

Multithreaded Fuzzy Logic based Web Services Mining Framework

A dissertation Presented by

Khurram Shehzad

(2007-NUST-MS PhD-CSE(E)-26)



Submitted to the Department of Computer Engineering in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Software Engineering

Advisor

Brig. Dr. Muhammad Younus Javed

College of Electrical & Mechanical Engineering

National University of Sciences and Technology

2010

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

THE COMMITTEE

Multithreaded Fuzzy Logic based Web Services Mining Framework

A dissertation Presented by

Khurram Shehzad

Style and content approved by

Brig. Dr. Muhammad Younus Javed *Supervisor*

Dr. Assia Khanam *GEC Member*

Dr. Farooq-e-Azam *GEC Member*

Dr. Saad Rehman *GEC Member*

Brig. Dr. Muhammad Younus Javed

Associate Dean

Department of Computer Engineering

DEDICATION

To my parents and family

ACKNOWLEDGEMENT

First of all, I am extremely thankful to Almighty Allah for giving me courage and strength to complete this challenging task and to compete with international research community. I am also grateful to my family, especially my parents who earnestly prayed for my success, which enabled me to brave the difficulties and overcome the crisis which I came across during the whole process..

With a deep sense of gratitude I thank **Dr. Muhammad Younus Javed** for his valuable suggestions and continuous guidance throughout my research work. I am highly grateful to **Dr Aasia Khanam, Dr. Farooq e Azam Khan** and **Dr Saad Rehman** for their help and guidance. I am also thankful to teachers who have been guiding me throughout my course work and contributed in the enhancement of my knowledge and technical know how. Their knowledge, guidance and training helped me a lot in the fulfillment of this research work.

I am thankful to my parents and family who earnestly prayed for my success which enabled me to brave the difficulties and overcome the crisis which I came across during the whole process.

ABSTRACT

Multithreaded Fuzzy Logic based Web Services Mining Framework

Finding valuable and attractive web services is becoming difficult, due to massive number of web services. Requirement of web services mining like data mining is vital these days. Comparative study of web services composition with mining concepts is presented in this report. A web services mining frame work, based on fuzzy logic, fuzzy set theory and fuzzy matching algorithm is proposed. This framework helps in finding valuable services and composing those services into composite web services. Mined services are further filtered in the rules matching and evaluation phases where specified rules are matched. Framework is tested with different UDDI registries of large sizes and the results are compared with existing techniques.

The proposed model is divided into different steps and phases, to reduce the model complexity and simplify different integrating processes. The problems, faced in mining process are complexity of the search space and pattern matching. The complexity model is targeted by introducing the concept of threading for parallel processing. A new thread is initiated for every member of fuzzy set and mines the search space for required computation. This parallel processing approach helps in optimizing the search and matching process and for efficient discovery of individual web services and composition of web services.

The first step in proposed framework is scope and rules specifications. Scope and the rules are specified by a web service domain expert and these are according to required mining results. For example, domain expert is looking for web services, related to traveling or in the field of medicine. Rules specified by the domain expert will be matched in constraint satisfaction and evaluation phases for filtering and validating of found web services and their compositions. Based on the scope, specified by the domain expert and weights, fuzzy set is generated and accordingly assigned to each number of fuzzy set. Weights are calculated based on the probability model and with the help of local database. This local database is used to store members of fuzzy set and helps in calculating weights. Every member of the fuzzy set is used as

input to the next searching phase and after the phase of assigning weights, a new thread is initiated for every member of the fuzzy set. Based on the fuzzy matching algorithm, this thread explores the UDDI registry and looks for relevant services. Outputs of the found web services are further used for discovery of web services, which are using these as their input parameters for composing individual web services into composite web services.

Web service mining results sorted in the indexing phase based on weights assigned and these sorted results are filtered in the rules satisfaction phase, where constraint specified in the first phase are matched with the publisher's constraint. Publisher specifies any service relevant constraint in the web service description document and at this step of our proposed model, these rules are satisfied for filtering and validation of found results. These filtered results are used as input to evaluation phase where these results are gone through objective and subjective evaluation.

The performance of the proposed approach is evaluated using different factors like precision, recall and f-measure. Framework is tested for web services mining and the values for precision, recall and f-measure are calculated. Also, these values compared with the existing frameworks shown where proposed framework has improved the web services mining. After discovery, the services are available for composition. Mining time for UDDI registries of different sizes is recorded. At the end, comparison is given with an existing technique to present the improvements of proposed framework.

Table of Contents

THE COMMITTEE	i
DEDICATION.....	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT.....	iv
List of Abbreviations	x
List of Figures.....	xi
List of Tables	xii
Chapter 1 : Introduction	1
1.1 Web Services.....	1
1.1.1 Web Service Architecture.....	1
1.1.2 Web Services Standards.....	2
1.1.3 Engaging a Web Service.....	2
1.1.4 Web Service Invocation.....	3
1.1.5 Web Services Addressing	5
1.2 Web Services Discovery	5
1.2.1 Web Services Discovery Approaches.....	6
1.3 Web Services Composition	7
1.3.1 Web Services Composition Steps.....	8
1.3.2 Web Services Composition Classification.....	8
1.3.3 Web Services Composition Drawbacks.....	9
1.4 Web Services Mining.....	9
1.4.1 Web Services Composition versus Web Services Mining.....	9
1.4.2 Web Services Mining Issues.....	11
1.5 UDDI.....	12

1.5.1	How UDDI is used.....	13
1.5.2	UDDI Architecture.....	13
1.6	Motivation	15
1.7	Background	16
1.8	Problem Statement	17
1.9	Problem Solution.....	18
1.10	Organization.....	19
Chapter 2 : Related Work.....		20
2.1	Web Services Architecture.....	20
2.2	Service Mining on the Web.....	20
2.3	Service Pattern Discovery of Web Service Mining	22
2.3.1	Service mining in service registry-repository.....	22
2.3.2	System Architecture	23
2.4	An Improved Way to Facilitate Composition-Oriented Semantic Service Discovery...	24
2.5	Service Mining for Web Service Composition	26
Chapter 3 : Proposed Approach.....		28
3.1	Proposed Web Services Mining Framework.....	28
3.1.1	Context and Rules Specification.....	29
3.1.2	Fuzzy Set Generation.....	30
3.1.3	Weights Calculation and Assignment.....	30
3.1.4	Fuzzy Rules.....	30
3.1.5	Multithreaded Model for Mining Services and Fuzzy Matching.....	31
3.1.6	Constraint Satisfaction	33
3.1.7	Evaluation	35
Chapter 4 : System Design		36

4.1	System Flow Chart	36
4.2	Sequence Diagram.....	36
4.3	Use Cases	38
4.4	Use Case Diagram.....	44
Chapter 5 : Implementation.....		46
5.1	UDDI Server (Apache jUDDI)	46
5.1.1	Setting up UDDI Server.....	46
5.1.2	jUDDI Server Implementation Details	48
5.1.3	jUDDI Data Structures.....	48
5.1.4	Handling Publication Requests with jUDDI.....	49
5.1.5	Handling Inquiry Requests with jUDDI.....	51
5.1.6	Handling Authentication Requests with jUDDI	51
5.2	RUDDI	51
5.2.1	Ruddi Characteristics	51
5.2.2	Ruddi Usage.....	53
5.2.2.1	Querying an UDDI registry	53
5.2.2.2	Saving and updating information in an UDDI registry.....	53
5.2.2.3	Suppressing information from an UDDI registry	53
5.2.3	Various Ruddi™ API examples.....	54
5.3	Approximate String Matching.....	54
5.3.1	Levenshtein Algorithm	54
5.3.2	Dice's coefficient Algorithm	55
5.3.3	Longest Common Subsequence Algorithm	56
5.4	WSDL4J	56
Chapter 6 : Results and Discussion.....		57

6.1	System Requirements	57
6.2	Evaluation Criteria	58
6.2.1	Number of Services Discovered and Composed	58
6.2.2	Precision.....	59
6.2.3	Recall	59
6.2.4	F-measure.....	59
6.3	Dataset.....	60
6.4	Performance Evaluation	61
6.4.1	Number of Services Discovered and Composed	61
6.4.2	Average Precision	62
6.4.3	Average Recall.....	65
6.4.4	Average F-measure	66
6.4.5	Evaluation Time of Services	69
Chapter 7 : Conclusion and Future Work.....		72
7.1	Overview of Research	72
7.2	Achievements	72
7.3	Limitations	73
7.4	Future Work	73
APPENDIX A		74
References.....		88

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BPEL4WS	Business Process Execution Language for Web Services
CSP	Constraint Satisfaction Problem
DNS	Domain Name System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
JUDDI	Java implementation of UDDI
OWL-S	Ontology Web Language Semantics
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
WS	Web Service
WSCI	Web Service Choreography Interface
WSDB	Web Service Database
WSDL	Web Service Description Language
WSDL4J	Web Service Description Language for Java

List of Figures

FIGURE 1.1: WEB SERVICE ARCHITECTURE [1].....	1
FIGURE 1.2: GENERAL PROCESS OF ENGAGING A WEB SERVICE [1]	3
FIGURE 1.3: WEB SERVICE INVOCATION [1]	4
FIGURE 1.4: WEB SERVICES COMPOSITION [9]	7
FIGURE 1.5: COMPARISON OF WEB SERVICES COMPOSITION AND MINING	11
FIGURE 1.6: THE UDDI INITIATIVE [4]	14
FIGURE 2.1: TOP-DOWN COMPOSITION VERSUS BOTTOM-UP MINING [3]	21
FIGURE 2.2: PATTERN-DISCOVERY ENABLED REGISTRY-REPOSITORY ARCHITECTURE [5].....	23
FIGURE 2.3: STRUCTURE OF INVERTED INDEXING [6]	25
FIGURE 2.4: STANDARD WEB SERVICE MODEL [9].....	26
FIGURE 2.5: WEB SERVICE MODEL EXTENDED FORM [9].....	27
FIGURE 3.1: WEB SERVICES MINING FRAMEWORK.....	29
FIGURE 3.2: FUZZY RULES FOR WEB SERVICES MINING FRAMEWORK.....	31
FIGURE 4.1: FLOW CHART OF WEB SERVICES MINING FRAMEWORK.....	37
FIGURE 4.2: SEQUENCE DIAGRAM OF WEB SERVICES MINING FRAMEWORK.....	39
FIGURE 4.3: USE CASE DIAGRAM OF WEB SERVICES MINING FRAMEWORK.....	45
FIGURE 5.1: HIGH LEVEL ARCHITECTURE.....	46
FIGURE 5.2: JUDDI WELCOME PAGE.....	47
FIGURE 5.3: REQUEST PROCESS FLOW DIAGRAM	49
FIGURE 5.4: PROCESS FLOW FOR A SAVE_BUSINESS REQUEST	50
FIGURE 5.5: LEVENSHTAIN ALGORITHM	55
FIGURE 6.1: MINING RESULTS	63
FIGURE 6.2: AVERAGE PRECISION.....	65
FIGURE 6.3: AVERAGE RECALL.....	67
FIGURE 6.4: AVERAGE F-MEASURE.....	69
FIGURE 6.5: EVALUATION TIME OF WEB SERVICES	71

List of Tables

TABLE 4.1: USE CASE FOR WEB SERVICES MINING	ERROR! BOOKMARK NOT DEFINED.
TABLE 4.2: USE CASE FOR ADDING UDDI REGISTRY	41
TABLE 4.3: USE CASE FOR EDITING UDDI REGISTRY	42
TABLE 4.4: USE CASE FOR DELETING UDDI REGISTRY	43
TABLE 6.1: SYSTEM REQUIREMENTS	57
TABLE 6.2: LEVENSHTAIN ALGORITHM PARAMETERS	58
TABLE 6.3: DICE'S COEFFICIENT ALGORITHM.....	58
TABLE 6.4: LONGEST COMMON SUBSEQUENCE ALGORITHM.....	59
TABLE 6.5: STATIC EVALUATION FACTORS FOR WEB SERVICE	60
TABLE 6.6: DYNAMIC FACTORS FOR EVALUATION OF WEB SERVICE	60
TABLE 6.7: STATISTICAL FACTORS FOR EVALUATION OF WEB SERVICE	60
TABLE 6.8: MINING RESULTS.....	62
TABLE 6.9: AVERAGE PRECISION.....	64
TABLE 6.10: AVERAGE RECALL.....	66
TABLE 6.11: AVERAGE F MEASURE.....	68
TABLE 6.12: EVALUATION TIME OF WEB SERVICES	70

Chapter 1 : Introduction

This chapter introduces the research work that has been taken in this thesis report. Motivation for this specific research task is presented in detail. Moreover, the problem statement, solution and objectives are also discussed in this chapter.

1.1 Web Services

Web Service is a software application, identified by a *URI*, whose interfaces and bindings are capable of being *defined*, *described*, and *discovered* by XML artifacts and this supports direct interactions with other software applications using XML-based messages via internet-based protocols

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (WSDL). Other systems interact with the web service in a manner, prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web related standards.

1.1.1 Web Service Architecture

Before going into details, take a closer look at the web service architecture given in Figure 1.1[1].

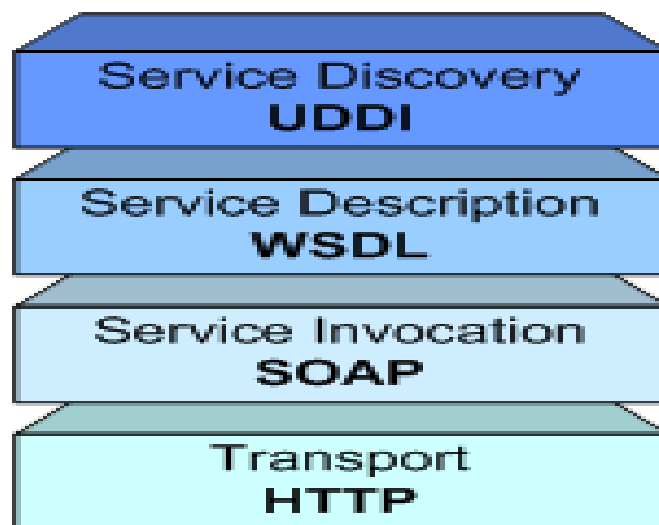


Figure 1.1: Web Service Architecture [1]

- **Service Discovery:** This part of the architecture helps in finding web services which meet certain requirements. This part is usually handled by UDDI (Universal Description, Discovery, and Integration).
- **Service Description:** One of the most interesting features of web services is that they are self describing. This means that, once a web service is located, it can be described and tells about what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).
- **Service Invocation:** Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how to format requests to the server, and how the server should format its responses. In theory, other service invocation languages can be used (such as XML-RPC, or even some *ad hoc* XML language).
- **Transport:** Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP.

1.1.2 Web Services Standards

Few web service standards are given in this section. This includes:

- BPEL4WS (a.k.a. BPEL) – Business Process Execution Language for Web Services
- IBM and Microsoft
- WSCI – Web Services Choreography Interface
- Sun, SAP, BEA, and Intalio
- BPML – Business Process Management Language
- BPMI.org (chartered by Intarlio, Sterling Commerce, Sun, CSC, and others)

1.1.3 Engaging a Web Service

There are many ways that a requester entity might engage and use a web service. In general, the following broad steps are required, as illustrated in Figure 1.2[1].

- In step 1, the requester and provider entities become known to each other (or at least one becomes known to the other).

- In step 2, the requester and provider entities somehow agree on the service description and semantics that will govern the interaction between the requester and provider agents.
- In step 3, the service description and semantics are realized by the requester and provider agents.
- In step 4, the requester and provider agents exchange messages, thus performing some task on behalf of the requester and provider entities. (I.e., the exchange of messages with the provider agent represents the concrete manifestation of interacting with the provider entity's Web service.)

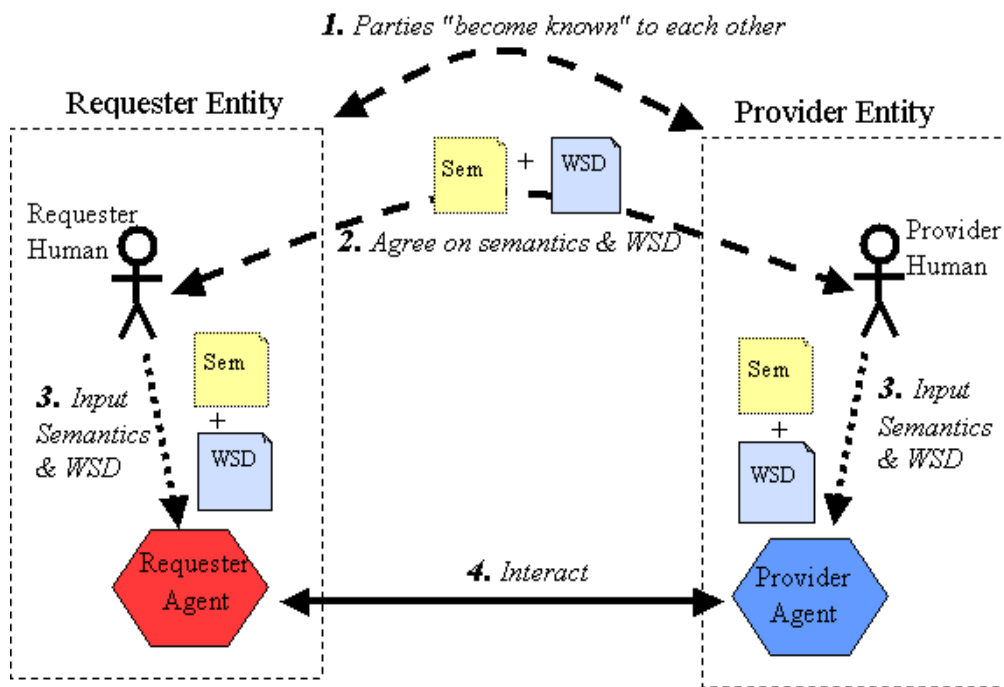


Figure 1.2: General Process of Engaging a Web Service [1]

1.1.4 Web Service Invocation

Web service invocation process is illustrated in Figure 1.3[1]. Explanation of each step involved in web service invocation process is given below.

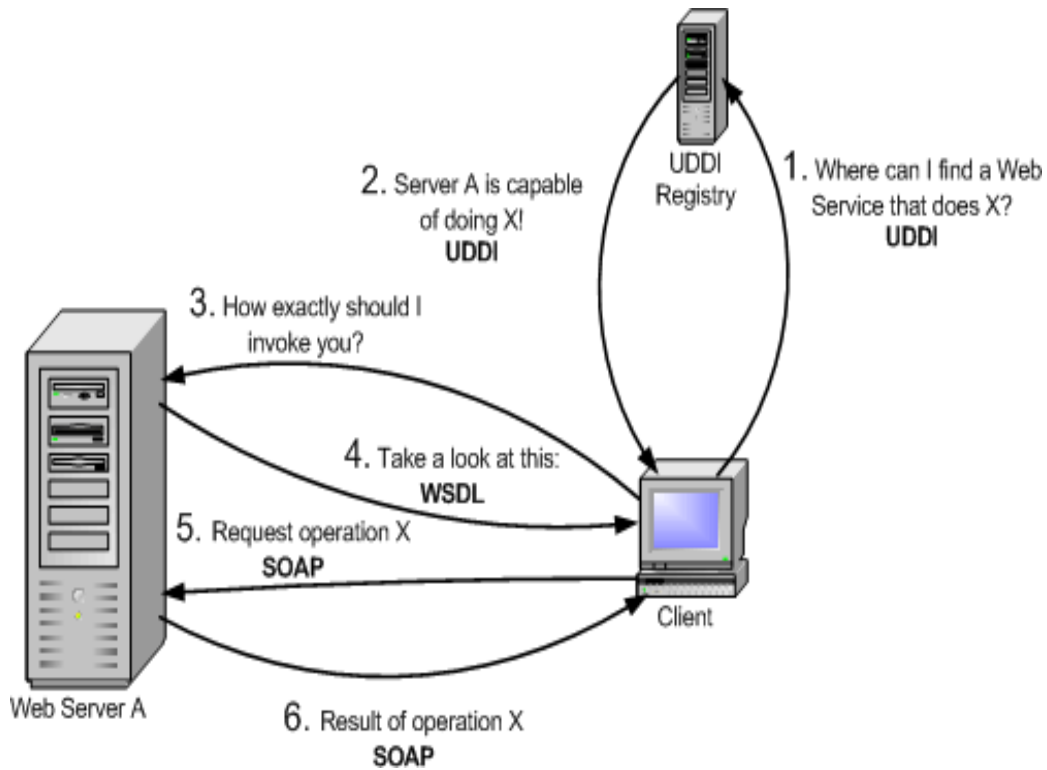


Figure 1.3: Web Service Invocation [1]

- A client may have no knowledge of what web service it is going to invoke. So, first step will be to *find* a web service that meets requirements. For example, a person might be interested in locating a public Web Service which can give the temperature in US cities. This will happen by contacting a UDDI registry.
- The UDDI registry will reply, telling what servers can provide the required service (e.g. the temperature in US cities).
- In above step, the location of a web service is known, but has no idea of how to actually invoke it. Sure, it can give the temperature of a US city but what is the actual service invocation? The method used to invoke might be called `Temperature getCityTemperature(int CityPostalCode)`, but it could also be called `int getUSCityTemp(string cityName, bool isFahrenheit)`. For this, need to ask the web service to describe itself.
- The Web Service replies in a language called WSDL.

- Finally, know about where the web service is located and how to invoke it. The invocation itself is done in a language called SOAP. Therefore, first send a SOAP request asking for the temperature of a certain city.
- The web service will reply with a SOAP response which includes the temperature asked for, or maybe an error message if SOAP request was incorrect.

1.1.5 Web Services Addressing

In last step a web service invocation process is explained. At one point, the UDDI registry 'told' the client *where* the web service is located. But, how exactly are web services addressed? The answer is very simple; just like web pages use plain and simple URIs (Uniform Resource Identifiers). For example, the UDDI registry might have replied with the following URI:

`http://webservices.mysite.com/weather/us/WeatherService`

This could easily be the address of a web page. However, web services are always used by software (never directly by humans). If a web service URI is typed into web browser, an error message will receive or some unintelligible code (some web servers *will* show a nice graphical interface to the web service, but that isn't very common). After finding a web service URI, next step is to give that URI to a program. In fact, most of the client programs will receive the Grid Service URI as a command-line argument.

1.2 Web Services Discovery

Web service discovery is a process of accurate matching of web service from UDDI and it becomes hard to locate a web service which feeds the exact user requirements and usually a single web service is not enough to meet the user requirements. Discovery process faces two main challenges:

- Finding exactly matched web service
- Satisfying end user needs with a single web service

The current UDDI search mechanism can only focus on a single search criterion, such as business name, business location, business category, or service type by name, business identifier, or discovery URL. In fact, in a business solution, it is very normal to search multiple UDDI

registries or WSDL documents and then aggregate the returned result by using filtering and ranking techniques.

With the popularity of the web services technology, more and more software systems functionalities become available by being published and registered as web services. Registered web services need to be dynamically discovered and invoked to meet service requestor's complex service needs.

1.2.1 Web Services Discovery Approaches

Though the field of web service discovery is rather new, yet much work has been lately devoted to the area. The effort in the bulk of the approaches is to enhance the discovery mechanisms in order to overcome the inadequacy of the standard, keyword-based matching, where often the user cannot discover the web service.

Several approaches for web services discovery have been identified and are actually deployed. There detailed discussion is given below.

- **Manual Procedures versus Intelligent Automation:** 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.
- **Centralized versus Decentralized Solutions:** A registry is an authoritative, centrally controlled store of information. The recommended representative of this category is the UDDI registry. 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. 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*. More elaborated decentralized solutions have also been proposed. These systems 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 which 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 aim 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.

1.3 Web Services Composition

Web services composition provides an open, standards-based approach for connecting web services together to create higher-level business processes. Standards are designed to reduce the complexity, required to compose web services. Hence reducing time and costs, and increase overall efficiency in businesses. In Figure 1.4[9] different entities are shown that are involved in composition process.

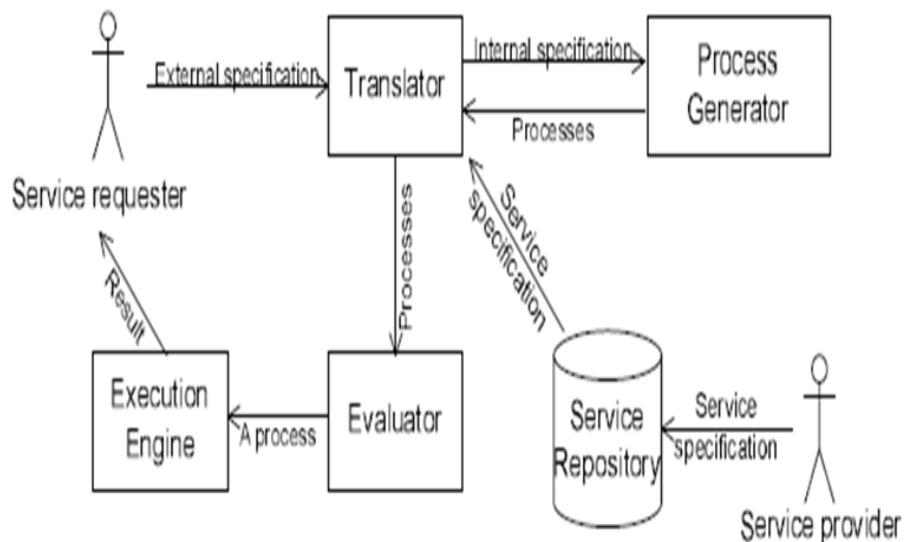


Figure 1.4: Web Services Composition [9]

Different methodologies of web services composition were presented to meet and find required results of user's query, where different web services are integrated and composed into a composite web service. A user is planning for a trip and locates the appropriate web service to execute plan. Different web services like hotel reservation web service, flight reservation service are individually available and need to be integrated to fulfill the user requirement and in plan execution. Composition technique will merge or integrate these individual services into a composite service.

1.3.1 Web Services Composition Steps

Following steps are used in web services composition process:

- A process model specifying control and data flow among the activities has to be created.
- Concrete services to be bound to the process activities need to be discovered. The service composer usually interacts with a broker, e.g. a service registry; in order to look up services which match with certain criteria.
- The composite service must be made available to potential clients. Again the broker is used to publish a description and the physical access point of the service.
- During invocation of a composite service, a coordinating entity may manage the control flow and the data flow according to the specified process model.

1.3.2 Web Services Composition Classification

Web services composition is divided into following classes:

- **Proactive composition:** Offline composition of available services, when services are stable and always running. For example ticket reservation service.
- **Reactive composition:** Dynamically creating a composite service when composite service not often used and service processes not stable. For example tour manager where the itinerary is not predefined
- **Mandatory composition:** All subcomponents must participate to yield a result. For example service that calculates the averages of stock values for a company.
- **Optional composition:** Subcomponents are not obligated to participate for a successful execution. For example services that include a subcomponent that is an optimizer.

1.3.3 Web Services Composition Drawbacks

Since the composer is typically only aware of and consequently interested in some specific types of compositions, the scope of such a search is usually very narrow. Aiming at exploring the full potential of the service space without prior knowledge of what exactly is in it, another view that approaches service composition from the bottom-up is building up recently. Instead of starting the search with a specific goal, a service engineer may be interested in discovering any interesting and useful service compositions that may come up in the search process.

Despite the web service composition benefits, the composition process faces the following drawbacks:

- For finding a specific web service or composite web service, user needs to provide precise query which reduces the size of search space and end user is not getting the advantage of complete search space.
- When user is looking for specific web service or composite web service the user may find the required service or the result in empty set.
- Traditional web services composition is known as top down approach and the web services queried by the composer must be available.

1.4 Web Services Mining

Web services mining provides benefits over compositions techniques and takes the full advantage of search space. In mining process someone can find the relevant usage patterns which are usually not explored in the composition process. Web services are being added to the web at an accelerating rate and increasing the size of search space provides an opportunity for web services mining process to discover and compose interesting web services from the existing web services in unexpected ways which usually are not found in the traditional ways.

1.4.1 Web Services Composition versus Web Services Mining

A key characteristic, distinguishing web services mining from traditional web services composition approaches as governed by standards such as WSFL, XLANG, BPEL4WS, DAML-S and OWL-S is, that web services mining is driven by the desire to find *any* unanticipated and interesting compositions of existing web services. Traditional composition approaches are usually driven by a *top down* strategy, which first requires a user to provide a *goal* containing a

fixed set of *specific* criteria. It then uses these criteria to search for matching component web services. Since the goal provided by the user already implies the type of compositions, the user anticipates, the evaluation of composition *interestingness* is not a major concern in these approaches. In the absence of such top-down query, web service mining techniques need to address how *interestingness* of service compositions can be determined. The lack of specific goals in web services mining also lends itself naturally to being carried out using the bottom-up strategy. The simplest approach following this strategy would be an exhaustive search for composability between all web services. This approach does not scale well since it would inevitably result in a “combinatorial explosion” problem when faced with a large number of web services.

Web services mining provides benefits over compositions techniques and takes the full advantage of search space. In mining process relevant usage patterns are discovered which are usually not explored in the composition process. The benefits offered by web services mining are 1) Walk around the complete web service search space without a specific target in mind 2) Performance issues are solved by cutting down the search space at different levels and providing more suitable search results efficiently. 3) Web services mining results may include other significant web services which are relatively important. 4) Unanticipated web services are discovered in the mining process. 5) Bottom up process which results in unexpected interesting and useful individual web services and composition of web services

In Figure 1.5 a picture of top down web services composition versus bottom up web services mining process is given. Composition results in finding the required web service or composition of web services or an empty set if no match is found where as mining process found all the interesting and relative web services and the composition of web services which are of the user’s interest. Web services are being added to the web at an accelerating rate and increasing size of search space is providing an opportunity for web services mining process to discover and compose interesting web services from the existing web services in unexpected ways which usually are not found in the traditional ways.

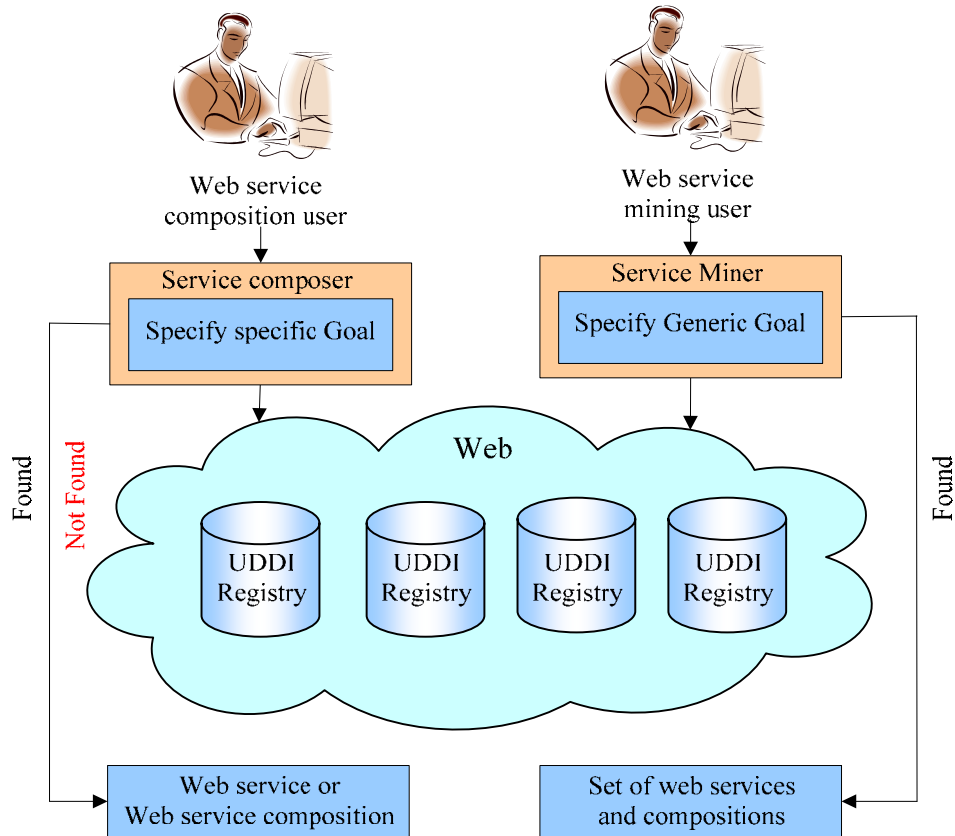


Figure 1.5: Comparison of Web Services Composition and Mining

1.4.2 Web Services Mining Issues

Web services mining process faces the following two main problems which are addressed by using suitable techniques.

- **Combinatorial explosion:** As the number of registered web services increases at an accelerating rate, such an approach can quickly become unfeasible due to the overwhelming computation resulting from a “combinatorial explosion.” Large size of the search space is the major obstacle in finding and composing web services in efficient way. This problem is solved by cutting down the search space size at various phases and by use of multiple threads for parallel searching of web services based on a fuzzy set.
- **Interestingness and Usefulness:** Second main problem with the mining process is finding useful and interesting patterns from the existing web services in the search space. To cater for this problem, fuzzy based matching and constraint satisfaction is applied at filtering and evaluation phases of the mining process.

1.5 UDDI

The Universal Description, Discovery, and Integration (UDDI) Project[4] provides a standardized method for publishing and discovering information about web services. The UDDI project is an industry initiative that attempts to create a platform-independent, open framework for describing services, discovering businesses, and integrating business services. UDDI focuses on the process of *discovery* in the service-oriented architecture.

This section presents an overview of UDDI and how to put it to work. It includes a discussion about the information stored in a UDDI registry, the different potential uses of UDDI, and its technical architecture; the specifications that comprise the UDDI effort, with a focus on their relevance to developers and a list of different Java approaches for programming with UDDI; and an introduction to interacting with a UDDI registry programmatically. The following sections cover the UDDI data structures and XML APIs available for accessing a registry.

Prior to the UDDI project, no industry-wide approach was available for businesses to reach their customers and partners with information about their products and web services. Nor was there a uniform method that detailed how to integrate the systems and processes that are already in place at and between business partners. Nothing attempted to cover both the business and development aspects of publishing and locating information associated with a piece of software on a global scale.

Conceptually, a business can register three types of information into a UDDI registry. The specification does not call out these types specifically, but they provide a good summary of what UDDI can store for a business:

- **White pages:** Basic contact information and identifiers about a company, including business name, address, contact information and unique identifiers such as D-U-N-S numbers or tax IDs. This information allows others to discover web services based upon business identification.
- **Yellow pages:** Information that describes a web service using different categorizations (taxonomies). This information allows others to discover web services based upon its categorization (such as being in the manufacturing or car sales business).

- **Green pages:** Technical information that describes the behaviors and supported functions of a web service hosted by business. This information includes pointers to the grouping information of web services and where the web services are located.

1.5.1 How UDDI is used

UDDI has several different uses, based on the perspective of who is using it. From a business analyst's perspective, UDDI is similar to an internet search engine for business processes. Typical search engines, such as AskJeeves, organize and index URLs for web sites. However, a business exporting a web service needs to expose much more than a simple URL. A business analyst can browse one or more UDDI registries to view the different businesses that expose web services and the specifications of those services. However, business users probably won't browse a UDDI registry directly, since the information stored within it is not necessarily reader friendly. A series of marketplaces and business search portals could crop up to provide business analysts with a more user-oriented approach to browsing the services and businesses hosted in a UDDI registry.

Software developers use the UDDI Programmer's API to publish services (i.e., put information about them in the registry) and query the registry to discover services matching various criteria. It is conceivable that software will eventually discover a service dynamically and use it without requiring human interaction.

Both business analysts and software developers can publish new business entities and services. Business analysts can use portals attached directly to a particular UDDI server or to a more general search portal that supports UDDI.

1.5.2 UDDI Architecture

Details of UDDI project are shown in Figure 1.6[4]. The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system, built from multiple nodes that has their data synchronized through replication. A series of operator nodes, each hosts a copy of the content. The global grouping of operator nodes is jointly known as the UBR. Operator nodes replicate content among one another. Accessing any individual operator node, provides the same information and quality of service, as any other operator node. Content inserted into the UBR is done at a single node, and that operator node becomes the master owner of that content. Any

subsequent updates or deletes of the data must occur at the operator node where the data was inserted.

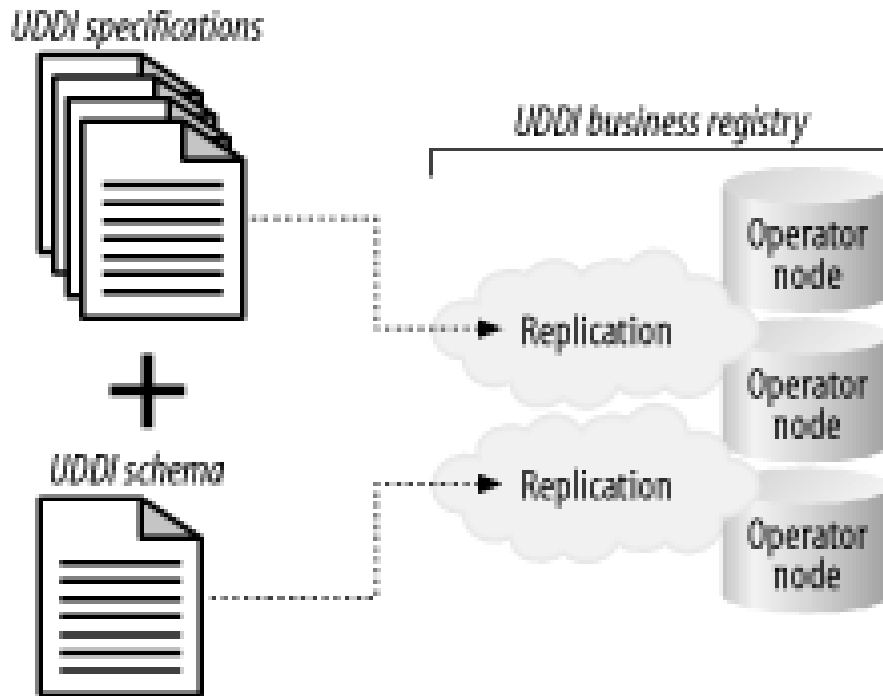


Figure 1.6: The UDDI Initiative [4]

The scope of the UDDI project is much more than the UBR; a company can provide 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. A grouping of companies can also create a "private cloud" of nodes that have information replicated between their private nodes, but that replication sequence will not have any interaction with the UBR nodes.

The UBR has widely accessible inquiry services, but services may be published only by authenticated entities. Any business can create an operator node and make it available over the Internet and part of the UBR. Private operator nodes can define the access rules for their nodes on a case-by-case basis. They can follow the same model as the UBR or make the restrictions looser or tighter.

Companies will likely set up private UDDI nodes. Even though use of these nodes will probably be limited in the near future, quite a few companies are showing interest in setting up private registries for internal or B2B operations. Industry groups are also discussing options for meeting the demands of their individual sector.

Many products have either been created or are being expanded to allow companies to create their own public and private UDDI registries. For example, BEA WebLogic Server and IBM WebSphere both intend to ship a fully compliant UDDI Server embedded within the application server sometime in 2002. Other companies, such as Systinet, HP, Oracle, SAP, Cape Clear, The Mind Electric, and Silverstream, have created J2EE-compliant UDDI implementations that work with existing application servers, including Tomcat, BEA, and IBM. Microsoft has an implementation based upon .NET. Additionally, two open source J2EE UDDI projects are in development: Bowstreet's jUDDI (<http://www.juddi.org/>) and JP Moresmau's pudding (<http://www.opensorcerer.org/>).

1.6 Motivation

All the information on the web is being presented in the form of web services. A large number of web services are being added by different companies and other sources of web service providers. Web services are being created and published by one company on the internet and are used by the web service requestors.

The current service oriented architecture of the web requires automatic discovery of individual web services and composition of interactive and useful web services into composite web services. Web services users, who are looking for a single web service or composition of web services, to satisfy their potential needs, face the problem of finding exciting web service from the large space of available web services, same as user looking for static information on the web. Instead of starting the search with a specific goal, a service engineer may be interested in discovering any interesting and useful service compositions that may come up in the search process

Similar to different data mining techniques, a web service mining framework is required to cater these problems and to satisfy service mining requests. Web service mining process explores the full potential of the service space, without prior knowledge of what exactly is in it. Web service mining provides benefits over compositions techniques and takes the full advantage of search space. In mining process you can find the relevant usage patterns which are usually not explored in the composition process.

The benefits offered by web service mining are:

- Walk around the complete web service search space without a specific target in mind.

- Performance issues are solved by cutting down the search space at different levels and providing more suitable search results efficiently.
- Web service mining results may include other significant web services which are relatively important.
- Unanticipated web services are discovered in the mining process.
- Bottom up process which results in unexpected interesting and useful individual web services and composition of web services.

1.7 Background

Web services are becoming the basis for electronic commerce of all forms. Companies invoke the services of other companies to accomplish a business transaction. In an environment in which only a few companies participate, managing the discovery of business partners manually would be simple. After all, how difficult would it be to figure out if one of few business partners has an access point that adheres to requirements? This model breaks down. However, as the number of companies that need to interact grows along with the number and types of interfaces they export. How to discover all the business partners that can do business? If attempted to account for them manually, user could never be sure that user has discovered every partner.

Information allied with the web services is categorized as “web service provider information” like business details of the provider, protocol used to connect with the web service like HTTP, any constraint specified for the web service and specific patterns associated with the published web service. UDDI, Universal Description, Discovery and Integration has become the de facto standard for publishing web services by web service providers and for discovering information about web services in a standard way. UDDI project was initiated by Microsoft, HP and IBM and provides standard interfaces for communication with the UDDI registries. Two interfaces are provided by the UDDI specification, one for creating and storing information in UDDI and other is the inquiry interface for finding web services. UDDI stores three types of information:

- Company’s detail like contact information
- Web service description details
- Web service functions and supported features.

All the three types of UDDI information is used in the discovery, composition and mining processes. Web service interfaces are described using Web Service Description Language (WSDL) and used for specifying web service metadata description in UDDI registries.

Web service discovery is a process of accurate matching of web service from UDDI and it becomes hard to locate a web service which feeds the exact user requirements and usually a single web service is not enough to meet the user requirements. Discovery process faces two main challenges:

- Finding exactly matched web service
- Satisfying end user needs with a single web service.

Different methodologies of web service composition were presented to meet and find required results of user's query where different web services are integrated and composed into a composite web service. A user is planning for a trip and locates the appropriate web service to execute his plan. Different web services like hotel reservation web service, flight reservation service are individually available and need to be integrated to fulfill the user requirement and in plan execution. Composition technique will merge or integrate these individual services into a composite service.

Despite the web service composition benefits, the composition process faces the following drawbacks:

- For finding a specific web service or composite web service, user needs to provide precise query which reduces the size of search space and end user is not getting the advantage of complete search space.
- When user is looking for specific web service or composite web service the user may find the required service or the result in empty set.
- Traditional web service composition is known as top down approach and the web service queried by the composer must be available.

1.8 Problem Statement

Users, who request either simple or composite web services, face the problem of identifying “what is out there on the web” that is similar to the search problem faced by the users looking for available text content. Just as users looking for text content need web mining, users looking for services need service mining. Web service users who are looking for a single web

service or composition of web services to satisfy their potential needs, face the problem of finding exciting web service from the large space of available web services same as user looking for static information on the web. Similar to different data mining techniques, a web service mining framework is required to cater these problems and to satisfy service mining requests.

1.9 Problem Solution

A threaded model for web service mining based on fuzzy set and fuzzy logic with rules satisfaction is proposed. The proposed model is divided into different steps and phases to reduce the model complexity and simplify different integrating processes. The problems faced in mining process are complexity of the search space and pattern matching. These problems are discussed in detail in introductory part. The complexity model is targeted by introducing the concept of threading for parallel processing. A new thread is initiated for the every member of fuzzy set and mines the search space for required computation. This parallel processing approach helps in optimizing the search and matching process and for efficient discovery of individual web services and composition of web services.

The first step in proposed framework is scope and rules specifications. Scope and the rules are specified by a web service domain expert and these are according to required mining results. For example domain expert is looking for web services related to traveling or in the field of medicine. Rules specified by the domain expert will be matched in constraint satisfaction and evaluation phases for filtering and validating of found web services and their compositions. Fuzzy set is generated, based on the scope specified by the domain expert and weights are assigned to each member of the fuzzy set. Weights are calculated based on the probability model and with the help of local database. This local database is used to stored members of fuzzy set and helps in calculating weights. Every member of the fuzzy set is used as input to the next searching phase and after the phase of assigning weights, a new thread is initiated for every member of the fuzzy set. This thread explores the UDDI registry and looks for relevant services based on the fuzzy matching algorithm. Outputs of the found web services are further used for discovery of web services which are using these as their input parameters for composing individual web services into composite web services.

Web service mining results sorted in the indexing phase, based on weights assigned and these sorted results are filtered in the rules satisfaction phase, where constraint specified in the

first phase are matched with the publisher's constraint. Publisher specifies any service relevant constraint in the web service description document and at this step of our proposed model, these rules are satisfied for filtering and validation of found results. These filtered results are used as input to evaluation phase where these results are gone through objective and subjective evaluation.

1.10 Organization

Current chapter comprises an overview of Web Services, Dynamic Web Services composition and Web services composition approaches with brief explanation. Also the Problem statement and contributions to work are briefly stated.

Chapter 2 This chapter is on the research papers that are used as references for our thesis.

Chapter 3 This chapter is about the Methodology and Techniques used in the thesis.

Chapter 4 This chapter is about the implementation of proposed algorithm.

Chapter 5 This chapter is concerned with analysis and Results.

Chapter 6 This chapter includes the conclusion and future work.

Chapter 2 : Related Work

Web mining is a process of retrieving useful information from the web using artificial intelligence techniques. Whereas, focus of this report is on exploring the useful and interesting patterns from web services called web services mining. The need for the dynamic web services discovery and composing services is arising. In this chapter different relevant web service mining techniques are discussed.

2.1 Web Services Architecture

Complete understanding of web service architecture creating, publishing and discovering web services is suggested by [1]. Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. More details about web services are already discussed in chapter 1.

2.2 Service Mining on the Web

George Zheng, Athman Bouguettaya et al. “*Service Mining on the Web*” [3] propose a web service mining framework that allows unexpected and interesting service compositions to automatically emerge in a bottom-up fashion. As a novel application of this framework, authors demonstrate its effectiveness and potential by applying it to service-oriented models of biological processes for the discovery of interesting and useful pathways. This paper discusses the top down web services composition versus bottom up web services mining techniques.

For an illustration, authors of this paper show Figure 2.1[3] that a service engineer sets out to find any interesting and useful services with a general interest in Chinese medicine in mind. What comes out of the search process, might be quite surprising. For example, in addition to discovering the possibility of composing a service for translating Tsalagi1 to Chinese, the

engineer also discovers, with the help of a service mining tool, a service composition that takes as input a biological sample from a subject, determines the corresponding genome and the possible diseases, the subject is predisposed to, and finally generates a list of treatment recommendations and/or life style suggestions. Thus, unlike the search process in the top down approach that is strictly driven by the search criteria, the search process in the bottom-up approach is serendipitous in nature, i.e., it has the potential of finding interesting and useful service compositions that are unexpected.

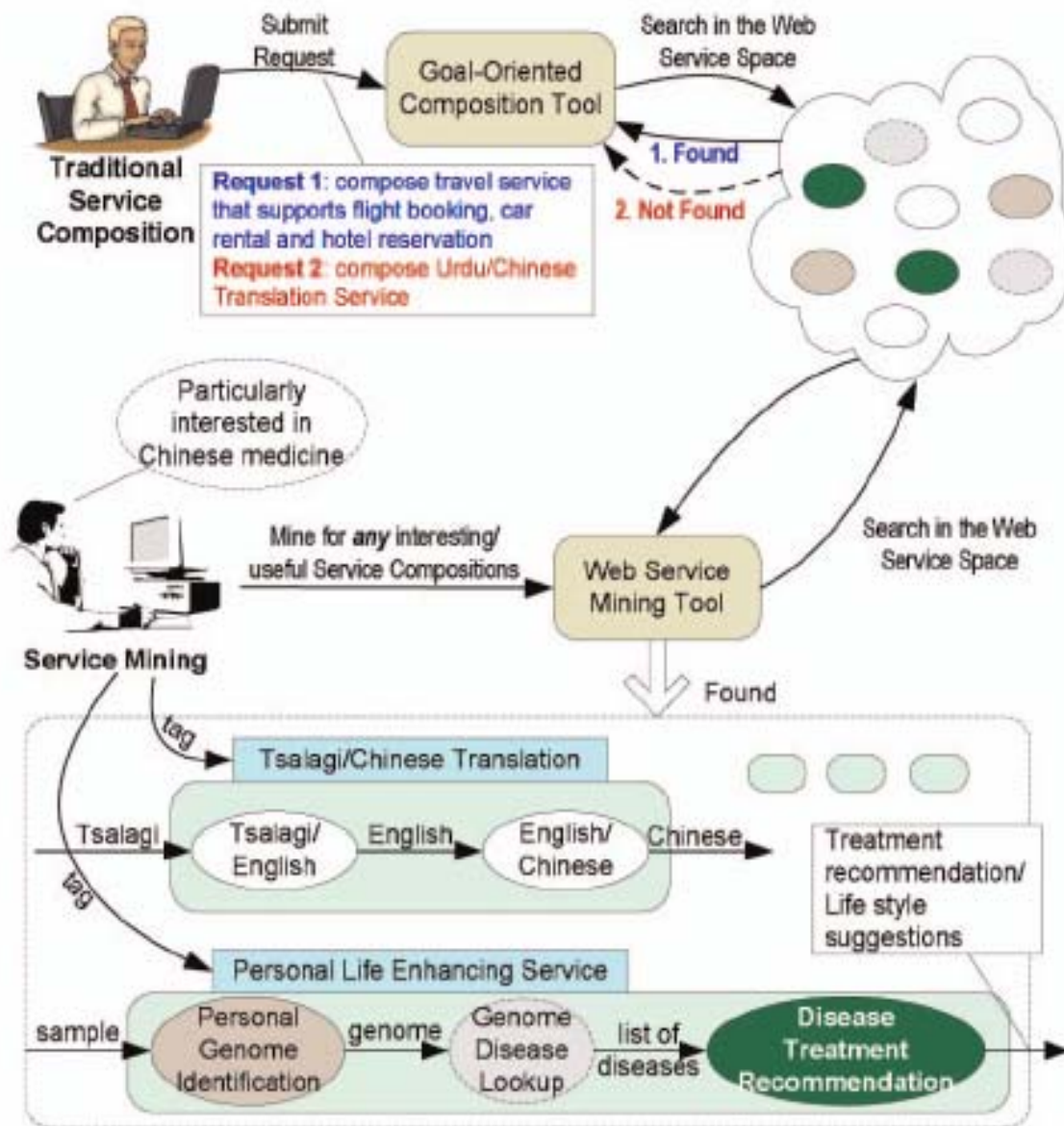


Figure 2.1: Top-down Composition versus Bottom-up Mining [3]

2.3 Service Pattern Discovery of Web Service Mining

Qianhui Althea LIANG, Jen-Yao CHUNG, Steven MILLER and Yang OUYANG et al. “*Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository*”[5] presents and elaborates the concept of web service usage patterns and pattern discovery through service mining. Authors of this paper have defined three different levels of service usage data.

- User request level
- Template level
- Instance level

At each level, authors have investigated patterns of service usage data and the discovery of these patterns. An algorithm for service pattern discovery at the template level is presented. Authors of this paper have shown the system architecture of a service-mining enabled service registry repository. Web service patterns, pattern discovery and pattern mining supports the discovery and composition of complex services, which in turn supports the application of web services to increasingly complex business processes and applications.

2.3.1 Service mining in service registry-repository

Service mining is defined as the automated discovery in this paper and analysis, of how web services are used in a collective way. It aims at discovering services that meet the specified requirements. How a web service is described is essential to service mining. If web services are defined and described in a machine understandable manner, their discovery will be a much easier job. Adopting OWL-S as the service description language, knowledge of web services consists of four parts, which are listed below.

- Service Profile information, such as service provider’s contact information and service operation input and output information.
- Service grounding information, such as the protocol used to interact with the service.
- Service constraint information, such as the conditions that limit the use of the service.
- Service usage data, such as patterns associated with the use of the service.

Service mining is meant to discover all four types of knowledge. In particular, goals of service mining are as the followings:

- When constructing a new service for a business process by service composition, predict the correct service functions to select.
- When building such services, predict the good-profiled service partners to collaborate with.
- Optimize composite service execution for performance issue due to the scalability of the composite service.
- Classify services.

2.3.2 System Architecture

Registry-repository architecture is shown in Figure 2.2 [5].

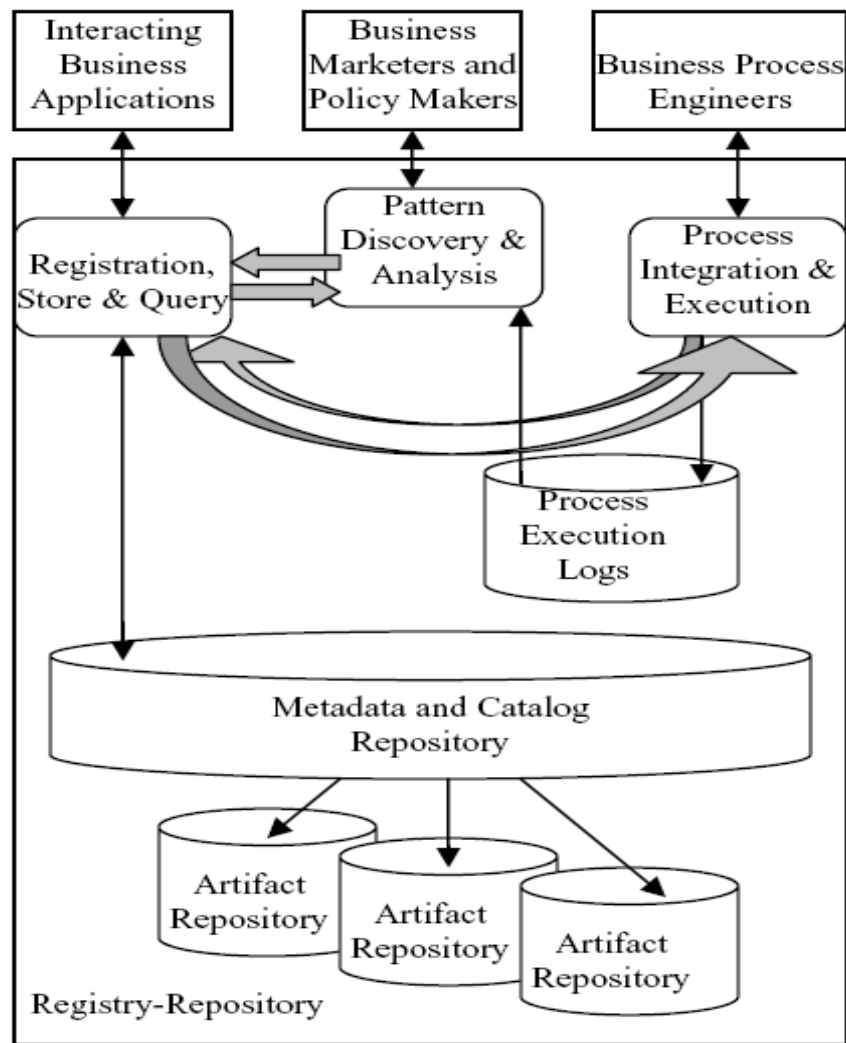


Figure 2.2: Pattern-Discovery Enabled Registry-Repository Architecture [5]

The registry-repository has a collection of artifact repositories. Each such repository, holds one type of artifacts or similar types of architects. For example, all WSDL documents can be stored in one repository and all OWL-S documents can be put in another. A metadata and catalog repository is connected to all artifact repositories to facilitate catalog and metadata management services of the contents in the repositories underneath it. These services are provided by various functional components of the registry to be discussed shortly. The registry also includes storages for the logs of executed business process instances, which make up one important information source for service analysis. The registry repository primarily offers registration service for businesses to publish services, responds to service queries of service requesters, provides storage for all service-related artifacts, discovers and analyzes service usage patterns and supports automated business process integration and engineering.

2.4 An Improved Way to Facilitate Composition-Oriented Semantic Service Discovery

Gao Ting, Wang Haiyang, Zheng Naihui, Li Fei et al. “*An Improved Way to Facilitate Composition-Oriented Semantic Service Discovery*” [6] simplify the web service discovery and composition definition, and present an approach for automatic service discovery and composition based on semantic description of web services, which is on the foundation of using inverted indexing to facilitate composition-oriented semantic service discovery. Authors of this paper use the inverted indexing to facilitate composition-oriented semantic service discovery. Inverted file is the kind of index found in most commercial library systems. One type of lexicographical index, the inverted file, is drawn on in this paper. The concept of the inverted file type of index is as follows. Assume, there is a set of documents. Each document is assigned a list of keywords or attributes. An inverted file is then the sorted list of keywords (attributes), with each keyword having links to the documents containing that keyword. Authors improve the technology based on the characteristics of web services. It can be applied to the service discovery and service composition, and the performance is greatly improved.

The structure of inverted indexing presented for the first time comes from inverted file technology. An inverted file is the sorted list (or index) of keywords (attributes), with each keyword having links to the documents containing that keyword. For the web service repository, keyword is outputs of service. Each output has a corresponding list of service links and each

service pointed by the link in the list all has the output. Each link has a corresponding service. Structure of inverted indexing is given in Figure 2.3[6].

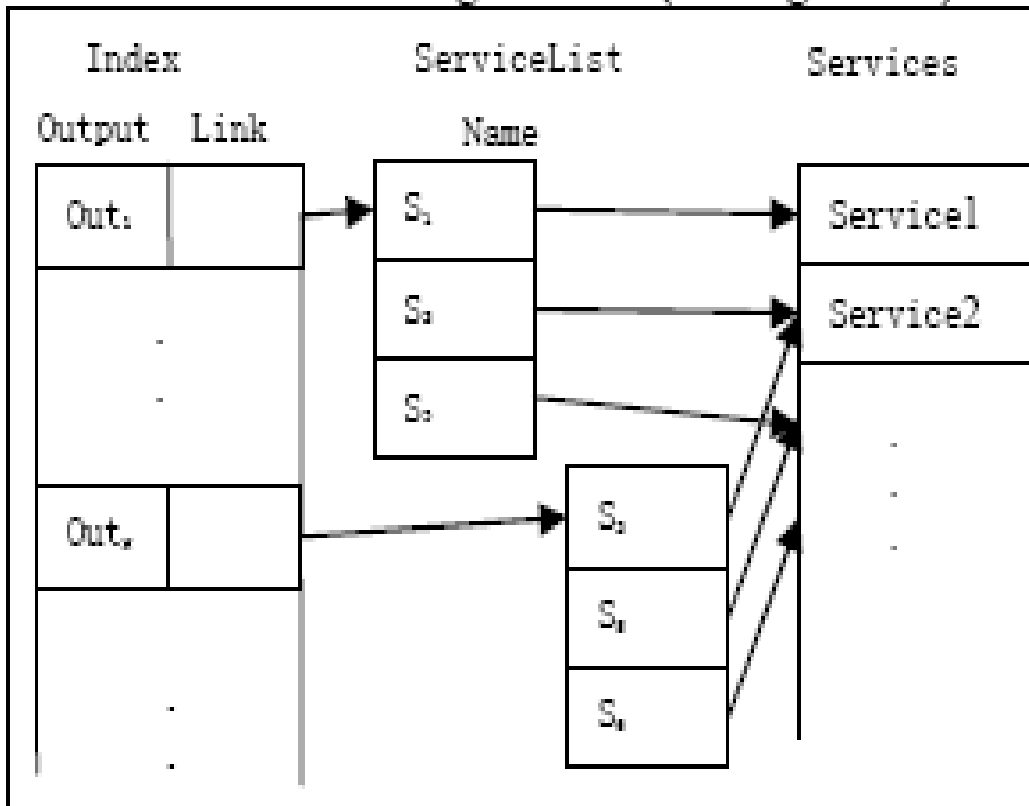


Figure 2.3: Structure of Inverted Indexing [6]

The web service discovery algorithm based on inverted indexing has three major steps:

- Atomic service discovery,
- Breadth- Composite service discovery
- Depth- Composite service discovery

This paper presents an improvement of service discovery, which is on the foundation of using inverted indexing to facilitate composition-oriented semantic service discovery. Furthermore, the algorithm decreases searching space cost and increase the precision. The web service providers register and advertise their services in an open repository. Applications find and choose the appropriate services automatically. Given a repository of Web services and a query requesting of a special service, the application should find one or some service that match the query requirements.

2.5 Service Mining for Web Service Composition

Qianhui Althea LIANG, Steven MILLER and Jen-Yao CHUNG “*Service Mining for Web Service Composition*” [9] propose the concept of service mining and show how this mining helps in the automatic assemblage of services into complex aggregates, called composite services. Differences of service mining with Web mining are explained in this paper. With a framework for composite service processing as background, possible aspects of service mining are also explored. Authors have introduced the concept of service constraint processing (SCP) for service mining, and model it as a 2-tier constraint satisfaction problem (CSP) problem. An algorithm for solving the SCP problem is also given.

In Figure 2.4[9] a standard web service model is given which is extended by authors.

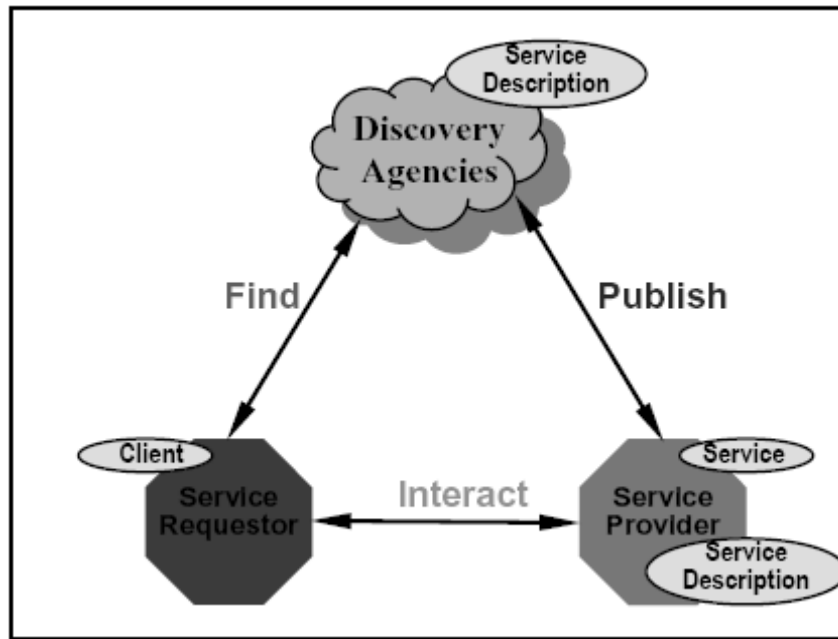


Figure 2.4: Standard Web Service Model [9]

WSDL is extended with constraint specifications resulting in the exposure of more business/technical details of the services. Service requestors' requirements and service providers' constraints are matched to effectively mine for useful services. In addition to basic operations like “publish”, “find” and “save” given in the UDDI specification, the Intelligent (service) Registry in the extended model provides supports for the construction, description and invocation of composite Web Services as well. The other component, called Composite Service

Processor (Csp) as shown in Figure 2.5[9], follows the specification and schedules the enactment of the composite service.

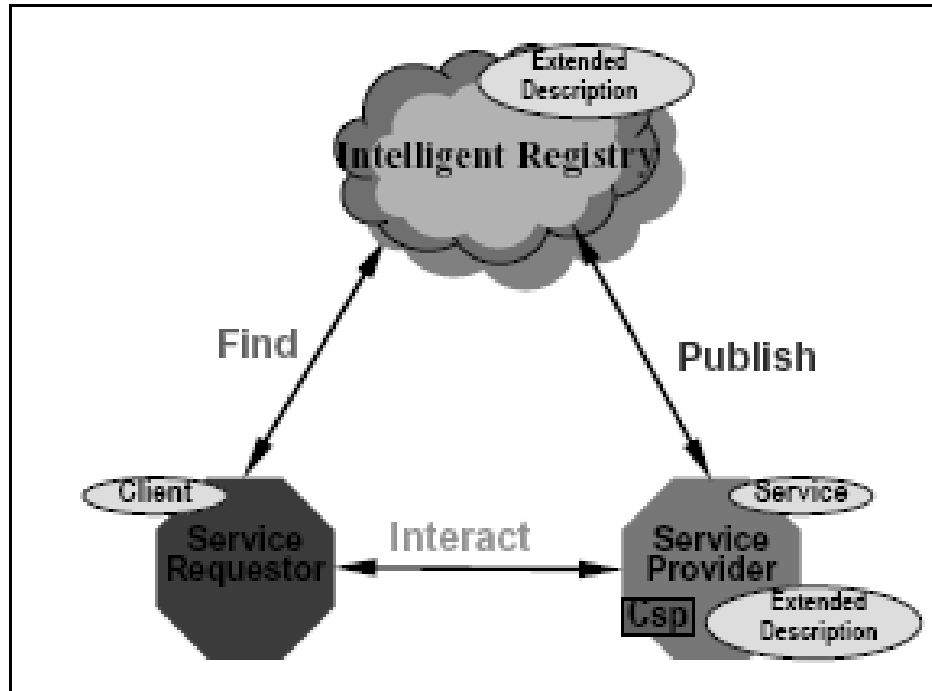


Figure 2.5: Web Service Model Extended Form [9]

In this paper, the Intelligent Registry uses a semiautomatic approach to dynamically compose services, using service mining to identify appropriate component services. The iterative and interactive composing process consists of mining existing Web Service interfaces to construct a composite service template(s) for the approval, and mining service providers to instantiate the template after the template is discovered and approved. The Intelligent Registry is composed of five components: the *Constraint-based Broker* is an implementation of the UDDI registry that answers service queries only with services not conflicting with requestors' requirements. The *Query Composer* interacts with the requestor to produce a service query. The *SDG (service dependency graph) Generator* builds an And-Or graph to describe the interdependent relationship of services. The *Service Composer* forms a composite service by searching the SDG and by solving a service constraint processing problem. The *Composite Service Specification (CSS) Generator* generates the description document of the discovered composite service.

Chapter 3 : Proposed Approach

In this chapter proposed system is discussed. The main focus of this chapter is to describe different components of the system and design of the system. Initially, all of the components of the architecture are mentioned after that their details i.e. their purpose and their working is discussed.

3.1 Proposed Web Services Mining Framework

A threaded model for web service mining based on fuzzy set and fuzzy logic with rules satisfaction is proposed. Figure 3.1 gives a complete picture of the projected idea and demonstrates different phases involved in the mining process. The proposed model is divided into different steps and phases to reduce the model complexity and simplify different integrating processes. The problems faced in mining process are complexity of the search space and pattern matching. The complexity model is targeted by introducing the concept of threading for parallel processing. A new thread is initiated for every member of fuzzy set and mines the search space for required computation. This parallel processing approach helps in optimizing the search and matching process and for efficient discovery of individual web services and composition of web services.

The first step in proposed framework is scope and rules specifications. Scope and the rules are specified by a web service domain expert and these are according to required mining results. For example domain expert is looking for web services, related to traveling or in the field of medicine. Rules specified by the domain expert will be matched in constraint satisfaction and evaluation phases for filtering and validating of found web services and their compositions. Fuzzy set is generated based on the scope specified by the domain expert and weights are assigned to each member of the fuzzy set. Weights are calculated on the probability model and with the help of local database. This local database is used to store members of fuzzy set and helps in calculating weights. Every member of the fuzzy set is used as input to the next searching phase and after the phase of assigning weights a new thread is initiated for every member of the fuzzy set. This thread explores the UDDI registry and looks for relevant services based on the fuzzy matching algorithm. Outputs of the found web services are further used for discovery of

web services which are used as their input parameters for composing individual web services into composite web services.

Web service mining results sorted in the indexing phase, based on weights assigned and these sorted results are filtered in the rules satisfaction phase, where constraint specified in the first phase are matched with the publisher’s constraint. Publisher specifies any service relevant constraint in the web service description document and at this step of proposed model these rules are satisfied for filtering and validation of found results. These filtered results are used as input to evaluation phase where these results are gone through objective and subjective evaluation.

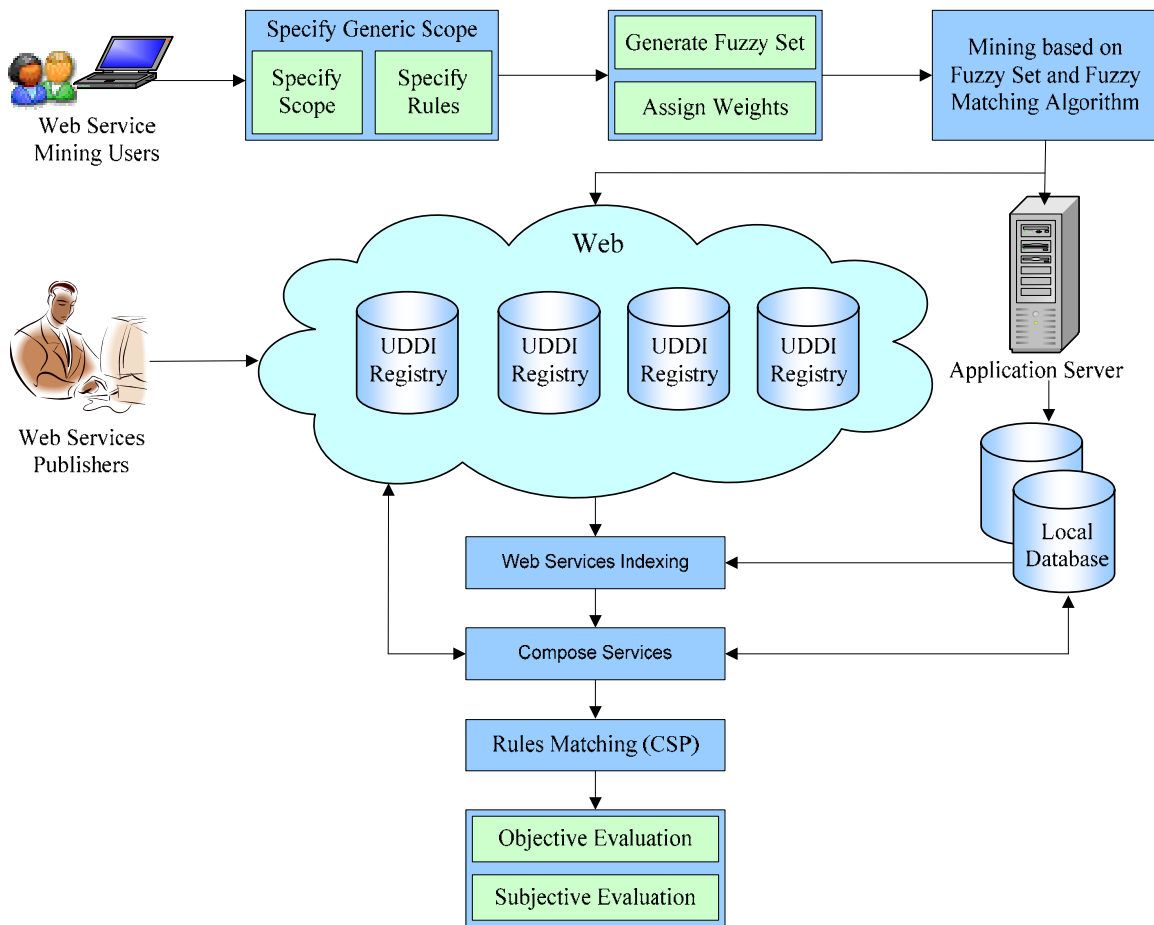


Figure 3.1: Web Services Mining Framework

3.1.1 Context and Rules Specification

Web services mining framework starts with the specifying scope by a web service domain expert. This scope may include a generic domain, like mining in the field of traveling, insurance or medicine. Web service domain expert will also specify a set of rules on the domain

of mining. These rules may include some generic specifications or specific rules to the input, output parameters and web service description. These rules are matched in the filtering phase with the web service description document. For example, looking for web services for books where defined the upper bound and lower bound on the price of books or looking for some insurance policy related web service of a particular period.

3.1.2 Fuzzy Set Generation

Fuzzy set is generated based on the context, specified by the domain expert. This set contains the synonyms of the scope specified in the earlier stage and weights are assigned to this set. Specified fuzzy set is

$$S = (s_1 , s_2 , s_3 \dots s_n)$$

Where $S_i = \mu_s(x_i)$ and x_i is element of X and X is the domain of the web services. For original members of the search string we will assign weight 1 and it has the membership value as

$$\exists x \in X \bullet \mu_s(x) = 1$$

And for all those members where degree of truth is non zero define the relationship as:

$$x \in X \mid \mu_s(x) > 0$$

3.1.3 Weights Calculation and Assignment

Value of degree of truth is assigned to every member of the fuzzy set. Degree of truth for the actual specified string is 1 and for other fuzzy set members degree of truth is calculated on basis of history of term used. A local database is maintained for storing the record of every term when it is used, and lately, these records are used for weights calculation. For example, different words like books, medicine, travel and insurance for web services mining are used and these words have a relative number of occurrences in the database. Total numbers of occurrences of all members of fuzzy set are added and number of occurrences of each member is divided with this sum to calculate the weight. This weight is assigned to the member of fuzzy set.

$$\text{Weight of a term} = \text{Number of occurrence of term} / \text{total occurrences}$$

3.1.4 Fuzzy Rules

Two fuzzy sets have been defined based on which fuzzy rules are determined. W is a fuzzy set of weights assigned in last step. It is defined as

$$W = \{\text{Short, Medium, High}\}$$

The other set D is the matching distance which will be used in distance or approximate matching algorithm. This set is defined as

$$D = \{\text{Exact, Close, Approximate}\}$$

Based on these fuzzy sets the following rules are defined:

- IF W = Short THEN D = Exact
- IF W = Medium THEN D = Close
- IF W = High THEN D = Approximate

Distance is increased in fuzzy matching algorithm as long as weight of input parameter is increasing. These weights and distances based rules are plotted in Figure 3.2.

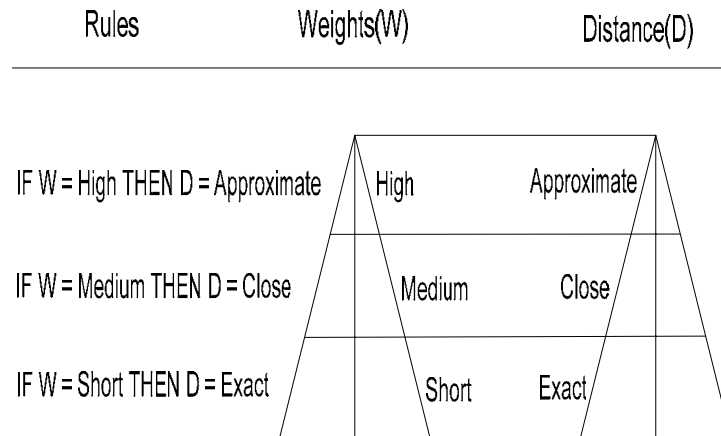


Figure 3.2: Fuzzy Rules for Web Services Mining Framework

3.1.5 Multithreaded Model for Mining Services and Fuzzy Matching

Based on the fuzzy set, which we have created in earlier phase, a separate thread is initiated for every member of the set which looks into the UDDI registry using public query interfaces and matches the results with the found web services based on fuzzy matching. Distance matching algorithms are implemented, which take service description, text pattern to match and distance as input parameters. This distance parameter is computed, based on the fuzzy rules, specified earlier using assigned weights to fuzzy set. Weights are calculated in earlier phase and for higher weights, assign longer distances. All those web services which have approximate matching greater than zero are stored in the database with their weights, distance

and matching value. These results are indexed in sorted order and constraint satisfaction is applied for the further filtering of the mining results. For every found web service, a new thread is initiated which will look output parameters as the input parameters of the other web services in the UDDI registry to find out interesting and useful compositions of web services. These results are also stored in the database and linked with the original web services in a connected way using a graph. Algorithms for fuzzy matching and composing web services are given below:

S[1..n]: Fuzzy Set

W[1..n]: Weights Set

O[1..n]: Web service output parameters

I[1..n]: Web service input parameters

Algorithm: FuzzyMatching

Input: S[1..n], W[1..n]

Output: services, composedServices

```
1: for i ← 1 to n do
2:   initiate new thread
3:   member ← S[i]
4:   weight ← W[i]
5:   if weight is High then
6:     distance ← Approximate
7:   else if weight is Medium then
8:     distance ← Close
9:   else if weight is Short then
10:    distance ← Exact
11:  end if
12:  service ← Fetch Web service
13:  result ← call ApproximateMatchingAlgorithm(service, member, distance)
14:  if result > 0 then
15:    Store service in database
16:  end if
17:  Sort stored services
18:  for each stored service
```

```

19:      initiate new thread
20:      O[1..n] ← service.outputParameters
21:      service ← Fetch Web service
22:      I[1..n] ← service.inputParameters
23:      temp ← false
24:      for i ← 1 to n do
25:          if O[i] = I[i] then
26:              temp ← true
27:          else
28:              temp ← false
29:              break loop
30:          end if
31:      end for
32:      if temp = true then
33:          link services and store in database
34:      end if
35:  end for
36: end for

```

3.1.6 Constraint Satisfaction

Service requestor requirements are matched with service provider's constraints and constraints specified in first step are satisfied with web service input, output and operations. Constraint matching model is presented as a 3-tuple of {I, O, OPR} where $I = \{i_1, i_2, i_3 \dots i_n\}$ is a set on input parameters of a web service, $O = \{o_1, o_2, o_3 \dots o_n\}$ is a set of output parameters of a web service and $OPR = \{d_1, d_2, d_3 \dots d_n\}$ is a set of operations of a web service. A set is specified by the service requestor on the input parameters, output parameters and service operations and constraints specified by the provider on these parameters and operations. Algorithm for constraint satisfaction is as follows:

I[1..n]: Web service input parameters

O[1..n]: Web service output parameters

OPR[1..n]: Web service operations

IC[1..n]: Constraints on input parameters

OC[1..n]: Constraints on output parameters

OPRC[1..n]: Constraints on operations

Algorithm: ConstraintSatisfaction

Input: I[1..n], O[1..n], OPR[1..n], IC[1..n], OC[1..n], OPRC[1..n]

Output: Filtered Web Services

```
1: for all members from database do
2:   service ← Fetch Web Service
3:   I[1..n] ← service.inputParameters
4:   O[1..n] ← service.outputParameters
5:   OPR[1..n] ← service.operations
6:   result ← false
6:   for i=1 to n do
7:     result ← Match I[i] with IC[i]
8:     if result = false then
9:       remove from database
10:      break loop
11:    end if
12:  end for
13:  for i=1 to n do
14:    result ← Match O[i] with OC[i]
15:    if result = false then
16:      remove from database
17:      break loop
18:    end if
19:  end for
20:  for i=1 to n do
21:    result ← Match OPR[i] with OPRC[i]
22:    if result = false then
23:      remove from database
```



```
24:         break loop
25:     end if
26: end for
27: end for
```

3.1.7 Evaluation

Evaluation process is used to finalize and fetch out interesting compositions of web services from the resultant pool of web services. Evaluation process involves two phases which are objective evaluation and subjective evaluation. Objective parameters like operation similarity are used in the objective evaluation phase. Two operations are considered similar if they have same input parameters and produce the same results for these input parameters. Objective evaluation is based on operation similarity where mined web services are matched with the required operations. Our final evaluation process involves taking subjective actions to find out attractive web services compositions leads. The subjective process is based on the knowledge of the domain expert and the requirements of end user. The size of web services pool is already reduced in the constraint satisfaction and the objective evaluation phases. The final web services and their compositions are selected by the domain expert based on experience and previous knowledge.

Chapter 4 : System Design

4.1 System Flow Chart

Flow charts are easy-to-understand diagrams showing how steps in a process fit together. This makes them useful tools for communicating how processes work, and for clearly documenting how a particular job is done. Furthermore, the act of mapping a process in flow chart format helps in understanding of the process, and helps about where the process can be improved.

A flow chart can therefore be used to:

- Define and analyze processes;
- Build a step-by-step picture of the process for analysis, discussion, or communication; and
- Define, standardize or find areas for improvement in a process

Figure 4.1 shows flow chart of the complete web services mining framework. All the major modules and systems of proposed framework are covered in this flow chart. Proposed framework is invoked by defining generic scope, and then rules are specified. Based on this generic scope a fuzzy set is generated and weights are assigned to this fuzzy set. In next step, for each member of generated fuzzy set, a new thread is initiated. This thread discovers services from UDDI based on fuzzy set using appropriate fuzzy algorithm. All the services discovered at this step are indexed and further used in composition process. Web service composition algorithm is invoked at this step of proposed framework and all the discovered services are passed as input parameter. Output parameters of each discovered web service are matched with input parameters of other services and a link is formed where these parameters matched. In final step all the composed and discovered web services are filtered.

4.2 Sequence Diagram

UML sequence diagrams are used to represent or model the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction, showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

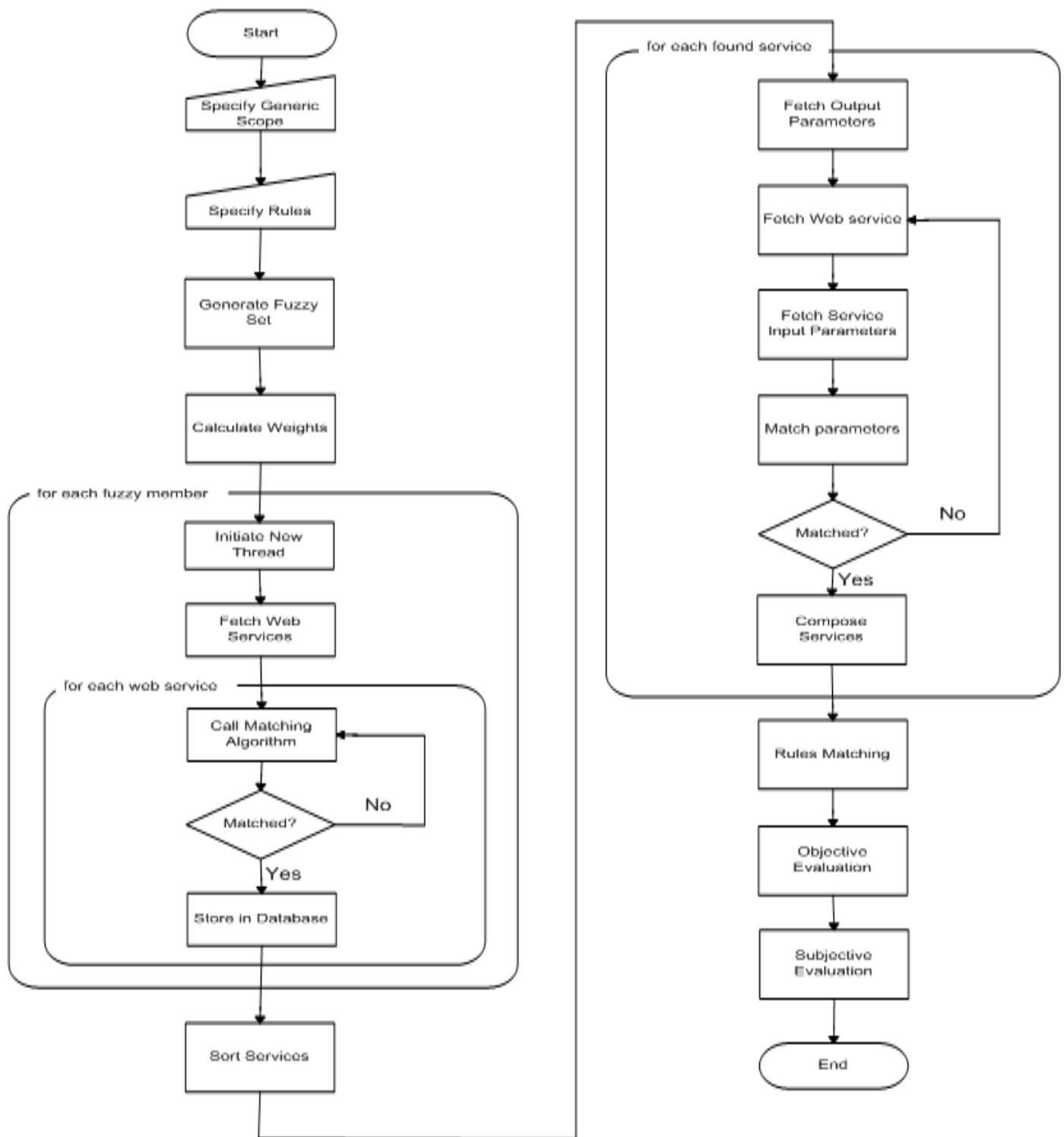


Figure 4.1: Flow Chart of Web Services Mining Framework

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.

In Figure 4.2 a complete flow of proposed web services mining framework is given in the form of sequence diagram. This sequence diagram shows all the interacting objects with respect to time and events generated. Objects that interact in proposed approach are user, system, UDDI registries and application server. In first step user specifies goal and rules. System generates fuzzy set based on goal and sends a request to application server to calculate weights for each member of fuzzy set. Application server calculates weights based on values in database and sends results back to system. System initiates a new thread for each fuzzy member and discovery process is called. Finally all the discovered web services are composed.

4.3 Use Cases

Use cases describe the system from the user's point of view. Use cases describe the interaction between one or more actors (an actor that is the initiator of the interaction may be referred to as the 'primary actor') and the system itself, represented as a sequence of simple steps. Actors are something or someone which exists outside the system ('black box') under study, and that take part in a sequence of activities in a dialogue with the system to achieve some goal. Actors may be end users, other systems, or hardware devices. Each use case is a complete series of events, described from the point of view of the actor.

In Table 4.1 a use case for web service mining process is given. In this table all the minor details of mining process are covered from the user's point of view. In Table 4.2 a use case for add a new UDDI registry is explained. Similarly in Table 4.3 a use case for editing a UDDI registry is given and in Table 4.4 a use case for deleting a UDDI registry is explained.

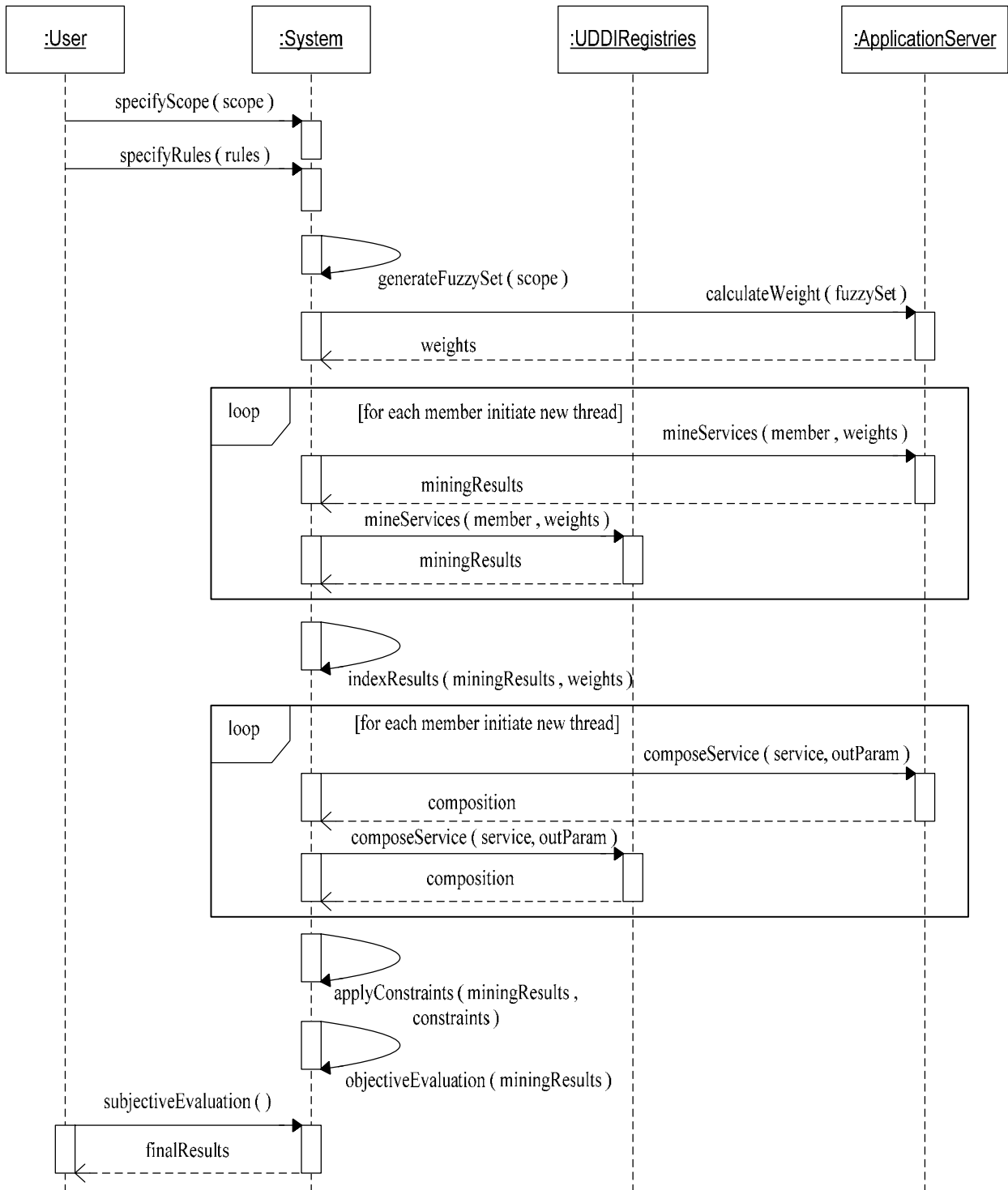


Figure 4.2: Sequence Diagram of Web Services Mining Framework

Table 4-1: Use Case for Web Services Mining

Use Case Name	Web Services Mining	
Scenario	Web Services Mining	
Triggering Event	User specify the generic scope for mining web services	
Brief Description	User specifies a generic scope for web services mining and system process the use request and generates results. These results are evaluated in different phases.	
Actors	User	
Related Use Cases		
Stakeholders	User	
Pre-conditions	Database Server is up and UDDI registries are connected.	
Post-conditions	System generates a set of web services and composition of web services.	
Flow of Events	Actor	System
	<ol style="list-style-type: none"> 1. User specifies generic scope. 2. User specifies rules. 3. User requests for web services mining based on scope and rules specified. 	<ol style="list-style-type: none"> 3.1 System generates fuzzy set. 3.2 System calculates weights for each member of fuzzy set. 3.3 System assigns these weights. 3.4 System mines for web services for each member of the fuzzy set based on the weights. 3.5 System looks for composition for each result

	4. User performs subjective evaluation.	found in the above step. 3.6 System applies rules satisfaction. 3.7 System performs objective evaluation.
Exception Conditions:		

Table 4-2: Use Case for Adding UDDI Registry

Use Case Name	Add New UDDI Registry	
Scenario	Adding new UDDI registry	
Triggering Event	User wants to add a new UDDI registry.	
Brief Description	In web services mining framework, we have tested it with different UDDI registries. User need to add or edit different UDDI registries connection information. For this given an interface.	
Actors	User	
Related Use Cases	Edit UDDI Registry, Delete UDDI Registry	
Stakeholders	User	
Pre-conditions	Database Server is up.	
Post-conditions	UDDI registry connection information is saved in the database.	
Flow of Events	Actor	System
	1. User enters UDDI registry name. 2. User enters information for inquiry interface. 3. User enters information for publish interface. 4. User enters username	

	<p>information.</p> <p>5. User enters password information.</p> <p>6. User saves the information in database.</p>	<p>6.1 System validates the information entered by user.</p> <p>6.2 System checks for duplication of information.</p> <p>6.3 System saves information in database.</p>
Exception Conditions:	6.2 If information already exists, system asks the user to review information.	

Table 4-3: Use Case for Editing UDDI Registry

Use Case Name	Edit UDDI Registry	
Scenario	Editing UDDI registry	
Triggering Event	User wants to edit already added UDDI registry information.	
Brief Description	In web services mining framework, we have tested it with different UDDI registries. User need to add or edit different UDDI registries connection information. For this given an interface.	
Actors	User	
Related Use Cases	Add UDDI Registry, Delete UDDI Registry	
Stakeholders	User	
Pre-conditions	Database Server is up.	
Post-conditions	UDDI registry connection information is updated in the database.	
Flow of Events	Actor	System
	<p>1. User updates UDDI registry name.</p> <p>2. User enters updated information for inquiry</p>	

	<p>interface.</p> <p>3. User enters updated information for publish interface.</p> <p>4. User updates username information.</p> <p>5. User updates password information.</p> <p>6. User updates the information in database.</p>	<p>6.1 System validates the information updates by user.</p> <p>6.2 System checks for duplication of information.</p> <p>6.3 System updates information in database.</p>
Exception Conditions:	6.2 If information already exists, system asks the user to review information.	

Table 4-4: Use Case for Deleting UDDI Registry

Use Case Name	Delete UDDI Registry		
Scenario	Deleting UDDI registry		
Triggering Event	User wants to delete already added UDDI registry information.		
Brief Description	In web services mining framework, we have tested it with different UDDI registries. User need to add, edit or delete different UDDI registries connection information. For this given an interface.		
Actors	User		
Related Use Cases	Add UDDI Registry, Edit UDDI Registry		
Stakeholders	User		
Pre-conditions	Database Server is up.		
Post-conditions	UDDI registry connection information is removed from the database.		
Flow of Events	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; text-align: center;">Actor</td> <td style="width: 50%; text-align: center;">System</td> </tr> </table>	Actor	System
Actor	System		

	<ol style="list-style-type: none"> 1. User selects a UDDI registry from the list of already added registries information. 2. User deletes selected UDDI registry. 3. User confirms the delete operation. 	<ol style="list-style-type: none"> 2.1 System shows a confirmation message to user. 3.1 System deletes UDDI registry information from the database. 3.2 System shows updates list of registries to user.
Exception Conditions:	2.1 If user does not confirm the delete operation, system returns control to main screen.	

4.4 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality, provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Figure 4.3 shows the proposed system from user's perspective.

Following use cases are plotted in use case diagram.

- Add UDDI Registry
- Edit UDDI Registry
- Delete UDDI Registry
- Specify Scope
- Specify Rules
- Web Service Mining



Figure 4.3: Use Case Diagram of Web Services Mining Framework

Chapter 5 : Implementation

This chapter covers the design and implementation details of the application. Different supporting APIs used to develop this system are discussed in depth. Details about setting up a UDDI server and creating a UDDI client using JAVA APIs are given in this chapter. Figure 5.1 shows a high level system architecture diagram.



Figure 5.1: High Level Architecture

5.1 UDDI Server (Apache jUDDI)

JUDDI (pronounced "Judy") is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for Web Services. jUDDI is used with existing authentication technologies, and with virtually any relational database including MySQL, DB2, Sybase, and others.

5.1.1 Setting up UDDI Server

To set up UDDI registry using Java, install following:

- Java 2 SDK—Sun's Java 2 SDK SE, version 1.6.
- Web server and/or servlet container—Apache Tomcat, version 6.0.24.

- SOAP processing framework—Apache Axis that ships with JUDDI.
- Data storage mechanism—MySQL relational database, version 5.0.
- UDDI registry framework—jUDDI.
- Once the JUDDI server is setup properly in the tomcat, use the following link to access the welcome page of the JUDDI.

http://localhost:8080/juddiv3

Screen shown in Figure 5.2 will appear.



Figure 5.2: jUDDI Welcome Page

5.1.2 jUDDI Server Implementation Details

Functions handle the actual logic for each UDDI invocation message. The `org.apache.juddi.function` package defines the function classes—one for each logical UDDI invocation message. The `org.apache.juddi.registry.RegistryEngine` class uses the `org.apache.juddi.function.FunctionMaker` class to lookup functions, based on the class name of the function. `FunctionMaker` keeps a cache of instances of the maker classes.

jUDDI uses Apache Axis to handle SOAP messaging. Axis defines a transparent transport framework that allows different transport protocols to be used. For the HTTP protocol, any servlet derived from the `org.apache.axis.transport.http.AxisServlet` class is a candidate for handling HTTP requests. In jUDDI, three servlets extend the `AxisServlet` class:

- `org.apache.juddi.transport.axis.AdminServlet`
- `org.apache.juddi.transport.axis.PublishServlet`
- `org.apache.juddi.transport.axis.InquiryServlet`

This all seems quite straightforward, however, there is a slight twist—jUDDI registers these three classes as servlets with an application server, but only uses them to determine the type of request that is made. The actual processing is handled by the `org.apache.juddi.transport.axis.AxisHandler` class which must be registered with the Axis handler-chain. Figure 5.3 shows a flowchart that illustrates the process for a typical request.

5.1.3 jUDDI Data Structures

JUDDI encapsulates the primary UDDI data structures (`businessEntity`, `businessService`, `bindingTemplate` and `tModel`) in classes following the `ValueObject` pattern. The classes are found subordinate to the `org.apache.juddi.datatype` package as follows:

- `org.apache.juddi.datatype.business.BusinessEntity`
- `org.apache.juddi.datatype.service.BusinessService`
- `org.apache.juddi.datatype.binding.BindingTemplate`
- `org.apache.juddi.datatype.tmodel.TModel`

Instances of each of these classes (along with all other UDDI data types) are acted on by jUDDI handlers and functions in order to process client requests, as shown in Figure 5.4.

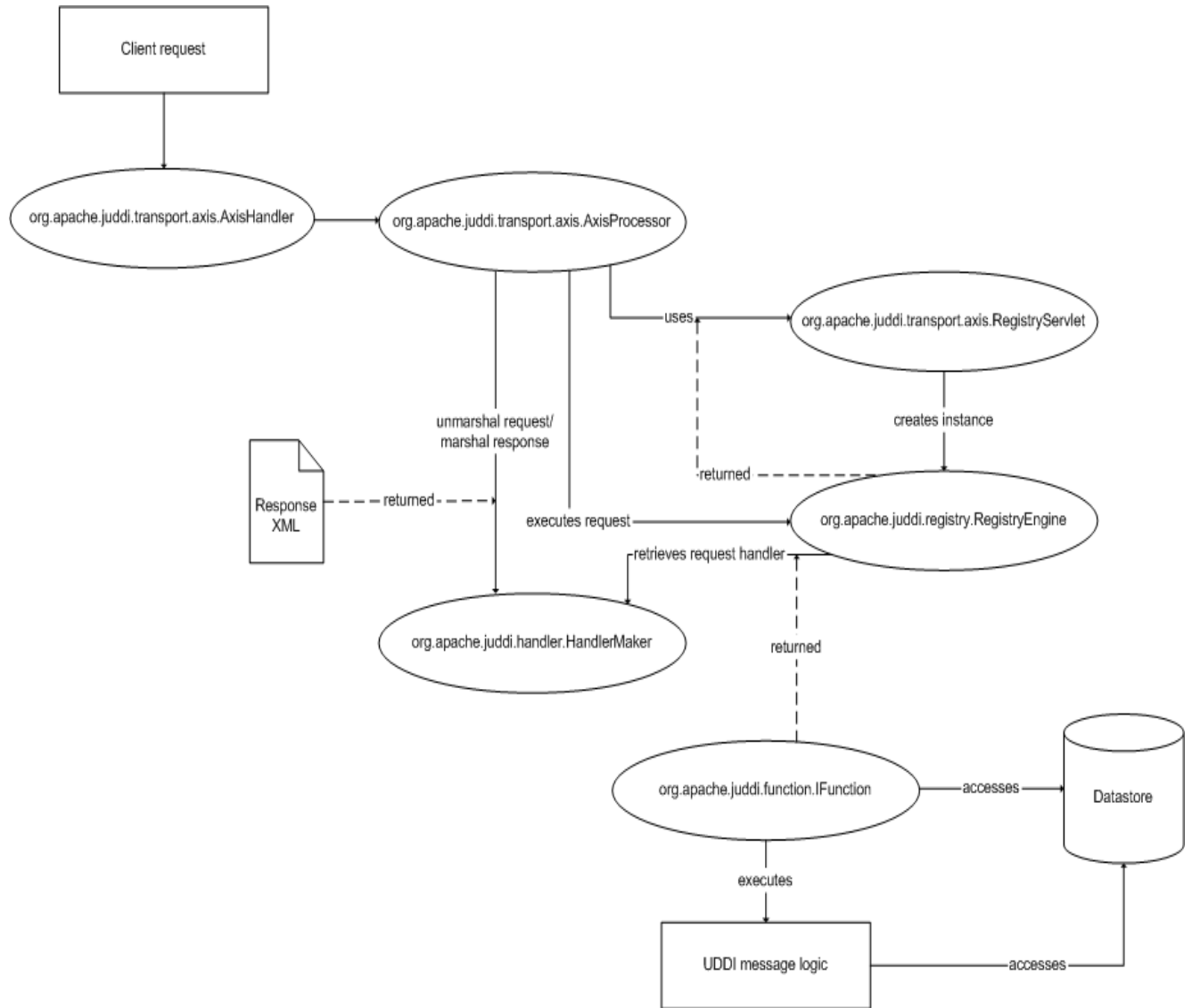


Figure 5.3: Request Process Flow Diagram

5.1.4 Handling Publication Requests with jUDDI

The JUDDI framework dispatches request messages through an Axis handler object named AxisHandler. The AxisHandler class uses the services of the RegistryEngine class to do the actual request processing. A URL mapping for a specific child of AxisServlet is used to classify *inquiry* requests, *publish* requests, and *admin* requests.

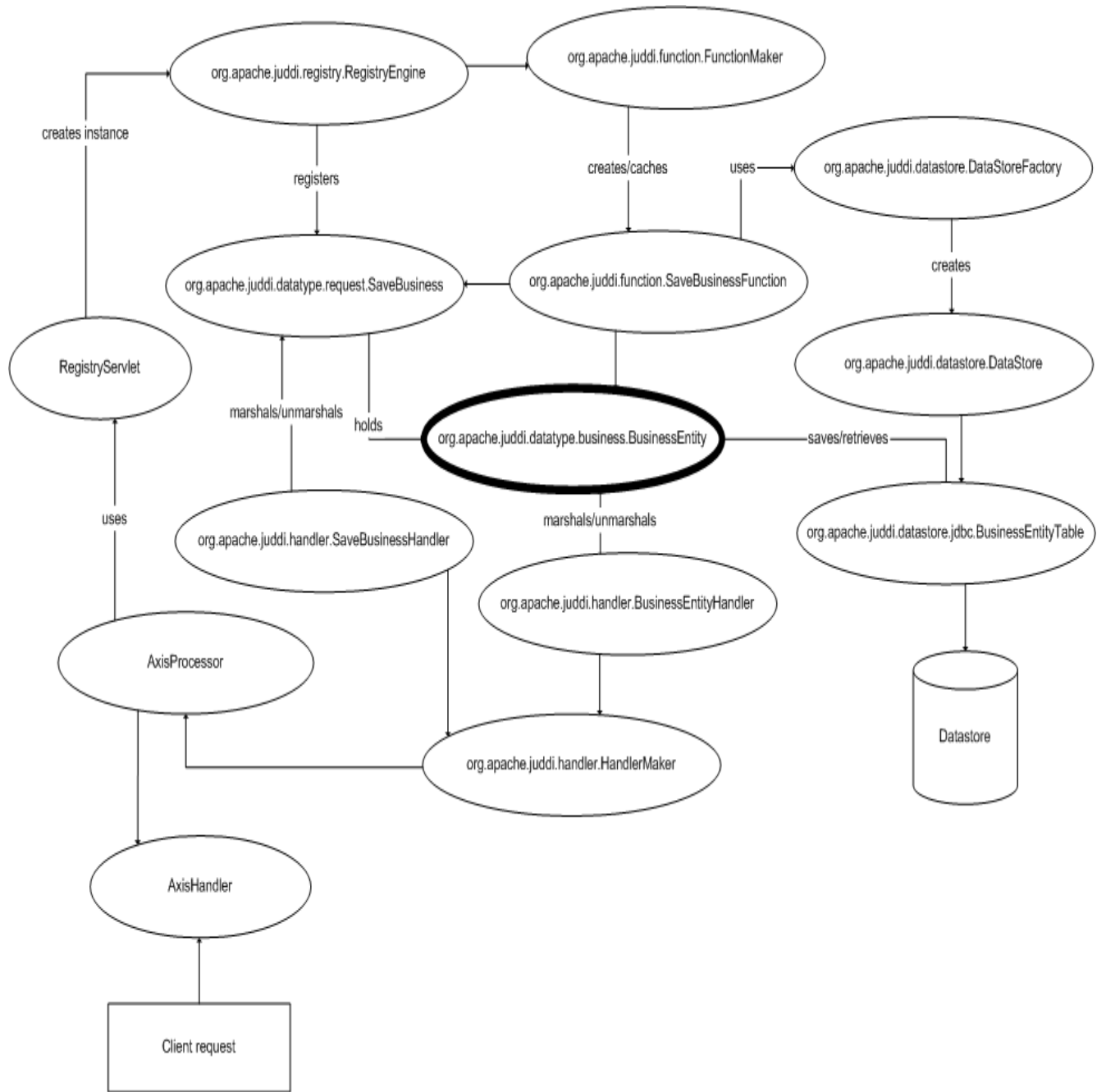


Figure 5.4: Process Flow for a Save_Business Request

The framework classifies each request according to the property value set in the request's MessageContext object for the transport.http.servlet key. Thus, the framework maps the following URL to the PublishServlet:

`http://localhost:8080/juddiv3/publish`

After receiving a request, the RegistryEngine class converts the XML-based UDDI request to Java objects (a process called *unmarshalling*), invokes the appropriate Java objects, and converts Java objects to XML-based responses (called *marshalling*).

5.1.5 Handling Inquiry Requests with jUDDI

As with a publish request, a URL mapping for a specific child of AxisServlet is used to classify inquiry requests. The transport.http.servlet property of the request's MessageContext object will return an instance of the InquiryServlet class and therefore be routed accordingly. Thus, jUDDI will map the following URL to the InquiryServlet:

`http://localhost:8080/juddiv3/inquiry`

5.1.6 Handling Authentication Requests with jUDDI

JUDDI uses the org.apache.juddi.auth.AuthenticatorFactory object to create the desired Authenticator instance in order to authenticate a client. AuthenticatorFactory is an implementation of the Factory pattern. Use it to create an implementation of the org.apache.juddi.auth.Authenticator interface. Retrieve the name of the specific Authenticator implementation to create from the "juddi.auth" property value. If pass a null value, then the AuthenticatorFactory creates a default Authenticator implementation "org.apache.juddi.auth.DefaultAuthenticator." The DefaultAuthenticator class applies no restrictions; therefore, it allows all requests. Production systems should supply an implementation of the Authenticator interface that can authenticate callers against an existing authentication system. The Authenticator implementation class is registered in the juddi.properties file.

5.2 RUDDI

5.2.1 Ruddi Characteristics

Ruddi is UDDI client library. UDDI client library implemented by Ruddi™ currently has the following characteristics:

- Ruddi™ provides access to UDDI registries using an expressive pure Java API. No specific knowledge of XML, SOAP or UDDI messaging is required.

- Ruddi™ fully implements the publishing and inquiry UDDI APIs of UDDI V3, V2 and V1.
- Ruddi™ has a tested interoperability with the public Microsoft, SAP and IBM UDDI Business Registries (UDDI V2 and V1 only, as far as V3 is not currently implemented by the public nodes).
- Ruddi™ transparently manages UDDI V3, V2 and V1 messaging. The runtime uses either UDDI V3, V2 or V1 messaging to communicate with a UDDI registry depending on a user-defined profile. As a result, it is possible to write applications that can alternatively interrogate UDDI V3, V2 or V1 registries with no code change.
- Ruddi™ has UDDI-specific collections library allowing writing expressive, strongly typed UDDI applications.
- Ruddi™ has a validation library allowing validating all UDDI data structures according to either the UDDI V2 or V1 specification (V3 under development). For example, a business entity name of 150 characters will be detected as “too long” if the library is configured for validation against the UDDI V1 specification but will be considered valid if the library is configured for validation against the V2 specification.
- Ruddi™ internally automates low-level UDDI interactions. For example, an authentication token will automatically be fetched using the appropriate information defined in a profile whenever a method of the publishing API is invoked.
- Ruddi™ has an extended query API providing a level of interaction equivalent to what JAXR proposes.
- Ruddi™ allows accessing UDDI registry replies as streams that can be used for example as an input to an XSLT processor (for XML => HTML scenarios, for example).
- Ruddi™’s message transport can be managed internally or be delegated to the Apache Axis V1 SOAP engine.
- Ruddi™ has a logging facility allowing monitoring the XML conversation between the UDDI client and the UDDI registry. System.out logging, as well as a Log4J-based and an experimental XML-based logging are supported.

- Ruddi™ is easy to install. Get up to speed in less than 5 minutes. Learn by example with the about 20 examples provided with the library.
- Ruddi™ has extensive documentation.

5.2.2 Ruddi Usage

The following examples demonstrate the most common uses of Ruddi™ to connect to UDDI registries.

- Querying an UDDI registry
- Saving and updating information in an UDDI registry
- Suppressing information from an UDDI registry
- Various Ruddi™ API examples

5.2.2.1 Querying an UDDI registry

- Finds a business entity by name.
- Finds a business service by name.
- Finds the technical models of a business entity.
- Finds the binding details of a business service.
- Gets detailed information on a business entity.
- Searches for business entities belonging to a given NAICS category.
- Finds a business entity by name using the Axis 1.0 SOAP implementation.

5.2.2.2 Saving and updating information in an UDDI registry

- Saves a business entity.
- Saves a business service.
- Saves a binding template.
- Saves a technical model.
- Saves a business entity.

5.2.2.3 Suppressing information from an UDDI registry

- Deletes a business entity.
- Deletes a business service.
- Deletes a technical model.

- Deletes a binding template.

5.2.3 Various Ruddi™ API examples

- Shows how to use the Ruddi™ collections API.
- Shows how to enable and disable logging.
- Shows the validation capabilities of Ruddi™.
- Shows the capabilities of Ruddi™ with regard to keys.
- Shows how Ruddi™ UDDI structures serializers can be used.
- Shows how Ruddi™ can be used to convert V2 structures to V3 structures

5.3 Approximate String Matching

Fuzzy matching is a programmatic process of determining similarity between two strings, such as names, addresses, drug names, materials (as in engineering), parts descriptions, etc. when there is knowledge or suspicion that there is a difference between the two strings, and that they may need to be merged, updated, purged, or simply identified. Optimally, exact matching should precede fuzzy matching. Some of the anomalies between two strings or bodies of text that call for approximate string matching are: Typing mistakes, abbreviations, different data entry conventions, truncation, inconsistencies in data formatting, and a number of others specific to the type of data.

Match entries with typing mistakes:

Divesh Srivastava vs. Divesh Shrivastava

Match entries with abbreviations:

Euroaft Corporation vs. Euroaft Corp.

Match entries with different conventions:

Comp. Sci. Dept. vs. Dept. of Comp. Sci.

Match entries with inconsistent formatting:

010104 vs. 01-01-04

5.3.1 Levenshtein Algorithm

The Levenshtein algorithm calculates the least number of edit operations that are necessary to modify one string to obtain another string. The closeness of a match is measured in terms of the number of primitive operations necessary to convert the string into an exact match.

This number is called the edit distance — also called the Levenshtein distance — between the string and the pattern. The usual primitive operations are:

- insertion (e.g., changing cot to coat),
- deletion (e.g. changing coat to cot)
- substitution (e.g. changing coat to cost).

Here is an example that features the comparison of "meilenstein" and "levenshtein":

		m	e	i	l	e	n	s	t	e	i	n
l	0	1	2	3	4	5	6	7	8	9	10	11
e	1	1	2	3	3	4	5	6	7	8	9	10
v	2	2	1	2	3	3	4	5	6	7	8	9
e	3	3	2	2	3	4	4	5	6	7	8	9
n	4	4	3	3	3	3	4	5	6	6	7	8
s	5	5	4	4	4	4	3	4	5	6	7	7
h	6	6	5	5	5	5	4	3	4	5	6	7
t	7	7	6	6	6	6	5	4	4	5	6	7
e	8	8	7	7	7	7	6	5	4	5	6	7
i	9	9	8	8	8	7	7	6	5	4	5	6
n	10	10	9	8	9	8	8	7	6	5	4	5
	11	11	10	9	9	9	8	8	7	6	5	4

Figure 5.5: Levenshtein Algorithm

5.3.2 Dice's coefficient Algorithm

Dice's coefficient, named after Lee Raymond Dice and also known as the Dice coefficient, is a similarity measure related to the Jaccard index. For sets X and Y of keywords used in information retrieval, the coefficient may be defined as:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

When taken as a string similarity measure, the coefficient may be calculated for two strings, x and y using bigrams as follows:

$$s = \frac{2n_t}{n_x + n_y}$$

Where n_t is the number of character bigrams found in both strings, n_x is the number of bigrams in string x and n_y is the number of bigrams in string y .

5.3.3 Longest Common Subsequence Algorithm

The longest common subsequence (LCS) problem is to find the longest subsequence. For example, here are two sequences having the same last element: (BANANA) and (ATANA).

- Remove the same last element. Repeat the procedure till you find no common last element. The removed sequence will be (ANA).
- The sequences now under consideration: (BAN) and (AT).
- The LCS of these last two sequences is, by inspection, (A).
- Append the removed element, (ANA), giving (AANA), which, by inspection, is the LCS of the original sequences.

5.4 WSDL4J

The Web Services Description Language for Java Toolkit (WSDL4J) allows the creation, representation, and manipulation of WSDL documents. Is the reference implementation for JSR110 'JWSDL' (jcp.org).

The IBM reference implementation of JSR-110 (Java APIs for WSDL), Web Services Description Language for Java Toolkit (WSDL4J) allows the creation, representation, and manipulation of WSDL documents.

Chapter 6 : Results and Discussion

Measuring the performance of web services mining framework is non-trivial. Generally a framework is evaluated by implementing the framework and then using a dataset to test the web services mining based on calculating Precision, Recall and F-measure. The fundamental factors for web service quality evaluation can be largely divided into static, dynamic and statistical factors. Static factors do not change as long as no changes occur within the service since they are dependent to the service in concern. Meanwhile, dynamic factors represent quality information that changes according to certain situations such as network traffic. Statistical factors are evaluated based on the statistical data of the service.

6.1 System Requirements

System prototype is developed using Netbeans 6.8 and java development kit 6 so for running this software prototype there is a requirement of Java Runtime Environment 6 and database handling is done using MySql 5.0 which must be installed and database should be configured for proper running of this software. In tabular form ideal requirements for this prototype are given in Table 6.1.

Table 6-1: System Requirements

System Processor	2.4 GHz
Hard Disk	40 GB
RAM	1 GB
Operating System	Windows 2000 Server, Windows 2003 Server, Windows XP
Runtime Environment	Java Runtime Environment 6
Database Server	MySql 5.0
Application Server	Tomcat 6.0

6.2 Evaluation Criteria

Following are main criteria on which proposed approach is evaluated and compared to existing techniques:

- Number of Services Discovered and Composed
- Precision
- Fallout
- F Measure
- Mining time of Services
- Composition time of Services

6.2.1 Number of Services Discovered and Composed

Proposed framework is tested with different approximate string matching algorithms. Following testing parameters are used for each algorithm:

- **Levenshtein Algorithm:** In Table 6.2 distances are defined for different weight ranges which are used in testing of proposed framework.

Table 6-2: Levenshtein Algorithm Parameters

Weight Range	Distance
0 – 0.3	0
0.3 – 0.6	1
0.6 - 1	2

- **Dice's Coefficient Algorithm:** In Table 6.3 different string matching ranges are given for weight ranges. This table is used as input rules for testing of proposed web service mining framework.

Table 6-3: Dice's Coefficient Algorithm

Weight Range	String Matching
0 – 0.3	0.8 - 1
0.3 – 0.6	0.7 – 0.8
0.6 - 1	0.6 – 0.7

- **Longest Common Subsequence Algorithm:** Fuzzy rules for Longest Common Subsequence Algorithm are given in Table 6.4 where string distance is given for different weight ranges.

Table 6-4: Longest Common Subsequence Algorithm

Weight Range	Distance
0 – 0.3	String Length - 0
0.3 – 0.6	String Length - 1
0.6 - 1	String Length - 2

6.2.2 Precision

Precision is the proportion of services that satisfies users' request in all the discovered services.

$$\text{Precision} = \frac{\text{Number of Relevant Services Retrieved}}{\text{Number of Retrieved Services}}$$

6.2.3 Recall

Recall is the fraction of the web services, which are relevant to the request, that are successfully retrieved.

$$\text{Recall} = \frac{\text{Number of Relevant Services Retrieved}}{\text{Number of Relevant Services}}$$

6.2.4 F-measure

This is the weighted harmonic mean of precision and recall. It trades off between precision and recall.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \text{ where } \beta^2 = \frac{1 - \alpha}{\alpha}$$

- Where F is F-measure, P is precision and R is recall

- The default well adjusted F-measure that fairly weights precision and recall uses the parameters

$$\alpha = 1/2 \text{ or } \beta = 1$$

Table 6-5: Static Evaluation Factors for Web Service

Factor	Description
Regulatory	What is the standard that the web service follows?
Security	Does the service abide by security factors such as WS-Security?

Table 6-6: Dynamic Factors for Evaluation of Web Service

Factor	Description
Service Availability	Is the service working properly?
Network Availability	How fast is the service dynamic network speed?
Execution Duration	How long does it take to receive a reply after requesting the service?

Table 6-7: Statistical Factors for Evaluation of Web Service

Factor	Description
Service Reliability	How stable is the operation of the service?
Network Reliability	How stable was the service network?
Execution Reliability	How frequently is the reply sent back within a standard period of time?
Reputation	How good is the reputation of the service compared with other services of the same type?

6.3 Dataset

The framework is implemented in Java 6 using Netbeans 6.8 integrated development environment. Apache jUDDI v3 is used to setup UDDIs. Apache jUDDI is an open source universal description discovery and integration. Apache Tomcat 6 is used to host the Juddi. RUDDI API is used to access Juddi from Java. WSDL4J (Web Service Description Language for Java) is used to parse the WSDL files that are used to describe the web service choreography and orchestration interfaces.

6.4 Performance Evaluation

The performance of the proposed approach is evaluated using all of the factors discussed above. Framework is tested for web services mining and logged the values for Precision, Recall and F-measure. Also, compared these values with the existing frameworks and show where proposed framework has improved the web services mining. After discovery, the services are available for composition. Mining time for UDDI registries of different sizes is recorded. At the end, comparison is given with an existing technique to present the improvements of proposed framework.

6.4.1 Number of Services Discovered and Composed

Following are the web services mining results using different fuzzy string matching algorithm. Table 6.8 lists the results of services discovered and composed, using proposed technique and compared it with the existing framework ^[7]. Table 6.8 provides results for registries of different sizes and from the table it clearly depicts that proposed framework has better results than existing approach.

In first column, total no of services are given that are used in testing of proposed framework. When the total no of services are 500, services discovered using Levenshtein algorithm are 6 and only 1 composition is formed. Similarly services discovered using Dice's Coefficient algorithm are 7 and only 1 composition is formed. Longest Common Subsequence algorithm has discovered 9 services and again only 1 composition is formed using these discovered services. Zheng approach [7] has discovered only 4 services and no composition is formed. Finally, when the total no of services are 5000, services discovered using Levenshtein algorithm are 19 and 3 compositions are formed. Similarly services discovered using Dice's Coefficient algorithm are 24 and 3 compositions are formed. Longest Common Subsequence algorithm has discovered 28 services and 4 compositions are formed using these discovered services. Zheng approach [7] has discovered only 8 services and 2 compositions are formed.

It is concluded from Table 6.8 that all the algorithms used in proposed approach gives better results. Particularly, Longest Common Subsequence algorithm has best result as compared to other algorithms and previous approaches.

Table 6-8: Mining Results

	Proposed Approach						Zheng Approach	
UDDI Registry	Levenshtein Algorithm		Dice's Coefficient Algorithm		Longest Common Subsequence Algorithm			
Number of Services	Number of Services Discovered	Number of Compositions Formed	Number of Services Discovered	Number of Compositions Formed	Number of Services Discovered	Number of Compositions Formed	Number of Services Discovered	Number of Compositions Formed
500	6	1	7	1	9	1	4	0
1000	7	1	8	1	9	1	4	0
1500	7	1	8	1	11	2	4	0
2000	9	2	10	2	11	2	5	1
2500	10	2	13	2	15	3	5	1
3000	13	2	13	2	17	5	5	1
3500	15	3	16	3	20	5	6	2
4000	15	3	17	4	24	6	6	2
4500	18	4	20	4	26	7	7	2
5000	19	5	24	5	28	7	8	2

All the values given in Table 6.8 are plotted in Figure 6.1. On x-axis, total numbers of services used in testing are given and on y-axis, numbers of services discovered and composed are plotted.

On x-axis total numbers of services ranging from 0 to 5000 are plotted. Scale of 500 is used on x-axis. While on y-axis different line formats are used to differentiate between plotted values. The chart given in Figure 6.1 clearly shows that total numbers of services discovered by longest common subsequent algorithm are higher than discovered service by other algorithms. Finally, when these discovered services are further used in composition process, the numbers of compositions formed are also higher than other approaches.

6.4.2 Average Precision

Various sets of services are taken and for each set 10 readings are made and then computed an average for that set. Testing is started with a service set of 500 web services and then keep on increasing the number of web services to 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 and finally 5000. In Table 6.9 average precision of proposed framework is given.

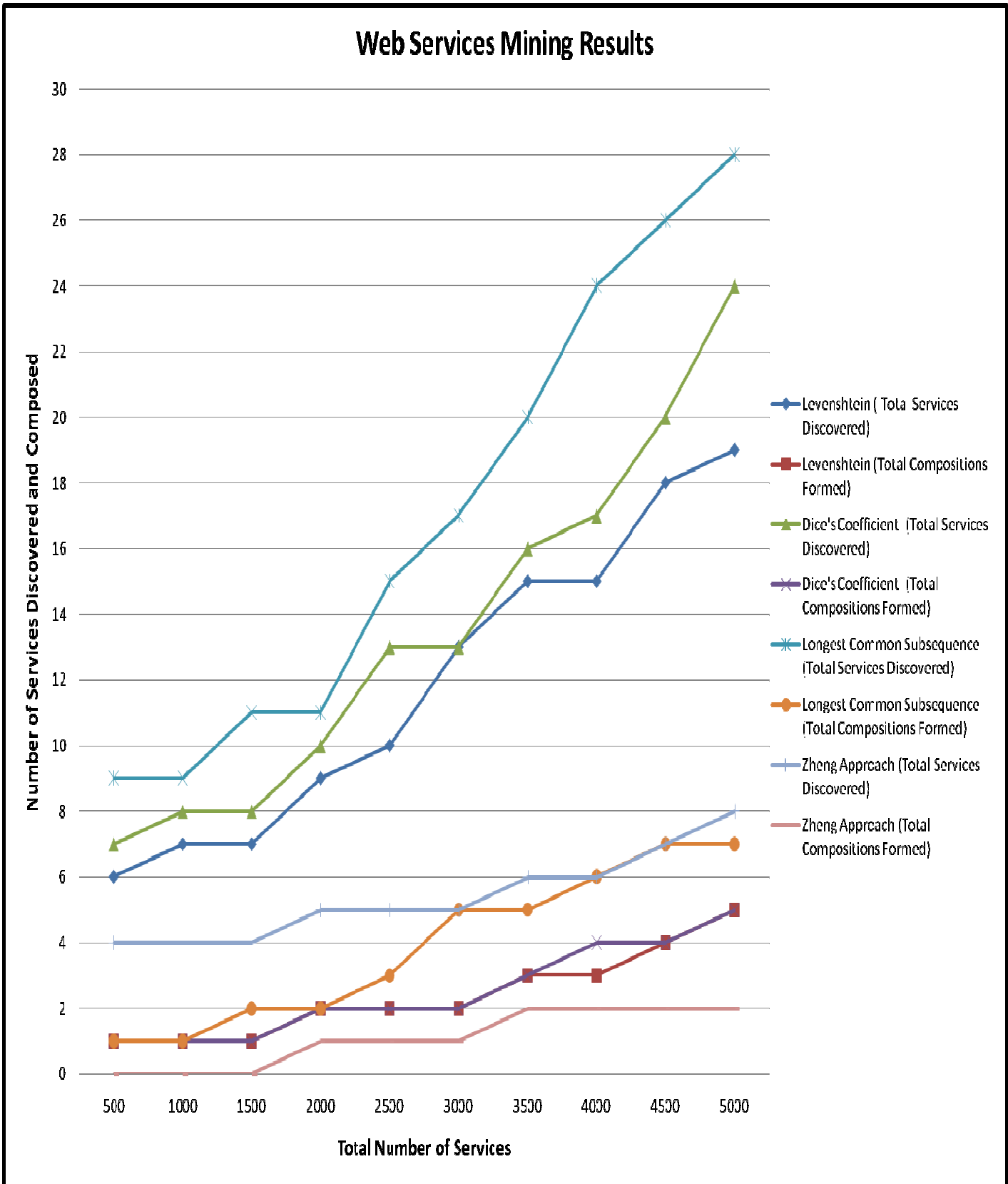


Figure 6.1: Mining Results

Average precision is calculated by dividing number of relevant services retrieved with number of retrieved services. The test is repeated on multiple sets of services and finally an average value is calculated using following algorithms.

- Levenshtein Algorithm
- Dice's Coefficient Algorithm
- Longest Common Subsequence Algorithm

Dice's coefficient algorithm is more precise as compared to Levenshtien algorithm and Longest Common Subsequence algorithm.

Table 6-9: Average Precision

UDDI Registry	Levenshtein Algorithm	Dice's Coefficient Algorithm	Longest Common Subsequence Algorithm
Number of Services	Average Precision %	Average Precision %	Average Precision %
500	100	100	88
1000	100	100	88
1500	100	100	82
2000	88	100	82
2500	90	92	80
3000	84	92	76
3500	80	87	75
4000	80	87	75
4500	78	85	73
5000	78	83	71

All the values of average precision given in Table 6.9 are plotted in Figure 6.2. On x-axis, total numbers of services ranging from 0 to 5000 are plotted. Scale used on x-axis is 500. On y-axis average precision ranging from 0 to 100 is plotted.

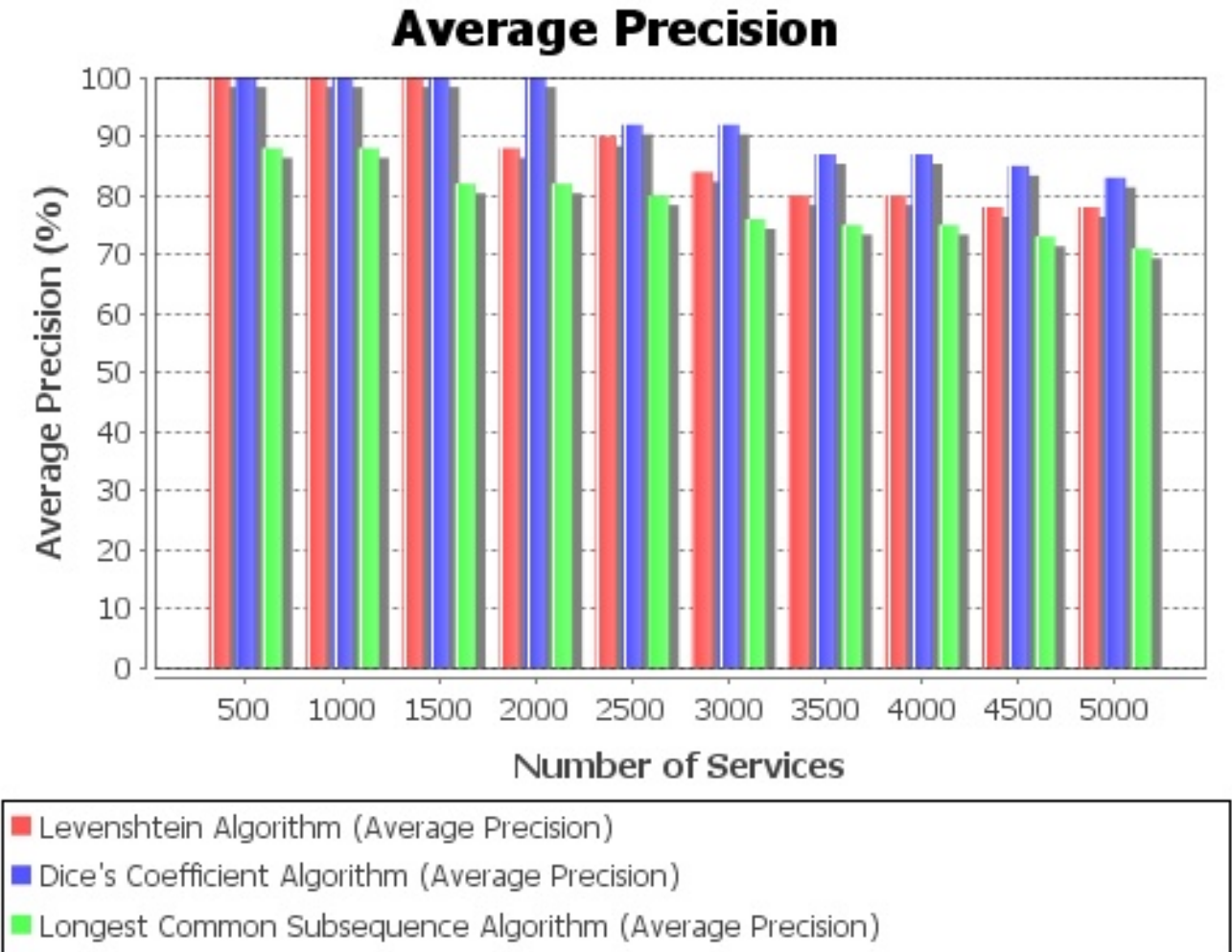


Figure 6.2: Average Precision

6.4.3 Average Recall

Various sets of services are taken and for each set 10 readings are made and then computed an average for that set. Testing is started with a service set of 500 web services and then keep on increasing the number of web services to 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 and finally 5000.

Average recall is calculated by dividing number of relevant services retrieved with number of relevant services. The numbers of relevant services are calculated by the domain expert based on knowledge of domain and expertise.

Average recall of proposed framework is given in Table 6.10. As already given that proposed approach is based on fuzzy set and fuzzy rules. Also approximate string matching algorithms are used that's why average recall of proposed framework is 100%.

Table 6-10: Average Recall

UDDI Registry	Levenshtein Algorithm	Dice's Coefficient Algorithm	Longest Common Subsequence Algorithm
Number of Services	Average Recall %	Average Recall %	Average Recall %
500	100	100	100
1000	100	100	100
1500	100	100	100
2000	100	100	100
2500	100	100	100
3000	100	100	100
3500	100	100	100
4000	100	100	100
4500	100	100	100
5000	100	100	100

All the values of average recall given in Table 6.10 are plotted in Figure 6.3. In figure 6.3, average recall is plotted on y-axis and on x-axis total numbers of services are given. On x-axis total number of services ranging from 0 to 5000 on a scale of 500 is plotted, whereas on y-axis average precision ranging from 0 to 100 is plotted.

6.4.4 Average F-measure

F-measure is the weighted harmonic mean of precision and recall. It trades off between precision and recall. Table 6.11 shows F-measure for UDDI registries of different sizes.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \text{ where } \beta^2 = \frac{1 - \alpha}{\alpha}$$

- Where F is F-measure, P is precision and R is recall
- The default well adjusted F-measure that fairly weights precision and recall uses the parameters

$$\alpha = 1/2 \text{ or } \beta = 1$$

Average Recall

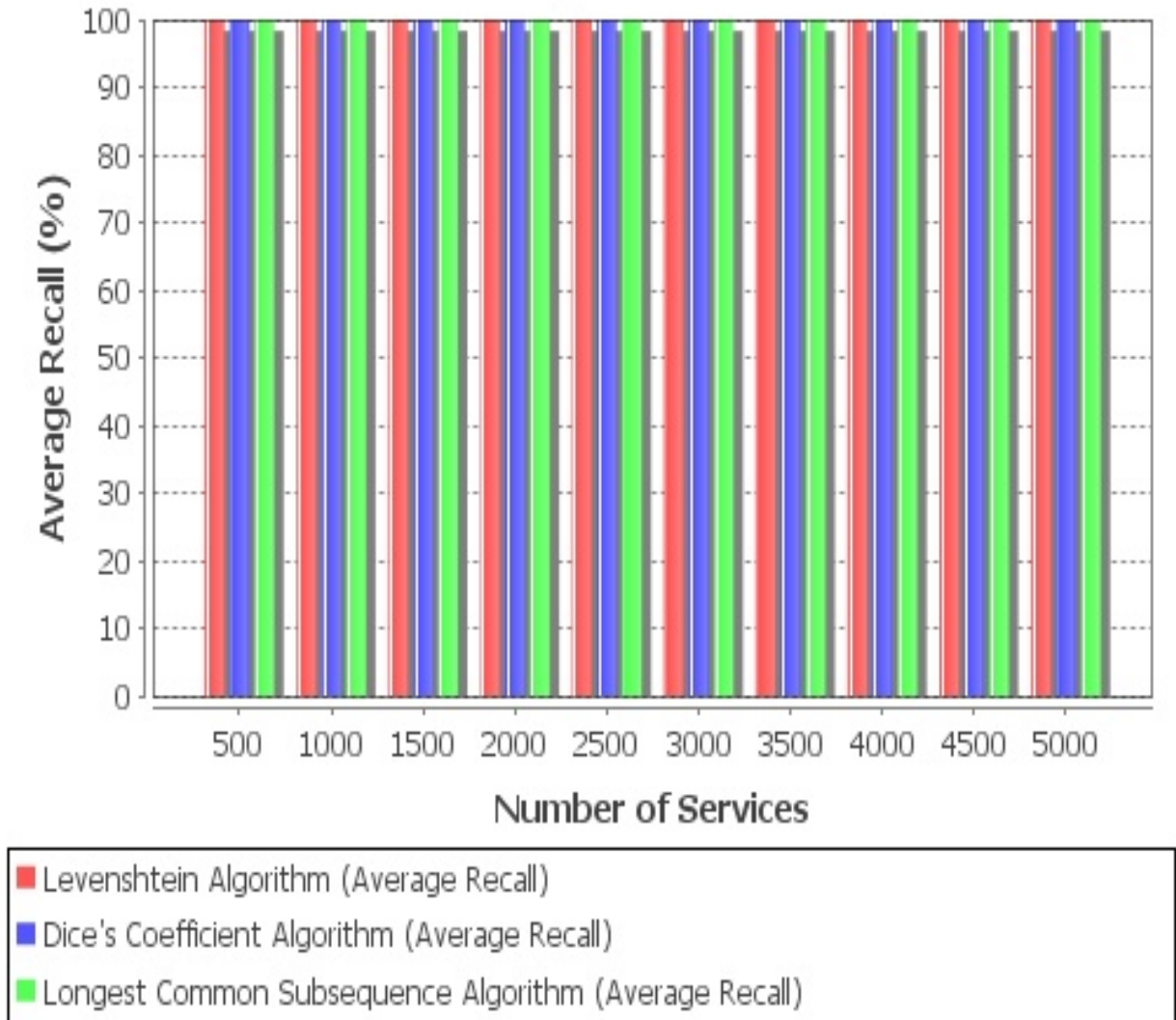


Figure 6.3: Average Recall

Table 6-11: Average F Measure

UDDI Registry	Levenshtein Algorithm	Dice's Coefficient Algorithm	Longest Common Subsequence Algorithm
Number of Services	Average F Measure %	Average F Measure %	Average F Measure %
500	100	100	94
1000	100	100	94
1500	100	100	90
2000	94	100	90
2500	95	96	89
3000	91	96	86
3500	89	93	85
4000	89	93	85
4500	88	92	85
5000	88	92	85

In Table 6.11, it is given that average f-measure is 100% for small testing data set but as data set size is increasing, value of f-measure is decreasing. Dice's coefficient algorithm has better f-measure value as compared to Levenshtein algorithm and Longest Common Subsequence algorithm.

All the values of f-measure given in Table 6.11 are shown in graphical format in Figure 6.4. On x-axis total numbers of services ranging from 0 to 5000 are plotted using a scale of 500. Whereas on y-axis average f-measure given in Table 6.11 ranging from 0 to 100 is plotted. A scale of 10 is used on y-axis. Different colors are used to differentiate between values of Dice's coefficient algorithm, Levenshtein algorithm and Longest Common Subsequence algorithm.

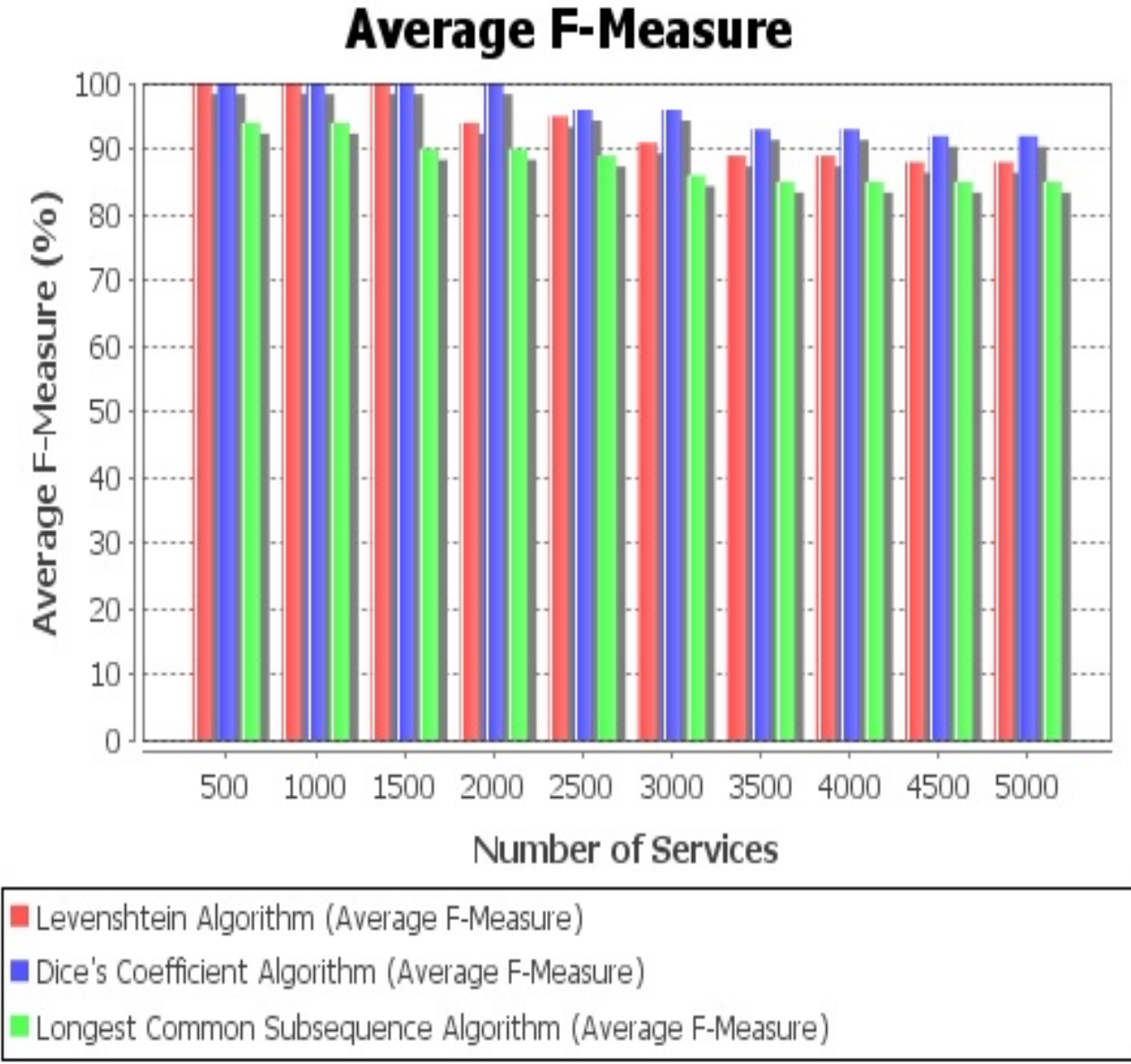


Figure 6.4: Average F-measure

6.4.5 Evaluation Time of Services

WSDL4J is used to parse the WSDL file of the web service. Once the service is discovered, one must know the methods that it presents to be used from outside world. Evaluation time of web services is logged for various numbers of methods exposed by the web services. Once again randomly evaluated the web services and then logged the timings. Following is the analysis of the evaluation time given in Table 6.12.

Table 6-12: Evaluation Time of Web Services

	Proposed Approach						Zheng Approach
UDDI Registry	Levenshtein Algorithm		Dice's Coefficient Algorithm		Longest Common Subsequence Algorithm		
Number of Services	Time (ms) Single Thread	Time (ms) Multithreaded	Time (ms) Single Thread	Time (ms) Multithreaded	Time (ms) Single Thread	Time (ms) Multithreaded	
500	2250	500	2200	450	2400	600	2500
1000	2330	550	2270	500	2500	650	2850
1500	2410	610	2350	560	2610	720	3010
2000	2490	680	2420	630	2700	800	3200
2500	2600	750	2500	700	2820	900	3430
3000	2730	820	2600	780	2950	990	3600
3500	2855	900	2710	850	3050	1080	3915
4000	2970	1000	2820	930	3180	1150	4250
4500	3090	1080	2940	1000	3300	1280	4410
5000	3200	1150	3050	1080	3450	1350	4720

Proposed framework is tested using single thread and multithreaded approach and time is calculated in mili seconds for Dice’s coefficient algorithm, Levenshtein algorithm and Longest Common Subsequence algorithm. Time for all these algorithms is noted and compared with Zheng approach [7]. Using multithreaded approach proposed framework has much better computation time as compared to existing approach.

All the values given in Table 6.12 are plotted in Figure 6.5. Again on x-axis, total numbers of services ranging from 0 to 5000 are plotted and on y-axis web services mining time is given in mili seconds.

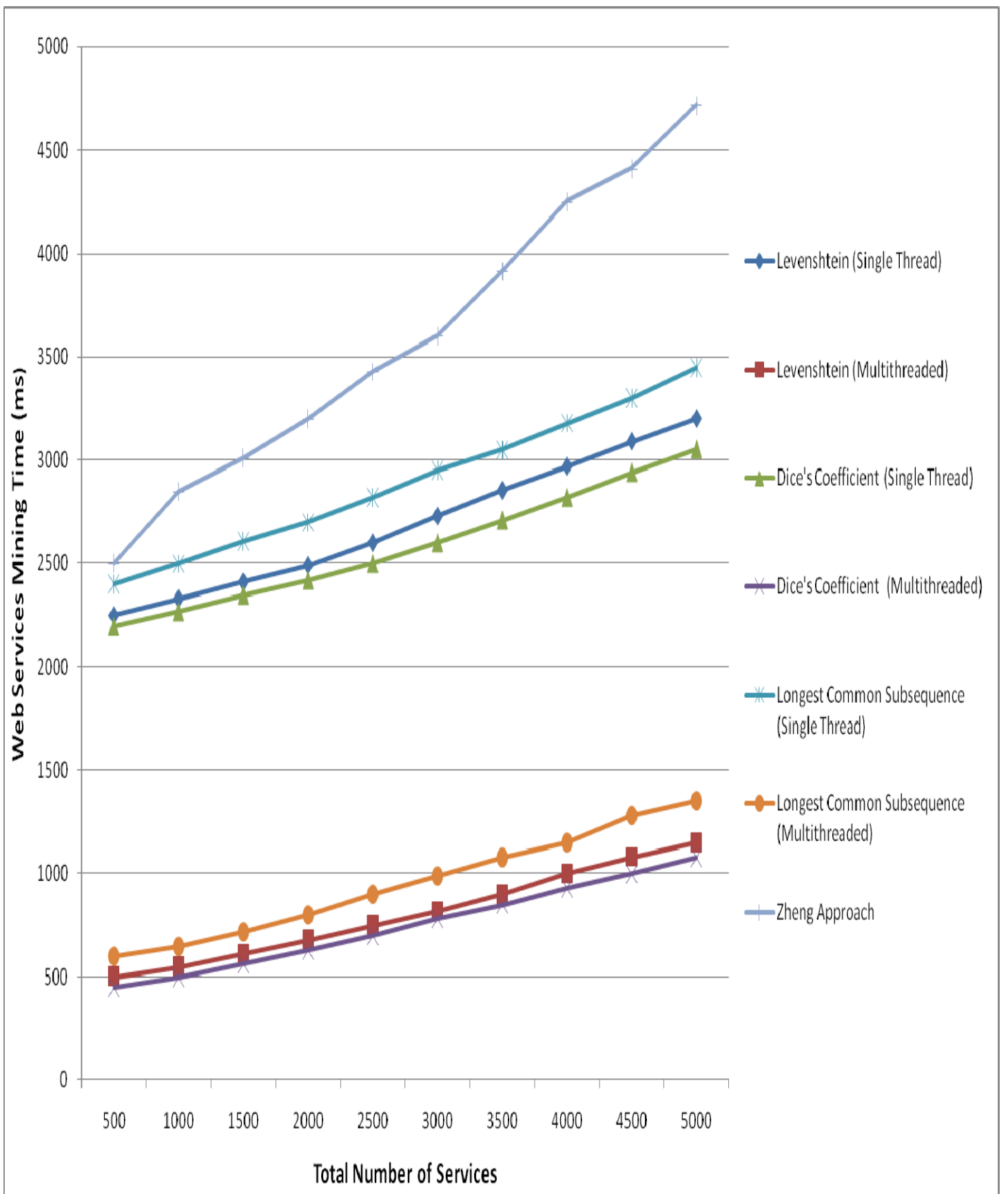


Figure 6.5: Evaluation Time of Web Services

Chapter 7 : Conclusion and Future Work

7.1 Overview of Research

A mining framework for “web services based on fuzzy set and constraint satisfaction” is proposed that proactively uncovers the interesting and useful individual web services and composes existing web services into composite web services. Weights are assigned to fuzzy set on the basis of probability calculation, in order to optimize the mining process and provide more efficient results as long as system matures. The framework is scalable with the growing number of web services repositories and provides efficient mining results. To mine huge web service repositories efficiently, multithreaded approach is applied where a separate thread is initiated for every member of the fuzzy set. This parallel processing approach for mining web services has improved the performance and made the framework scalable with growing web service search space. Mining queries and results are stored and managed locally with the system and are used for future probability calculation and are first locally discovered. Web service mining results are sorted and indexed based on weights assigned and further filtered in the evaluation phase. Future works include adding the pre and post screening phases for mining more relevant web services and includes the ontology based fuzzy mining of web services.

7.2 Achievements

In this thesis a “web services mining algorithm” is proposed to solve the mining issues related to data distribution, reliability, availability and QOS. A framework is proposed by combination of interface based rules. The proposed framework solves the issues related to unavailability of updated information and inaccessibility of web services from repository/databases due to any fault/failure. In proposed framework, multiple repositories and WSDB's have been introduced in order to make system more reliable and ensure data availability. By using multiple registries, data availability is guaranteed, whereas, by using aging factor user's can retrieve up to date information. It solves unavailability of updated information problem by adding aging factor in repository/WSDB(Web Services Database). Finally, algorithm eliminates the dynamic service composition issues, supports web service composition considering QOS(Quality of Services), efficient data retrieval and updating, fast service

distribution and fault tolerance. The proposed system is fault tolerant, reliable, performs fast data retrieval and Quality of services based.

7.3 Limitations

In this short paper discussion is around distributed technologies, execute ability issues, data distribution, QoS issues and how to avoid problems with execute ability issues. At this stage, automated web services mining process is still under development, although some automated tools and proposals are available. The full automation of this mining process is still an ongoing research activity.

7.4 Future Work

Nothing is perfect in this world and no work is ever perfect and there is always room for improvement. Similarly, in this, although a lot of hard work is done but still it can be further optimized and improved, providing more functionality. This step opens the path for others to march on. In future, the framework can be extended by Crawling the web for searching web services instead of querying the UDDI registries. Also when looking into deeper details of every component of the framework to ensure better and efficient mining.

Recent advancement in web services plays an important role in business to business and business to consumer interaction. In order to find a suitable service, discovery mechanism is used. Through discovery mechanism collaboration between service providers and consumers becomes possible by using standard protocols. A static web service discovery mechanism is not only time consuming but requires continuous human interaction. This paper presents a framework for automatic, web services mining. The framework is flexible, scalable and new services can easily be inserted and updated in local cache and UDDI registries.

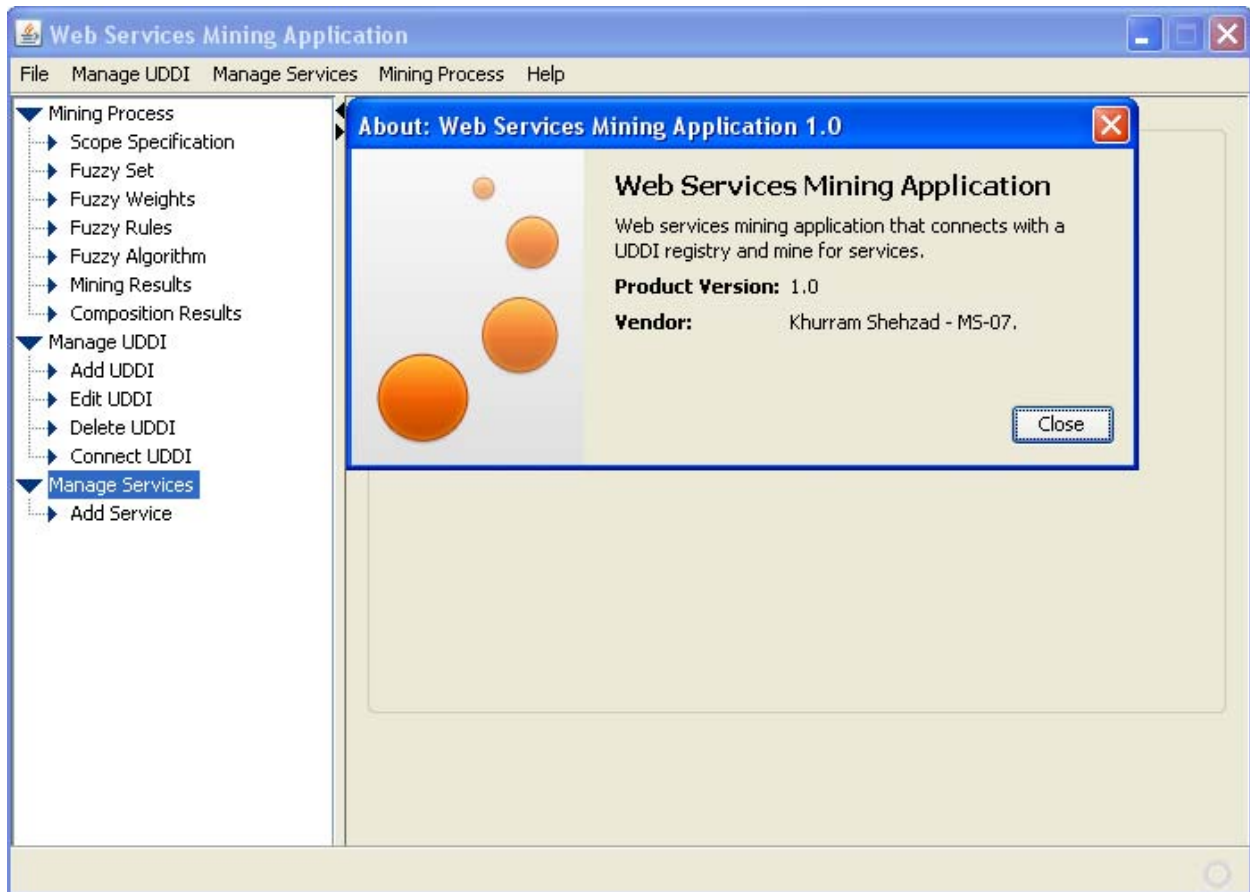
APPENDIX A

User Manual

Interfaces are necessary part of any system as these are used by the end users for interaction with the system. Therefore the more interactive, easy and user friendly the interfaces would be, the easier it would be for the end user to communicate use the system. In this section the screen shots of proposed web services mining tool are given with details.

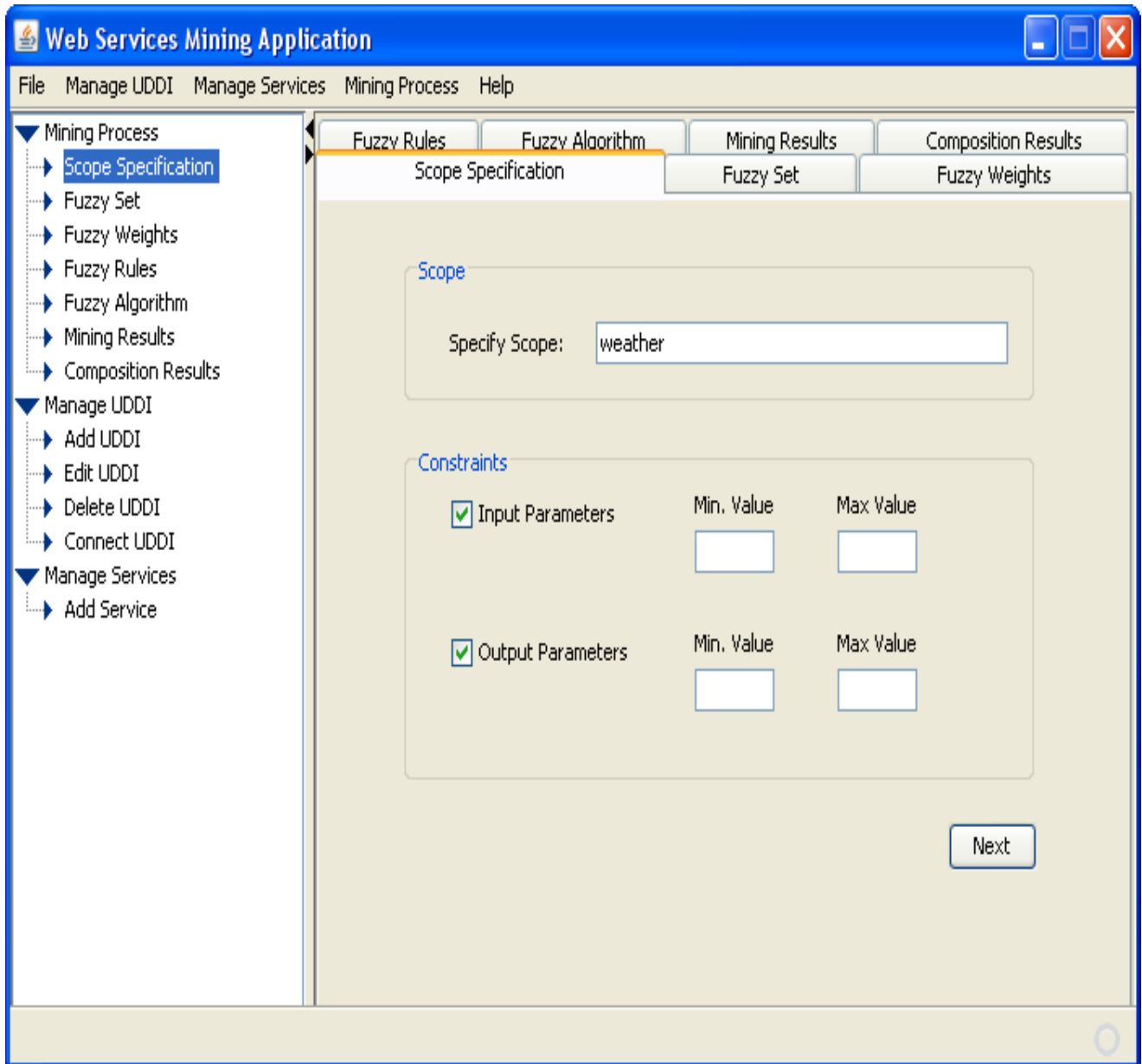
Main Screen

Double click the executable jar file to run the application or run it from Netbeans 6.8 IDE. The main page of the application appears as seen below. Left side of the main page shows the menu with different available options. The right part of the screen shows the detail panel where about page is displayed at start.



Scope Specification

Scope is specified by the domain expert on this screen. A screen shot of scope specification is given below. User also defines constraints on the input and output parameters in form of ranges. These rules are matched in filtering phase where services are filtered out.



Fuzzy Set

Below, screen shows a fuzzy set generated based on the scope specified by domain expert. This screen shows a list of fuzzy members with their description. These fuzzy members are used in the mining process.

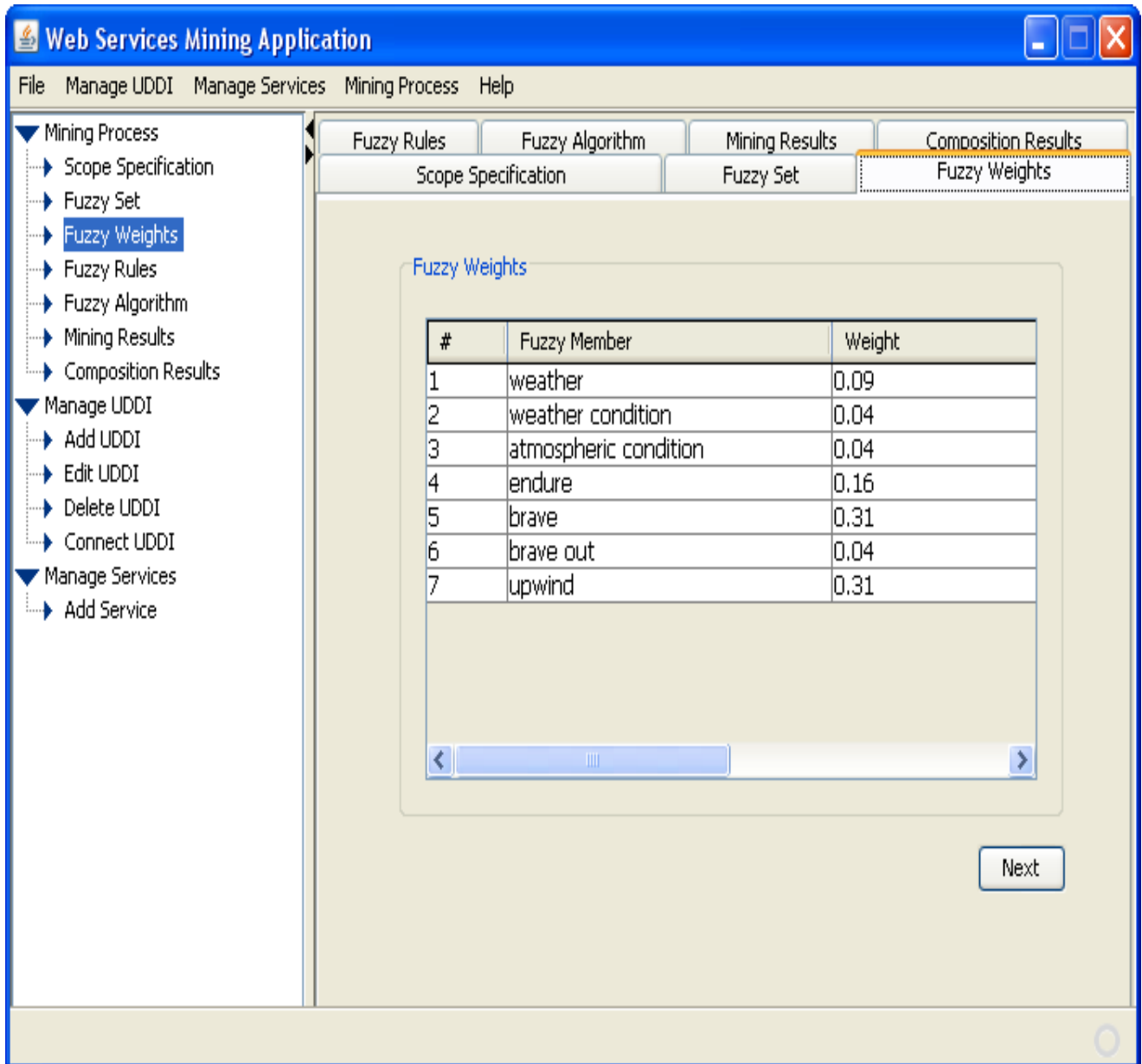
The screenshot displays the 'Web Services Mining Application' interface. The 'Mining Process' menu is expanded, and the 'Fuzzy Set' option is selected. The main window shows a table of fuzzy members with the following data:

#	Fuzzy Member	Description
1	weather	weather
2	weather condition	the meteorological con
3	atmospheric condition	the meteorological cor
4	endure	face or endure with co
5	brave	face or endure with co
6	brave out	face or endure with co
7	upwind	towards the side expo

A 'Next' button is visible at the bottom right of the main window.

Fuzzy Weights

Below, screen shows fuzzy weights for each member of fuzzy set. These weights are calculated on basis of probability from the local database. In next step fuzzy rules are defined on basis of these weights.



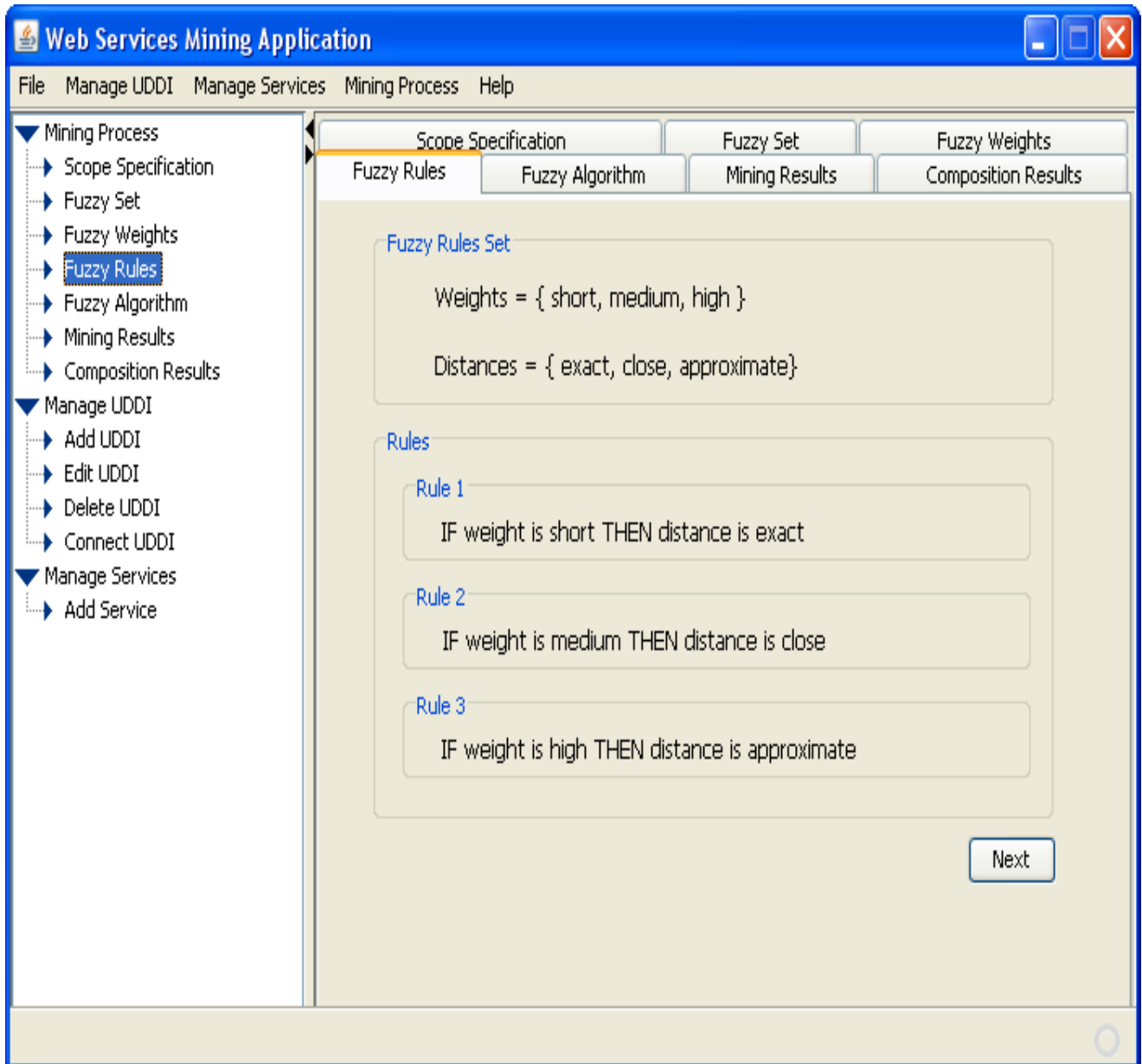
The screenshot shows the 'Web Services Mining Application' window. The 'Mining Process' tree view on the left is expanded to 'Fuzzy Weights'. The main content area displays a table titled 'Fuzzy Weights' with the following data:

#	Fuzzy Member	Weight
1	weather	0.09
2	weather condition	0.04
3	atmospheric condition	0.04
4	endure	0.16
5	brave	0.31
6	brave out	0.04
7	upwind	0.31

A 'Next' button is located at the bottom right of the main content area.

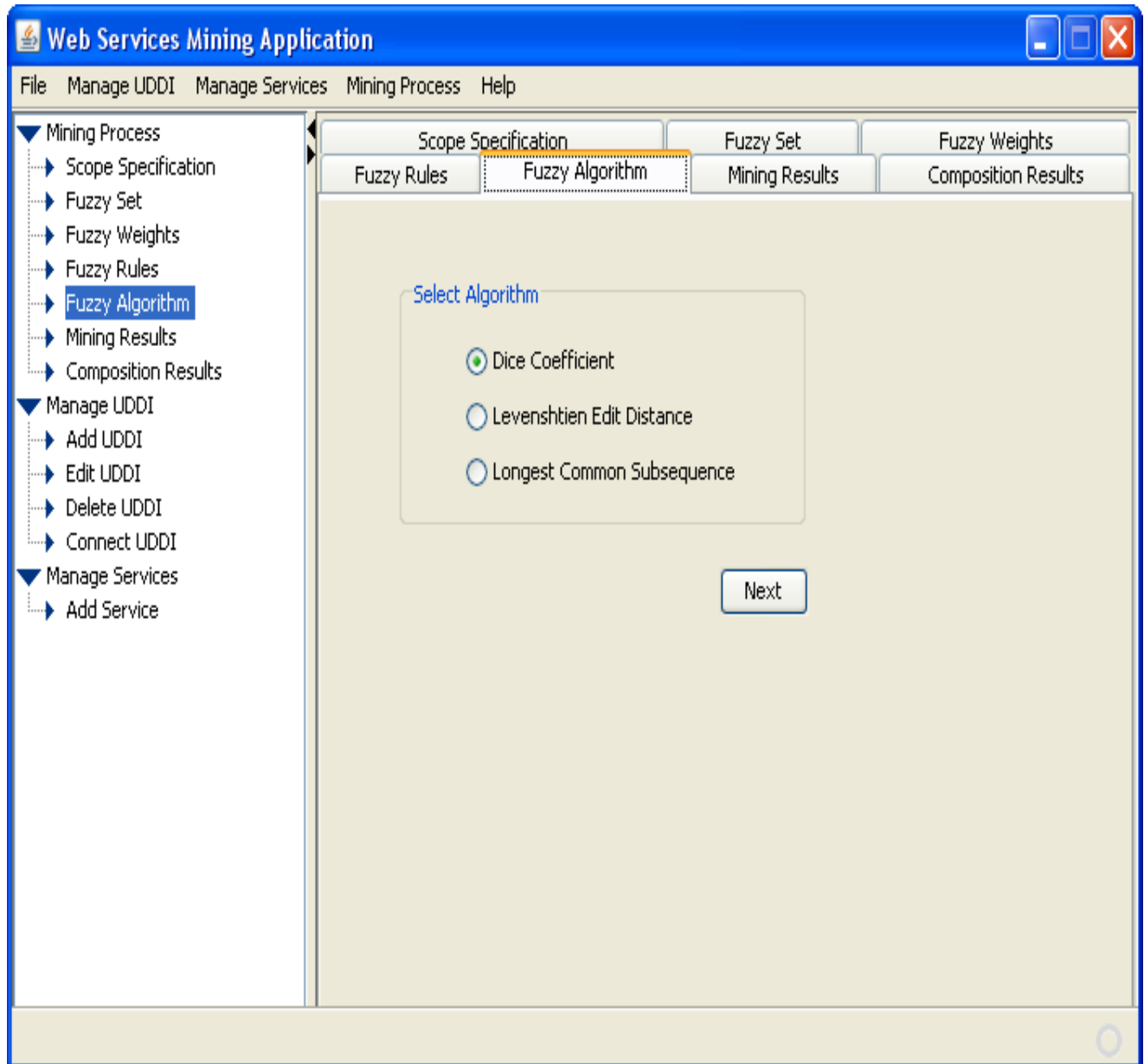
Fuzzy Rules

Fuzzy rules are defined using weights calculated in previous step. Below screen shows fuzzy rules for different ranges of weights. These rules define acceptance criteria for fuzzy matching algorithms which will be used in next step.



Fuzzy Algorithm

Different fuzzy based string matching algorithms are implemented in proposed system. Below screen list these fuzzy algorithms. User will select a fuzzy algorithm and mining results will be shown in next step.



Mining Results

Below screen shows the web services mining results. It has two sections. On left side all the mined services are listed, whereas on right side details of the selected web service is given. This web services list is further used in the composition of web services.

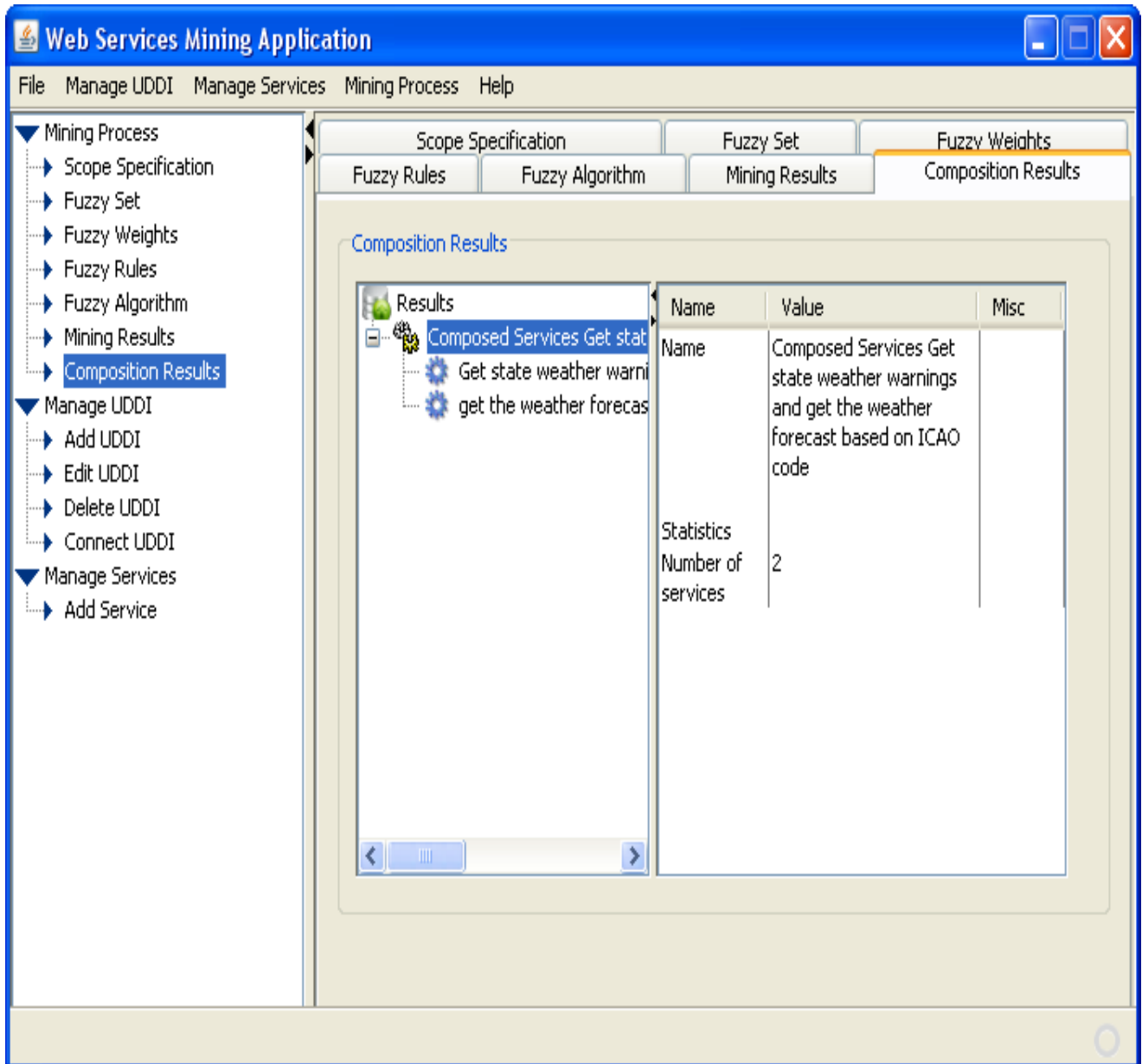
The screenshot displays the 'Web Services Mining Application' window. The interface is divided into several sections:

- Menu Bar:** File, Manage UDDI, Manage Services, Mining Process, Help.
- Left Panel (Tree View):** Mining Process (expanded), Scope Specification, Fuzzy Set, Fuzzy Weights, Fuzzy Rules, Fuzzy Algorithm, Mining Results (selected), Composition Results, Manage UDDI (expanded), Add UDDI, Edit UDDI, Delete UDDI, Connect UDDI, Manage Services (expanded), Add Service.
- Top Panel (Tabs):** Scope Specification, Fuzzy Set (selected), Fuzzy Weights, Fuzzy Rules, Fuzzy Algorithm, Mining Results (selected), Composition Results.
- Main Content Area (Mining Results):** A list of services on the left and a details table on the right.
 - Service List:** Includes 'Weather forecast service' (selected), 'Weather forecast data', 'Weather resource service', 'Weather data service', and 'US Weather forecasts by'.
 - Details Table:**

Name	Value	Misc
Id	uddi:juddi.apache.org:27a6807b-0da9-47b8-88c3-7a9f17180a0c	
Business Entity Id	uddi:juddi.apache.org:699ecea3-7b15-4e1a-a541-3fb4b4f86b0a	
Name	Weather forecast service	
Statistics		
Access Point	http://www.programmableweb.com/api/weather-underground	
- Bottom Right:** A 'Next' button.

Composition Results

Services discovered in the mining step are composed using interface based composition technique. Below screen shows the list of composed services where two or more services are integrated.

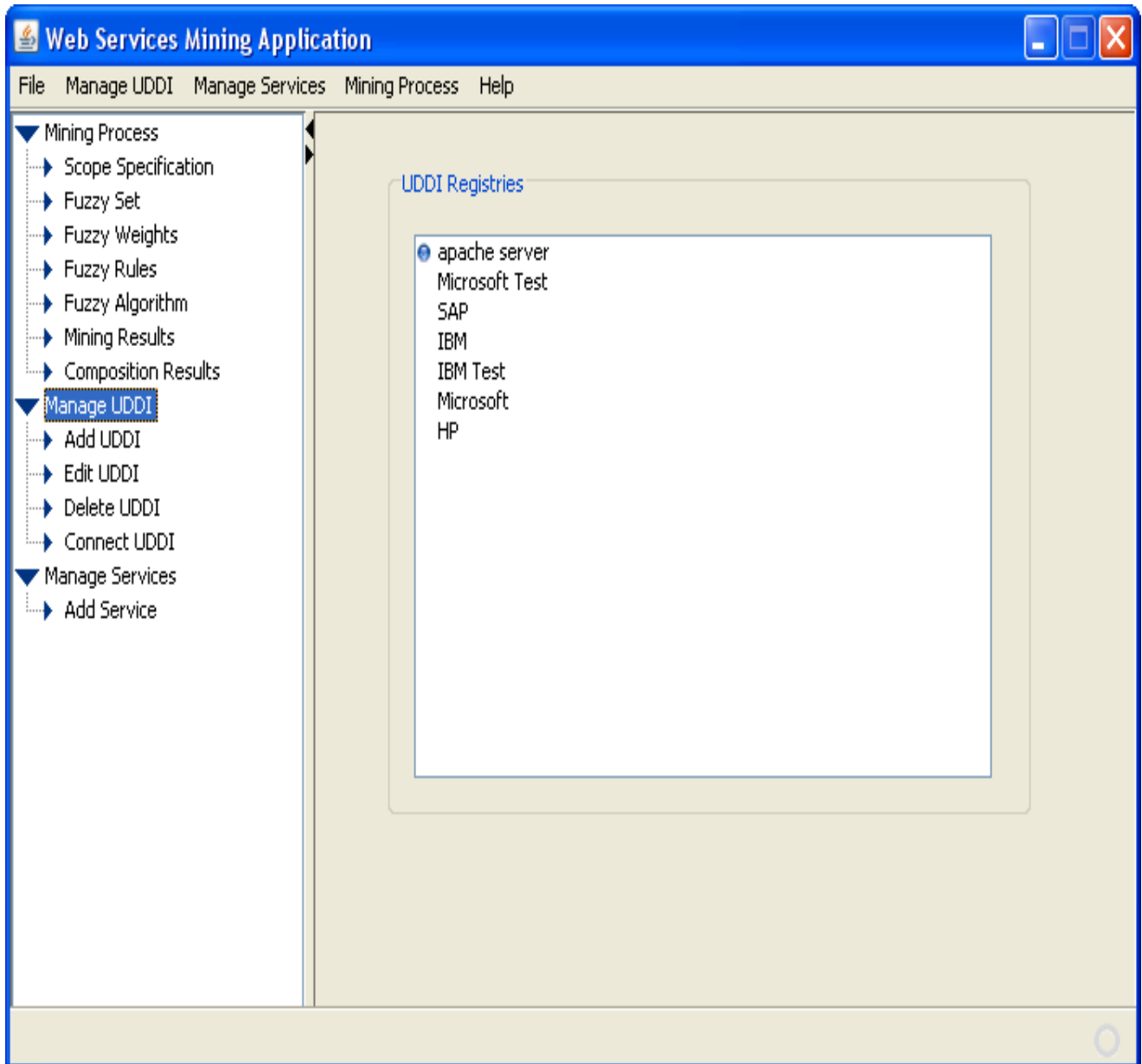


The screenshot displays the 'Web Services Mining Application' window. The interface includes a menu bar with 'File', 'Manage UDDI', 'Manage Services', 'Mining Process', and 'Help'. A left-hand navigation tree is expanded to 'Mining Process', with 'Composition Results' selected. The main workspace is divided into several tabs: 'Scope Specification', 'Fuzzy Set', 'Fuzzy Weights', 'Fuzzy Rules', 'Fuzzy Algorithm', 'Mining Results', and 'Composition Results'. The 'Composition Results' tab is active, showing a tree view of results and a table of data.

Name	Value	Misc
Composed Services Get state weather warnings and get the weather forecast based on ICAO code		
Statistics		
Number of services	2	

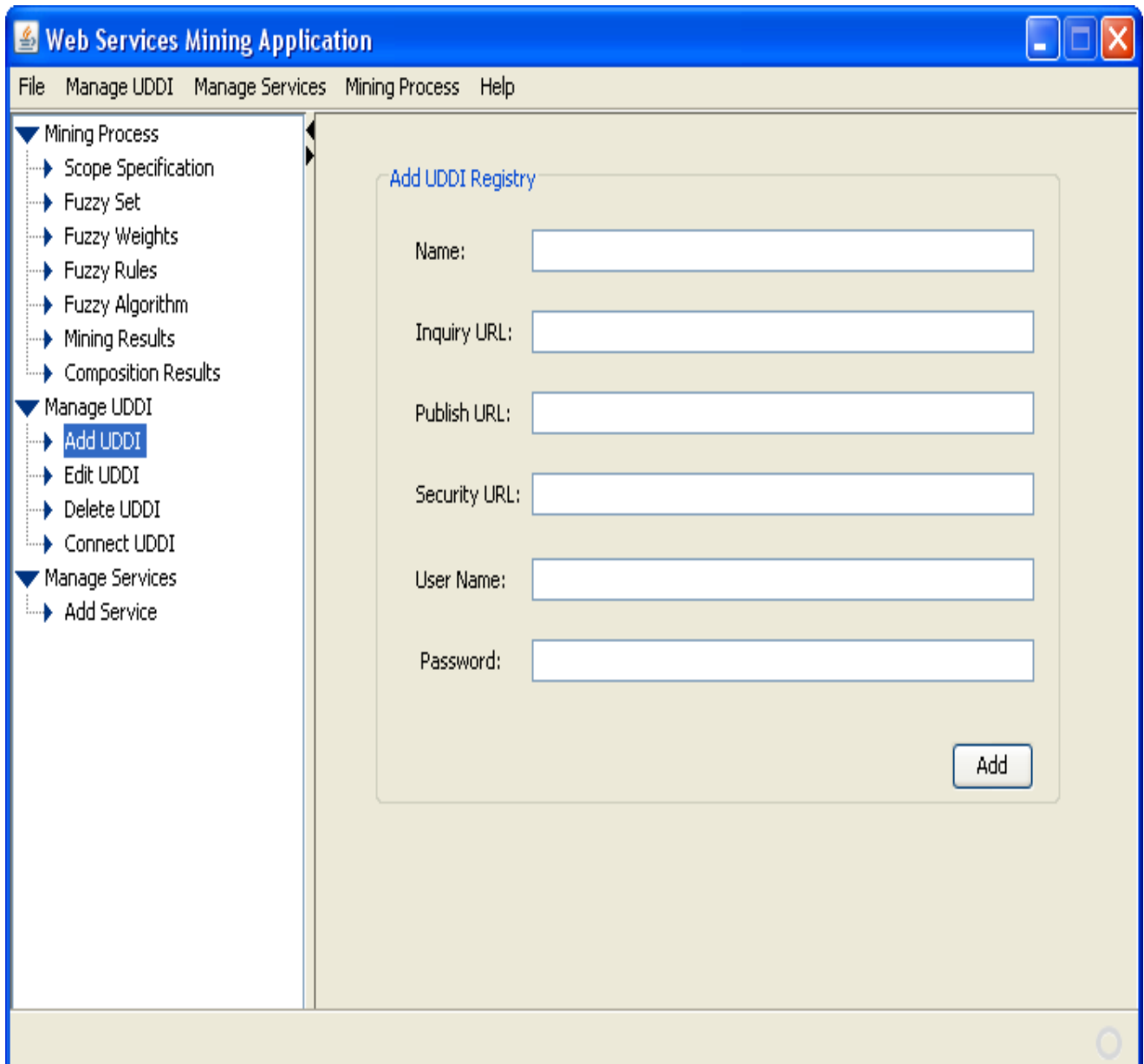
Manage UDDI

Below, screen shows a list of already added UDDI registries. Currently connected UDDI registry is show with a connector symbol. Different interface are available to manage these UDDI registries. This screen only list already added and connected UDDI.



Add UDDI

A new UDDI can be easily added to the system. Provide the name, inquiry, publish and security URLs with username/password (if applicable). After entering all the information into below screen click the add button and it will save the data into database. After that UDDI list is also updated.



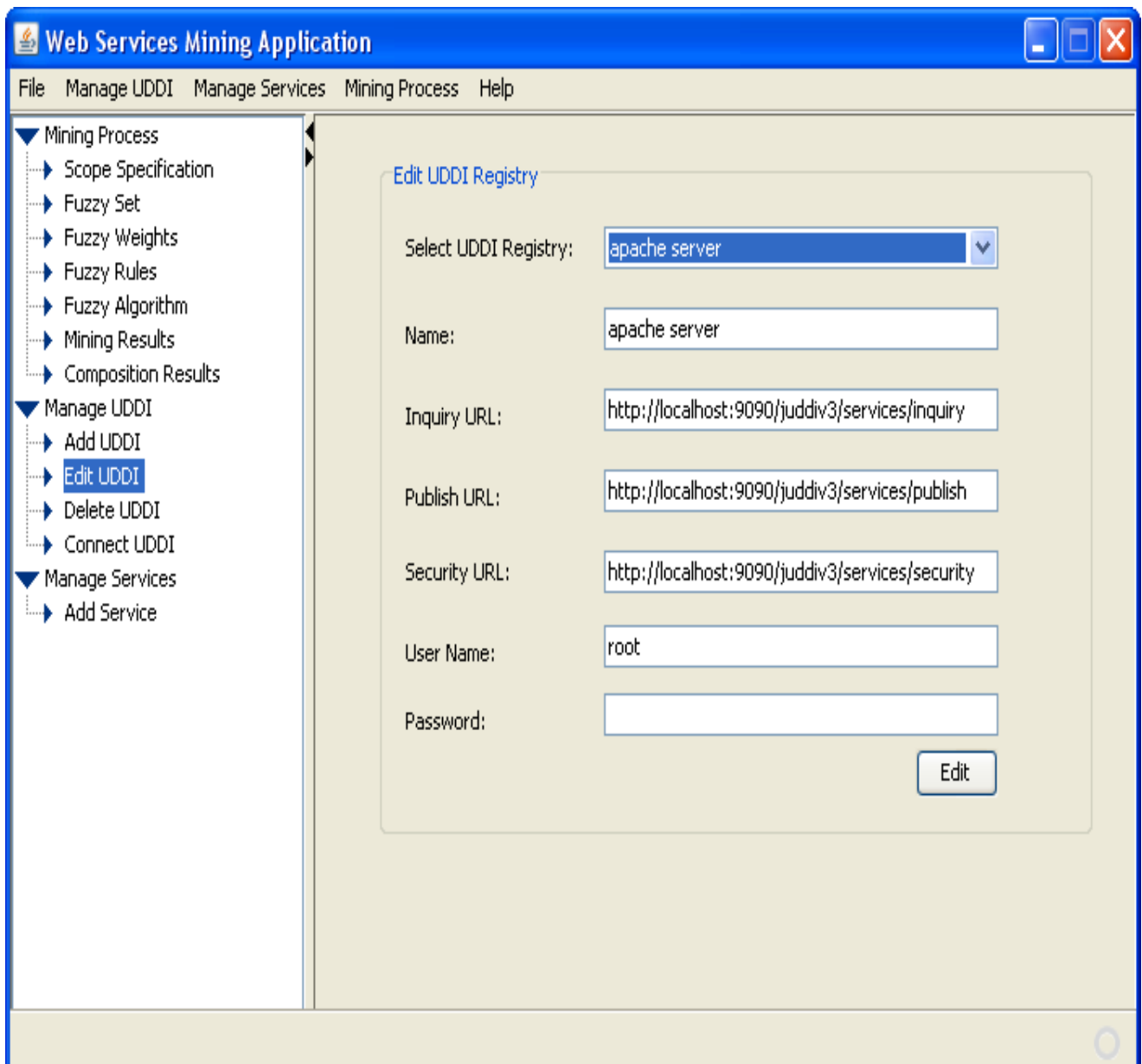
The screenshot displays the 'Web Services Mining Application' window. The title bar includes the application name and standard window controls. The menu bar contains 'File', 'Manage UDDI', 'Manage Services', 'Mining Process', and 'Help'. The left sidebar shows a tree view with 'Manage UDDI' expanded, and 'Add UDDI' selected. The main content area is titled 'Add UDDI Registry' and contains the following form fields:

- Name:
- Inquiry URL:
- Publish URL:
- Security URL:
- User Name:
- Password:

An 'Add' button is located at the bottom right of the form area.

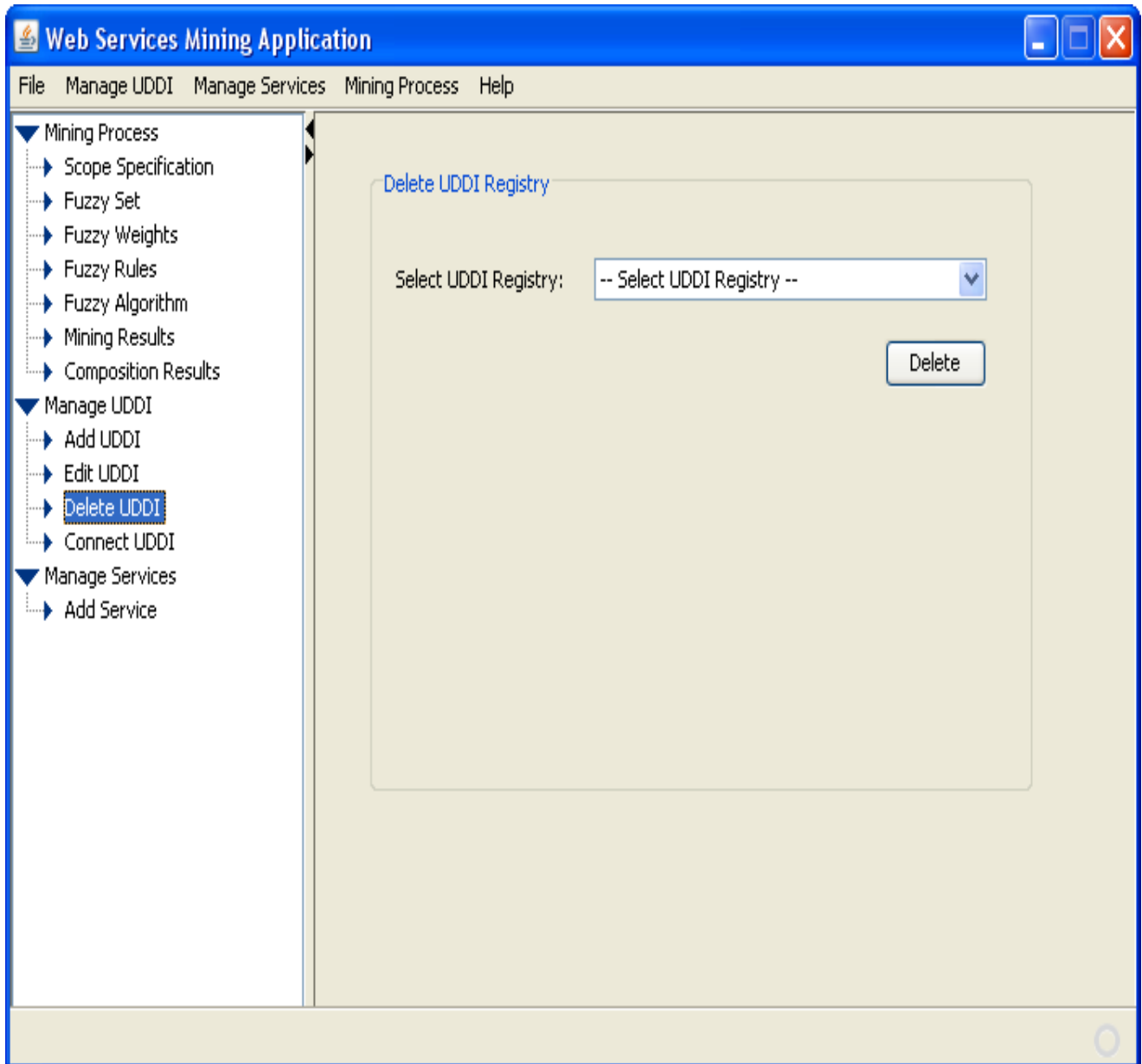
Edit UDDI

Already added, UDDI registry can be easily updated. For editing a UDDI registry below interface is provided. Select a UDDI registry from the drop down and it will populate all the fields with stored information. User will change information in any field and click the edit button; it will update information in database.



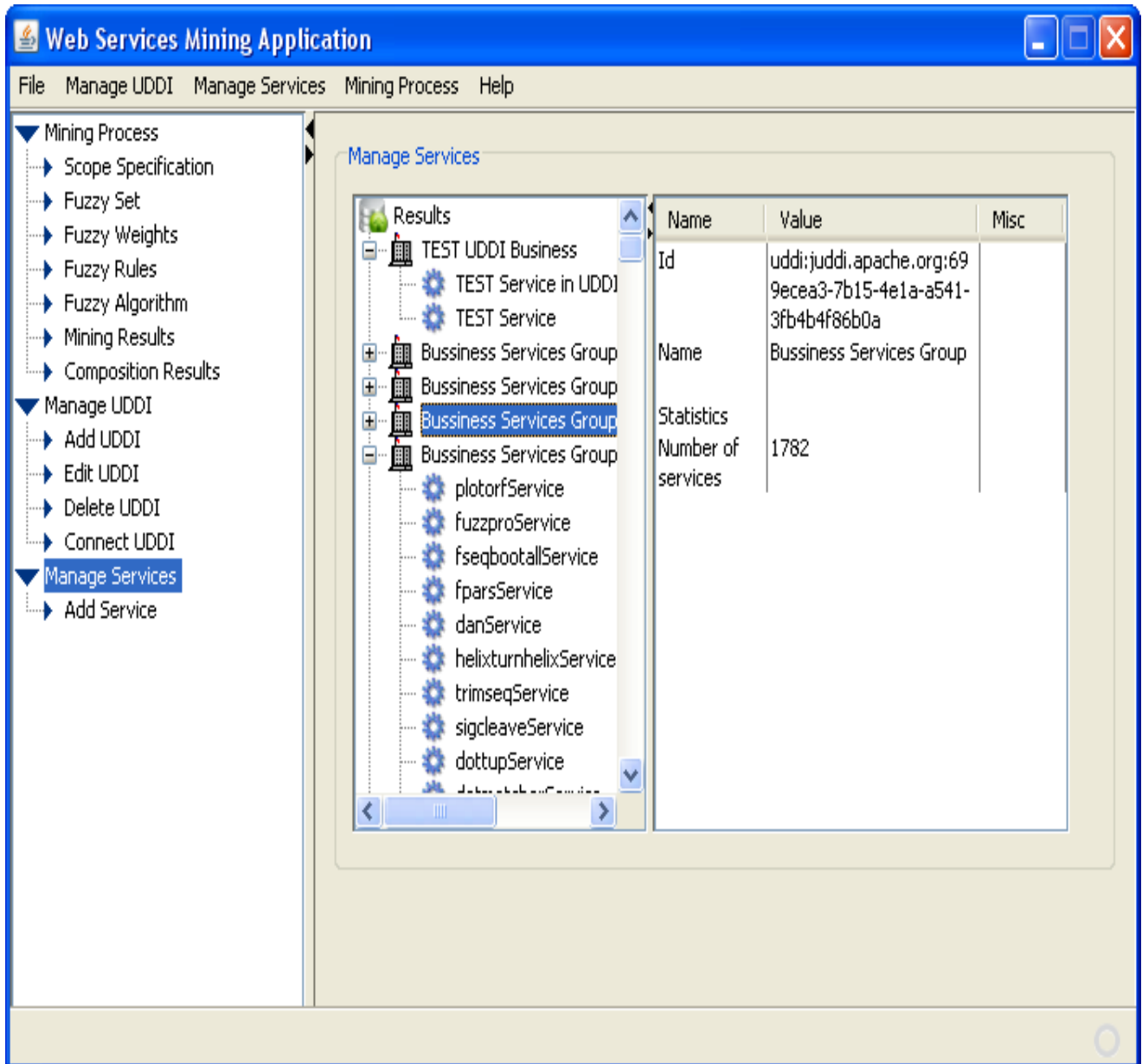
Delete/Connect UDDI

Delete and connect UDDI interfaces are same. Below is delete UDDI interface is given. User will select a UDDI from down and click on the delete button. It will remove the UDDI data from database.



Manage Services

All the businesses with their services are given on this screen. On left side business with services are listed whereas on right side details are given for any selected business or web service.



The screenshot displays the 'Web Services Mining Application' window. The 'Manage Services' menu item is selected in the left-hand navigation pane. The main area is titled 'Manage Services' and contains a tree view on the left and a table on the right.

The tree view shows the following structure:

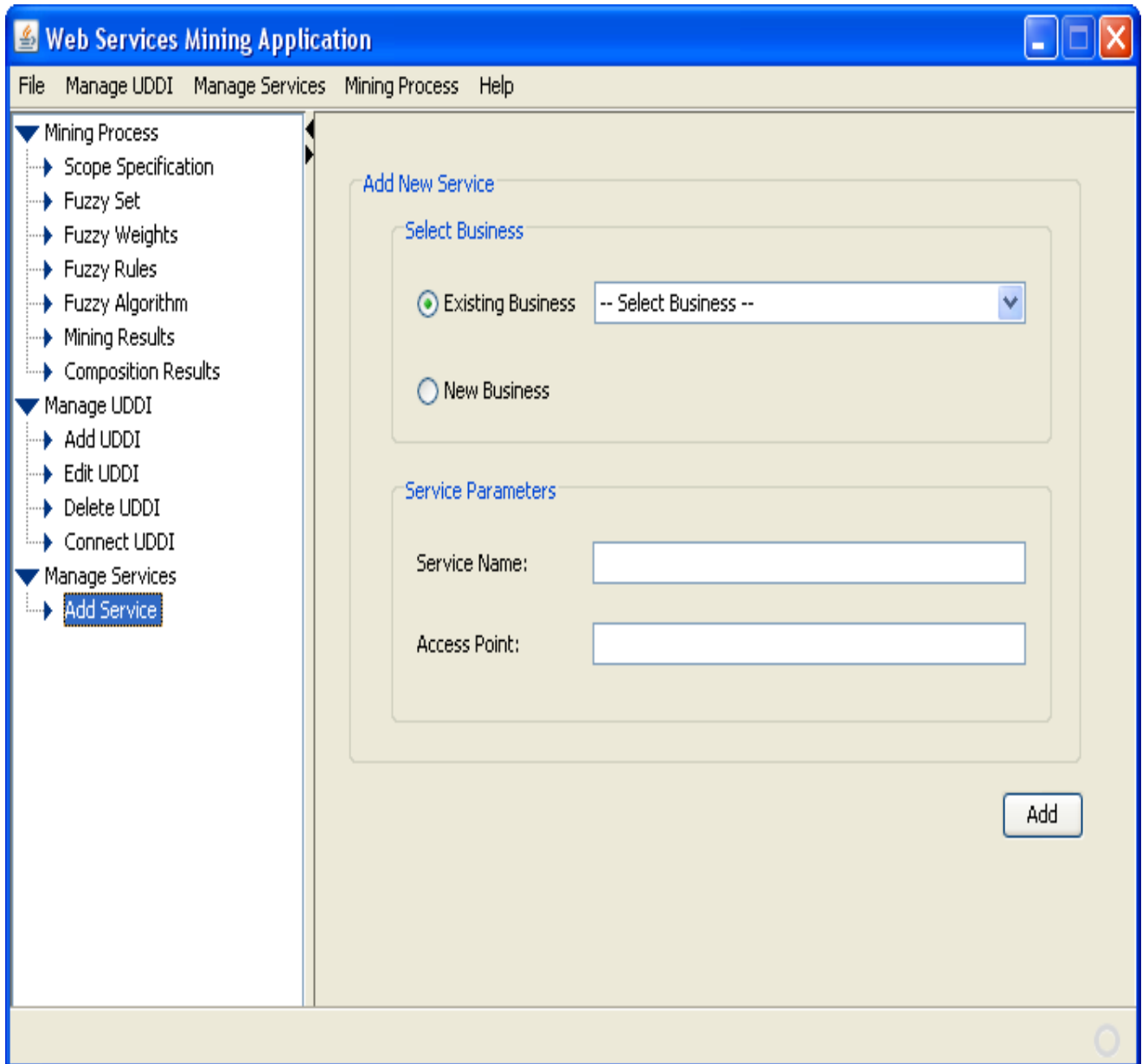
- Results
 - TEST UDDI Business
 - TEST Service in UDDI
 - TEST Service
 - Business Services Group (selected)
 - Business Services Group
 - Business Services Group
 - plotorfService
 - fuzzproService
 - fseqbootallService
 - fparsService
 - danService
 - helixturnhelixService
 - trimseqService
 - sigcleaveService
 - dottupService
 - detestabefService

The table on the right displays details for the selected 'Business Services Group':

Name	Value	Misc
Id	uddi:juddi.apache.org:699ecea3-7b15-4e1a-a541-3fb4b4f86b0a	
Name	Bussiness Services Group	
Statistics		
Number of services	1782	

Add Service

User can add new services into UDDI. For this user needs to select an existing business or need to provide a new business name. Service name and access point are provided by end user to add new service.



References

- [1] Web Services Architecture-[http:// www.w3.org/TR/2004/NOTE-ws-arch-20040211/](http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/), Feb. 2004.
- [2] Hong-Jie Dai¹, Chi-Hsin Huang¹, Jaimie Yi-Wen Lin, Pei-Hsuan Chou¹, Richard Tzong-Han Tsai², Wen-Lian Hsu¹ “A Survey of State of the Art Biomedical Text Mining Techniques for Semantic Analysis” 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing.
- [3] George Zheng and Athman Bouguettaya, Senior Member, IEEE “Service Mining on the Web” IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. 2, NO. 1, JANUARY-MARCH 2009.
- [4] UDDI: Universal Description, Discovery, and Integration—<http://www.oriely.com/>.
- [5] Qianhui Althea LIANG, Jen-Yao CHUNG, Steven MILLER and Yang OUYANG “Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository” IEEE International Conference on e-Business Engineering (ICEBE'06).
- [6] Gao Ting, Wang Haiyang, Zheng Naihui, Li Fei “An Improved Way to Facilitate Composition-Oriented Semantic Service Discovery” 2009 International Conference on Computer Engineering and Technology.
- [7] George Zheng and Athman Bouguettaya “A Web Service Mining Framework “2007 IEEE International Conference on Web Services (ICWS 2007).
- [8] Giuseppe Fenza, Vincenzo Loia, Sabrina Senatore “Concept Mining of Semantic Web Services By Means Of Extended Fuzzy Formal Concept Analysis (FFCA)” 2008 IEEE International Conference on Systems, Man and Cybernetics (SMC 2008).
- [9] Qianhui Althea LIANG, Steven MILLER and Jen-Yao CHUNG “Service Mining for Web Service Composition” IEEE 2005.
- [10] Gao Ting, Wang Haiyang, Zheng Naihui, Li Fei “An Improved Way to Facilitate Composition-Oriented Semantic Service Discovery” International Conference on Computer Engineering and Technology, 2009.
- [11] Shahab Bayati, Ali Farahmand Nejad, Sadegh Kharazmi, Ardeshir Bahreininejad “Using Association Rule Mining to Improve Semantic Web Services Composition Performance” IEEE, 2009.
- [12] George Zheng and Athman Bouguettaya “Mining Web Services for Pathway Discovery” VLDB '07, September 2328, 2007, Vienna, Austria. VLDB Endowment, ACM 9781595936493/07/09.