceptSet_{CS}) has a match score greater than 0.8, it signifies that the candidate concept and service template concept resemble each other and hence a context match score of 1.0 is returned.
 - If any concept mapping from (SameConceptSet_{ST}, DifferentConceptSet_{CS}) or (SameConceptSet_{CS}, DifferentConceptSet_{ST}) has match score equal to 1.0, it clearly signifies that the candidate concept and the Search Template concept model entirely different real world entities and hence the Context Match is -1.
 - If the average match score of (SameConceptSet_{ST}, SameConceptSet_{CS}) mappings is less than 0.5 and average concept match score for either (SameConceptSet_{ST}, DifferentConceptSet_{CS}) or (SameConceptSet_{CS}, DifferentConceptSet_{ST}) mappings is greater than 0.8, it can be assumed that the candidate concept and service template concept do not model the same real world entity and hence the Context Match is -1.
 - If none of the above cases occur then average match score of (SameConceptSet_{ST}, SameConceptSet_{CS}) mappings is calculated. Similarly average match score of (DifferentConceptSet_{ST}, DifferentConceptSet_{CS}) mappings is also calculated and then overall context similarity is given by the geometric mean of these two match scores.

9.3.5. Matching Concepts from Same Ontology

While matching concepts from the same ontology, we first calculate the coverage similarity. If the coverage similarity is not zero then that means that the concepts are linked to each other by a subsumption relationship. We then calculate the Shallow Property Match as shown in section . We also default the context match to 1.0 and accordingly the overall concept match is calculated. In the case where, the concepts are from the same ontology but are not connected to each other through any ancestors, the match is calculated in the same manner as matching concepts from different ontology.

The testing section describes a graph and an explanation to depict all the above dimensions of the Concept similarity with respect to the Search Template defined in section .

10. Evaluation and Empirical Testing

We have tested our algorithm on a set of 24 Web services from Stocks domain to see how functional similarity and concept similarity, help to eliminate false matches. This set contains real world Web services that we gathered from categorized Web service indexes of salcentral.com and bindingpoint.com. We also added a few more services derived from these services. We then annotated these Web services using our MWSAF tool [Patil et al., 2004] with two different ontologies. The first

ontology named *StocksOnt.owl*⁵ is based on nasdaq's web site and the online stock tutorial from investopedia whereas the second ontology *xIgniteStocks.owl*⁵ was obtained from concepts used by xIgnite, a Web service company. We consider our selection to be adequate and sufficient for preliminary testing purposes as this set represents 14 different operations modeling scenarios with respect to a Search Template (detailed in Graph 2 & 3) and contains 8 distinct cases that can arise while matching two ontological concepts (detailed in Graph 1). We believe that most of the Web services will adhere to one of these 14 cases. For comprehensive testing, we would require rich, real world ontologies and a vast set of well written existing Web services. Due to lack of either, we had to create ontologies which are rich enough to show the benefits of our approach.

The first graph, shows how the context and the coverage similarity used by the concept matching algorithm help us to get more accurate match scores between concepts. These improved match scores help us to eliminate false matches between service advertisements and templates as shown in the second graph.

We will see how the Concept matching algorithm performs better with the help of 8 distinct cases. We will consider the *StockQuote* concept from our example ST. This *StockQuote* concept is from the *StockOnt* ontology. While matching this concept with other concepts, eight distinct cases need to be considered. Four of these cases occur when the candidate concept is from the same ontology i.e. it is a direct match, a super-concept match, a sub-concept match or a completely different concept. The remaining four cases occur when the candidate concept is from a different ontology; again it could be a direct match, a super-concept match, a sub-concept match or a completely different concept. Graph 1 shows the similarity values for each of these eight cases. In this graph we show seven bars for each concept pair. The very first bar among these seven bars depicts the similarity value using only property similarity and syntactic similarity [Cardoso and Sheth, 2003]. The second bar shows the property similarity without considering the property penalty described in section 9.3.2. The third bar shows the syntactic similarity. The fourth bar shows the context similarity. The fifth bar shows the coverage similarity. The sixth bar shows the property penalty by considering the property penalty. We can compare the second and the sixth bar to see the improvement using the property penalty. The last bar shows the similarity values calculated by the MWSDI concept matching algorithm. We will see each of these conditions in detail and compare the MWSDI match score with the match scores considering only property and syntactic similarity (the first and the last bar) –

Candidate Concepts from Same Ontology

Case 1. The simplest and best case is when the candidate concept is the same as our ST concept. In this case all the four dimensions give a similarity of 1 and the overall match score is also 1. Here, considering only property and syntactic similarity will also give a match score of 1.

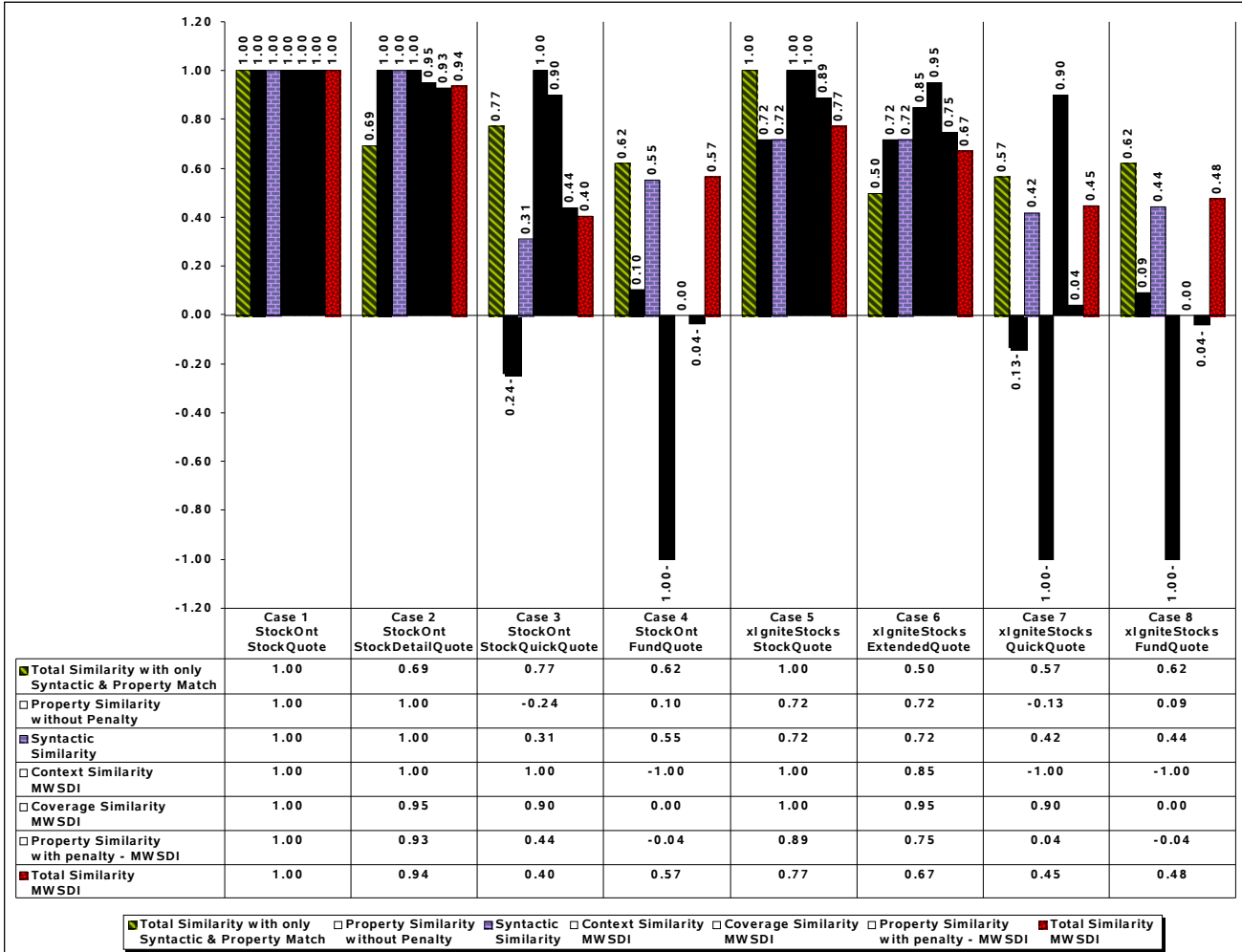
Case 2. The second case is when the candidate concept is a sub-concept of our ST concept. In such a case, even if our requirement is satisfied completely, it is still not the exact concept that we want and hence, for *StockDetailQuote*, the coverage similarity is reduced by 0.05 as it is the *immediate* child of *StockQuote*. Since we have a non-zero coverage match and the concepts are from the same ontology we default the context similarity to 1. This boosts the overall similarity even more to give a better match score compared to the similarity calculated by using only property and syntactic similarity.

Case 3. When the candidate concept is a super-concept of our requirement, it does not satisfy all the properties of our requirement. Here we apply a penalty of 0.05 for each unmatched property. This reduces the property similarity for *StockQuickQuote* to a negative value, which signifies that most of the properties of our requirement are not satisfied. We also deduct 0.1 from the coverage match, as the candidate concept is the *immediate* parent of our requirement. Again since we have non-zero coverage similarity and as the concepts are from the same ontology, we default the context similarity to 1. With the help of the negative property similarity we are able to reduce the overall concept similarity but we give advantage to this match as concepts are from same ontology with context similarity of 1 and avoid too much reduction in the final match score.

Case 4. If the candidate concept is from the same ontology but not related to the requirement concept, then we treat it in same way as concepts from two different ontologies. Here *FundQuote* gives us a context match of -1 as it is modeled as a sibling of *StockQuote*. The Coverage similarity is 0 as *StockQuote* and *FundQuote* do not have a subsumption relationship between them and the property similarity is also reduced by using the penalty for unmatched properties. This will give us an overall match score below zero, making it easier to discard this match. In contrast, without using the

⁵ A partial view of these ontologies is attached in appendix A

coverage and context similarities and the property penalty we will get a decent match score as property similarity without penalty is above 0.5.



Graph 1. Concept Matching in MWSDI

Candidate Concepts from different Ontology

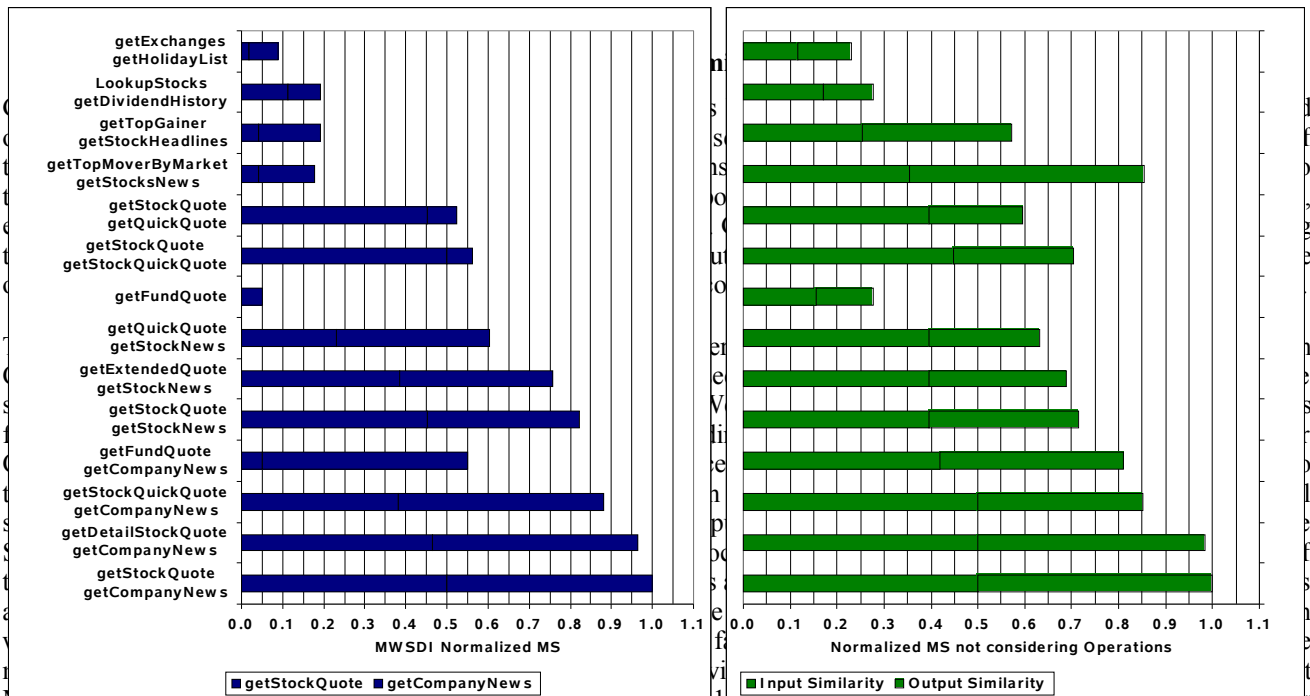
Case 5. A different ontology can model the same *StockQuote* concept with a bit of variety. The same is the case for the *StockQuote* concept from *xIgniteStocks* ontology. We can see that all the properties are matched as the property similarity values with and without penalty are the same. Since, both have the same name, the syntactic similarity is 1. As all the properties for both the concepts match each other with a value of above 0.7, we have Context similarity and coverage similarity also as 1 and they help to boost the overall match score.

Case 6. The next case is when the candidate concept from a different ontology matches better with a sub-concept of our requirement. In our example, *ExtendedQuote* from *xIgniteStocks* matches better with *StockDetailQuote* of *StockOnt*. As *StockDetailQuote* is the immediate child of the requirement concept, we have a coverage match of 0.1. Here, all the properties of the requirement are satisfied and we have a good context similarity of 0.85 which boosts the overall match score.

Case 7. The next is when we try to match a concept from different ontology which matches better with a super concept of our requirement concept. Here in case of *QuickQuote* as it matches to *StockQuickQuote* which is an immediate super concept of our requirement, we have a coverage match of 0.9. *QuickQuote* is modeled as the sibling of *StockQuote* in

its ontology and our requirement concept matches better with that *StockQuote* concept from *xIgniteStocks*, we have a similarity of -1. In addition to this we have a penalty for unmatched properties. This results in a very low overall match score for *StockQuote-QuickQuote* pair. It can be noted here that by considering only syntactic and property similarity we will still get a good match score for this pair.

Case 8. The last case is when we try to match an unrelated concept from a different ontology. In this case, we have *FundQuote* concept which has a decent property similarity without penalty. If we consider only syntactic and properties similarity this concept gives a match score near to 0.5 with our requirement. With human perception however, we know that this match should be discarded as it is surely not our requirement. Our algorithm excels here. The property penalty will help here to reduce the property similarity to a low value. In addition since this concept matches better (MS > 0.8) with the *FundQuote* concept from our ST ontology, which is a sibling of our requirement, we are able to assign it a score of -1 for context similarity. The coverage match is 0 as we do not have an indirect subsumption relationship. Overall, we get a negative score for this pair and we are able to discard this match with our algorithm.



MWSDI avoids this false match too. Candidate Services 13 and 14 have both the operations entirely different from our requirement. We can see that MWSDI is able to lower the match scores even further because it uses functional concepts to identify what the operations are.

With the help of context and coverage similarity we are able to boost the match scores for good matches from different ontologies by almost 15-20%. We are also able to temper down bad matches where some or all operations are different from our requirement by 10-20%.

Using a threshold of 0.7, MWSDI algorithm in graph 2 returned 5 matches and the algorithm considering just inputs and outputs returned 7 results. Of the seven, two cases (cases 4 and 11) lead to false matches, which are eliminated in MWSDI. The only input and output based algorithm also leaves out a good match i.e. case 6 and returns a bad match i.e. case 9. By bad match we mean a partial match where all the operations of request are not satisfied. MWSDI is able to eliminate both this problems. One of the false matches arises because the property names of concepts they are annotated with are same e.g. StockQuote and FundQuote outputs for getStockQuote and getFundQuote operations. The other false match scenario is a result of neglecting the input / output combination in a specific operation. When a threshold of 0.55 was used the algorithm considering just inputs and outputs returned a set of 11 services out of which three cases i.e. 4, 11 and 12 were false matches and two cases, case 9 and case 10 are bad matches. With the same threshold value MWSDI returns 8 matches out of which only one case i.e. case 4 is a false match. It also reduces the number of bad matches to 1.

Thus at different threshold values both the algorithms return different number of matches, but in all the cases the MWSDI algorithm worked better in terms of eliminating false matches.

11. Related Work

The approaches to Web services discovery can be classified as centralized and decentralized. UDDI falls under fully centralized approach that supports replication. Having realized that replicating the UDDI data is not a scalable approach several decentralized approaches have been proposed. [Thaden et al., 2003] argues that the trend in integrating UDDI features into general purpose enterprise registries results in rapid increase of private registries and limits the use of public registries. It also states that, from discovery perspective, it is impractical to replicate these private registries in the public counterparts. They propose to deal with this problem by creating a virtual global registry by connecting all private registries in a P2P network. They also support semantics based service discovery using OWL-S service descriptions and matchmaking. This work is similar to our work but they do not consider registry federations. [Schlosser et al., 2002] details the use of a global ontology to determine the organization of peers in their P2P topology thus enabling concept based search. It is about ontology based clustering of peers based on their capabilities. P2P based Web service discovery is also discussed in [Schmidt and Parashkar, 2003; Paolucci et al., 2003; Maedche and Staab, 2003]. None of the aforementioned works considers relationships between registries and registry federations and routing queries on the basis of them. Our work is different since we use XTRO to capture relationships among registries as well categorize the registries on the basis of business domains.

[Zhou et al., 2003] presents a federated architecture that supports QoS based discovery of services. It has a notion of UX ("UDDI Extension") server that performs federated discovery on behalf of a user request and aggregates results before sending them back to the requestor. The paper discusses different ways of maintaining links between the servers and how query is propagated. It also envisions linking UX servers across different domains. It points out that improvements could be made using semantic descriptions and matchmaking. In our approach, we have used JXTA and abstracted the network level issues. The emphasis is rather on utilizing UDDI data structure to store semantic description of a service for better service matchmaking, to establish a federation for carrying out discovery process in multiple registries and in exploiting an ontology for improving the registry selection mechanism for Web service publication and discovery.

Looking from a business perspective our work shares that the perspective of [Christoffel, 2001]. It emphasizes the idea of increasing the potential of individual traders by cooperation with other traders. It also argues that cooperation between traders can be useful when a trader cannot provide sufficient service to a customer. It also discusses the use of centralized and non-centralized federations. Our work applies this idea to Web service registries and argues the use of cooperation of registries where in registries can work together as a federation to get useful results during Web service discovery. Due to existence of tModel directory in federations, our federation can be termed as centralized federation. Federated Web service discovery is discussed in [Chandana et al., 2003]. However, it is limited to keyword based search on a predefined set of UDDI registries. Business Explorer for Web Services (BE4WS) [BE4WS, 2001] is an XML-based UDDI exploring engine to perform complex searches using a single query request on a single or multiple UDDI Registries. It does not however support tModel translation while performing a query across multiple registries as discussed in our work.

While selection of the registry is the first step to efficient discovery it is equally important to correctly match the users request to the advertisements. A lot of papers discuss the ontology based approach for service matching and discovery. [Klein and Bernstein, 2001] use a process taxonomy based approach for discovery. The matching algorithm uses the semantic relationships encoded in the process ontology to match the service process model against queries. [Gonzales et.al. 2001] [Paolucci et al., 2002] use subsumption relationships to find matches between two concepts. The former uses a DL reasoner whereas the latter determines the degree of similarity using the vertical distance between concepts. Both these approaches are based on DAML+OIL descriptions and are limited to services mapped with a single ontology. [Benatallah et al, 2005] presented an approach for Web service discovery based on rewriting of service advertisements using description logics. [Akkiraju et al., 2006] presented an approach for service discovery that combined schema based techniques along with ontology based matching. [Trastour et.al., 2001] convert services to RDF graphs and two nodes match if all their sub-nodes match. This approach is also limited for a single ontology based service description. [Sycara et.al., 1999] describes the matchmaking process using an agent capability description language, called LARKS (Language for Advertisement and Request for Knowledge Sharing). The matching engine of the matchmaker agent contains five different filters: Context matching, Profile comparison, Similarity matching, Signature matching, and Constraint matching. Although this approach uses a context filter, it differs from our approach in terms of what it means by the context. In LARKS, the context is defined by words that are used to define the overall domain of the service, whereas MWSDI uses context information for every concept. In addition LARKS assumes that even if there are different domain ontologies, they are derived from a common basic vocabulary. Another approach by [Cardoso and Sheth, 2003] specifies an algorithm for matching services annotated

with different ontologies. However, this approach also uses DAML+OIL and is plagued with problem of false matching. [Paolucci et al., 2002a] talk about using UDDI with OWL-S. They propose to import semantics from OWL-S service description into UDDI which can be used while searching for services. A parallel effort by the WSMO group is the implementation of the WSMF [Fensel and Bussler, 2002] framework. It concentrates more on how Web services should be described in order to implement a better service discovery approach.

12. Conclusion and Future work

In this paper we discussed how the metadata of a registry network stored in an ontology (XTRO) can be used in registry selection to perform Web service publication or discovery. In addition we described a discovery algorithm to find the appropriate services once a registry is selected. Empirical testing and preliminary evaluation of our discovery algorithm showed that considering the context and coverage of the annotated ontological concept helped us to better understand the user's service requirement and thus led to better quality matches. Our testing was hindered due to lack of annotated Web services and ontologies. We are working on building a testbed of annotated services, which will help us to evaluate our approach further and expose classes of problems that are not apparent in our preliminary evaluation. It will also help us to refine our algorithm. It is understood that the weights determine the result of the match and the ranking of services available in the result. The user of MWSDI is expected to assign the weights based on the nature of services the user is interested in, domain the services belong to and the number of services available for comparison. However, the user does not have to assign the weights every time the search has to be made. Once the weights are assigned for a particular domain, the user can use it as a default value set for that domain and the weights can be reset if needed or overridden in special cases.

We strongly believe that the XTRO based P2P based architecture and the MWSDI discovery algorithm will help applications, businesses, partners and suppliers to analyze the registries' characteristics and information to carry out the process of service discovery more effectively.

In the future we intend to extend this work to incorporate the following:

- Establishing contracts, trust and security policies among members of the federation
- Mediator to enhance interaction between federations
- Dynamic/Runtime association between registries
- Extend the service discovery algorithm to support fault matching.

Acknowledgements

This chapter is a slightly updated version of S. Oundhakar, K. Verma, K. Sivashanugam, A. Sheth and J. Miller, Discovery of Web Services in a Multi-Ontology and Federated Registry Environment, International Journal of Web Services Research, 2 (3), July-September, 2005, pp.1-32.

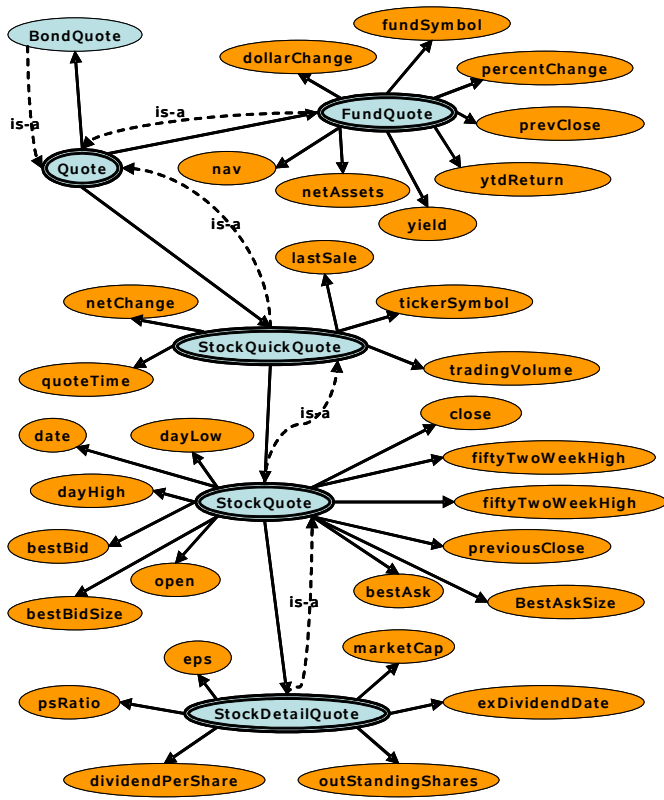
References

- [Akkiraju et al., 2006] R. Akkiraju, B. Srivastava, A. Ivan, R. Goodwin, T. Syeda-Mahmood: SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition, AAAI 2006.
- [Angell et. al., 1983] R. Angell, G. Freund, et al., "Automatic spelling correction using a trigram similarity measure." *Information Processing and Management* 19(4): 255-161, 1983
- [BE4WS, 2001] Business Explorer for Web Services, <http://www.alphaworks.ibm.com/tech/be4ws>
- [Benatallah et al, 2005] B. Benatallah, M. Hacid, A. Léger, C. Rey, F. Toumani, On automating Web services discovery. VLDB J. 14(1), 84-96 (2005).
- [Cardoso and Sheth, 2003] J. Cardoso and A. Sheth (2003), "[Semantic e-Workflow Composition](#)", Journal of Intelligent Information Systems (JIIS), 2003.
- [Chandana et al., 2003] C. Subasinghe, D. Cooray, N. Sadeep, L. Kumara, Wonder Room for Easy Web Services Invocation, Department of Computer Science and Engineering University of Moratuwa, December 2003.

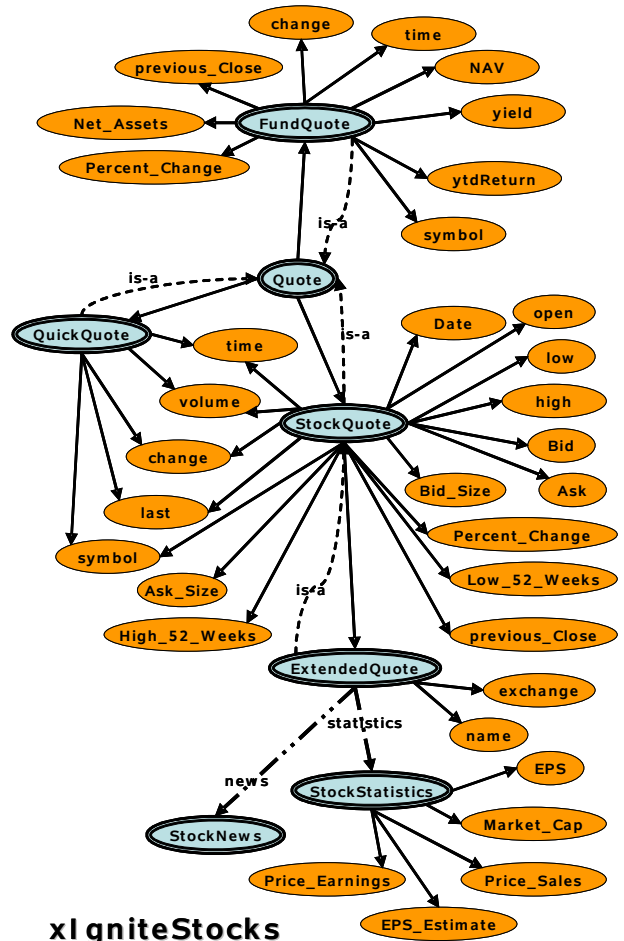
- [Christoffel, 2001] M. Christoffel, Cooperations and Federations of Traders in an Information Market, in Proceedings of the AISB Symposium Intelligent Agents in Electronic Commerce, York, 2001.
- [Colgrave et.al., 2004] J. Colgrave, R. Akkiraju, R. Goodwin, External Matching in UDDI, ICWS 2004
- [Fensel and Bussler, 2002] D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce: Research and Applications, 1 (2002) 113-137
- [Gonzales et.al. 2001] J. Gonzales-Castillo, D. Trastour, and C. Bartolini, Description logics for matchmaking of services, In Proc. of Workshop on Application of Description Logics, 2001.
- [JXTA] <http://www.jxta.org>
- [Klein and Bernstein, 2001] M. Klein and A. Bernstein, Searching for Services on the Semantic Web using Process Ontologies, in The First Semantic Web Working Symposium (SWWS-1). 2001. Stanford, CA USA.
- [Lee et al., 2003] J. Lee, R. Goodwin, R. Akkiraju, P. Doshi, Y. Ye 2003 SNoBASE: A Semantic Network-based Ontology Management. <http://alphaWorks.ibm.com/tech/snobase>.
- [Maedche and Staab, 2003] A. Maedche, S. Staab, Services on the Move - Towards P2P-Enabled Semantic Web Services, in Proceedings of the 10th International Conference on Information Technology and Travel & Tourism, ENTER 2003, Helsinki, Finland, 29th-31st January 2003.
- [Miller, 1990] G. Miller. "Special Issue, WordNet: An on-line lexical database" International Journal of Lexicography, Vol. 3, Num. 4, 1990
- [Paolucci et al., 2002] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, Semantic matching of Web Services capabilities, in Proc. of the 1st International Semantic Web Conference (ISWC), 2002.
- [Paolucci et al., 2002a] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, Importing the Semantic Web in UDDI, In Web Services, E-Business and Semantic Web Workshop, 2002.
- [Paolucci et al., 2003] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan, "Using DAML-S for P2P Discovery," in Proceedings of the First International Conference on Web Services (ICWS'03), Las Vegas, Nevada, USA, June 2003, pp 203-207.
- [Patil et.al., 2004] A. Patil, S. Oundhakar, A. Sheth, and K. Verma, [METEOR-S Web Service Annotation Framework](#), in proceedings of the 13th international conference on World Wide Web, 2004
- [Porter, 1980] M. Porter, "An algorithm for Suffix Stripping", *Program – Automated Library and Information Systems*, 14 (3):130-137, 1980
- [SAWSDL, 2006] Semantic Annotations for Web Services Description Language Working Group, <http://www.w3.org/2002/ws/sawSDL/>[Salton, 1988] G. Salton, "Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer", Massachusetts, Addison-Wesley, 1988
- [Schlosser et al., 2002] M. Schlosser, M. Sintek, S. Decker and W. Nejdl, A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services, Second International Conference on Peer-to-Peer Computing (P2P'02) September 05-07, Linköping, Sweden 2002.
- [Schmidt and Parashkar, 2003] A Peer-to-Peer Approach to Web Service Discovery, C. Schmidt and M. Parashkar, World Wide Web, Internet and Web Information Systems, Kluwer Academic Publishers, August 2003.
- [Sivashanmugam et al., 2003] 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) pp. 395-401.

- [Sivashanmugam et al., 2005] K. Sivashanmugam, J. Miller, A. Sheth, K. Verma, [METEOR-S Web Service Composition Framework](#), International Journal of E-commerce 9(2), pp. 71-106
- [Sheth and Larson, 1990] A. Sheth, and J. Larson: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, [ACM Comput. Surv. 22\(3\)](#): 183-236 (1990)
- [Sycara et al., 1999] K. Sycara, J. Lu, M. Klusch, S. Widom, Dynamic Service Matchmaking among Agents in Open Information Environments. Journal ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, Ouksel, A., Sheth, A. (ed.) (1999)
- [Thaden et al, 2003] U. Thaden, W. Siberski, and W. Nejd, A Semantic Web based Peer-to-Peer Service Registry Network, Technical Report, Learning Lab Lower Saxony.
- [Trastour et al., 2001] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo, (2001), A Semantic Web Approach to Service Description for Matchmaking of Services, Proc. 1st Semantic Web Working Symposium, CA.
- [UDDI, 2002] Evolution of UDDI, White Paper available at http://www.uddi.org/pubs/the_evolution_of_uddi_2002_0719.pdf
- [UDDIDS, 2002] UDDI Version 2.03 Data Structure Reference, UDDI Committee Specification, 19 July 2002
- <http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>
- [UDDI, 2003] UDDI Version 3.0.1, available at <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>
- [Verma et al., 2004] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller, [METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services](#), Journal of Information Technology and Management (to appear, 2005)
- [WSDL-S, 2004] WSDL-S, Adding Semantics to WSDL, available at <http://lstdis.cs.uga.edu/projects/WSDL-S/wsdls.pdf>
- [WSDL-S, 2005] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, Web Service Semantics - WSDL-S, W3C Member Submission, <http://www.w3.org/Submission/WSDL-S/>. [Zamora E., 1981] E. Zamora, J. Pollock, et al. "The Use of Trigram Analysis for Spelling Error Detection", *Information Processing and Management* 17(6): 305-316, 1981
- [Zhou et al., 2003] Zhou Chen, Chia Liang-Tien, Bilhanan Silverajan and Lee Bu-Sung, UX- An Architecture Providing QoS-Aware and Federated Support for UDDI, In proceeding of the first International Conference on Web Services (ICWS03), 2003

Appendix A



StocksOnt



xIgniteStocks

Figure 6. Part of StocksOnt.owl and xIgniteStocks.owl ontologies