

# Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?

John Garofalakis<sup>1,2</sup>, Yannis Panagis<sup>1,2</sup>, Evangelos Sakkopoulos<sup>1,2,\*</sup> and Athanasios Tsakalidis<sup>1,2</sup>

<sup>1</sup> Department of Computer Engineering & Informatics  
School of Engineering, University of Patras  
Rio Campus, 26500 Patras, Greece  
{garofala, panagis, sakkopul, tsak}@ceid.upatras.gr  
<sup>2</sup> Research Academic Computer Technology Institute  
Internet and Multimedia Technologies Research Unit  
61 Riga Feraiou Str. 26110 Patras, Greece

**Abstract.** The introduction of software development via Web Services has been the most significant web engineering paradigm, in the last years. The widely acknowledged importance of the Web Services' concept lies in the fact that they provide a platform independent answer to the software component development question. Equally important are the mechanisms that allow for Web Service discovery, especially as the latter has turned to an arduous task. This paper reviews the latest methods, architectures, models and concerns that have arisen in the Web Service Discovery area.

## 1 Introduction

Web Services (abbr. WS) have emerged as a dominating set of recommendations and standards (W3C, OASIS). They have marked current web engineering methodologies and are ubiquitously supported by IT vendors and users. In short they are interoperable software components that can be used in application integration and component based application development. As the demand for WS consumption is rising, a series of questions arise concerning the methods and procedures to discover the most suitable to use. In fact there is much hiding behind the discovery of a Web Service. This work aims to examine and analyze the different proposals in the area.

Initially a definition outline should be attempted of what discovery mechanisms stand for. A first description of discovery mechanisms for service providers appears in [35] as the *match-making process*. It is the process of finding an appropriate service provider for a service requester through a middle agent [6]. It includes the following general steps: a) Service providers advertise their capabilities to middle agents, b) middle agents store this information, c) a service requester asks a middle agent whether it knows of service providers best matching requested capabilities and d) the middle agent, in order to reply,

---

\* Corresponding author. Contact at sakkopul@ceid.upatras.gr Authors appear in alphabetical order.

Latest Web Service Discovery Survey by: Garofalakis, J., Panagis, Y., Sakkopoulos, E., and Tsakalidis, A.: "Contemporary Web Service Discovery Mechanisms", *Journal of Web Engineering*, Rinton Press, 5(3):265-290, 2006

tries to match the request against the stored advertisements and returns a subset of stored service providers' advertisements.

A more up-to-date approach [4] defines the WS Discovery mechanism in a broader sense as "the act of locating a machine-processable description of a WS that may have been previously unknown and that meets certain functional criteria." It is a service responsible for the process of performing discovery, a logical role, which could be performed by either the requester agent, the provider agent or some other agent.

*Motivation and Obstacles.* The main use of WSs up to now comprised the invocation of services from distance, by sending and receiving messages. This situation however is not efficient and the reasons are the needs suggested by the complicated applications, which on the one hand require access to complicated WSs and on the other hand need to be able to choose between an abundance of provided Web Services with the same functionality. Since WSs found in repositories can be tagged with a wealth of information, methods to narrow the discovery to those matching a particular technical fingerprint can be quite complicated.

Web Service Discovery mechanisms allow access to service repositories that can warehouse information about businesses, services and further details. In that sense such mechanisms should be capable to retrieve a wide spectrum of information concerning the service providers themselves beside their advertised services.

Moreover there is a need for dynamic discovery structures that will be always up-to-date providing efficient and available Web Service choices. The discovery mechanism should offer a number of capabilities, recognizable at both development and execution time. During development, one may search a web service repository for information about available services. At execution, client applications may use this repository to discover all instances of a web service that match a given interface.

The main obstacle affecting the Web Service Discovery mechanisms is heterogeneity between services. A high level approach is considered by the emerging Web service architecture [1]. Each examined solutions in this work try to overcome different aspects of this heterogeneity in order to match the best Web Service available. The identification of different kinds of heterogeneity gives an impression on what has to be considered in order to avoid or mitigate them [20]:

- Technological heterogeneities (different platforms or different data formats).
- Ontological heterogeneity (domain-specific terms and concepts within services that can differ from one another, especially when developed by different vendors).
- Pragmatic heterogeneity (different development of domain-specific processes and different conceptions regarding the support of domain-specific tasks).

Having in mind the above this work tries to critically present the existing solutions in Web Service Discovery and set future goals. In section 2 the main players in the discovery game are outlined. The architectural aspects are examined in section 3. The data models facilitating discovery are discussed in section 4. Quality of Web Service provisioning appears in section 5, before future steps and conclusions in section 6.

## 2 Roles in the Discovery Game: Description of Players

WS Discovery mechanisms include a series of registries, indexes, catalogues, agent-based and Peer to Peer-P2P solutions. The most dominating among them is the Universal Description Discovery and Integration-UDDI standard that is currently in version 3 [17]. It can be considered relatively mature as little has changed in depth since the first edition of the standard. The different main players are presented in the following subsections in order to have a first differentiation among the available solutions.

### 2.1 Catalogues

Web Service catalogues are the dominating technological basis for WS Discovery mechanisms. They are specialized repositories which implement a specification framework as metaschema. In particular, prior to the UDDI standard, organizations lacked a common approach to publish information about their products and web services for their customers and partners. UDDI established the first uniform method that included details for integration of already existing systems and processes between business partners.

UDDI allows the enterprises to discover and share information with regard to the web services and other electronic and non-electronic services that are registered in a registry. A UDDI registry service is a WS that manages information about service providers, service implementations, and service metadata. In order to find a web service using the UDDI and much information regarding the required service is needed. The requirements include key words, part of the service's name and patience, in order to select the suitable service through the results of the registry. The available search tools are very simple and do not take into consideration any cross-correlations between web services and the qualitative characteristics of each web service, forcing the user to repeat the search from the beginning using new key words.

*The UDDI specifications* include a) SOAP APIs that allow querying and publishing of information, b) XML representation for the registry data model and the SOAP message formats, c) WSDL interface definitions of the SOAP and d) APIs Definitions of various technical models that facilitate category systems for identification and categorization of UDDI registrations

**Table 1.** UDDI Registries' Instances

Type	Description
Public	Querying and matching information is public to all web service consumers. In that sense public UDDI appears to be a WS itself. Publishing information into the registry is supported through secure channels (e.g. https), but this does not spoil its public character. Data communication with other registries is supported.
Protected	The notion of trust between collaborators characterizes this kind of registries. Such registries are implemented within a closed-group environment with monitored access to third parties. Administrative features may be delegated to trusted parties. Data communication with other registries is allowed only if explicitly specified.

Private It is about isolated registries fully secured. They are usually domain specific registries in an internal network. Data communication with other registries is not allowed.

Realization of a UDDI registry can have different end-user purposes (Table 1).

Using the above specifications, it is commonly recognized, though not specifically mentioned, that there are three types of information supported by the catalogue. These types included registration of white, yellow and green pages.

*White pages* include basic contact information and identifiers such as organization name, address, contact information, and other unique ids.

*Yellow pages* describe a web service using different categorizations (taxonomies). This way it is possible to discover a Web Service based upon its category.

*Green pages* include the technological information that describes the behaviors and support functions of a Web Service.

Before proceeding to the P2P solution logic of WS Discovery the simplified implementation of an index should be included in the catalogue type. It is in short a list of references for Web Services. Such compilations are neither authoritative nor validated. They are usually harvested collections of published information by the service providers (usually using web spiders).

## 2.2 P2P-based Solutions

Peer to Peer (P2P) platforms provide a good arena for the Web Service Discovery mechanisms' implementations. A P2P overlay network provides an infrastructure for routing and data location in a decentralized, self-organized environment in which each peer acts not only as a node providing routing and data location service, but also as a server providing service access. P2P can be considered a complete distributed computing model. Recently proposed P2P systems include CAN [24], Pastry [26] and Chord [33]. The above systems arrange the network of peers to a ring. Nodes are assigned IDs drawn from a global address space. Peers are also assigned a range of keys from the global address space that they are responsible for. Each peer also stores auxiliary information in order to appropriately route key lookups. Usually a key lookup is initiated at a peer. In this case the peer consults its look-up table in order to successfully route the query to the peer that stores the queried key. In the case of Chord routing with the aid of look-up table, simulates binary search on the address space of all peers, thus a request in an  $N$  peer network can be routed in  $O(\log N)$  time.

Chord mainly has been adopted as the overlay P2P network distributed web service architectures. The hosts in the P2P network publish their service descriptions to the overlay, and the users access the up-to-date Web Services. Architectural aspects are briefly discussed in this section and the interested reader may want to continue in section 4.1 for more details on data models.

In [14] an architecture called P2P-based Web service Discovery (PWSD) was presented. The authors use a Chord P2P protocol as overlay, consisting of Service Peers (SP). Each SP is mapped to several Logical Machines (Different Machines corresponding to the same hardware). Each Logical Machine maintains the necessary interfaces to map and search WSs in the P2P network. Service Descriptions as well as queries are hashed and routed in the Chord network.

The Speed-R system [32], is a WS storage and retrieval system that uses ontologies and a P2P infrastructure. Some nodes in the P2P subsystem are assigned registries, which in turn are partitioned according to their specific domain. An ontology is assigned to each domain. The P2P system is based on JXTA [9] implementation. Its architecture is based on role assignment to peers (for example some nodes have undertaken the role of controlling updates and propagating them), thus their system may suffer from single point failure.

Closing this section, catalogues and P2P solutions are the major players in Web Service Discovery. Realization of these mechanisms includes several different flavors that follow in the sections below.

### **3 Architecture: Aspects and Approaches**

There are types of WS Discovery approaches based on different architectural perception. In the next sections we distinguish the WS mechanisms according to a) the level of automation they provide, b) topological issues in terms of network involvement, c) compliance with standards & recommendations and d) platforms available.

#### **3.1 Manual Procedures vs. Intelligent Automation**

An early approach on this type of categorization appears in [4]. Web service discovery, carried out manually (e.g. by implementers during built-time) but also automatically (e.g. by self-assembling applications during run-time), is a three phase-process consisting of Web service search, Web service assessment and selection of Web services for the configuration process.

Under manual discovery, a requester human uses a discovery service (typically at design time) to locate and select a service description that meets the desired functional and other criteria. Under intelligent automated discovery, a requester agent performs and evaluates this task, either at design time or run time.

The several UDDI processes and mechanisms cover solely operational aspects of the UDDI cloud, data management, and replication aspects. They are designed and are suitable for dealing with explicitly published changes to the registry data, which are typically done by operators or publishers. While these processes can be regarded as an approach to automatically handle changes in the registry, they do not represent a solution for the problem of dynamic service invocation or fault tolerance.

A more careful approach, as proposed in [13], is to postpone the decision of which service to bind to until execution time by querying UDDI for the access points of services that are known to implement this WSDL.

#### **3.2 Centralized vs. Decentralized Solutions**

**Centralized Services.** A registry is an authoritative, centrally controlled store of information. The recommended representative of this category is the UDDI registry [17]. A lightweight version of a registry is the centralized service of indexes. Index

is a compilation or guide to information that exists elsewhere. It is not authoritative and does not centrally control the information that it references. The key difference between the two approaches is not just the difference between a registry itself and an index. Indeed, UDDI could be used as a means to implement an individual index: just spider the Web, and put the results into a UDDI registry. Rather, the key difference is one of control: Who controls what and how service descriptions get discovered? In the registry model, it is the owner of the registry who controls this. In the index model, since anyone can create an index, market forces determine which indexes become popular. Hence, it is effectively the market that controls what and how service descriptions get discovered. [4]

**Decentralized Solutions.** There is one primitive, though well-known and widespread, network decentralization approach. Publicly available UDDI nodes connected together form a service that, while appearing to be virtually a single component, is composed of an arbitrary number of operator nodes. They are called the UDDI *cloud* or *federation* [25]. An operator node is responsible for the data published at this node: in UDDI terms, it is the *custodian* of that part of the data.

Data consistency issues are resolved by the invocation of data replication procedures, inherent in the UDDI. Re-querying the registry faces invocation failures caused by static service caching

More elaborated decentralized solutions have also been proposed. These systems [29], [32] build on Peer-to-Peer (P2P) technologies and ontologies to publish and search for Web Services descriptions. A *Peep-to-Peer solution* (P2P) is also proposed in [14]. They present a Peer-to-Peer (P2P) indexing system and associated P2P storage that supports large-scale, decentralized, real-time search capabilities. *Agent based* solutions include [15]. This approach aims to describe an environment called DASD (DAML Agents for Service Discovery) where WS requesters and providers can discover each other with the intermediary action of a Matchmaking service.

Distant ancestors of the distributed lookup registries are the Whois++ [36] and rWhois [27] look-up protocols. Both protocols provide online look-up of individuals, network organizations, key host machines etc. Their key attribute is their hierarchical and distributed architecture, in a similar vein but different context with the contemporary decentralized lookup protocols.

### 3.3 Complying with Recommendations vs Overriding them

**Following the UDDI standard.** In order to enrich the UDDI specification, information that is required for the procurement of WS is added to the administrative information sections, namely *white* and *yellow* pages [13]. When performance is a requisite, data are included on the quality of service (e.g. expected meantime between failures, maximum response time, maximum data throughput, and so on), which are crucial to assess its applicability. In order for discovery to be able to query about security of a Web service, security details have to be included as well (see also Section 5.1).

A further aspect to discovery is that of applying improvements to the *green pages* [20]. The behavior of Web services (strictly speaking of its particular methods) has to be documented by specifying the pre- and post-conditions of the methods that are being published to their interfaces. Thereby, designing by contract [24] is supported. A pre-

condition expresses the constraints under which an invoked method returns correct results. A post-condition describes the state resulting from a method's execution and thus guarantees that it will satisfy certain conditions. Constraints regarding the ordered invocation of Web service methods can occur between Web services. They are called coordination constraints and help configuring a Web service on the basis of application processes.

**Introduction of other approaches.** Though providing basic support for remote service invocation, UDDI does not support dynamic service invocation within a network of distributed services. Active UDDI's [11] basic approach is an extension of the existing UDDI infrastructure without requiring changes to the data structures or the APIs themselves but using a totally new web service that plays the role of a man-in-the-middle. This solution provides a proxy based approach in order to dynamically provide registry updates.

A different use for Web Service discovery is presented in [16]. In order to build an open, large-scale and inter-operable, multi-agent system in the context of Grid computing, an attempt to integrate agent technologies with Web Services is made. The Grid problem is defined as flexible, secure, coordinated resource sharing, among dynamic collections of individuals, institutions and resources [7]. An extension of UDDI registry is used, with additional information (meta-data) about agents and an ontology-based pattern-matching in order to accommodate the kind of searching that is required to locate an agent service according to the performative it supports. The proposed extension of UDDI contains WSDL descriptions of all agents that have been registered. In this way dynamic discovery and invocation of services by software through common terminology and shared meaning is enabled.

Fig. 1 presents the above mentioned approaches according to their architecture.

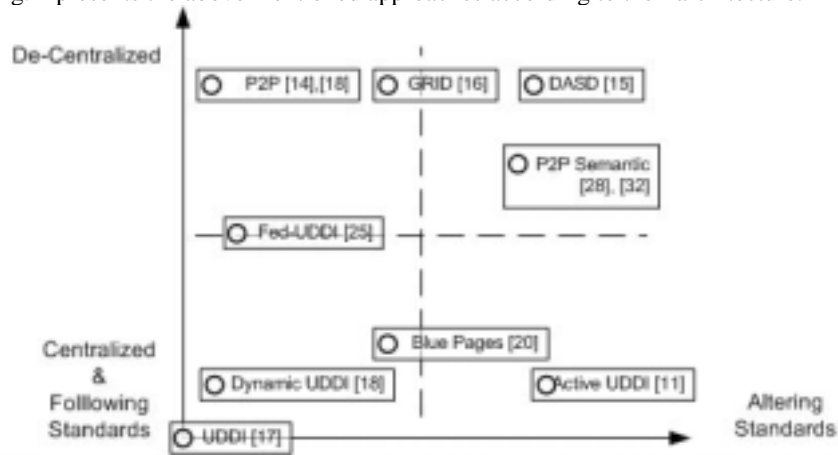


Fig. 1. Categorization based on the architectural approach

### 3.4 Industrial Platforms

Following the surge for WS Discovery, major industrial implementation platforms have included search facilities. Windows 2003 has a UDDI server preinstalled with the OS, whereas many J2EE vendors build UDDI instances into their application servers (see [13]).

**Java 2 Enterprise Edition (J2EE).** Sun Microsystems is positioning its Java API for XML Registries (JAXR) as a single general purpose API for interoperating with multiple registry types. JAXR allows its clients to access the Web Services provided by a Web Services implementer exposing Web Services built upon an implementation of the JAXR specification.

**Microsoft .NET.** At first, Microsoft had the discovery of Web Services with DISCO in the form of a discovery (DISCO) file. A published DISCO file is an XML document with links to other resources describing the Web Service. Since the wide spread adoption of UDDI, however, Microsoft has supported it in order to maximize interoperability between solutions in what is, after all, a set of specifications for interoperability. In addition to providing a .NET UDDI server, the UDDI SDK provides support for Visual Studio .NET and depends on the .NET framework. Products such as Microsoft Office XP offer support for service discovery through UDDI.

**Java-Based APIs.** The UDDI specifications do not directly define a Java-based API for accessing a UDDI registry. The Programmer's API specification only defines a series of SOAP messages that a UDDI registry can accept. Thus, a Java developer who wishes to access a UDDI registry can do so in a number of ways: 1) using Java-based SOAP API 2) Using a custom Java-based UDDI client API or 3) Using JAXR

## 4 Data Models for Web Services

An important aspect of the Web Services discovery concerns the issue of the way the Services themselves are modeled. The term *model* in this context refers to the representation of Web Services, a process that takes place before their discovery. In this section we present two alternative viewpoints: the Information Retrieval approach and the Semantics approach.

### 4.1 The Information Retrieval approach

The simplest Data Model is the *Catalogue/Keyword Based*. This model is followed by the legacy UDDI Standard and the discovery mechanism it supports. In a nutshell, the textual description that accompanies each Web Service is stored in the UDDI catalogue along with the tModel that provides the Service functionality. The retrieval stage comprises a user or a search program, entering a query to the catalogue. The query consists of keywords, which are matched against the stored descriptions. The matched Web Services are then returned as a candidate answer set and the user browses them in order to find which one of them really suits her needs or, what tends to be a frequent case, resubmits another query.

The above approach followed by the current UDDI registries, resembles the classic Boolean Information Retrieval model [2]. Despite its simplicity and ease of implementation, it suffers from either lots of returned results or very few returned ones.

A mainly architectural drawback of the approach is the fact that usually a centralized registry hosts the majority of descriptions and thus receives millions of requests being thus a bottleneck point. Decentralized proposals are discussed in section 3.2

An elegant approach to tackle the inadequacy of keyword-based Web Service Discovery was proposed in [28]. The key concept in their approach is to represent service descriptions as document vectors, a dominant approach in the IR field (see [2]). A description text, thus, corresponds to a vector  $\vec{d}$  in the vector space spanned by all the terms used in all Service description texts. They go one step further by representing all the document vectors, as columns in a term-document matrix  $A$ .

Another IR technique is afterwards applied, which transforms the matrix  $A$  achieving a representation of the document collection by its more significant semantic concepts, or what is called *Latent Semantic Indexing* (LSI) [3]. This method is observed and proved to be able to return documents of the modelled text collection, which are more closely related to the semantics of the expressed query, regardless of exact matching or not with the query terms.

When applying LSI to the discovery of Web Service they observed that description vectors resulting from the transformation of the original matrix, were mapped more closely to the vector space representation of the query, than the respective representations of plain keyword-based descriptions.

A Web Service modelled as  $d$ -dimensional vectors, can also be thought of as a point in  $d$ -dimensions. In that respect a geometric data structure for indexing multidimensional data can be deployed in indexing and querying Web Services. Instead of transferring the problem to high-dimensions, Schmidt and Parashar [30], use a transformation, which injectively maps points in higher dimensions, to numbers. This transformation is called *space-filling curve*. Many space-filling curves have been proposed (see [8]), but among those, the Hilbert curve has the important property that adjacent intervals are mapped to nearby regions in  $d$ -dimensions.

In the system of Schmidt and Parashar [30], a unique ID is generated for each Web Service, through the Hilbert curve mapping. The IDs are then stored in a Chord [33] of Web Service Peers. Thus, the storage and retrieval of WS inherits the load balancing capability and the dynamic nature of Chord. The main advantage of the model followed by [30] is that it can efficiently support partial match queries. These queries are realized efficiently mainly due to the clustering properties of Hilbert curve. The querying procedure is further enhanced, by some query optimization heuristics.

Li et. al. [14] combine keyword matching with P2P storage presenting a system that also maps the XML Service Descriptions over a P2P Network, using distributed hashing. In that sense peers act both as service providers and request generators. The XML Service Descriptions are parsed in order to extract service keywords and keywords are hashed with the MD5 hash function. The underlying P2P Network Protocol is also Chord [33] and the modified system is called XChord. Chord's distribution policy is enforced to route the generated hash descriptions to nodes. A Web Service query starting at a peer node, is also decomposed into keywords which are subsequently sought for using the Chord searching principle. XChord is proved more stable, load balanced and less space consuming according to the conducted experiments.

## 4.2 Semantics approach

Some recent work has focused on performing semantic matching for Web Services Discovery. This development is increasingly significant since it seems to be able to tackle some of the UDDI catalogue inadequacies. The predominant problem is the restrictions posed by keyword matching that do not allow retrieval of WS with similar functionality; two WSDL descriptions can be used to describe the same Service but with different words. However, when modeling web services with ontologies the semantic representation of concepts and their relations can be exploited and thus semantic matching to be performed. Semantic descriptions of Web Services can be obtained with the use of DAML-S [5] or OWL-S [21] languages.

Paolucci et. al. [22] present a framework to allow WSDL and UDDI perform semantic matching. Web Services are modeled as ontologies, or *Service Profiles* as they are called, with the use of the DAML-S [5]. Typically, a Service Profile contains information for the *Actor* (provider), *Functional Attributes* like Geographic Location and *Functional Descriptions* such as inputs outputs of the service. By maintaining ontology hierarchies, it is possible to perform semantic matching, which is subsequently performed by exploiting the subsumption capabilities of DAML. In order to combine ontologies with the UDDI Registry, the authors define a separate layer the *DAML-S Matchmaker*. The matchmaker does not extend any of the UDDI page categories, but it is treated as an add-on, which undertakes semantic matching and the mapping of ontologies to UDDI Descriptions. Semantic matching especially when using DAML-S has several advantages. First of all it provides matching flexibility, because results are returned that can differ syntactically with the input query. It also provides accuracy since no matching is performed unless this is derived from the hierarchy and finally the concept of matching degree can be supported.

A recent development was the introduction of a new language, OWL-S, to combine semantic annotation of Web Services with their discovery and invocation with WSDL and SOAP (see [34]). This approach has led to the *OWL-S Matchmaker* module.

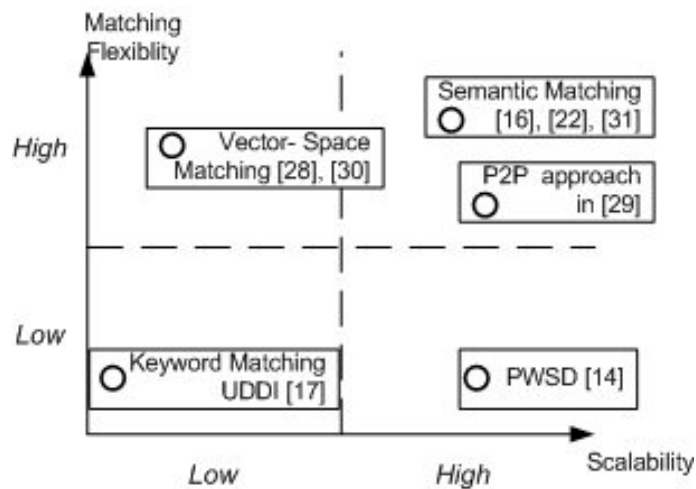
The approach of adding DAML extensions to UDDI descriptions is also adopted by [31]. Instead of providing a separate layer, they perform a simpler construct by enabling both WSDL descriptions and UDDI registrations contain semantic information. This information is simply a mapping between WSDL entries and DAML+OIL ontologies. In the case of UDDI, different tModels are provided to represent functionality, input output, etc. A matching procedure that uses templates and exploits semantic descriptions to provide semantic matching is performed.

Moreau et. al. [16] also perform some kind of semantic matching but in a different context. They describe agents performing Grid computations, as WSs. Hence, they transform agent ontologies into XML and semantic matching is performed by validating structurally expressed queries against agent description schemas. In the approach of [15] a combination of semantic annotation of web services and agent-based publishing and discovery is followed.

Interesting implementation frameworks for semantic matching are proposed in [10] and [20]. The first paper provides a framework that uses ontologies to discover (“bind” in that case) the web services that best match a specific operation domain (desired set of operations). The available data are represented with domain ontologies and the available operations, with operation ontologies. Generalization relationships on both models are encoded in XML formats and so are binding relationships. Binding can thus be per-

formed by a binding ontology that decides what fits where according to binding relations. Overhage in [20] proposes the implementation of semantic descriptions as an extension of the UDDI protocol, termed as E-UDDI. E-UDDI introduces “blue pages”, sections that contain semantic descriptions of Web Services; the latter are implemented in DAML-S. The model described in [20], also provides extensions to the green page section of the UDDI, by adding the capability to define constraints in the WS execution sequence. However E-UDDI seems to be more a vision than an implementable system.

Fig. 2, presents a brief taxonomy of Discovery Models.



**Fig. 2:** Taxonomy of discovery models with respect to Scalability and Matching Flexibility

## 5 Quality of Web Service Provisioning

The Quality of WS Provisioning (abbr. QoWS), is an issue which was more or less set aside in most the work in WS area. Therefore, neither concrete definition nor globally accepted notions of QoWS exist. Some recent work however, breaks new ground, in an attempt to define some of the QoWS parameters and methods of delivery.

### 5.1 QoWS parameters

Ran [23], Ouzzani [19] and Ouzzani-Bouguettaya [18] highlight the predominant parameters that define the Quality of Web Service. We will refer only to the most important of them, due to space limitations.

**Computational behavior** We are interested in parameters such as: *Execution Attributes* (e.g. *Latency*, *Accuracy*, *Throughput*), *Security* (*Encryption*, *Authentication* etc.), *Privacy* (is there any privacy policy implemented?), *Availability* (the probability of the service being available).

**Business Behavior** Mainly referring to *Execution Cost* (how much will a single execution cost?) and *Reliability* of the service publishing company.

**Metadata Constraints** Constraints that have to be followed regarding UDDI/WSDL parameters such as location, specific companies etc.

The necessary modifications of the tModel to include QoWS characteristics are also described in [23].

## 5.2 QoWS Provisioning

The need for QoWS Provisioning has emerged in complex Web Service applications. In a typical scenario a user is executing a complex query which is transparently translated to a set of WSs, which may have to be executed in a specific order. This execution sequence is also referred to as *execution plan*. A single WS in this plan can possibly be provided by more than one distinct access points. The goal in this case is to select the best execution plan in order to maximize the delivered QoWS.

In order to select the proper execution plan for the delivery according to quality constraints, it is proposed in [18],[19] to assess each of the QoWS parameters by a quantity called *the quality distance*. Quality distance  $dQ_i$ , effectively computes the distance between the advertised and the delivered quality of a service  $i$ . The quality distance for an execution plan is given by Eqn. (1).

$$QoWSdist = \sum_{QoWSi \in pos} (pQ_i - aQ_i) + \sum_{QoWSi \in neg} \left( \frac{1}{pQ_i} - \frac{1}{aQ_i} \right) \quad (1)$$

In the above *pos* and *neg* are the sets of QoWS parameters that one wishes to maximize or minimize, respectively. For example response time and execution time need to be minimized while availability needs to be maximized. Moreover,  $pQ_i$  and  $aQ_i$  represent provisioned and advertised values for parameter  $i$ , respectively. Actually, Eqn. 1 provides with a rating of how close the execution of a WS is to its advertised value.

The above rating will influence the choice of the optimal execution plan. Hence, an objective function  $F(p)$  (Eqn. 2) is defined for each execution plan  $p$ .

$$F(p) = \left( \sum_{Q_i \in neg} \frac{Q_i^{max} - Q_i}{Q_i^{max} - Q_i^{min}} + \sum_{Q_i \in pos} \frac{Q_i - Q_i^{min}}{Q_i^{max} - Q_i^{min}} \right) \quad (2)$$

where  $Q_i$  denotes the measured value for a parameter and  $Q_i^{max}$  ( $Q_i^{min}$ ) the maximum (minimum) value over all available choices for service  $i$ . The  $F(p)$  function can be further tuned by weighting appropriately each sum term according to the rating that the corresponding parameter receives from Eqn. 1. Subsequently, optimization techniques are employed to discover the optimum execution plan that will maximize  $F(p)$  and thus deliver QoWS.

We must also note that the DAML-S based architecture, mentioned in [22], includes some QoWS parameters of "Metadata Constraint" category, as part of their defined ontologies. Semantic matching is subsequently taking these parameters into account in order to compute the matching degree, however this procedure is tacitly assumed.

## 6. Conclusions

The Web Service Discovery mechanisms appearing above strive to achieve a set of goals to enhance efficiency in the matching and binding procedure. Among others, discovery mechanisms should enable search and assessment solely based on a Web Service's outer view. Assessment is based on multi-criteria decision-making [12].

Furthermore, focus should be given on defining QoWS metrics. This is important so as to refine WS Discovery mechanisms in order to reach minimum standards in performance, security and availability in the matching & binding results. Support of load balancing in the WSs delivery starting at the moment of choosing one will then be possible. This can boost performance especially at situations with excessively increased workload is met.

The work on Discovery mechanisms should try to reach resulting structures not only applicable to WSs, but web-based or other software components in general. This would require introducing some additional specifications about the platform, the system requirements, the type of reuse, the type of code, and the scope of supply [20]. These specifications could be added within the administrative information. Consequently, even a unified specification of software components could eventually be achieved (which could be the basis for future component catalogues & CASE tools).

Concluding this discussion, several approaches have been discussed using different view-points. Beside UDDI, emerging decentralized aspects such as P2P solutions are promising. Enhancement of data models is also possible by elaborating both IR techniques and ontologies, especially when taking into account that research about the Semantic Web is particularly popular. WS Discovery mechanisms have a role even more important than Web searching, because they facilitate the need for collaboration among business processes and consumers over widely accepted Web standards.

## Acknowledgements

The authors wish to thank the anonymous referees, whose insightful comments have helped at improving the presentation of this paper.

## References

1. Austin, D., Barbir, A., Ferris, C., Garg, S. (eds.): Web Service Architecture Requirements. W3C Working Group Notes (2004) <http://www.w3.org/TR/wsa-reqs>
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, Addison-Wesley, (1999)
3. Berry, M. W., Dumais, S. T., and O'Brien, G. W.: Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4). (1995) 573-595
4. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion M., Ferris, C., Orchard, D. (eds.): Web Services Architecture. W3C WG Note. <http://www.w3.org/TR/ws-arch/>
5. DAML-S Coalition: DAML-S: Web Service Description for the Semantic Web, Proc. 1st Int'l Semantic Web Conf. (ISWC 02), 2002.
6. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. Proc. 15th IJCAI. Nagoya, Japan. (1997) 578-583

7. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid. Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*. (2001)
8. Gaede, V., Gunther, O.: *Multidimensional Access Methods*. ACM CSUR, 30(2). (1998)
9. Gong L., "JXTA: A Network Programming Environment". *IEEE Internet Computing*, (5)3:88-95, May/June 2001.
10. Hu, Z.: Using Ontology to Bind Web Services to the Data Model of Automation Systems Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems. (2002) 154 – 168
11. Jeckle, M., Zengler, B.: Active UDDI - an Extension to UDDI for Dynamic and Fault-Tolerant Service Invocation, Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services and Database Systems , LNCS. (2003) 91-99
12. Konito, J.: A Case Study in Applying a Systematic Method for COTS Selection. In: Proc., 18th Int. Conf. on Soft. Eng. (ICSE). IEEE Computer Society Press (1996) 201–209
13. Lacey, P.: Uddi & Dynamic Web Service Discovery. <http://www.ddj.com/articles/2004>
14. Li, Y., Zou, F., Wu, Z., Ma, F.: PWSO: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network, In Proc. APWeb04, LNCS 3007. (2004) 291-300
15. Montebello, M., C. Abela, C.: DAML Enabled Web Services and Agents in the Semantic Web. Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services and Database Systems , LNCS. (2003) 46 – 58
16. Moreau, L., Avila-Rosas, A., Dialani, V., Miles, S. and Liu, X. Agents for the Grid: A Comparison with Web Services (part II: Service Discovery). In Proceedings of Workshop on Challenges in Open Agent Systems , Italy. (2002) 52-56
17. OASIS UDDI Specifications TC - Committee Specifications <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>
18. Ouzzani M., Bouguettaya A.: Efficient Access to Web Services. *IEEE Internet Computing*, March/April (2004) 34-44
19. Ouzzani, M., Efficient Delivery of Web Services, PhD Thesis, Virginia Polytechnic. (2004)
20. Overhage, S.: On Specifying Web Services Using UDDI Improvements. 3rd Annual International Conference on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World Net.ObjectDays, Germany (2002)
21. OWL-S Specifications, <http://www.daml.org/services/owl-s/1.0/>
22. Paolucci, M., Kawamura, T., Payne, T. R., Sycara, K.: Semantic Matching of Web Services Capabilities. In Proceedings of the 1st Int. Semantic Web Conference (ISWC2002). (2002)
23. Ran S.: A Model for Web Services Discovery with QoS, ACM, ACM SIGecom Exchanges, Volume 4 , Issue 1 Spring. (2003) 1 - 10.
24. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. Proceedings of ACM SIGCOMM'01, San Diego, September (2001)
25. Rompothong, P., Senivongse, Tw.: A Query Federation of UDDI Registries, ISICT (2003)
26. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proc. of the 18th IFIP/ACM Middleware, Germany. (2001)
27. rWhois RFC, <http://rfc.net/rfc2167.html>
28. Sajjanhar, A., Hou, J., Zhang, Y.: Algorithm for Web Services Matching, In Proc. APWeb 2004, Lecture notes in Computer Science 3007, Springer Verlag, (2004) 665-670
29. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services. Proc. 2nd Int. Conf. P2P'02, Sweden. (2002) 104–111.
30. Schmidt, C., Parashar, M. A.: Peer-to-Peer Approach to Web Service Discovery. *World Wide Web: Internet and Web Information Systems*, 7. (2004) 211-229
31. Sivashanmugam, K., Verma, K., Sheth, A. P., Miller, J. A.: Adding Semantics to Web Services Standards. Proceedings of the Int. Conf. on Web Services, ICWS '03. (2003) 395-401
32. Sivashanmugam, K., Verma, K., Mulye, R., Zhong, Z., and Sheth, A.: Speed-R: Semantic P2P environment for diverse Web Service registries, <http://webster.cs.uga.edu/~mulye/SemEnt/Speed-R.html>. (2004)