Chapter 2

Review of Literature

The Web Service Discovery issue in distributed applications has been handled since 2001. The present study demanded a comprehensive understanding of different approaches and mechanism used for discovering web services dynamically. The result of literature survey is presented here.

2.1 Web Service Discovery

Web Service Discovery is “the act of locating a machine-processable description of a web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions”. The goal is to find an appropriate Web service-related resource.\[97\] Traditionally, the Web service discovery processes involved manual intervention. A set of Web service descriptions are discovered according to user requirements. These service descriptions are manually scanned and those services that satisfy user requirements are selected and composed. In the context of distributed system integration, such manual intervention is unrealistic, cumbersome and time consuming.

The approaches to Web services discovery can be classified as centralized and decentralized. UDDI falls under fully centralized approach that supports replication where central registries are used to store Web service descriptions. Having realized that replicating the UDDI data is
not a scalable approach several decentralized approaches have been proposed. Three major operators, namely IBM, Microsoft, and ARIBA provide public UDDI service.

Web service 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.

2.2 Web Service Discovery Mechanisms

Web service discovery mechanisms allow accessing to service repositories and/or “crawling the Web” in the search for services. Since large amount of information is associated with web services, methods to narrow the discovery can be quite complicated and use such semantic information. Search engines such as Google and Yahoo have become a new source for finding Web services. However, search engines do not easily separate and expose to users the basic service properties (i.e. binding information, operations, ports, service endpoints, among others), as they are instrumented or crawling and indexing generic content. In addition, search engines generally crawl Web pages from accessible Web sites while publicly accessible WSDL documents reside on Web servers; hence they are not designed to be fetched and analyzed by normal crawlers.

Web Service Discovery mechanisms are broadly classified into three types:

- Peer-to-Peer mechanisms based on decentralized approach
- UDDI and ebXML registry based mechanisms based on centralized approach
- Alternative mechanisms

2.2.1 Peer-to-Peer mechanisms based on decentralized approach

Peer-to-Peer (P2P) mechanisms are based on decentralized approach in which web services are not discovered on a single registry but it allows web services to be discovered dynamically on the network. from peer-to-peer. All peers in the network are functionally equal and co-operate with each other for responding to the user request. At discovery time, a service requester queries its neighbors in search of a suitable web service. If any one of them matches the request, then it replies. Otherwise each queries its own neighboring peers and the query propagates through the network until a particular hop count or other termination criterion is reached. As peer-to-peer architectures do not need a centralized registry and any node on the
network is able respond to the queries it receives, this architectures do not have a single point of failure, such as a centralized registry. Additionally, each peer may contain its own indexing of the existing web services. But at the same time, the reliability provided by the high connectivity of peer-to-peer systems comes with performance costs and no assurance of predicting the path of propagation. Every node in peer-to-peer architecture must have the resources needed to ensure query propagation and response routing. This results that each node acts as a relay of information that may be of no use for the node itself. If the number of nodes on the network are increased, connectivity increases and this results in reducing the efficiency of the system and increasing overhead. Still there may be no guarantee that a request will be propagated across the entire network, and hence there is no guarantee to find the desired web service.

**Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications” 2001 [41].** Chord is a distributed lookup protocol which is designed to efficiently locate the node that stores a particular data item. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Chord is scalable, with communication cost and the state maintained by each node scaling logarithmically with the number of Chord nodes.

**Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, Hai Jin, “Chord4S: A P2P-based Decentralised Service Discovery Approach” 2008 [72] proposes a peer-to-peer based decentralised service discovery approach named Chord4S. Chord4S utilises the data distribution and lookup capabilities of the popular Chord to distribute and discover services in a decentralized manner. Data availability is further improved by distributing service descriptions of functionally-equivalent services to different successor nodes that are organised into a virtual segment in the Chord circle. In addition, the Chord routing protocol is extended to support efficient discovery of multiple services with single request. This enables late negotiation of service level agreements between a service consumer and multiple service**
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

providers. They claim that Chord4S achieves higher data availability and provides efficient query with reasonable overhead.

Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, Amr El Abbadi, “A Peer-to-Peer Framework for Web Service Discovery with Ranking” 2004 [19]. They have proposed a structured peer-to-peer framework for web service discovery in which Web services are located based on both service functionality and process behavior. It represents the process behavior of the web services with finite automata and use these automata for publishing and querying the web services within the system. The model is scalable and robust due to the underlying peer-to-peer architecture. Web services can join and leave the system dynamically. We also propose an efficient and scalable reputation model based on sketch theory. Thus the returned services are ranked based on the trust and quality ratings of the services using the proposed reputation model.

Ioan Toma, Brahmananda Sapkota, James Scicluna, Juan Miguel Gomez, Dumitru Roman, and Dieter Fensel, “A P2P Discovery mechanism for Web Service Execution Environment” 2005 [40]. They have presented a scalable approach for automatic discovery of services over distributed execution environments. The solution is based on P2P technologies that proved to be scalable, efficient and robust solutions for distributed systems. As shown in Fig. 2.1, equal WSMX peers which participate in the service discovery process have to match the local registered services against a broadcasted query. A major aspect that is to be considered in this context is the topology of network. For message routing the topology of the network has significant impact on the overall performance of the service discovery process. The approach that they have adopted to address these aspects is the HyperCuP approach. HyperCuP decreases the big overhead of network communication by providing a topology based on a structure called hypercube: a generalization of a 3-cube to n dimensions. In the resultant graph the connection between neighbored nodes can be associated with a specific dimension of the hypercube. This allows us to define a message broadcast scheme with certain guarantees: nodes receive a message exactly once and the number of messages sent is linearly dependent on the number of nodes in the network. A set of structuring ontology concepts is used to build a hypercube consisting of distinct concept clusters. A query which consists of a logical
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

combination of service and domain ontology concepts is routed to all relevant concept clusters. Within a concept cluster the message is broadcasted to all contained peers. If the query formulation matches the conceptual description of a service the representing peer is reacting by sending an according response to the requester.

Figure 2.1 Peer-to-Peer approach for Distributed Discovery in WSMX

Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi, “WSPDS: Web Services Peer-to-peer Discovery Service” 2004 [22]. They have introduced WSPDS (Web Services Peer-to-peer Discovery Service), a fully decentralized and interoperable discovery service with semantic-level matching capability. They claim that a peer-to-peer architecture of the semantic-enabled WSPDS not only satisfies the design requirements for efficient and accurate discovery in distributed environments, but also is compatible with the nature of the web Services environment as a self-organized federations of peer service-providers without any particular sponsor. WSPDS is a distributed discovery service implemented as a cooperative service. A network of WSPDS servants collaborate to resolve discovery queries raised by their
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

peers. Fig. 2.2 depicts an unstructured peer-to-peer network of WSPDS servants. Each servant is composed of two engines, communication engine and local query engine, playing two roles: (1) Communication and Collaboration: the communication engine provides the interface to user and also represents the servant in the peer-to-peer network of servants. This engine is responsible for receiving service queries from users, resolving the queries by local query (through the local query engine) and global query (via its peer servants), and finally merging the received responses to reply to the user query; and receiving queries from its neighbors in the peer-to-peer network, resolving the queries by local query, and sending the response (if not empty) to the network as well as forwarding the query to other neighbors in the network. (2) Local query: the local query engine receives the queries from the communication engine, queries the local site (where the servant is running) for matching services, and sends responses to the communication engine.

![Figure 2.2 WSPDS Architecture](image)

**Figure 2.2 WSPDS Architecture**

*Sivashanmugam, K., Verma, K., Mulye, R., Zhong, Z., and Sheth, A., “Speed-R: Semantic P2P environment for diverse Web Service registries” 2004 [95].* They have proposed Speed-R system for publishing and discovering web services that uses ontologies and a P2P infrastructure. Some nodes in the P2P subsystem are assigned registries, which in turn partitioned according to their specific domain. An ontology is assigned to each domain. Its architecture is based on role assignment to peers. e.g. some nodes have undertaken the role of
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

controlling updates and propagating them, thus their system may suffer from single point failure. Fig. 2.3 presents architecture of Speed-R system. Each Peer runs ‘Operator Peer’ to control semantic access to its registry (direct registry access without support for semantic discovery is allowed). Peers support Domain Ontology and Operator Services (if ontology is not used, no semantic discovery can be provided, search defaults to keyword search). Each Registry can be accessed using API, which is dependent on its implementation and standard that it conforms to. Registries Ontology (i.e., the upper ontology, only one for the whole P2P cloud) is present in the P2P network. Any given time peers are aware of the updated Registries Ontology.

![Figure 2.3 Speed-R Architecture](image)

**Figure 2.3 Speed-R Architecture**

Gang Zhou, Jianjun Yu, Rui Chen, Hui Zhang, "Scalable Web Service Discovery on P2P Overlay Network" 2007 [25]. They have developed the ServiceIndex system for service discovery which merges advantages of P2P computing and Semantic Web Services into web services world. The ServiceIndex system tries to solve the problem of semantic search in distributed environment and support complex search, tree lookups, locality sensitivity, and ontology based service discovery. It is possible to construct a dynamic and pure P2P overlay network for service discovery and achieve considerable system performance.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

**Summarizing the above papers**, it can be concluded that peer-to-peer discovery mechanism provide an efficient and scalable solution for the discovery of services in distributed systems as it is fully decentralized and do not have a single point of failure, such as a centralized registry. Additionally, each peer may contain its own indexing of the existing web services. In case of semantic search, ontology can also be specified.

### 2.2.2 UDDI and ebXML registry based mechanisms based on centralized approach

In centralized approach, UDDI (Universal Description, Discovery and Integration) and ebXML (electronic business XML) are the two types of registries which are storing and managing web service information centrally. UDDI is a vendor-sponsored initiative led by IBM, Microsoft, and Ariba, whereas, ebXML is a UN/CEFACT (United Nations center for Trade Facilitation and Electronic Business) / OASIS sponsored initiative for creating a single global electronic market. UDDI and ebXML, make it possible for business organizations to publish information on the Internet about their products and web services, where the information can be readily and globally accessed by clients who want to do business. UDDI Registry is a web-based registry that exposes information about a business providing web service, web service and its technical interfaces. A service provider makes its services available to public users by publishing information about the service in a UDDI registry.

The information about Web services in a UDDI registry includes a description of the business and organizations that provide the services, a description of a service’s business function, and a description of the technical interfaces to access and manage those services [92]. A UDDI registry which is an XML-based registry consists of instances of four core data structures including the businessEntity, the businessService, the bindingTemplate and the tModel. This information comprises everything a user needs to know to use a particular Web service. The businessService is a description of a service’s business function, businessEntity describes the information about the organization that published the service, bindingTemplate describes the service’s technical details, including a reference to the service’s programmatic interface or API, and tModel defines various other attributes or metadata such as taxonomy and digital signatures [92]. UDDI (Universal Description, Discovery and Integration) plays a key role in the web service architecture. It provides a structured and standard description of the web
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

... service functionalities as well as searching facilities to help in finding the providers that better fit client requirements. Generally speaking, a UDDI registry contains the information about businesses and services these business organization offers. These services may not be always web services or computer related services at all. In fact, UDDI was designed in the intention of holding arbitrary information about a business. It serves not only as an access point for service related information, but also about the businesses themselves. Structure of UDDI is similar to telephone directory, in the way that phone numbers are stored and catalogued.

With ebXML, companies are able to define how to conduct business using a specific vocabulary. Core components are used to build predefined documents. Messages are sent using standardized protocols and formats. All of this information is stored in ebXML registries. Business Processes and Business Document has to be created prior to their use. Specification of these both describes the workflow of business processes and the information exchanged between the partners respectively. These documents can be composed of reusable and extendable Core Components. An ebXML Registry provides means for finding organizations, business processes, core components and other objects. Therefore it does not store the actual objects but metadata and associations between them. Business partners register their services in an ebXML registry along with their Collaboration Protocol Profiles (CPPs). During the search the registry is queried for a business partner that offers the required service. Based on the CPPs of both partners a Collaboration Protocol Agreement (CPA) is formed which specifies what kind of business is to be performed and how. Usually CPA is negotiated after being proposed by one party. Based on the agreement it is now possible to configure an ebXML enabled application and execute the business process.

Ali ShaikhAli, Omer F. Rana, Rashid Al-Ali, David W. Walker, “UDDIe: An Extended Registry for Web Services” 2004 [79]. They implement UDDIe an extension to UDDI, which supports the notion of “blue pages”, to record user defined properties associated with a service and to enable discovery of services based on these. UDDIe enables a registry to be more dynamic, by allowing services to hold a lease – a time period describing how long a service description should remain in the registry. UDDIe can co-exist with existing UDDI – and has been implemented as an opensource software.

Extensions in UDDIe are based on four types of information: business information; service information, binding information; and information about specifications for services. A service
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

may be discovered by sending requests based on service information. The extensions provided in UDDIe consist of the following:

**Service Leasing**: Service providers may want to make their service available for limited time periods (for security reasons, for instance) – or the service may change often. UDDIe supports “Finite” and “Infinite” leases – where a finite lease can be immediate, or based on a future lease. When using finite leases, service providers must define the exact period for which the service should be made available for discovery in the registry. The lease period is restricted by the maximum allowable lease period defined by the UDDIe administrator. Depending on the type of application domain for which the UDDIe registry is to be used, the value of the maximum allowable lease may change. This parameter is left to the UDDIe administrator to set. For example, if a service provider is interested in publishing a service in UDDIe for two hours, but the maximum granted lease is one hour, publication of the service will be rejected by the registry. A “future lease” allows a service provider to make the lease period start at a future time – the service will only be discoverable once this lease has been activated. Alternatively, service providers may want to publish their services for an infinite period of time. Such leases are allowed in UDDIe, but only if the ratio of finite/infinite lease services is within a threshold (a parameter set by the UDDIe administrator).

**Replication**: The UDDI Business Registry (UBR) is conceptually a single system built from a group of nodes that have their data synchronized through replication. A series of operator nodes each host a copy of the content, thereby replicating content among one another. Content may be added to the UBR at a single node, and that operator node becomes the content master. Any subsequent updates or deletes of the data must occur at the operator node where the data was inserted. UDDIe can be used as a private operator node that is not part of the UBR. Private nodes do not have data synchronized with the UBR, so the information contained within is distinct. The availability of private nodes is significant if an organization considers sharing
their service content a security problem. This is useful in instances where a company does not want to expose certain service offerings and business processes to others – for instance, suppliers set up to handle large contracts may not be able to handle individual customers.

In UDDI a business Service, structure represents a logical service – and is the logical child of a business Entity – the provider of the service. Service properties are contained in the property Bag entities – such as the Quality of Service (QoS) that a service can provide, or the methods available within a service that can be called by other services. Figure 2.4 illustrates the attributes associated with a property – and consists of a propertyName, propertyType and propertyValue. Some of these are user defined attributes – such as propertyType – and can be number, string, method etc. Range based checks, for instance, are only allowed if the propertyType is a number. The API for interacting with the registry system extends three classes within existing UDDI implementations. The extensions provided in the API include: _ saveService: This set of APIs is mainly used for publishing service details. This has been extended from the original UDDI system to introduce dynamic metadata for services. Such metadata could be used to represent attributes such as cost of access, performance characteristics, or usage index associated with a service, along with information related to how a service is to be accessed, and what parameters the service will return. The saveService call utilises the propertyBag mechanism provided in UDDI. _ findService: This set of APIs is
mainly used for inquiry purposes. In particular we extend this set of API from the original UDDI to include queries based on various information associated with services, such as Service Property and Service leasing.

They claim that extensions to the UDDI registry and query mechanisms would add a great search flexibility, making UDDI a more powerful search engine. The ability for UDDIe to co-exist with standard UDDI version is also an important aspect of this work – as they do not break compatibility with existing UDDI deployments.

*Phil Bonderud, Sam Chung, Barbara Endicott-Popovsky,* “Toward Trustworthy Service Consumers and Producers” 2008 [11]. They proposed a S-QoS4WS approach that utilizes ‘PublisherAssertion’ tags within the UDDI to satisfy Security and QoS issues. This approach makes use of existing mechanisms within UDDI version 3 to resolve current issues involving trust and non-repudiation. S-QoS4WS takes into consideration security and QoS issues with respect to establishing trust and nonrepudiation. The approach adds an optional third party entity to the web services paradigm whose sole purpose is to certify information about each respective business partner. The third party service certifier certifies that services offered by a service producer meet the specifications used to describe the service in the UDDI. The third party consumer validation entity authenticates that its service consumer partner is a trustworthy and legitimate business. Each third party entity is expected to publish its own web service whose sole purpose is to provide an automated way of obtaining information.

In Figure 2.6, solid lines represent interactions that require human intervention. It is expected that in order for a service to be adequately certified or a consumer to be validated, that some degree of human involvement will be required. Dashed lines represent transactions that are fully automated. S-QoS requires that a service producer select a third party entity (A) which will certify that any statistics and requirements it wishes to advertise in the UDDI, about a service, are accurate. This communication is expected to require human involvement, which is indicated by the solid line in Figure. Upon reaching final agreement concerning a service’s certification (B), the certifying entity publishes a web service to a UDDI. This service, published by the certifier, holds the results of a service’s certification. The service producer (B)
also publishes its service to the UDDI, if it has not already done so. Both the service producer and the third party certification entity make identical ‘publisher Assertions’ for this service,

![Figure 2.6: Service model of UDDI](image)

which will be explained in detail in the next section. By making identical ‘publisher Assertions’ for this service (C), service consumers can query the UDDI for ‘status:complete’ certified services. S-QoS mirrors the interactions between the service producer and its certification entity to produce consumer validations (1 – 3). Whether or not a unique UDDI is used as diagrammed, which caters only to service consumers, is irrelevant to this research and not a requirement for the success of this approach. Equivalent to communications represented by line A, communications between a service consumer (1) and its respective third party service consumer validation entity is expected to require human involvement. Upon reaching final agreement over the information to be published (2), the validation entity publishes a web service to a UDDI that holds the results of a consumer’s validation. The service consumer (2) also publishes an informational service to the UDDI that represents itself, if it has not already done so. Both the service consumer and the third party validation entity make identical ‘publisher Assertions’ for the consumer. By making identical ‘publisher Assertions’ for the
consumer (3), service producers obtain an added bonus of being able to query a UDDI for ‘status:complete’ validated consumers. This added bonus enables service producers to proactively market their services to viable organizations. Within each UDDI businesses have the option of providing a service description statement. Service consumers will enter ‘Service Consumer’ as their descriptor, consumer evaluators will enter ‘Consumer Validation’, service certification entities will use ‘Service Certification’, and service producers default to any description.

YoungKon Lee “Web Services Registry implementation for Processing Quality of Service” 2008 [103]. This paper presented the design principle for integrating quality management on Web service registry developed in UDDI specification and Web service quality management system (WSQMS). WSQMS, developed by NIA1 can measure and collect the quality information of Web services by its agency system installed on the Web service system. Web service registry is core system for registering and searching WSDL(Web Service Description Language). In a Web service registry, WSDL is referenced in a tModel, which is a container for a reference to the WSDL. Because tModel is devised to include the detail information about a Web service, it is natural conclusion that we modify tModel to be proper for including the reference to WSQDL. There are two choices. First is to make a new reference data object to WSQDL in <overviewDoc> as the form described as WSDL. This way is trivial, so it enables

![XML schema for WSQDL complex type.](image)

Figure 2.7 XML schema for WSQDL complex type.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

users to find out at once that WSDL and WSQDL describe characteristics for the same target Web service. This way, however, restricts severely the usage of reference to quality data. That is, user cannot search the quality data rapidly because there are no classification schemes for quality data. Figure 2.7 shows <wsqrlURL> in <overviewDoc> and XML schema for WSQDL complex type.

Second way is to make a specific tag, <qualityBag>, in tModel to store the reference to WSQDL. This requires additional processing modules, but enables the quality data to be used more widely. For example, this method allows the reference to WSQDL in <qualityBag> to be handled as the form of tModel, resulting that process related with tModel could have still flexibility. However, it requires updates of considerable part of the registry because the registry system should process two types of tModel for: WSDL and WSQDL. However, it is impossible to search a Web service effectively on the basis of quality data, because tModel has only reference data to WSQDL. Thus, it is desirable to implement architecture for referring Web service quality data by using the quality classification scheme. Figure 2.7 shows the tModel component structure and XML schema including <uddi:qualityBag>.

![Figure 2.8 tModel Component and Schema including qualityBag](image-url)
In Figure 2.8 the structure of <overviewDoc> is the same as previous tModel, but <qualityBag> is a new structure for referring any number of tModel. Another way is to add quality context information to tModel for quality classification scheme. This allows the registry to have quality context in <qualityBag>, as corresponding WSDL through tModel. As the previous search method of Web service registry by using <categoryBag>, a registry parses previously the quality data in <qualityBag> and stores the quality context so that users may just search a Web service satisfying some criteria by using the quality context or in quality classification. To represent quality context data consistently and to manage it requires further study. Figure 2.8 shows the <qualityBag> component structure and its XML schema including <qualityContext>, whose structure could include any type of character string. <qualityBag> stores any number of required <qualityContext> and represent any type of quality data. For example, as digital signature for message consistency and proof of message sender, a <qualityContext> as type of /eval/sec/Dsig/keySize/ could be made and we say that a system is safer when it has its value of 128 rather than 64 in the respect of digital signature safety. <qualityContext> representing Web service quality information should be registered on a registry and user can search the quality data according to the value of <qualityContext>. The registry requires the additional APIs for processing the quality data in the relationship with WSQMS. Firstly, it is required for WSQMS to have APIs searching the new registered Web service. The APIs correspond to the functionality of searching Business Entity, Service, Binding, and tModel. APIs for representing the reference to the quality information sent from WSQMS are required. If the reference to the quality data is stored in tModel, the additional APIs for processing tModel operation are required. Besides, it’s required the APIs for modifying and updating Web service quality information and synchronizing the Web service information between WSQMS and registries.

Massimo Paolucci and Katia Sycara, “Autonomous Semantic Web Services” 2003[66]. In this paper, the authors presented a mechanism that begins to bridge the gap between the Web services infrastructure and the Semantic Web. They adopted the vision of Web services as autonomous goal-directed agents that select other agents to interact with and that flexibly negotiate their interaction models, acting variously in client–server and peer-to-peer modes. The resulting web services called as autonomous Semantic Web services, use ontologies and
semantically annotated web pages to automate the fulfillment of tasks and transactions. In particular, these services use the Semantic Web to support capability-based discovery and interoperation at runtime. A first step toward this vision was to develop formal languages and inference mechanisms for representing and reasoning with core Web service concepts. The DARPA Agent Markup Language for Services (DAML-S) is the first attempt to define such a language. One objective behind the Semantic Web is to provide languages for expressing the content of Web pages and making that information accessible to agents and computer programs. More precisely, the Semantic Web is based on a set of languages such as the Resource Description Framework (RDF), DAML+OIL, and the more recent Web Ontology Language (OWL), which can be used to annotate Web content. These languages have well-defined semantics and inferential procedures that let agents draw inferences from the languages’ statements. Using the semantic markup for the US National Oceanic and Atmospheric Administration’s page reporting Pittsburgh’s weather conditions, for example, an agent could learn that the current condition is heavy snow. The agent might further learn from the Pittsburgh school board site’s semantic markup that all schools are closed on days of heavy snow. Combining the two pieces of information, the agent could infer that Pittsburgh schools are closed today. The Semantic Web’s second element is a set of ontologies that provide conceptual models for interpreting the information provided. An ontology of weather might contain concepts such as temperature, snowy, cloudy, and sunny, for example, and relationships between the terms. The Semantic Web vision is about transforming the Web into an Internet-wide knowledge-representation system in which ontologies provide the conceptual framework for interpreting the information provided by Web pages. To produce the types of inferences they have described, the Semantic Web requires computational processes and agents that can interpret semantic content and derive consequences from the information they collect. The Semantic Web also supports a more distributed computational model in which a requester transacts with multiple Web services, solving problems through collaboration and negotiation. Within this scheme, ontologies not only define a shared conceptualization for interpreting semantic markup of Web sites, but also provide a shared vocabulary that lets services across the Web use the same terminology to interpret each other’s messages. Ultimately, the Semantic Web will provide the basic mechanisms for extracting information from Web pages and the basic knowledge that Web services will use in all transactions. In addition to knowledge,
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

however, Web services need an infrastructure that facilitates reliable communication — registries to locate other services, reputation services, guarantees of secure and private transactions, and so on. Such an infrastructure falls outside the current view of the Semantic Web’s scope.

Web services advertise or request through the communication module using DAML-S. Advertisements are stored in the UDDI registry, and requests are sent to the DAML-S matching engine. The *service profile* provides a high-level view of a given Web service. It is the DAML-S analog to the Web service representation that UDDI provides in the Web services infrastructure, although the two have some sharp differences as well as similarities. Some information, such as a Web service’s provider, is present in both descriptions, but the service profile supports properties such as the representation of capabilities — the tasks the service performs — that UDDI does not support. On the other hand, UDDI describes the ports the Web service uses, whereas DAML-S relegates this information to other modules of the description, such as the grounding (described below). The *process model* specifies the tasks a Web service performs, the order in which it performs them, and the consequences of each. A client can use the process model to derive the service’s choreography, or message-exchange pattern, by figuring out what inputs it expects, when it expects them, what outputs it reports, and when. The process model’s role is similar to emerging standards such as BPEL4WS and WSCI, but focuses more on the effects of executing a service’s different components. The *service
grounding binds the abstract description of a Web service’s information exchanges —defined in terms of inputs and outputs in the process model — with an explicit WSDL operation, and through WSDL to SOAP messages and transport layer information. DAML-S’s reliance on DAML+OIL, as well as WSDL and SOAP, shows how proposed Web services standards can be enriched with semantic information. DAML-S adds formal content representations and reasoning about interactions and capabilities to Web service specifications. Therefore, DAML-Senabled Web services use UDDI, WSDL, and SOAP to discover other services and interact with them, and they use DAML-S to integrate these interactions, in their own problem solving.

Managing Web Services with DAML-S

They have implemented tools for Semantic Web service discovery and invocation making use of DAML-S and complementing current Web services systems. They describe the DAML-S/UDDI Matchmaker and the architecture of a DAML-S-empowered Web service.

DAML-S-Enabled Service Discovery

The DAML-S service profile relies on ontologies to specify what type of information the Web service reports and what effects its execution produces. At discovery time, a Web service generates a request that contains a profile for the ideal service it wants to interact with. The discovery process selects a Web service provider’s profile that matches the request. Although DAML-S profiles and UDDI Web-service descriptions contain different information, they share the goal of facilitating Web-service discovery. The combination could thus provide rich representations for Web services. Using UDDI’s TModels to encode DAML-S capability descriptions, we can reconcile the differences between the two. Once the capabilities encoded, a new module is added to UDDI: the matching engine performs inferences based on DAML+OIL logics and effectively adds capability matches to UDDI. The result is the DAML-S/UDDI Matchmaker for Web services. The Matchmaker receives Web-service advertisements, information inquiries, and requests for capabilities through the communication module, which implements a simple inquiry-and-publish API. The communication module then sends the advertisements and inquiries to UDDI through the DAML-S/UDDI translator, which transforms DAML-S encoded advertisements into UDDI format. The communication module directs requests for capabilities to the DAML-S matching engine, which selects those Web services whose advertised capabilities match the request. The matching is complicated by the fact that providers and requesters have different views on Web-service functionality. Thus, the
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Matching engine can’t base the selection on strings or keywords. Rather, it must match semantic descriptions of capabilities to access the deeper meaning of the advertisements and requests. Consider, for example, a service provider advertising that it sells pet food, and a requester looking to buy dog food. A UDDI-style registry would be unable to match the request because keyword matching is not powerful enough to identify the relationship between pet food and dog food. Instead, DAML-S profiles let service providers express concepts that are explicitly related via ontologies. In this case, the provider could specify that dog is a type of pet, and the DAML-S matching engine could recognize a semantic match between the request and the advertisement. The DAML-S matching algorithm accommodates the differences between an advertisement and a request by producing flexible matches — recognizing degrees of similarity — on the basis of available ontologies. Basically, the matching engine attempts to verify whether the requested outputs are a subset of those generated by the advertisement, and whether the advertisement’s inputs subsume those of the request. When these conditions are satisfied, the advertised service generates the outputs that the requester expects and the requester can provide all the inputs the Web service expects. The degree of satisfaction between these two rules determines the degree of match between provider and requester.

Katia Sycara, Massimo Paolucci, Julien Soudry, and Naveen Srinivasan, “Dynamic Discovery and Coordination of Agent-Based Semantic Web Services” 2004 [87]. Matchmaking and brokering are multiagent coordination mechanisms for Web services. Both have performance trade-offs, but the Web Ontology Language for Semantic Web Services (OWL-S) can handle extensions that address some of the shortcomings. In this article, the authors focus on the broker, analyzing both its interaction protocol and reasoning tasks. The authors also describe OWL-S’s exec extensions, detail their implementation’s basic features, and explain how these features address the broker’s reasoning problems.

M. Adel Serhani, Rachida Dssouli, Abdelhakim Hafid, Houari Sahraoui, “A QoS broker based architecture for efficient web services selection” 2005 [78]. In this paper, the authors presented a QoS broker based architecture for web services. The main goal of the architecture was to support the client in selecting web services based on his/her required QoS. To achieve...
this goal, researchers proposed a two-phase verification technique that is performed by a third party broker.

The first phase consists of syntactic and semantic verification of the service interface description including the QoS parameters description. The second phase consists of applying a measurement technique to compute the QoS metrics stated in the service interface and compares their values with the claimed one. This is used to verify the conformity of a web service from the QoS point of view (QoS testing). A methodological approach to generate QoS test cases, as input to QoS verification is used. They implemented a prototype that included the verification and certification components of the broker. They performed experiments to evaluate the importance of verification and certification features in the selection process using real web services. The architecture extends the standard Service Oriented Architecture (SOA) [1] [2] with QoS support for web services. It includes QoS description during the service publication, and performed dynamic QoS-aware invocations. In addition, it verified, certified, confirmed and monitored QoS dynamically via a web service-based broker. The architecture involves four main participating roles the web service broker, the web service provider, the
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

client, in addition to a QoS enabled UDDI registry [15].

Figure 2.10 presents an architecture based broker with features such as support of service selection based on client requirement, QoS verification and certification. QoS verification is the process of validating the correctness of information described in the service interface as well as the described QoS parameters. The QoS verification is performed using an approach that generates test cases to measure QoS parameters. The verification will be used as input for the certification process that will be issued when the verification succeed. The broker arbitrates the negotiation process between clients and their providers until they reach an agreement. During web service invocation, the broker measures dynamically QoS attributes and uses their values to monitor the provision of the selected QoS level; then, it notifies the interested entities of any violation. The broker updates, regularly, its database whenever significant changes happen. In the architecture, the certification process goes beyond certifying just the QoS provider’s claims.

Wenli Dong “QoS Driven Service Discovery Method Based on Extended UDDI”, 2007 [99]. In this paper the author proposed a QoS driven service discovery method based on extended UDDI with the help of Semantic Web. First, a Extending UDDI Model based on QoS driven was proposed, QoS ontology was analyzed to reduce misunderstanding. Second, a matching algorithm based on fuzzy correlation calculate was proposed to filter the unqualified service to improve the discovery accuracy. Third, a discovery process based on policy was built based on Semantic Web technology. The experience results showed that the QoS driven Web service discovery method possessed high discovery accuracy.

The QoS certifier was added in the proposed extended UDDI model to support QoS filtering function as shown in Figure 2.11. The QoS certifier’s role is to verify service provider’s QoS claims. According to the author the proposed registry differed from the current UDDI model by having information of the function description of the Web service as well as its associated quality of service registered in the registry repository. Lookup could be made by function description of the desired Web service, with the required quality of service attributes as lookup constraints. QoS is a combination of several non-functional characteristics. QoS publication helps selecting among services with the same functionality based on OoS. There are many aspects of QoS that are important to Web services.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 2.11 Architecture of Web Service Publication and Discovery with QoS certifier

Huimin HE, Haiyan DU, Dongxia HAN, Yuemei HE, “Research on the Models to Customize Private UDDI Registry Query Results” 2008 [37]. This paper presents three models which enable the customization of Universal Description, Discovery and Integration (UDDI) query results, based on some pre-defined and/or real-time changing parameters. These proposed models detail the requirements, design and techniques which make ranking of Web service discovery results from a service registry possible. They present an extension to the UDDI inquiry capabilities to customize or rank the query results, based on business requirements. Authors proposes three models to achieve the customization of UDDI query results. All three share some common architecture components as shown in Fig.2.12.

Fig 2.12 Common Architecture components of the Models to Customize Private UDDI Registry Query Results

Research study by Netra Patil
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

They are: UDDI server, UDDI Proxy and User Interface. These components will interact with other external components. The customization criteria required is the ranking of list of business or service list to User Interface. Load balancing also can be improved by keeping the User Interface and UDDI Proxy on separate servers. The most basic feature of UDDI is to allow businesses to publish their services in a directory and enable other business representatives to locate partners and to form business relationships based on the web services they provide. They introduce two types of parameter: static and dynamic. The static parameter will hold certain values which has been fixed and do not change during run-time. Only Administrator access can modify its values. Examples of static parameter are vendor ranking, cost per transaction and advertisement priority. Unlike static, the dynamic parameter will be used to store value which is real-time changing and gets updated during run-time. The updating frequency will depend on mechanism defined within the criteria. One usage of dynamic parameter is to keep track of service or business popularity. The criteria used to customize the UDDI query results will be represented by static and dynamic parameters.

Model where parameters are saved and retrieved from UDDI server

In this first model, we propose the use of only UDDI Proxy and UDDI Server components, where the parameters will be saved inside the UDDI server itself.

![Diagram](image)

**Figure 2.13 Model 1 - Parameter values to be saved and retrieved from UDDI server**

This will require a new tModel definition to describe the parameters information. Each business entity and service will then contain a reference to this tModel in their record. The term “bag” indicates a generic container of multiple values, and enables a company to register multiple business identifiers. i. Retrieving Parameters Values In this model, all the parameter values are stored using XML schema inside the UDDI server. Whenever a request is made by consumer to get a list of services, the UDDI Proxy will invoke the UDDI Find functions of the inquiry API. Certain Find Qualifiers can also be used to enable more precise search criteria. Let us take an example of mobile user who requests for online stock quote service. All static and dynamic parameters related to the services are embedded in the list. This is very important
as the UDDI Proxy will use some of this parameter values as ranking criteria. Based on the criteria preferences defined by administrator, if the ranking feature is enabled, the UDDI Proxy will further process the list accordingly, using the embedded parameters values. Once processing is done, the new list which contains ranked and sorted services will be sent to user interface, all the parameters values will be discarded. ii. Saving Parameters Values Saving of parameters values to UDDI Server will be handled by the UDDI Proxy using the Save functions of the UDDI publishing API. For static parameters, its values can be edited only by the administrator. This can be achieved by having UDDI Proxy to display and save the parameter values directly to UDDI server. The save frequency is solely depending on the registry administrator. As for the dynamic parameters, its values will be updated each time the Proxy detect a request has been made to access the respective business or service links. If the dynamic parameter is used to store an incremental number such as vendor ranking or popularity, first the UDDI Proxy is required to read the current parameter value, increment the value by 1 before it invoke the save function. The main advantage of the first model is the criteria data are stored and bind with its associated business or service entity. This will be beneficial for private registry operator who wishes to extend UDDI capabilities to support ranking with minimal changes to his present system architecture. However, there might be certain performance issue if the Proxy accesses launch too many queries, too frequently to the UDDI server.

Model Where Parameters are retrieved from Server Logs

A private registry system normally consists of several application and server components. A typical UDDI server is often hosted together with application server and SOAP server or being part of a integrated solution package. As with the UDDI server, these servers do provide cross-language logging services for purposes of application debugging and auditing. Web service log data could provide information such as Web service usage, supporting information concerning business transaction and quality of service. These logs data could provide useful semantic information for ranking criteria. Fig.2.14 shows the components and data flow of this second model. Note this model does not support the retrieving or saving of static parameters.
Figure 2.14 Model 2 – Parameter values to be retrieved from logs data

Retrieving Dynamic Parameter Values In this second model, we propose the creating of dynamic parameter values by extracting and processing the data from log files of SOAP server, application server and UDDI server. A function used to search, match and count for each parameter type is required within the UDDI Proxy.

ii. Saving Dynamic Parameter Values
Since dynamic parameters values are extracted from the log files and the log processing is handled by the respective server logging services, there will be no saving mechanism introduced here. The only important requirement is to ensure all the servers logging service are turned on, or to the minimum level where UDDI will be created within the logs. The main advantage of the second model is the criteria data can be automatically generated from the server logs. This will simplify implementation procedures and ensure data received are the most recent. Registry administrator who does not require static parameters for their criteria will find this model suitable for their need. Besides, this model can be further extended to monitor the health of registry servers as described in.

Model Where Parameters are saved and Retrieved from External File

In this model, researcher proposes keeping both parameter values in external files, one file for each parameter type. As shown in Fig. 2.15, the files should be accessible directly from the Proxy, outside the UDDI server. The flat ASCII file can either be in pipe-delimited or even XML format. File A is used to store values for static parameters and it can be modified by administrator only. File B is used to store values for dynamic parameters and gets updated by certain functions within the UDDI Proxy.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 2.15 Model 3 - Parameter values to be saved and retrieved from external files

Unlike the first model where saving of parameter values will be added to existing UDDI record based on XML schema, this model will have its own data structure to store business/service parameters values. The third model introduces distributed storage of the parameters data; it has the advantages of lowering the UDDI Server load, and gives administrator more control over the external files. However, with more control available at the administrator interface, the UDDI Proxy will have to provide more complex functions to support these requirements and file handling processing. This model will best suite registry operator who has long list of criteria parameters, require full control of the parameters data, and has to generate complex criteria on the registry query results.

Claudia Diamantini, Domenico Potena, Jessica Cellini, “UDDI registry for Knowledge Discovery in Databases services” 2007 [18]. In this paper the authors discussed the design and implementation of the UDDI service broker, a core element of the platform. They analyze the information needed to describe a tool in our platform, showing limitations of the present UDDI standard. Then, they present the solution to overcome such limitations and to extend UDDI broker capabilities In this paper, they discuss how to extend the UDDI registry in order to manage information needed to describe a service in the KDDVM platform, focusing on the description of KDD tools. UDDI specifications define two ways to add new information into a registry. One possibility is to define a tModel in order to address, by the overviewDoc field, WSDL description. In this way, WSDL and UDDI work together for web services advertisement. As a matter of fact, a WSDL document defines how to invoke a service. It provides information on the data being exchanged, the sequence of messages for an operation, the location of the service and the description of bindings (e.g. SOAP or HTTP). The other way
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

is to use a category-\-Bag to classify services based on their functionalities. A categoryBag is a collection of `keyedReference` structures. Each keyedReference provides a `<name,value>` pair, that assumes values in a particular domain described by the related tModel.

**Eyhab Al-Masri and Qusay H. Mahmoud**, “*Toward quality driven web Service Discovery*” 2008 [3]. In this paper, the authors provide quality-driven discovery using our *Web service broker* (WSB), shown in Figure. In the WSB model, service providers publish service information in the UDDI or search engines. The WSB collects Web services disseminated throughout the Web and continuously monitors their behavior based on various QWS metrics. WSB requires no human intervention because it performs these functions automatically. Service providers can also submit their Web services to the WSB. The WSB interface lets clients articulate proper service queries based on QWS. When clients receive response messages, they can invoke services. To assess a particular Web service’s quality, the service must contain at least one accessible operation - that is, it must have a valid service end point. However, a Web service might contain one or more operations but the service end point is inaccessible, so the service can’t be monitored or considered serviceable. WSB performs a series of tests to determine a collected Web service’s serviceability.

![Figure 2.16 High-level architecture of WSB model.](image)

The WSB automatically collects Web services disseminated throughout the Web and monitors their behavior using various QWS metrics. Clients use the WSB interface to enter QWS-based queries. For example, a Web service interface might contain two or more operations, but the actual service end point to invoke these operations requires authentication or contains an
invalid location. This makes achieving trialability impossible. Therefore, amplifying Web services prior to any QWS monitoring can ensure their trialability and that they’re serviceable.

Eyhab Al-Masri and Qusay H. Mahmoud, “Investigating Web Services on the World Wide Web” 2008 [21]. In this work, the authors conduct a thorough analytical investigation on the plurality of Web service interfaces that exist on the Web today. Using their Web Service Crawler Engine (WSCE), we collect metadata service information on retrieved interfaces through accessible UBRs, service portals and search engines. This data can be used to determine Web service statistics and distribution based on object sizes, types of technologies employed, and the number of functioning services. This statistical data can be used to help determine the current status of Web services. UDDI Business Registries (UBRs) UBRs are used for publishing and discovering Web services into registries. There are several key UBRs that currently exist and were used for this method including: Microsoft, XMethods, SAP, National Biological Information Infrastructure (NBII), among others. Web-based crawling involves using an existing search engine API to discover WSDL files across the Web such as Google and Yahoo search APIs. Using this method, a crawler engine can continuously parse search results from an existing search engine when looking for Web services throughout their indices. This involves the use of search engine specific features to collect Web service information. For example, Google Search API provides a way to search for files with any extension such as WSDL, DISCO, or WSIL. There were several key search engines indices that were used for crawling these types of service resource including: Google, Yahoo, AlltheWeb, and Baidu. The crawling tools consist of a verifier, validator, and metadata collector. A Web service is passed to the WSCE crawler tools after a resource is examined. Crawlers are used to build the backend index for search engines by following links from one page to another. However, Web service crawling is relatively distinctive from Web page crawling.
**Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism**

Eyhab Al-Masri and Qusay H. Mahmoud “Discovering the Best Web Service” 2007 [20]. This work introduces the Web Service Relevancy Function (WsRF) used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences. The main focus of their approach is to design an intelligent system that has the potential of examining web service’s QoS properties in an open and transparent manner, and enabling clients to select the best available web service by taking advantage of client QoS preferences, Web service capabilities, and service provider features. This is achieved through the WS-QoSMAn service broker. The architecture of the proposed WS-QoSMAn solution is shown on

![Figure 2.17 Architecture based on WS-QoSMAn service broker](image)

QoSMetrics uses overviewURL to point to an XML-based file generated by WS-QoSMAn and that contains QoS metrics for a specific Web service. WsRF is used to measure the relevancy ranking of a particular Web service wsi. Clients can submit their requests to WS-QoSMAn (i.e. via a GUI) which will process these requests and compute WsRF values for all available services related to search query. A Web service with the highest calculated WsRF value is the most desirable and relevant to the client based on his/her preferences. In order to calculate WsRF(wsi), we need the maximum normalized value for each set of QoS parameters.

Ivan Magdalenic, Ivo Pejakovic, Zoran Skocir, Mihaela Sokic, Marina Simunic, “Modeling ebXML Registry Service Architecture” 2003 [55]. In this paper, the authors have modeled
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

ebXML Registry/Repository architecture and concentrates on its Registry Service part. In their implementation they have made many improvements over the existing open-source implementation. They made the ebXML Registry Services server distributed which is very important knowing that processing of the business documents is highly resource demanding. Also the system of Registry Services can share same registry items and metadata about them.

Stefan Schulte, Melanie Siebenhaar, and Ralf Steinmetz. “Integrating Semantic Web Services and Matchmaking into ebXML Registry” 2010 [76]. In this paper, the authors presented a solution extending the ebXML Registry by capabilities to handle and provide SWS. This includes a concept for the integration of SWS into ebXML Registry as well as a prototypical implementation using SAWSDL and the open source framework freebXML. They have proposed a quite lightweight interface for matchmakers. The interface is based on the assumption that service requests are formulated using a “query by example” approach.

Summarizing above papers, UDDI and ebXML have many things in common and can complement each other. Both technologies provide solutions to integration problems, both use XML over Internet for Message interchange, and both approaches share a common high-level architecture. Observing the e-Business world reveals the evolution from tactical systems with limited scope to strategic e-Business initiatives. This does not mean, however, that UDDI will soon be abolished and replaced by ebXML. UDDI is a well established and widely adopted standard. A multitude of experienced developers use the numerous available libraries and frameworks to guarantee short time to market for their products. In addition to those strengths, the UDDI domain is much broader than that of ebXML and its architecture is simpler and easier to handle. As a successor of other middleware technologies, UDDI excel in intra-enterprise request/response type application integration environments. The major drawbacks of ebXML are that the specification is not entirely complete and that industry support is still lacking. If industry fails to provide affordable implementations of ebXML, this standard might follow the destiny of EDIFACT, which was not widely adopted due largely to its cost. Since ebXML is powerful, implementations are likely to be complex and might not be easy to handle. Templates for the most common demands of companies might help to decrease the time-to-market for system providers that use ebXML implementations. While ebXML is always intended for e-Business, UDDI is a bottom-up technology that focuses on the technical aspects of middleware functionality. However, for many in-house projects companies do not need full
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

grown e-Business suites. Instead, they need smaller, more reliable, and easier to handle technologies that have reached a sufficient level of maturity.

**Summarizing above discussion**, it can be concluded that the two emerging standards which could have very well impact on the way of conducting e-business in future are UDDI (Universal Description, Discovery and Integration) and ebXML (electronic business XML). UDDI is a vendor-sponsored initiative led by IBM, Microsoft, and Ariba, whereas, ebXML is a UN/CEFACT (United Nations center for Trade Facilitation and Electronic Business) / OASIS sponsored initiative for creating a single global electronic market. UDDI and ebXML, make it possible for business organizations to publish information on the Internet about their products and web services, where the information can be readily and globally accessed by clients who want to do business. Using UDDI based mechanism, WSCE collect metadata service information on retrieved interfaces through accessible UBRs, service portals and search engines. Broker based mechanisms allow user to specify the functional requirement and QoS parameter values for searching the services. For semantic web service discovery, DAML-S can be used. DAML-S uses semantic annotations and ontologies to relate each web service’s description to a description of its operational domain. For example, a DAML-S description of a stock-reporting service might specify the data it reports, its delay versus the market, and the cost of using the service.

**2.2.3 Alternative mechanisms**

**2.2.3.1 Federated Registry**

*Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth*, “*Discovery of Web Services in a Federated Registry Environment*” 2004 [85]. They have presented the 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 federation or domains. An ontology based approach is used to classify registries and locate them based on the user requirements. They have also presented the way in which web service discovery is carried out within a federation. In their initial, naïve implementation registries could only be categorized based on business domains. Extended Registries ontology (XTRO), represented in OWL, is a comprehensive ontology containing details of Domains, Registries, Ontologies and Registry.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Federation and network of relationships among them. All the classes and few important object properties in XTRO are shown in Fig. 2.18.

Abraham Bernstein, Mark Klein, “Discovering services: Towards High-Precision Service Retrieval” 2004 [52]. They described a novel service retrieval approach based on the sophisticated use of process ontologies. They claim that this approach offers qualitatively higher retrieval precision than existing (keyword and table based) approaches without sacrificing recall and computational scalability. In this approach, the salient behavior of a service is captured using process models, and these process models, as well as their components (subtasks, resources, etc.), are placed in the appropriate locations in the process ontology. Queries can then be defined (using a process query language – PQL) to find all the services whose process models include a given set of entities and relationships. The greater expressiveness of process models, as compared to keywords or tables, offers the potential for substantively increased retrieval precision, at the cost of requiring that services be modeled in this more formal way. This process-based approach offers qualitatively increased retrieval precision, and beside this it can be achieved with a reasonable expenditure of service modeling effort. The approach has the functional architecture shown in Figure.

Figure 2.18 Classes and their Relationships in XTRO

Abraham Bernstein, Mark Klein, “Discovering services: Towards High-Precision Service Retrieval” 2004 [52]. They described a novel service retrieval approach based on the sophisticated use of process ontologies. They claim that this approach offers qualitatively higher retrieval precision than existing (keyword and table based) approaches without sacrificing recall and computational scalability. In this approach, the salient behavior of a service is captured using process models, and these process models, as well as their components (subtasks, resources, etc.), are placed in the appropriate locations in the process ontology. Queries can then be defined (using a process query language – PQL) to find all the services whose process models include a given set of entities and relationships. The greater expressiveness of process models, as compared to keywords or tables, offers the potential for substantively increased retrieval precision, at the cost of requiring that services be modeled in this more formal way. This process-based approach offers qualitatively increased retrieval precision, and beside this it can be achieved with a reasonable expenditure of service modeling effort. The approach has the functional architecture shown in Figure.

Figure 2.19 Service retrieval approach based on process ontology
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

**Jorge Cardoso and Amit Shet**, “Semantic e-Workflow Composition” 2002 [47]. They have presented a methodology and a set of algorithms for web service discovery based on three dimensions: syntax, operational metrics, and semantics. This approach allows for web service discovery not only based on functional requirements, but also on operational metrics.

![Multidimensional approach to Web Service Discovery and Integration](image)

**Figure 2.20 Multidimensional approach to Web Service Discovery and Integration**

**Jinghai Rao, Dimitar Dimitrov, Paul Hofmann and Norman Sadehw**, “A Mixed Initiative Approach to Semantic Web Service Discovery and Composition: SAP’s Guided Procedures Framework” 2006 [45]. They described a mixed initiative framework for semantic web service discovery and composition that aims at flexibly interleaving human decision making and automated functionality in environments where annotations may be incomplete and even inconsistent. Fig. 2.21 depicts overall architecture of the system.

![Architecture of Semantic Web Service Discovery and Composition](image)

**Figure 2.21 Architecture of Semantic Web Service Discovery and Composition**

Research study by Netra Patil
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Patrick C. K. Hung And Haifei Li, “Web Services Discovery Based on the Trade-off between Quality and Cost of Service: A Token based Approach” 2003 [67]. They have proposed a token based approach for web services discovery based on the trade-off between Quality and Cost of Service (QoS and CoS) to quantify the QoS and CoS for achieving integrative solutions. In this model, the QoS relates to performance-oriented capabilities and the CoS relates to services’ resource requirements. To achieve an integrative solution, both parties have to evaluate the list of QoS and CoS alternatives for obtaining an appropriate combination. One of the negotiation strategies for achieving integrative solutions for both parties is called logrolling. Logrolling is an important step in web service discovery process in which both web services providers and web services requestors can find appropriate partners.

Summarizing above papers, it can be concluded that service discovery mechanism should be based on not only functionality and QoS of the service desired by the user, but also it should allow them to specify the domain to which that service belongs. Also the user should be able to evaluate tradeoffs between QoS and CoS in selecting perfect service.

2.3 Limitations of existing mechanisms

A significant amount of literature is available on web service discovery mechanism and techniques. Still, the pros and cons of these mechanisms and techniques have not been adequately studied with respect to their performance and interface.

M. Paolucci, T. Kawamura, T. Payne, and K. Sycara [2002] focused on discovering Web services through a centralized UDDI registry. Although centralized registries can provide effective methods for the discovery of Web services, they suffer from problems associated with having centralized systems such as a single point of failure, and bottlenecks. In addition, other issues relating to the scalability of data replication, providing notifications to all subscribers when performing any system upgrades, and handling versioning of services from the same provider have driven researchers to find other alternatives.

Jorge Cardoso and Amit Sheth [2002] presented a methodology and a set of algorithms for Web service discovery based on three dimensions: syntax, operational metrics, and semantics.
This approach allows for Web service discovery not only based on functional requirements, but also on operational metrics.

Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl [2002] proposed a graph topology which allows for very efficient broadcast and search, and provide an efficient topology construction and maintenance algorithm which, crucial to symmetric peer-to-peer networks, does neither require a central server nor super nodes in the network.

Bernstein, Abraham, and Mark Klein [2002] described a novel service retrieval approached based on the sophisticated use of process ontologies. This approach offers qualitatively higher retrieval precision than existing (keyword and table based) approaches without sacrificing recall and computational tractability/scalability.

Patrick C. K. Hung And Haifei Li [2003] proposed a token based approach for web services discovery based on the trade-off between Quality and Cost of Service (QoS and CoS) to quantify the QoS and CoS for achieving integrative solutions. One of the negotiation strategies for achieving integrative solutions for both parties is called logrolling. Logrolling is an important step in web service discovery process in which both web services providers and web services requestors can find appropriate partners.

D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGunneess, B. Barsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara [2004] and D. Roman, H. Lausen, and U. Keller [2004] attempted to provide a formal way of expressing service provider's capabilities and user's requirements. These initiatives are mainly focused on knowledge representation aspects. Apart from knowledge representation, the web service discovery is a complex task and need to consider the context of its availability and usability.

U. Keller, R. Lara, A. Polelres, I. Toma, M. Kifer, and D. Fensel [2004] described different levels of service matches. It is understood that service matches are mandatory but not sufficient for Web service discovery.

K. Sivashanmugam, K. Verma, and A. Sheth [2004] proposed METEOR-S Web Service Discovery Infrastructure(MWSDI), an ontology based infrastructure to provide access to private and public registries divided based on business domains and grouped into federations for enhancing the discovery process. METEOR-S provides a discovery mechanism for
publishing Web services over a federated registry sources but, similar to the centralized registry environment, it does not provide any means for advanced search techniques which are essential for locating appropriate business applications. In addition, having a federated registry environment can potentially provide inconsistent policies to be employed which will significantly have an impact on the practicability of conducting inquiries across the federated environment and can at the same time significantly affect the productiveness of discovering Web services in a real-time manner across multiple registries.

K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller [2004] presented METEOR-S Web Services Discovery Infrastructure (MWSDI), a scalable infrastructure for semantic publication and discovery of Web services. We have presented two algorithms for semantic publication and discovery using WSDL descriptions.

Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, Amr El Abbadi [2004] proposed a structured peer-to-peer framework for web service discovery in which web services are located based on both service functionality and process behavior. In addition, they integrate a scalable reputation model in this distributed peer-to-peer framework to rank web services based on both trust and service quality.

Shou-jian Yu, Xiao-kun Ge, Jing-zhou Zhang, Guo-wen Wu [2006] presented a flexible Web service discovery architecture by combining semantic Web service with P2P networks. This system does not need a central registry for Web service discovery. They use an ontology-based approach to capture real world knowledge for semantic service annotation.

Eyhab Al-Masri and Qusay H. Mahmoud [2007] proposed a solution by introducing the Web Service Relevancy Function (WsRF) used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences for the purpose of finding the best available Web service during Web services’ discovery process based on a set of given client QoS preferences or QoS search criteria.

Gang Zhou, Jianjun Yu, Rui Chen, Hui Zhang [2007] proposed a peer-to-peer framework, which adopts an enhanced Skip Graph named ServiceIndex as the overlay network for service discovery. To guarantee discovery efficiency, ServiceIndex schemed WSDL-S (Web Services Semantics) as Semantic Web Services description language and extracted its semantic attributes as indexing keys in Skip Graph.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Jacek Kopecký [2007] intended to research an approach to SWS offer discovery that will significantly simplify the needed semantic descriptions and thus help ease the adoption of SWS technologies in the industry.

Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, Hai Jin [2008] proposed a peer-to-peer based decentralized service discovery approach named Chord4S. To improve data availability, Chord4S distributes the descriptions of functionally-equivalent services. An efficient routing algorithm is provided to facilitate queries of multiple candidate service providers.

Eyhab Al-Masri and Qusay H. Mahmoud [2008] proposed Web Service Crawler Engine (WSCE), a crawler that is capable of capturing service information from various accessible resources over the Web, to help in conducting investigation of Web services on the Web.

Shuiguang Deng, Zhaohui Wu, Jian Wu and Ying Li [2008] proposed a two-phase semantic-based service discovery mechanism to discover services in an accurate, efficient and automatic way. Compared to other approaches, the new method has two salient characteristics: (a) it takes into account the interface dependencies implied within an operation while performing matchmaking; (b) it supports two-level matchmaking, namely operation matchmaking and operation-composition matchmaking.

2.4 The Pilot study

Prior to the main research work, a pilot survey study was conducted in which a questionnaire was filled up by the around 220 service consumers from different groups of people like students, teachers, homemakers, software engineers etc. Sampling technique used for conducting the pilot study was Convenience Sampling and Purposive Sampling. Out of 220 consumers, 20 were service engineers who need to discover the service over the web for integrating it in their applications for some system related tasks whereas 197 were the direct users of the service who utilizes online services either for shopping, booking, bank transactions, bill payments etc. for their own purpose. Among 200 people who filled up the pilot study survey questionnaire, 3 people had never used any web service. Out of 200 customers, 89 were satisfied, 81 were not satisfied and 30 can’t say anything about the online services use available over the internet.
It is very obvious that a user’s perception varies from person to person. However, based on the experiences as an end user who uses online services available over the internet, following result is inferred on how they perceive the quality of service as compared to their expectation.

<table>
<thead>
<tr>
<th>Service Domain and No. of Customers</th>
<th>No. of Customers Satisfied with</th>
<th>Overall all Satisfied Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping (68)</td>
<td>Response Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>Booking (57)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Bank Transactions (44)</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>Bill Payments (28)</td>
<td>14</td>
<td>28</td>
</tr>
</tbody>
</table>

**Table 2.1 : Relation between Good Response Time and Satisfied Customer**

From Table 2.1, the computed correlation coefficient (0.99) is positive and significant. Hence there is a strong relationship between Good Response Time and Satisfied Customer.

<table>
<thead>
<tr>
<th>Service Domain and No. of Customers</th>
<th>No. of Customers Satisfied with</th>
<th>Overall all Satisfied Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping (68)</td>
<td>Reliability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>31</td>
</tr>
<tr>
<td>Booking (57)</td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td>Bank Transactions (44)</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Bill Payments (28)</td>
<td>26</td>
<td>28</td>
</tr>
</tbody>
</table>

**Table 2.2 : Relation between High Reliability and Satisfied Customer**

From Table 2.2, the computed correlation coefficient (0.56) is positive. Hence there is a good relationship between High Reliability and Satisfied Customer.

<table>
<thead>
<tr>
<th>Service Domain and No. of Customers</th>
<th>No. of Customers Satisfied with</th>
<th>Overall all Satisfied Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping (68)</td>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>31</td>
</tr>
<tr>
<td>Booking (57)</td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>Bank Transactions (44)</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>Bill Payments (28)</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

**Table 2.3 : Relation between High Availability and Satisfied Customer**

From Table 2.3, the computed correlation coefficient (0.76) is positive and significant. Hence there is a strong relationship between High Availability and Satisfied Customer.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

<table>
<thead>
<tr>
<th>Service Domain</th>
<th>No. of Customers</th>
<th>Satisfied with Response Time</th>
<th>No. of Customers</th>
<th>Satisfied with Reliability</th>
<th>No. of Customers</th>
<th>Satisfied with Availability</th>
<th>No. of Customers</th>
<th>Satisfied with Price</th>
<th>Overall Satisfied Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping (68)</td>
<td>16</td>
<td>66</td>
<td>60</td>
<td>45</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Booking (57)</td>
<td>7</td>
<td>28</td>
<td>18</td>
<td>21</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bank Transactions (44)</td>
<td>11</td>
<td>40</td>
<td>30</td>
<td>39</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bill Payments (28)</td>
<td>14</td>
<td>26</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4 : Relation between Good Price and Happy Customer

From Table 2.4, the computed correlation coefficient (0.66) is positive and significant. Hence there is a good relationship between Good Price and Satisfied Customer.

<table>
<thead>
<tr>
<th>Service Domain</th>
<th>No. of</th>
<th>Satisfied with Response Time</th>
<th>No. of</th>
<th>Satisfied with Reliability</th>
<th>No. of</th>
<th>Satisfied with Availability</th>
<th>No. of</th>
<th>Satisfied with Price</th>
<th>Overall Satisfied Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping (68)</td>
<td>16</td>
<td>66</td>
<td>60</td>
<td>45</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Booking (57)</td>
<td>7</td>
<td>28</td>
<td>18</td>
<td>21</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bank Transactions (44)</td>
<td>11</td>
<td>40</td>
<td>30</td>
<td>39</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bill Payments (28)</td>
<td>14</td>
<td>26</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.5 : Customers in different service domain Satisfied with different QoS

Figure 2.22 : Customers in different service domain satisfied with different QoS

In Figure 2.22, a graph of number of customers belonging to different service domain satisfied with different QoS parameter values shows that, customers are not satisfied only with the service functionality but they also want a desired level of QoS parameter value. E.g. for the
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

customers performing bank transactions online, reliability may be the highest priority QoS parameter than any other parameter and so on.

Based on consumer’s experiences as a service engineer who needs to find appropriate web services available over the internet, while designing and developing software applications, following result is inferred on how the quality of service affects the business application in terms of complaints received from the customers.

<table>
<thead>
<tr>
<th>Complaint type</th>
<th>Often</th>
<th>Sometimes</th>
<th>Rarely</th>
<th>Never</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow response</td>
<td>13</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Service temporarily unavailable</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Transaction not completed successfully</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Costing Issue</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.6 : Service Engineers receiving complaints about service

In Figure 2.23, it is observed that more number of service engineers are facing the complaints from the customers for ‘Slow response’, ‘Service temporarily unavailable’ and ‘Transaction not completed successfully’. Many service engineers also fill that the cost of integrating a web service in an application should not be more than developing the whole application on their side. Out of 20 engineers, not a single engineer is having zero complaints about service. QoS parameters have become equally important for the customers along with the service functionality.

Research study by Netra Patil
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

The pilot study survey result obtained as above signifies the study of existing web service discovery mechanisms and proposing an efficient mechanism which should be able to discover the most appropriate web service as per the consumer’s requirement of functionality as well as quality of service (QoS) and the priority of QoS.

2.5 The present study

The present research study has the objective to identify and evaluate the significant web service discovery technique which will be efficient in discovering the most appropriate web service according to the consumer’s requirement of functionality as well as quality.

In the present research study, the approach is to return maximum number of relevant web services of desired functionality and quality efficiently and rank them according to users’ preference of selecting his choice of QoS.

Thus, the study suggests to evaluate other implementations of algorithms for matching, ranking and selecting web services efficiently. The evaluation of performance is done on various parameters identified under the environmental limitations.

A new tool having a great interface for specifying service requirement, choosing right QoS values, and setting preference of QoS for ranking the services is also proposed. Algorithms are proposed for matching, ranking and selecting the web services.