Dynamic Web Services Integration and Execution

A dissertation Presented by

Farhan Hassan Khan

(2007-NUST-MS PhD-CSE(E)-27)

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


بسم الله الرحمن الرحيم

THE COMMITTEE

Dynamic Web Services Integration and Execution

A dissertation Presented by

Farhan Hassan Khan

Approved as to style and content by:

Dr. Muhammad Younus Javed , *Supervisor*

Dr. Farooq-e-Azam, *Member*

Dr. Rasheed Ahmed, *Member*

Dr. Aasia Khanam, *Member*

Dr. Muhammad Younis Javeed

Department Chair, Computer Engineering

DEDICATIONS

Dedicated to my parents, teachers, friends and family

ACKNOWLEDGEMENTS

Nothing worthwhile was ever achieved in isolation. I can not claim to have written this thesis without the significant influence of others. I would like to thank my supervisor **Brig. Dr. Muhammad Younus Javed**. It was invaluable to have the benefit of his decades of experience in the field of Information Technology standing behind his advising of me. I am most grateful that he allowed me to work with him for this thesis. I would like to thank **Dr. Farooq e Azam Khan, Dr. Rasheed Ahmed** and **Dr. Aasia Khanam** for serving on my committee.

I thank my parents and family for standing by me in times of crisis and for being patient with my endless years of study.

ABSTRACT

Dynamic Web Service Integration and Execution

The use of web services has dominated software industry. Existing technologies of web services are extended to give value added customized services to customers through composition.

Automated web service integration is a very challenging task. This research presents the solution of existing problems and proposes a technique by combination of interface based and functionality based rules. The framework also solves the issues related to unavailability of updated information and inaccessibility of web services from repository/databases due to any fault/failure. It provides updated information by adding aging factor in repository/WSDB (Web Services Database) and inaccessibility is solved by replication of WSDB. We discuss data distribution techniques and propose our framework by using one of these strategies.

Finally, the framework eliminates the dynamic service integration and execution issues, supports efficient data retrieval and updation, fast service distribution and fault tolerance.

Table of Contents

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
ebXML	Electronic based XML
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
JAXR	Java API for XML based Registries
JUDDI	Java implementation of UDDI
OWL-S	Ontology Web Language Semantic
QoS	Quality of Service
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
VSDL	Value based Service Description Language
WS	Web Service
WSCI	Web Service Choreography Interface

WSDB	Web Service DataBase
WSDL	Web Service Description Language
WSDL4J	Web Service Description Language for Java
WSIF	Web Service Invocation Framework
WSIL	Web Service Inspection Language

Chapter 1

1 Introduction

1.1 Overview of Web Services

Web services are software components designed to support interoperable machine to machine interaction over network. They have a standardized way of integrating web applications using XML, SOAP, WSDL and UDDI over internet backbone. XML is used to code and decode data, SOAP is used for transfer of data, WSDL is used to describe the available services and UDDI is used for listing which services are available. Web services allow organizations to communicate with each other and with clients without intimate knowledge of each other's IT systems behind the firewall.

Unlike traditional client/server models, web services do not provide the user with a GUI. Instead they share business logic, data and processes through programmatic interface across the network. Web services can be added to a GUI (such as a web page) to offer specific functionality to users. Web services allow different applications from different sources to communicate with time consuming and custom coding because all communication is in XML format. They do not require any specific language or operating system. For example Java can communicate with Perl, Windows application can talk with UNIX applications, etc.

Web services have two types of uses:

1. Reusable application components
2. Connect existing softwares

Web services offer application components like: currency conversion, weather reports, or even language translation as services. They also solve interoperability problems by providing a way to different applications to link their data. Data can be exchanged between different applications with different platforms.

Nowadays, an increasing amount of companies and organizations only implement their core business and outsource other application services over Internet. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. This trend has triggered a considerable number of research efforts on the composition of Web services both in academia and in industry.

1.2 Web Services Composition

Web service composition lets developers create applications on top of service-oriented computing's native description, discovery, and communication capabilities. Such applications are rapidly deployable and offer developers reuse possibilities and user's seamless access to a

variety of complex services. There are many existing approaches to service composition, ranging from abstract methods to those aiming to be industry standards.

Despite all these efforts, the Web service composition still is a highly complex task, and it is already beyond the human capability to deal with the whole process manually. The complexity, in general, comes from the following sources. First, the number of services available over the Web increases dramatically during the recent years, and one can expect to have a huge Web service repository to be searched. Second, Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up to date information. Third, Web services can be developed by different organizations, which use different concept models to describe the services, however, there is no unique language to define and evaluate the Web services in an identical means.

For business to business and enterprise level application integration, Composition of web services plays an important role. Sometimes a single web service does not fulfill the user's desired requirements and different web services are combined through composition method in order to achieve a specific goal. Service compositions reduce the development time to create new applications.

Web services can be categorized in following two ways on the basis of their functionality:

1) Semantic annotation describes what the web service does and

2) Functional annotation describes how it performs its functionality.

WSDL is a description language that is used for specification of messages that are used for communication between service providers and requesters.

There are two methods for web services composition. One is static web service composition and other is automated/dynamic web service composition. In static web service composition, integration of web services is performed manually, that is each web service is executed one by one in order to achieve the desired goal/requirement. It is a time consuming task which requires a lot of effort. In automated web service composition, agents are used to select a web service that may be composed of multiple web services but from user's viewpoint, it is considered as a single service.

1.3 Dynamic Web Services Composition

The automated/dynamic web service composition methods generate the request/response automatically through agents. Most of these methods are based on AI planning. In automated web services composition, request is given to the agent and agent performs the composition steps. First request goes to Translator which performs translation from external language to a language used by system, then the services are selected from repository that meet user criteria and goes to Process Generator which compose these services. If there are multiple composite services that meet user criteria then Evaluator evaluates them, selects and returns the best service to Execution Engine through which results are returned to clients (Requester). There

should be well defined methods and interfaces through which clients interact with the system and get the response. The generalized dynamic web services composition framework is shown in Fig 1.

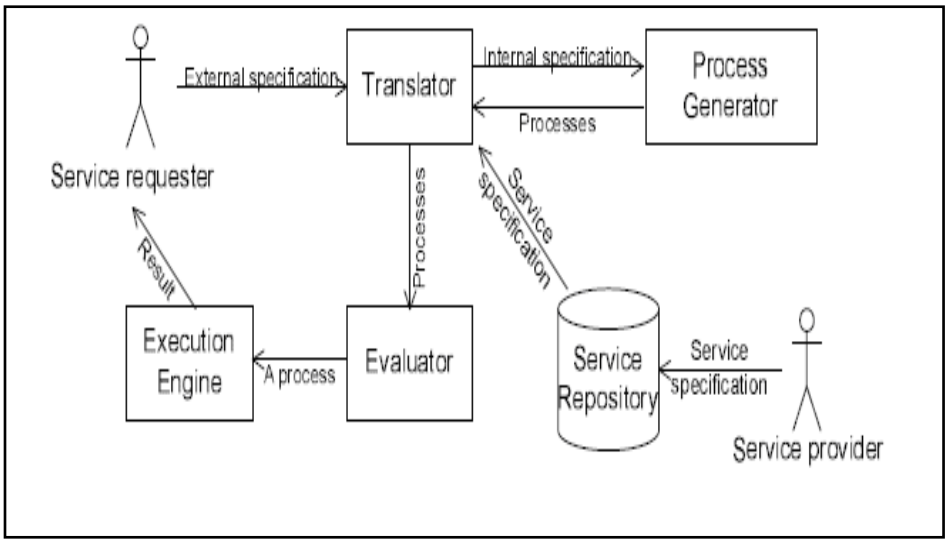


Fig 1: Generalized framework for dynamic web services composition

More precisely Dynamic web services composition include the following phases:

1.3.1 Presentation of single service

Web service providers advertise atomic web services on web. There are several languages used for advertising for example, UDDI and DAML-S service profile. The essential features for advertising a web service include signature, states and non-functional values. Signature is represented by service inputs, outputs and exceptions. It provides the information about data transformation during service execution. States describe the transformation from one

state to another. Non-functional values are used for evaluating the web services, such as cost, quality of service issues, etc.

1.3.2 Translation of Languages

Translator performs the translation of the user's request in order to search for a particular service. It translates the external language used by requester/provider into language used by system.

1.3.3 Generation of composition process model

Service requester can also request the service in service specification language. Process generator then compose the atomic services advertised by service providers. Process generator usually takes the functionality as input and presents a composed web service as output. Process model contains set of web services, control flow and data flow of these services.

1.3.4 Evaluation of Composite Service

It may be possible that planer generates more then one composite web services that fulfill the requirements. In that case, composite web services are evaluated based on non-functional attributes. The most commonly used is utility function. Requester specifies weights to each attributes and the best web service is one which is ranked on top.

1.3.5 Execution of Composite Service

Execution of web service includes the sequence of message passing according to process model. The dataflow of composite service is defined as the actions output data of a former executed service transfers to the input of a later executed atomic service.

1.4 Problem Statement

The main interest of web service compositions is to give value-added services to existing web services and introduce automated web services. Also they provide flexibility and agility. There are few problems in dynamic web service composition are:

- First, the numbers of web services are increasing with time and it is difficult to search the whole repository for desired service in order to use it for the fulfillment of specific goal.
- Second, Web services are dynamically created and updated so the decision should be taken at execution time and based on recent information.
- Third, Different web service providers use different conceptual models and there is a need of one structure so that web services easily access each other without any technical effort.
- Forth, only authorized persons can access few of these web services.

1.5 Existing Approaches

This section provides the overview of some recent methods that provide automation to Web service composition. The automation means that either the method can generate the process model automatically, or locate the correct services if an abstract process model is given.

The two approaches in web services composition are:

1) Centralized dataflow

2) Decentralized dataflow

Both of these approaches have advantages and some limitations. The limitation of centralized dataflow is that all component services must pass through a composite service. This results in the bottleneck problem which causes the decrease in throughput and increase the response time. The disadvantage of decentralized dataflow is that each web service directly shares data with web servers. This results in increasing the load at each node and delay in response time and throughput. The decentralized dataflow is very efficient in case of dynamic web services composition as it reduce the tight coupling between clients and servers by adding a middleware (UDDI, WS coordination or WS transaction etc).

Mostly automated composition techniques are:

1) Interface based and

2) Functionality based

In interface based composition, users get composite services on the basis of inputs and outputs through interfaces and after composition desired results are achieved. The drawback of this approach is that functionality is not guaranteed, whereas in functionality based composition, with inputs and outputs user provides the formula that gives logic into interface information.

Polymorphic process model (PPM) uses a method that combines the static and dynamic web services composition. Static settings are based on processes that consist of abstract sub-processes i.e. sub-processes that have functionality description but lack in implementation. Abstract sub-processes are implemented by services and bound at run time. The dynamic part is based on service based processes. Service is described by a state machine that includes the possible states and transitions of a service. In the setting, the dynamic service composition is enabled by the reasoning based on state machine.

AI Planning based web services composition is another way to tackle dynamic web services composition problems. DAML-S (also called OWL-S) is the only language that has direct connection with AI planning. The state change during web service execution is specified through per-condition and effect properties in Service Profile in DAML-S. Golog is a logic programming language built on top of situation calculus. Golog programs can be customized by incorporating the service requester's constraints. For example, the service requester can use the nondeterministic choice to present which action is selected in a given situation, or use the sequence construct to enforce the execution order between two actions.

Theorem proving is an idea for web services composition proposed by Weldinger. It is based on automated deduction and program synthesis. Initially web services and user requirements are described in first order language and then structive proofs are generated with SNARK theorem prover. Finally, service composition descriptions are extracted from particular proofs.

1.6 Contribution

The major contribution of this thesis is that it presents a method for automated and dynamic web service composition by combination of interface based and functionality based approaches. It focuses on the data distribution issues, QoS issues and defines how execution problems can be avoided. This research also resolves the problems of decentralized dataflow and provides a framework that has minimum latency, maximum throughput and response time. This thesis proposed a solution for researchers who are facing the problems of web service composition due to constant changes in input/output parameters, networking issues and independent nature of different web services. We have given generic implementation of proposed model to prove the correctness of algorithm.

1.7 Organization

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

Chapter 2 This chapter describes the related research papers.

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

Chapter 4 This chapter presents the implementation of proposed algorithm.

Chapter 5 This chapter is concerned with analysis and Results.

Chapter 6 This chapter includes the summary and conclusion.

Chapter 2

2 Related work

Literature Review

[1] Incheon Paik, Daisuke Maruyama et al. "*Automatic Web Services Composition Using Combining HTN and CSP* " proposes a framework for automated web services composition through AI planning technique by combining logical combination (HTN) and physical composition (CSP). This paper discusses the real life problems on the web that is related to planning and scheduling and provides task ordering to reach at desired goal. OWL-S and BPEL4WS are used for composition which removes the limitations of HTN that are lack of interactive environment for web services, lack of autonomy etc. Then the proposed model is compared with HTN according to web services invocation. It tackles the following given problems faced by planner alone, that are, it does not deal with various web users requests for information; second, it is inefficient for automated finding of solutions in given state space. Third, its maintenance is weak due to frequent user requests. The proposed framework gives intelligent web services for web users due to CSP which provides problem space for planning, scheduling and to automate the desired tasks.

[2] Faisal Mustafa, T. L. McCluskey et al. "*Dynamic Web Services Composition"* outlined the main challenges faced by automated web services composition that are related to distributed, dynamic and uncertain nature of web. The proposed model is semi-automatic static composition model and fixes some issues of dynamic web services composition that are listed as follows.

First, Repository has large number of web services and it is not possible to analyze and integrate them from repository. Second, updated web service information is required from repository when it is selected to fulfill the specific task. Third, Multiple services are written in different languages and there is a need of conceptual model to describe them in a single service. The proposed technique has a drawback that if a server goes down then input/output issues arises. Second, new uploaded information is not available in repository as it does not update its contents. The framework is shown in Fig.

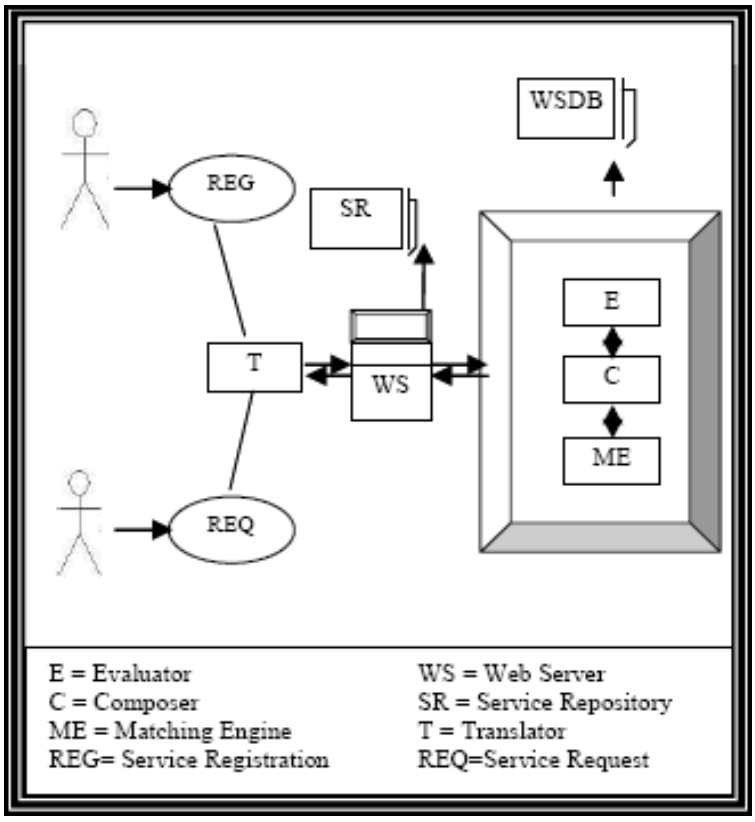


Fig 2.1: Composition framework by Faisal & McClusky

[3] Pat. P. W. Chan and Michael R. Lyu et al. *"Dynamic Web Service Composition: A New Approach in Building Reliable Web Service"* proposed the dynamic web service composition technique by using N- version programming technique which improves the reliability of system for scheduling among web services. If one server fails, other web servers provide the required services. Web services are described by WSDL and their interaction with other web services is described by WSCI (web services choreography interface). The composed web services are deadlock free and reduce average composition time. Also the proposed system is dynamic, as it works with updated versions without rewriting the specifications. The reliability of system has been improved by replication. At the end experimental evaluation and results are presented to verify the correctness of algorithm. The framework is shown in Fig 2.2

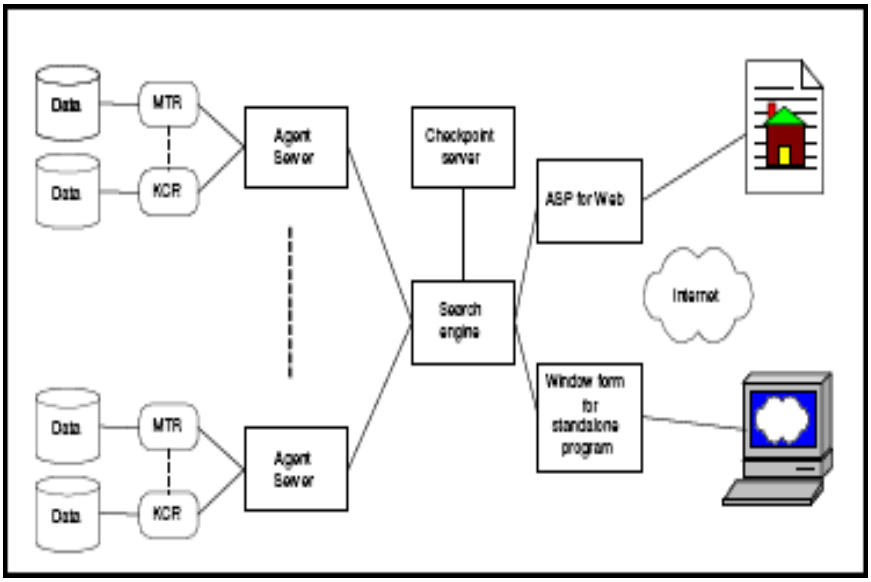


Fig 2.2: Best Route finding system architecture

[4] LIU AnFeng, CHEN ZhiGang, HE Hui, GUI WeiHua et al. *"Treenet: A Web Services Composition Model Based on Spanning Tree"* presents the technique based on web services interfaces and peer to peer ontology. It provides an overlay network (WSCON) with peer to peer technologies and provides a model for web services composition. The web services composition is based on domain ontology and Distributed Hash Table (DHT) is used for discovery and composition. The analyses shows that it is easy to understand because of loosely coupled due to the separation of interfaces from underlying details. The proposed model is based on ontology and service composition interface and the process is fault tolerant. The advantages of proposed model are: it provides web services composition based on QoS, fast composition rate, fault tolerant, efficient for dynamic discovery and composition. The framework is shown in Fig 2.3

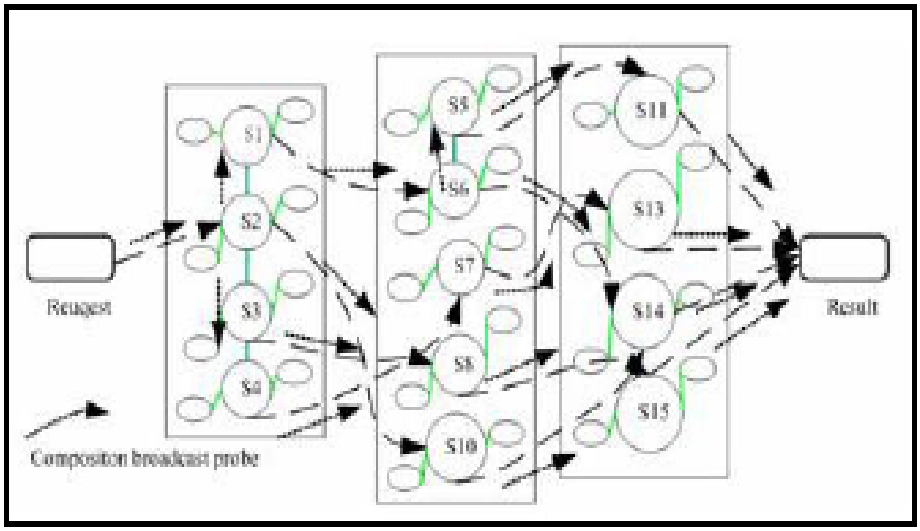


Fig 2.3: Probe broadcasting in forming web services composition paths

[5] Kazuto Nakamura, Mikio Aoyama et al. *"Value-Based Dynamic Composition of Web Services"* proposed a technique for dynamic web service composition that is value based and provide composed web services based on QoS. Value meta-model and it's representation language VSDL is presented. Values are used to define quality of web services. Value added service broker architecture is proposed to dynamically compose the web services and value-meta model to define relationship among values. Value models are stored into value repositories in value added service broker which provides value based service composition and evaluated the results through three dictionary services. Service brokers provide dynamic composition of web services and returned them to requesters. The results explained that resultant composite services can provide more values of quality of contents as compared to previous discovered web services. Although a number of dynamic web services composition techniques have been introduced, there is a need of dynamic approach to handle the large number of increasing web services and their updation in repositories. This paper provides an automated, fault tolerant and dynamic web services composition framework. The framework is shown in Fig

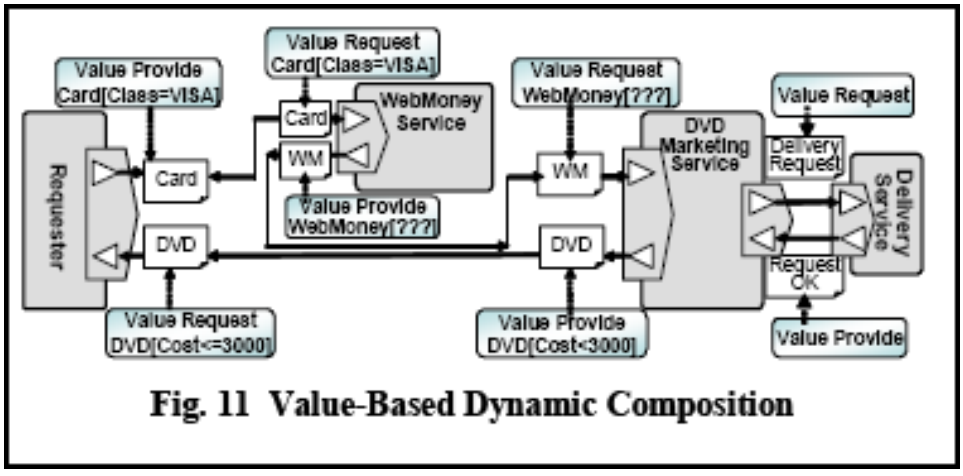


Fig 2.4: Value based dynamic composition

[6] Jinghai Rao and Xiaomeng Su et al. *"A Survey of Automated Web Services Composition Methods"* presents the overview of research efforts of automatic web service composition both from the workflow and AI planning research community. First it proposed the five step model for web services composition process. The composition model comprises of Presentation, Translation, process generation, evaluation and execution. Each step is based on different languages, platforms and methods. This method is enabled either by workflow research or AI planning. The workflow methods are usually used in situation when request is already defined in process model but automatic program is required to find atomic services to fulfill the requirements. The AI methods are used when requester has no process model but has a set of constraints and preferences. Hence process model can be automatically generated by the program. The author also concludes that although different automatic web services composition techniques are available, it is not true that higher automation is better. The web services environment is highly complex and it is not good to generate everything automatically. Usually, the highly automated methods are suitable for generating the implementation skeletons that can be refined into formal specification. The framework is shown in Fig 2.5

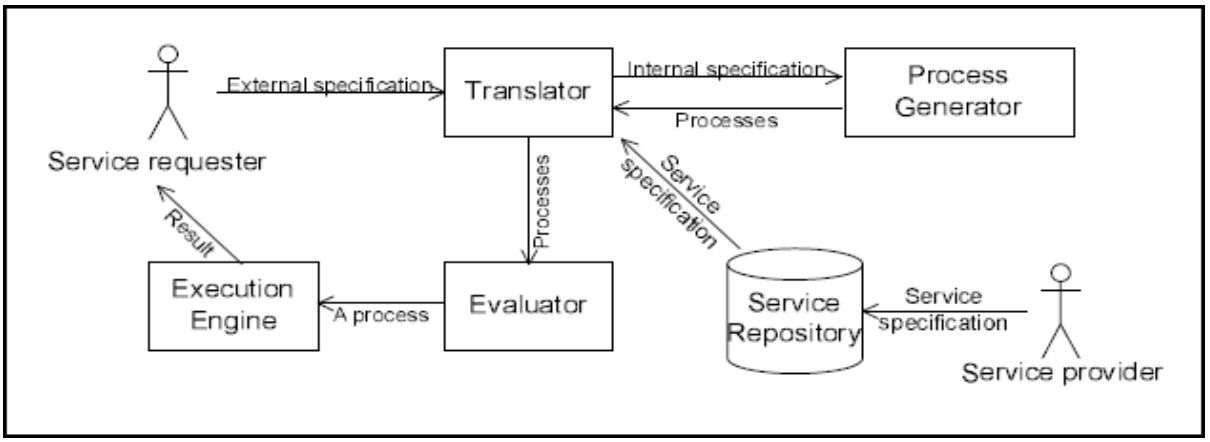


Fig 2.5: A framework of service composition system

[7] Biplav Srivastava, Jana Koehler et al. *"Web Service composition: Current Solutions and Open Problems"* explored the web services composition problems and compared the two major approaches to this problem. The industrial approach and Semantic web approach, with each other. The industrial approach is primarily syntactical and is based on XML standards which are used for web services specification. This approach is used for several Businesses to Business and enterprise applications integration. On the other hand Semantic web approach is based on semantic description of preconditions and effects by focusing on reasoning about web resources. For the composition of web services, they draw on the goal oriented inferencing from planning. Both approaches are developed independently from each other. Several sub problems are identified related to AI planning perspective. It is concluded that it is not possible to directly apply AI planning technology to them.

[8] Junmei Sun, Huaikou Miao et al.*"A Formal Architecture Supporting Dynamic Composition of Web* Services" proposed formal description to web services architecture using formal specification notation Z which is different from workflow technology. The specification was written in Z environment and checked which validates the complete process of composition. Then it is explained with an example which depends on proposed architecture. The architecture decompose the system in components, direct and validate the composition level at high abstract level and provides a top down reusing way based on components. Composition behavior is divided into two phases, A matching phase and an interaction phase. Connections for interactions are established after matching phase. Service provider, service requester and registry are used as components. Service provider can also play the role of service requester and vice versa. SOAP messages are used for remote procedure calls. Ports are abstract access points to component

services. The advantage of this approach is precise and concise description of web services discovery and composition.

[9] Jun Fang, Songlin Hu, Yanbo Han et al.*"A Service Interoperability Assessment Model for Service Composition"* present service interoperability assessment model for service composition. The model helps users to invoke properly their required composite services. The interoperability between autonomous services is evaluated in open and large scale distributed environment. The assessment model describes whether the interaction between two services is correct or to choose most suitable services from candidate. The paper also considers the measure quantity about interoperability will be more attractive and concrete than simple judgment, i.e. yes or no. By using proposed model, the cost of computation time is small. There is no scalability problem in proposed model as query space is smaller in comparison with service space. The advantage of this approach is that assessment model is multifactor decision problem, depends on separate evaluation of multiple levels. The framework usage is shown in Fig 2.6

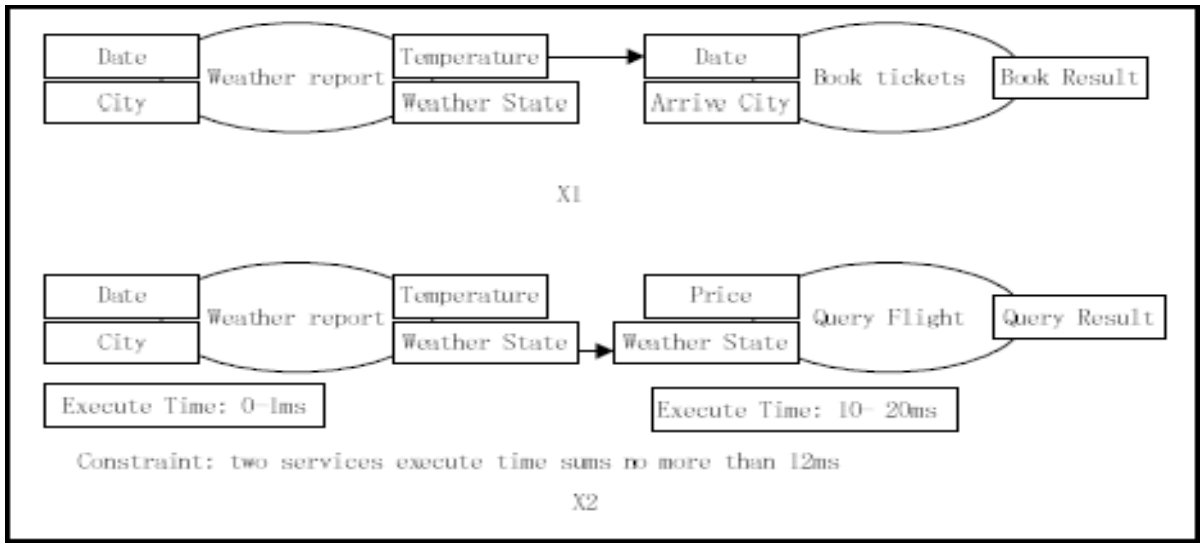


Fig 2.6: First use pattern Example

[10] Jiamao Liu, Juntao Cui, Ning Gu et al. *"Composing Web Services Dynamically and Semantically"*. In proposed approach, web services are based on some rules whose head and bodied are semantic ontology related which eliminates the conflicts in composition process. The algorithm is non-back-trace backward chaining that is used to compose the existing web services in an efficient way. By giving inputs and outputs, the approach automatically performs the composition process and converts it into BPEL4WS that can be executed and returns results. The advantage of this approach is that whole composition process can be done dynamically and automatically. The proposed framework is shown in Fig 2.7

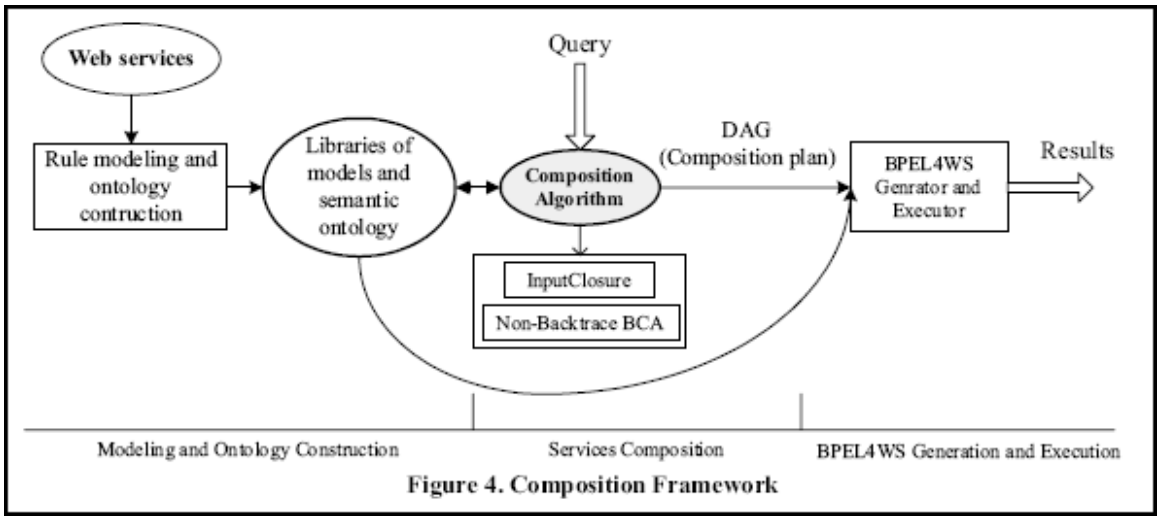


Fig 2.7: Composition framework

Chapter 3

3 Proposed Approach

3.1 Problem Statement

The main interest of web service compositions is to give value-added services to existing web services and introduce automated web services. Also they provide flexibility and agility. There are few problems in existing approaches of dynamic web service composition.

- First, the numbers of web services are increasing with time and it is difficult to search the whole repository for desired service in order to use it for the fulfillment of specific goal.
- Second, Web services are dynamically created and updated so the decision should be taken at execution time and based on recent information.
- Third, Different web service providers use different conceptual models and there is a need of one structure so that web services easily access each other without any technical effort.
- Forth, only authorized persons can access few of these web services

3.2 Suggested improvements

Following improvements are suggested in proposed dynamic web services composition framework.

3.3 Proposed Framework

Proposed framework of web services composition includes the following components

3.3.1 Service Provider

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization, such as a business or an individual. The *provider entity* is the person or organization that provides an appropriate agent to implement a particular service.

Service providers register their web services in registries and make it accessible to clients. New services are registered in registry through service registration process. There are several registries that different companies (service providers) maintained and all of them are synchronized after regular interval.

3.3.2 Service Requester

A *requester entity* is a person or organization that wishes to make use of a provider entity's Web service. It will use a *requester agent* to exchange messages with the provider entity's *provider agent*.

Clients that need a particular service send request through service request module. Service requester requests the service from registry and if desired service is found, it accesses that service through its service provider.

In most cases, the requester agent is the one to initiate this message exchange, though not always. Nonetheless, for consistency we still use the term "requester agent" for the agent that interacts with the provider agent, even in cases when the provider agent actually initiates the exchange.

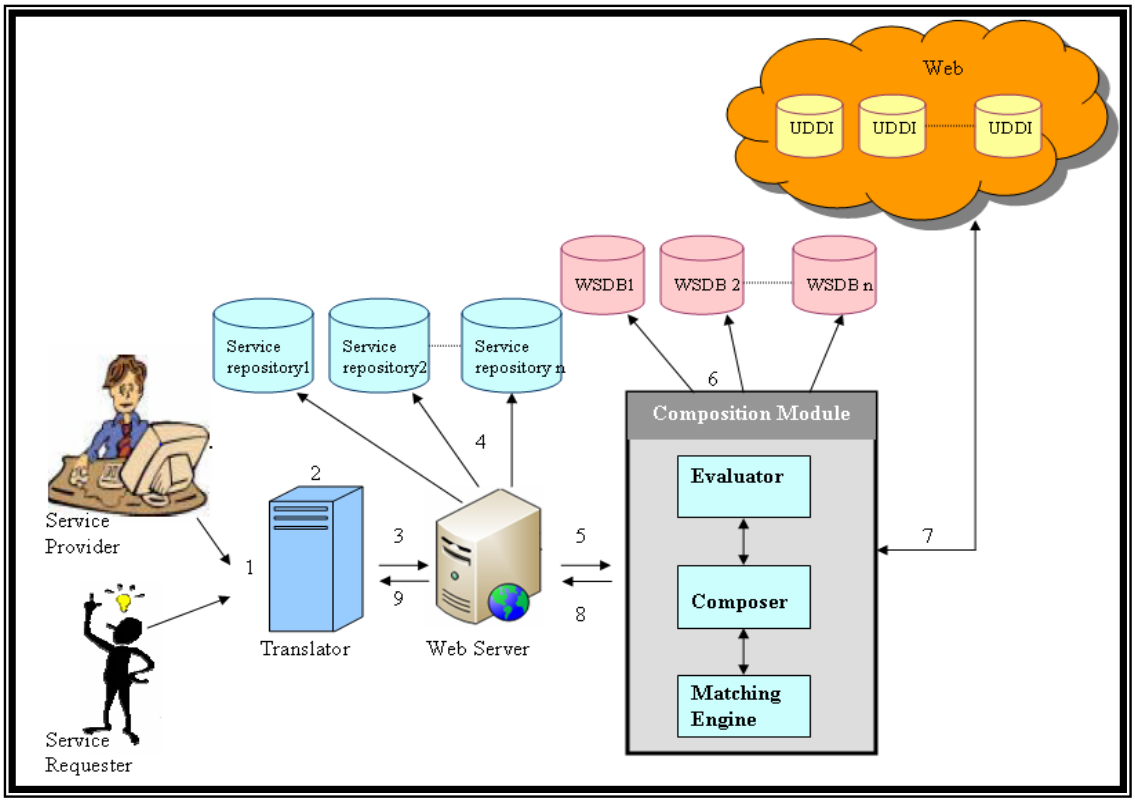


Fig 3.1: Proposed Framework for dynamic web services composition

3.3.3 Web Server

Web server is a computer program that accepts the user requests through http request and responds them back by http response which has data Contents. In proposed framework, service requester requests the desired service through web server; web server locates the desired service from different repositories and returns the address of service provider of desired service.

The primary function of a web server is to deliver web pages (HTML documents) and associated content (e.g. images, style sheets, JavaScripts) to clients. A client, commonly a web browser or web crawler, makes a request for a specific resource using HTTP and, if all goes well, the server responds with the content of that resource. The resource is typically a real file on the server's secondary memory, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes a way of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Many generic web servers also support server-side scripting (e.g. Apache HTTP Server and PHP). This means that a script can be executed by the server when a client requests it. Usually, this functionality is used to create HTML documents on-the-fly as opposed to return fixed documents. This is referred to as dynamic and static content respectively.

Highly riched web servers can be found in devices such as printers and routers in order to ease administration using a familiar user interface in the form of a web page.

3.3.4 Translator

The purpose of translator is to translate request/response from one language to another. We use translator so that all of the services are registered from external language to a language used by system and vice versa.

Translation is a pretty simple job, and can be broken down into the following subtasks:

1. Get text for translation and encode it into a HTTP POST request
2. Send the data to the web server.
3. Read the response back into a big string
4. Remove all the HTML and formatting and send the raw translated string back to the client.

Translation Web Services will work with other standards by providing them with the means of transferring content.

3.3.5 Evaluator

The evaluator evaluates selected web services on the basis of interface base and functionality base rules and returns the best selected service based on specified criteria.

3.3.6 Composer

The composer composes the selected component services in order to make a single desired web service.

3.3.7 Matching Engine

The purpose of matching engine is to match the user's request from the web services database. If match is found, it returns results back to web server. If not then select the web services from web, store/update them in database and then return results back to requested composer.

3.3.8 Service Repository/WSDB

The service registry and Web services database are used to register the web services by web service providers. Also they are used to request the user's desired web services.

Service repository build a customized, scalable and automated environment, enabling you to manage, trust and secure services, while eliminating costly redundancies, lowering maintenance costs and maximizing your existing IT investments.

- Ensure awareness of available applications, services and documents throughout the organization by publishing and finding them quickly, reliably and flexibly.

- Minimize critical outages and inefficiencies by helping manage and automate service upgrades and service level expectations.
- Quickly align business goals with IT and implement recommended practices by enabling consistent enforcement of operational and lifecycle governance policies.
- Reduce complexity and lower costs of securing services and applications with a scalable, standards-based solution.
- Help achieve and maintain industry/regulatory compliance with robust data protection, policy enforcement, and auditing capabilities to demonstrate compliance.

3.4 Working

The proposed framework includes the following phases:

3.4.1 Service Registration

Service registration is the process of registering new web services in database and service repository. The service providers register their web services in order to make it accessible for clients. It is the process of specification of web services to the system. New services are registered in registry through service registration process. There are several registries that different companies (service providers) maintained and all of them are synchronized after regular interval.

Service registry is a catalogue of services which helps in service definition, service selection and in enforcing service policies. Service repository consists of various artifacts/assets about the

services including functional specs, user and other documentation and various other service artifacts including SLAs that define transaction capacity, maximum throughput, downtime etc. While service registry is normally used to accommodate run-time assets service repository is used both for design time and run-time assets.

Following steps are used for web services registration:

1. Define the service's interface. This is done with *WSDL*
2. Implement the service. This is done with *Java*.
3. Define the deployment parameters.
4. Compile everything and generate a JAR file.
5. Deploy service.

To make a newly deployed Web service known to the community, the service producer should register it with an online Web service directory. Currently, several such directories are available, including www.xmethods.com and www.webservices.org. Service registration at these portals is accomplished by completing and submitting an online registration form. For example, www.xmethods.com registration form is available from www.xmethods.com/service. It requires the registrant to provide certain information, including service description, SOAP endpoint URL, method names, and WSDL URL. In the future, when UDDI becomes standardized and widely deployed, new Web service listings will be entered into UDDI registries via a standard process.

3.4.2 Service Request:

Through Service request procedure, clients request for desired web services either from repository or database.

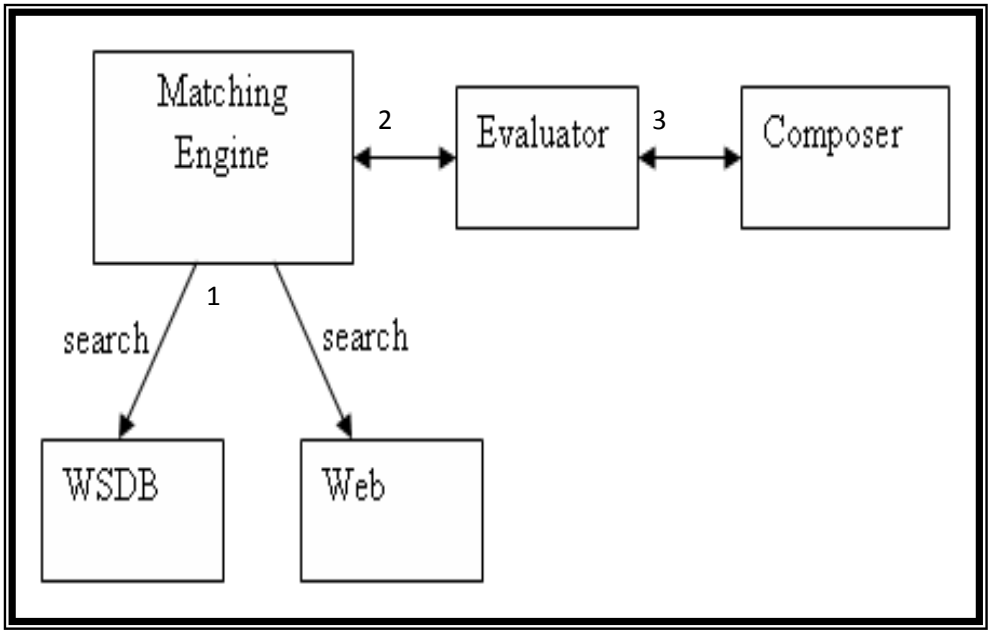


Fig 3.2: Composition module

3.4.3 Translation

It translates the user's request in order to search for a particular service. It translates the external form used by requester/provider into form used by system.

3.4.4 Web Server:

In the proposed model Web server is a software application which looks for a particular service from repository in order to fulfill user's request. It also passes the request to composition module if it does not find the particular service in its repository.

A Web Server does not function independently. It stores data and exchanges with other machines. Thus minimum two participants are required to exchange information. A client would request for the information and the server will have the information stored on it.

Each side needs software for exchanging the data. As far as the client is concerned, he has to use a Web browser like Internet explorer, Mozilla, Netscape, etc. On the server side, a variety of softwares are available. What type of server software you are able to run depends on the Operating System chosen for the server.

For example, Microsoft Internet Information Server is a popular choice for Windows NT, while those who prefer Unix choose Apache Web server.

The communication between the client machine and the Web server transpires as follows:

1. The client's browser divides the URL into different parts including address, path name and protocol.

2. The Domain Name Server translates the domain name into the corresponding IP address. This numeric combination represents the site's true address on the Internet.

3. The browser now decides which protocol should be used. A protocol, in common parlance, is a language which the client's machine uses to communicate with the server. FTP, HTTP, etc. are some such protocols.

4. The server sends a GET request to the Web Server to retrieve the address it has been given. It verifies that the given address exists, finds the necessary files, runs the appropriate scripts, exchanges cookies if necessary, and returns the results back to the browser.

5. The browser now converts the data into HTML and displays the results to the user. If the server cannot locate the file, the server sends an error message to the browser and eventually to the client.

This process continues for every request sent by the browser until the client leaves the site.

Besides this the Web server also has an additional number of responsibilities. Whereas a Web browser simply translates and displays data it is fed, a Web server has to distinguish between various error and data types.

A Web Server must, for example, designate the proper code for any sort of internal error and send that back to the browser immediately after it occurs. It also has to distinguish between various elements or file types on a Web page (such as GIFs, JPEGS, live audio or video files, etc.) so that the browser knows which files are saved in which format.

Depending on the site's function, a Web server may also have numerous additional tasks to handle, including logging statistics, handling security and encryption, serving images for other sites (for banners, pictures, etc), generating dynamic content, or managing e-commerce functions.

Web servers are responsible for storing and exchanging information with other machines/servers. So because of this, at least 2 participants are required for each exchange of information: a client, which requests the information, and a server, which stores it. Each side also requires a piece of software to negotiate the exchange of data; in the case of the client, a browser like Netscape or Internet Explorer is used. On the server side, however, things are not very simple. There is a myriad of software options available, but they all have a similar task: to negotiate data transfers between clients and servers via HTTP, the communication protocol of the Web. What type of server software you are able to run depends on the Operating System installed on the server. For example, Microsoft Internet Information Server is a popular choice for Windows NT, while many Linux fans choose Apache Web server.

3.4.5 Service Composition

This module has three sub modules that are Composer, Evaluator and Matching Engine. Composer composes the individual web services found from databases, Evaluator evaluate the selected web selected on the bases of its functionality and interfaces and Matching Engine match the requests against selected web services and return results back to web server.

3.4.6 Service Repository/WSDB:

The service repository and Web services database are used to register web services. Web server locates the requested service from Service repository, if it finds the service from repository, it returns result back. If not then it sends request to composition module which search the requested web services from WSDBs.

3.4.7 Web

In proposed framework, web is World Wide Web network where all service providers register their web services in UDDI registries. If desired web services are not found in repository or database then matching engine will search them from UDDI registries and save it in database for current and future use.

3.5 Methodology

The methodology followed of proposed model is given as:

1. The web services are registered in service repository.
2. Translator converts the query into language used by internal system, if it is required.
3. The request arrives at web server, which checks for the requested service from its service repository. If it finds the desired interface base service composition then it sends results back to requester.

4. Multiple repositories are used in which there is replication of data. The purpose of replicated repositories is, if one goes down or corrupted due to any failure/fault, then web server can locate services from other repository.
5. If server does not found requested service composition from repository then it send request to composition module.
6. In composition module matching engine select the desired services from web services database, Evaluator evaluates these web services in two steps. In first step it evaluates the web services on the basis of interface based search, whereas in second step it performs the evaluation on basis of functionality based rule. After evaluation it sends selected services to composer. The purpose of composer is to compose these component web services. Multiple WSDB's are introduced, so that if one goes down then we can make use of other DB's.
7. A timestamp (aging) is maintained with each URI in WSDB. If the request arrives before the expiration of that time then it look for the service in WSDB, otherwise request goes to the Web and from there it search the web services from multiple repositories and save them in WSDB. Also if matching engine does not finds the requested service from WSDB's then it sends request on the web for difference UDDI registries. The purpose of aging is also that it maintains the updated information about web services as the contents are refreshed each time when aging time expires.
8. The addresses of composed web services are sent back to composition requestor through matching engine.
9. Web server maintains a copy of these composed services in its WSDB for future use and returns the results back to client.

4 SYSTEM DESIGN

4.1 Data Flow Diagram:

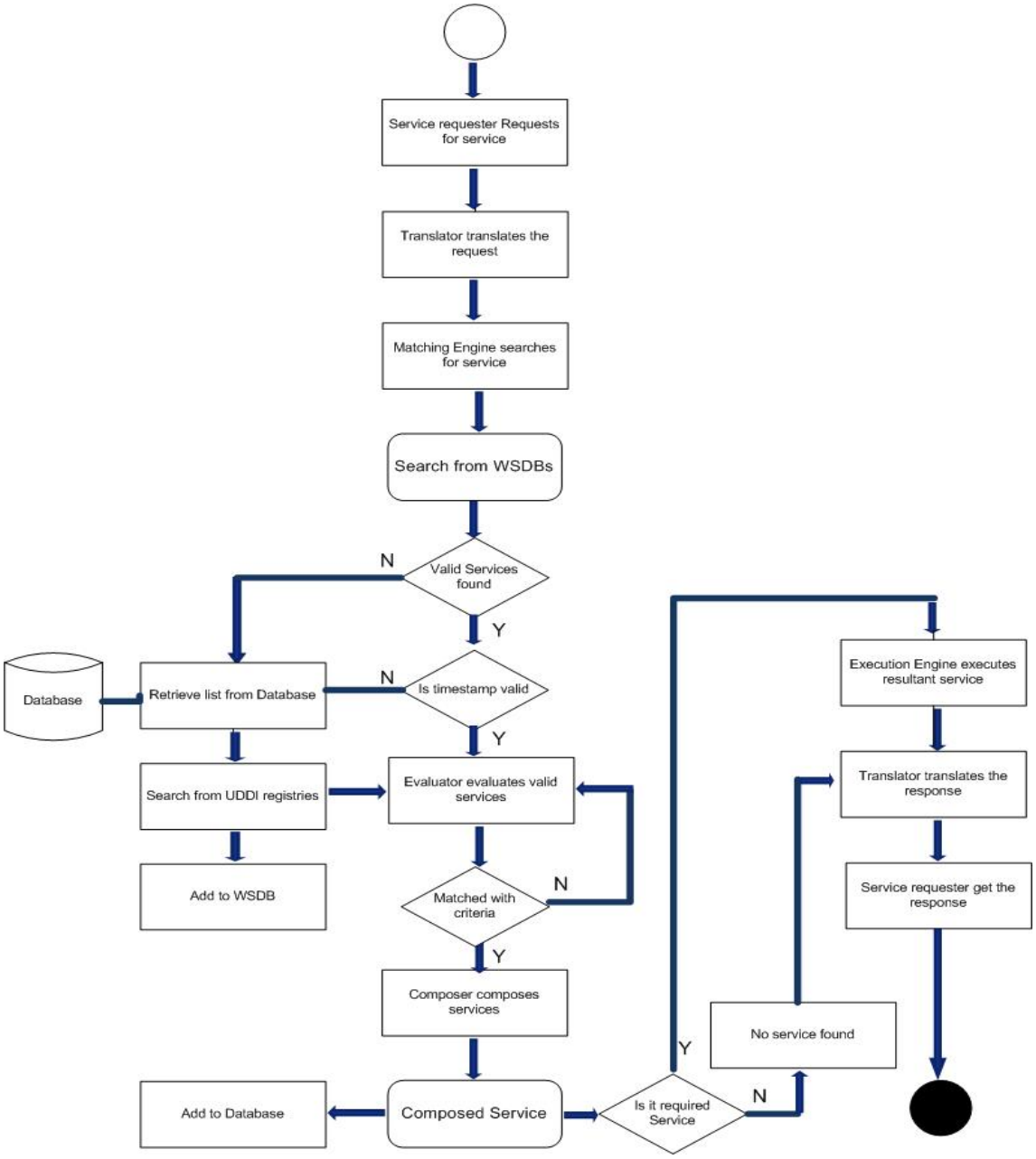


Fig 4.1: Web services composition DFD

This data flow diagram shows the flow of data for web services composition. The flow begins by entering the user request and then searching the desired services. The services are then composed in order to fulfill the desired request. The desired service is then discovered and composed. The composed service is returned to service requestor through translator. The complete flow is shown in figure.

4.2 Sequence Diagram

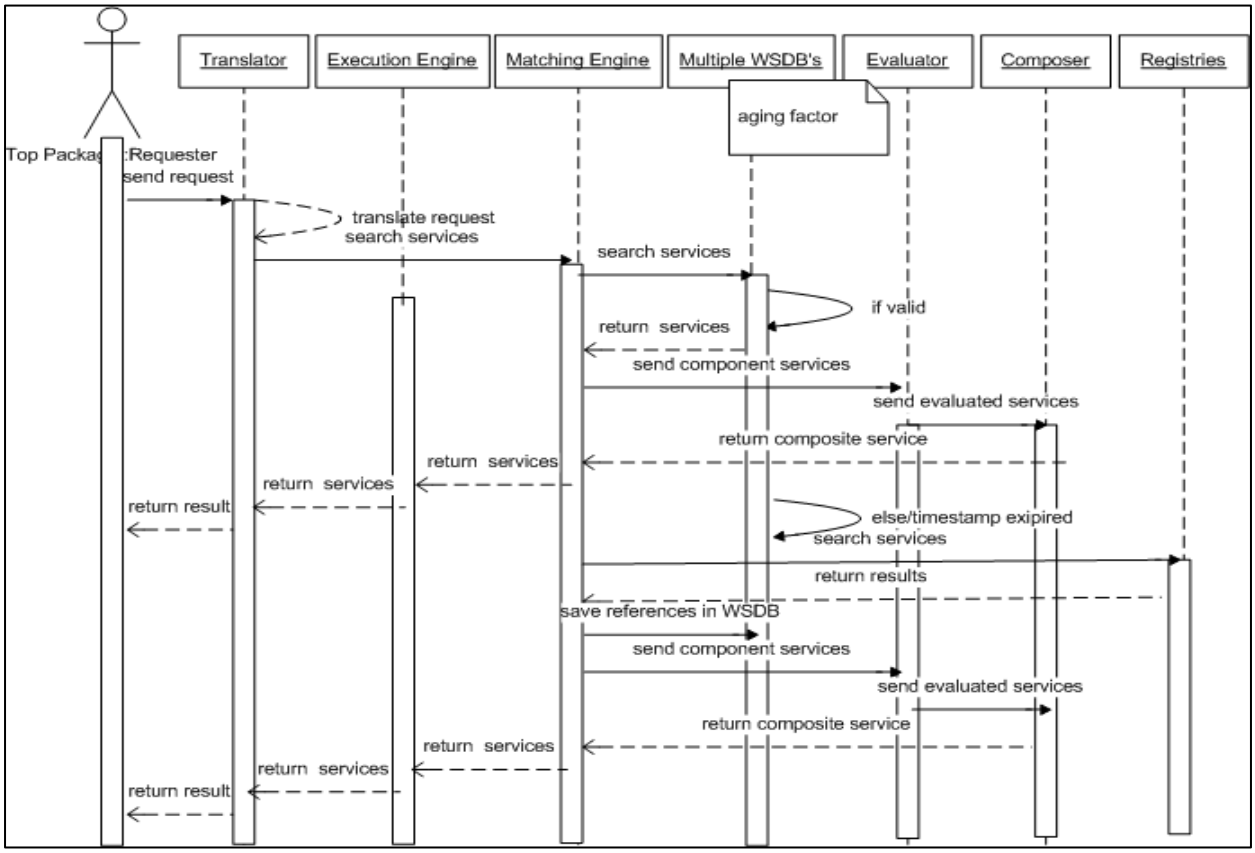


Fig 4.2: Sequence diagram of proposed framework

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the

order in which they occur. The given sequence diagram shows the complete sequence of steps starting from service request to web service composition. Matching engine searches the requested service and Execution engine executes the discovered service. Evaluator evaluates the discovered services on the basis of interface based and functionality based approaches. Finally composer composes and returned the desired service to Matching Engine.

4.3 Use Cases:

Design for Dynamic web services composition Application:

4.3.1 General Use cases for Interface

Use Case:

This use case diagram illustrates a set of use cases for the system, the actor APPLICATION USER and the relationship between actor and the use cases. In this use case diagram, following use cases have been shown: List available UDDIs, Create new UDDI, Delete UDDI, Edit UDDI, List available Businesses, List available Services, Compose Web Services and Available Composed Services.

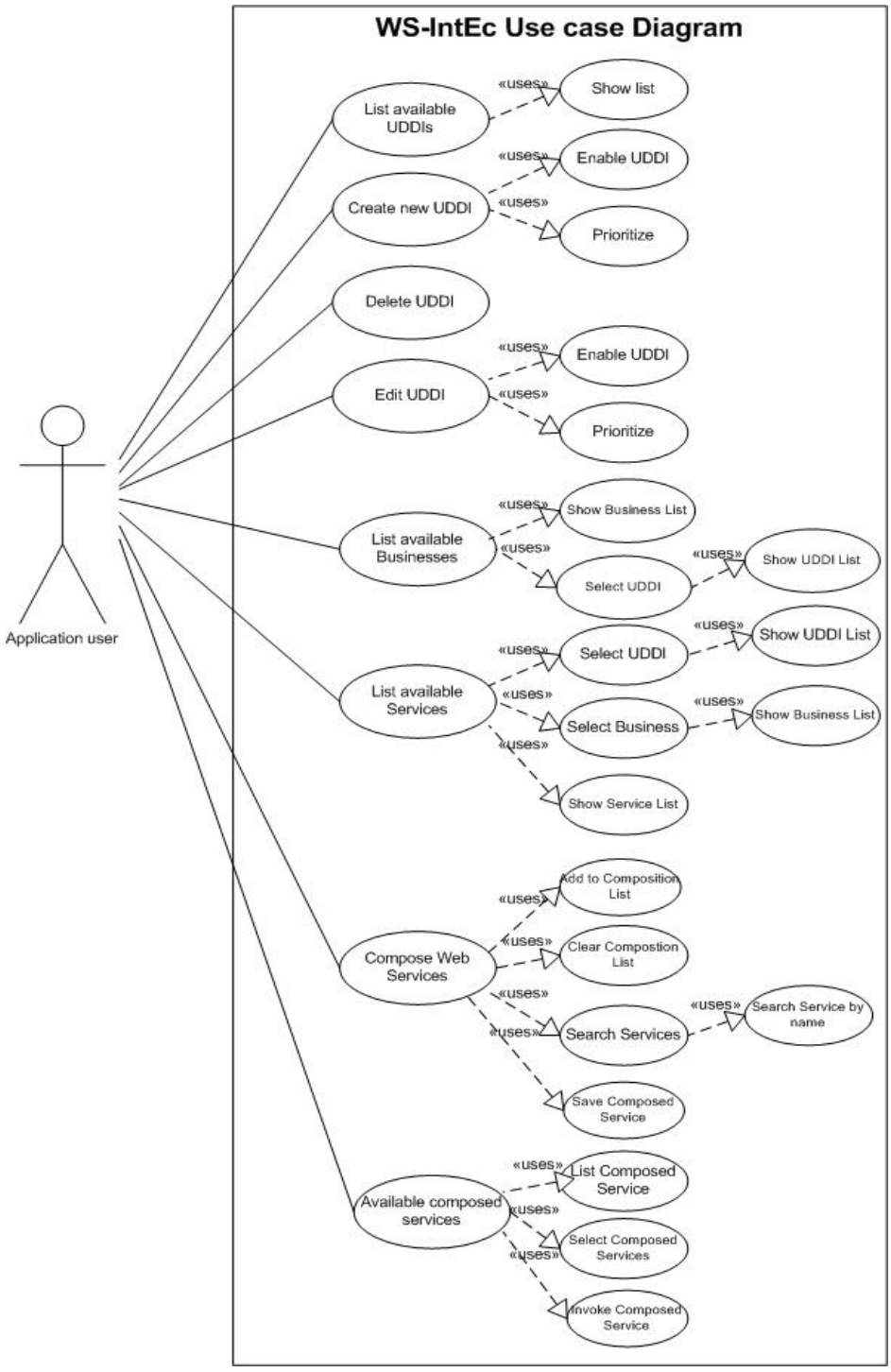


Fig 4.3: Web Services Composition Use Case Diagram

4.3.2 Extended Use cases for Interface

4.3.2.1 List available UDDIs:

Use case: List available UDDIs

Actors: APPLICATION USER

Pre Condition: APPLICATION USER views available UDDIs.

Post Condition: UDDIs are listed.

Description: APPLICATION USER lists UDDIs from application interface by clicking on list available UDDIs option.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER clicks on list UDDI option.	
	2. System shows the available UDDIs and its related information.

Alternative courses

Line 2: No UDDI is available, indicate an error.

4.3.2.2 Create new UDDI:

Use case: Create new UDDI

Actors: APPLICATION USER

Pre Condition: APPLICATION USER adds new UDDI.

Post Condition: New UDDI added in database.

Description: APPLICATION USER adds new UDDI from application interface by adding its relevant information.

Typical Course of Events

ACTIONS:	**SYSTEM RESPONSE:**
1. This use case begins when APPLICATION USER clicks on add new UDDI option.	
	2. System shows the new form asking relative information.
3. User enters details about UDDI that he wants to add.	
	4. New UDDI added in database
	5. System shows the enable UDDI option.
6. User checks Enable UDDI.	

	7. UDDI Enabled

Alternative courses

Line 2: USER enters Invalid information about the UDDI, indicate an error.

4.3.2.3 Edit UDDI:

Use case: Edit UDDI

Actors: APPLICATION USER

Pre Condition: APPLICATION USER edits UDDI.

Post Condition: UDDI information changed.

Description: APPLICATION USER edits UDDI related information from application interface by changing the relevant information in fields.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER clicks on edit UDDI option.	
	2. System shows the relevant form, listing all UDDIs.
3. USER than changes the UDDI related information by	

selecting particular UDDI.	
	4. System updates the UDDI details.
	5. System shows Enable UDDI option.
6. User Checks the Enable option.	
	7. UDDI Enabled.

Alternative courses

Line 2: USER enters Invalid information about the UDDI, indicate an error.

4.3.2.4 Delete UDDI:

Use case: Delete UDDI

Actors: APPLICATION USER

Pre Condition: APPLICATION USER deletes UDDI.

Post Condition: UDDI removed.

Description: APPLICATION USER deletes UDDI from application interface by selecting the particular UDDI and clicking on delete option.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER clicks on delete UDDI option.	
2. USER than select the particular UDDI to be deleted.	
	3. UDDI is selected.
4. User click on delete option.	
	5. UDDI deleted from database.

Alternative courses

Line 2: USER deletes the wrong UDDI.

4.3.2.5 List available Businesses:

Use case: List available Businesses

Actors: APPLICATION USER

Pre Condition: APPLICATION USER lists the Business.

Post Condition: Businesses Listed from UDDI database.

Description: APPLICATION USER views all the Businesses stored in database by clicking on list option.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when PUBLISHER clicks on list Business option.	
	2. System asks for selecting UDDI from drop down list.
3. User selects particular UDDI.	
4. User Clicks on show list button.	
	5. System makes available all the Businesses that are stored in selected UDDI database.

Alternative courses

Line 2: No Business is added in database, indicate an error.

4.3.2.6 List available Services:

Use case: List available Services

Actors: APPLICATION USER

Pre Condition: APPLICATION USER lists the Services.

Post Condition: Services Listed from UDDI database.

Description: APPLICATION USER views all the Services stored in database.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on list available Services option.	
	2. System asks for selecting UDDI.
3. User selects particular UDDI.	
4. User Clicks on show list button.	
	5. System makes available all the Services that are stored in selected UDDI database.

Alternative courses

Line 2: No Service is added in database, indicate an error.

4.3.2.7 List available Services:

Use case: List available Services

Actors: APPLICATION USER

Pre Condition: APPLICATION USER lists the Services.

Post Condition: Services Listed from UDDI database.

Description: APPLICATION USER views all the Services stored in database.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
6. This use case begins when USER clicks on list available Services option.	
	7. System asks for selecting UDDI and Selecting Business from drop down lists.
8. User selects particular UDDI.	
9. Now User selects Business from list.	
10. User Clicks on show list button.	
	11. System makes available all the Services that are stored in selected UDDI database.

Alternative courses

Line 2: No Service is added in database, indicate an error.

4.3.2.8 Compose Web Services:

Use case: Search Services

Actors: APPLICATION USER

Pre Condition: APPLICATION USER searches the composed service.

Post Condition: Composed service is searched.

Description: APPLICATION USER wants the composed service and searches the components services to make available the composed service.

Typical Course of Events

ACTIONS:	**SYSTEM RESPONSE:**
1. This use case begins when USER clicks on compose web services option.	
	2. System shows the search services tab.
	3. System asks the Service name.
4. User enter service name he wants to search.	
5. User Clicks on search button.	
	6. System makes available all the relevant Services stored in database.

7. User selects the service from available list.	
8. User clicks on “add to composition list” button.	
	9. System shows the complete detail of selected service.

Alternative courses

Line 2: No Service is added in database, indicate an error.

4.3.2.9 Compose Web Services:

Use case: Service Invocation

Actors: APPLICATION USER

Pre Condition: APPLICATION USER searches the composed service.

Post Condition: Composed service is searched.

Description: APPLICATION USER wants the composed service and searches the components services to make available the composed service.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on compose web	

services option.	
	2. System shows the service invocation tab.
	3. System shows the selected service WSDL URL, selected in previous use case.
4. User Clicks on “Get info” button.	
	5. System makes available all the methods available or that particular service.
6. User selects the method from list.	
7. User enters arguments for selected method.	
8. User clicks on “Invoke” button.	
	9. System shows the result.

Alternative courses

Line 2: No Service WSDL URL displayed, indicate an error.

4.3.2.10 Compose Web Services:

Use case: Compose Services

Actors: APPLICATION USER

Pre Condition: APPLICATION USER searches the composed service.

Post Condition: Composed service is searched.

Description: APPLICATION USER wants the composed service and searches the components services to make available the composed service.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on compose web services option.	
	2. System shows the compose services tab.
	3. System shows the list of services available for composition.
4. User enters service name and its method name.	
5. User enters input arguments.	
6. User clicks on “compose” button.	
	7. System shows the composed service.

Alternative courses

Line 2: No Service is added in database, indicate an error.

4.3.2.11 List Composed Services:

Use case: List Composed Services

Actors: APPLICATION USER

Pre Condition: APPLICATION USER lists the Services.

Post Condition: Services Listed from UDDI database.

Description: APPLICATION USER views all the Services stored in database by clicking on list option.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on list Service option.	
2. USER views all the added Services.	
	3. System makes available all the Services that are stored in UDDI database.

Alternative courses

Line 2: No Service is added in database, indicate an error.

4.3.2.12 Clear Log:

Use case: Clear Log

Actors: APPLICATION USER

Pre Condition: APPLICATION USER clears log.

Post Condition: Log is cleared.

Description: APPLICATION USER wants the Log detail to clear and log is cleared by clicking on clear log button.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on clear Log button on main screen.	
	2. System removes all Log detail and log is cleared.

Alternative courses

Line 2: User wants to clear the Log that is already cleared, indicate an error.

4.3.2.13 Application Exit:

Use case: Application Exit

Actors: APPLICATION USER

Pre Condition: APPLICATION USER wants to exit from Application.

Post Condition: Application closed.

Description: APPLICATION USER wants the exit from application and clicks on exit button. As a result application closed.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
3. This use case begins when USER clicks on “exit” button on main screen.	
	4. Application close.

Alternative courses

Line 2: User double clicks on exit button, indicate an error.

Chapter 5

5 Implementation

5.1 Apache jUDDI

jUDDI is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for Web Services.

5.1.1 jUDDI Features

- Open Source
- Platform Independent
- Use with any relational database that supports ANSI standard SQL (MySQL, Oracle, DB2, Sybase, Derby etc.)
- Deployable on any Java application server that supports the Servlet 2.3 specification
- jUDDI registry supports a clustered deployment configuration.
- Easy integration with existing authentication systems
- Supports InVM embeddable mode

5.1.2 jUDDIv3

- UDDI Specification version 3.0 compliant
- JDK 1.6 Recommended, but supports for JDK 1.5 and later
- Build on JAXB and JAX-WS standards, tested on CXF

- Build on JPA standard, tested with OpenJPA and Hibernate
- Pre-configured bundle deployed to Tomcat
- UDDI portlets
- Pre-configured portal bundle download
- User and Developer Documentation

5.1.3 jUDDIv2

- UDDI version 2.0 compliant
- Supports for JDK 1.3.1 and later

5.1.4 jUDDI Architecture:

jUDDI consist of a core request processor that un-marshals incoming UDDI requests, invoking the appropriate UDDI function and marshalling UDDI responses (marshalling and un-marshalling is the process of converting XML data to/from Java objects).

To invoke a UDDI function jUDDI employs the services of three configurable sub-components or modules that handle persistence (the DataStore), authentication (the Authenticator) and the generation of UUID's (the UUIDGen). jUDDI is bundled and pre-configured to use default implementations of each of these modules to help the registry up and running quickly.

5.1.5 Persistence (jUDDI DataStore)

jUDDI needs a place to store it's registry data so jUDDI is pre-configured to use JDBC and any one of several different DBMSs to do this.

jUDDI is Apache's Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for web services. It's integration with Managed Method's JaxView is a significant enhancement that was requested by the customers themselves. jUDDI registry integration is especially helpful to smaller companies with a limited budget for managing their SOA or cloud architecture.

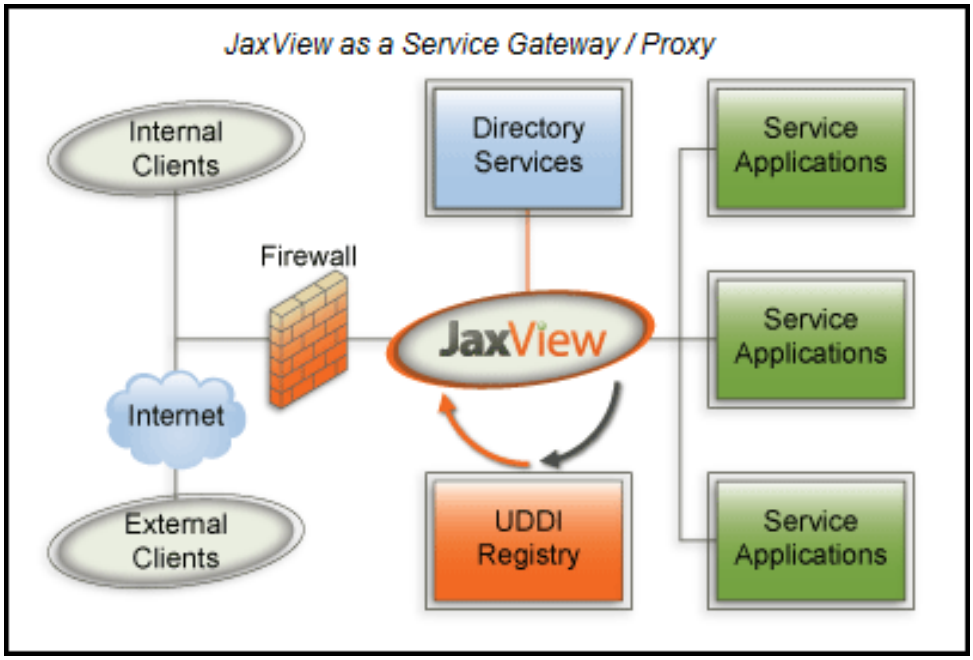


Fig 5.1: JaxView

JaxView now provides closed loop policy publishing (security, routing, etc.) to jUDDI as WS-Policy attachments, and it provedes closed loop governance with the jUDDI registry and repository. Service status reports and enforcement results can be attached to the jUDDI service asset. SLA policies can be modified by pushing them down to JaxView from jUDDI or pushing modified SLA policies into jUDDI. The integration also allows JaxView to read new policies from the jUDDI repository and automatically enforce them at runtime.

jUDDI is built on JAXB, JAX-WS, and JPA standards. The UDDI is a major component of web architectures that allows organizations and applications to quickly find and use web services. UDDI also supports the maintenance of operational registries. JaxView adds more capabilities with jUDDI and other governance solutions including IBM, Oracle, and Software AG.

JaxView includes tools for monitoring and managing standards-based (SOAP) and REST web services. It enables proxy, patent pending agent-less, and agent based deployments for managing cloud or SOA environments. JaxView is built on a thin-client, server-based application architecture that is accessible through a web browser.

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 again 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.

1. Querying an UDDI registry
2. Saving and updating information in an UDDI registry
3. Suppressing information from an UDDI registry
4. Various Ruddi™ API examples

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.

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.

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 JAXR

The Java API for XML Registries (JAXR) provides a uniform and standard Java API for accessing different kinds of XML Registries. An XML registry is an enabling infrastructure for building, deploying, and discovering Web services.

JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. Simplicity and ease of use are facilitated within JAXR by a unified JAXR information model, which describes content and metadata within XML registries.

Current implementations of JAXR support ebXML Registry version 2.0, and UDDI version 2.0. More such registries could be defined in the future. JAXR provides an API for the clients to interact with XML registries and a service provider interface (SPI) for the registry providers so they can plug in their registry implementations. The JAXR API insulates application code from the underlying registry mechanism. When writing a JAXR based client to browse or populate a registry, the code does not have to change if the registry changes, for instance from UDDI to ebXML.

JAXR provides rich metadata capabilities for classification and association, as well as rich query capabilities. As an abstraction-based API, JAXR gives developers the ability to write registry client programs that are portable across different target registries. This is consistent with the Java philosophy of "Write Once, Run Anywhere." Similarly, JAXR also enables value-added capabilities beyond those of the underlying registries.

The current version of the JAXR specification includes detailed bindings between the JAXR information model and both the ebXML Registry and the UDDI Registry v2.0 specifications.

JAXR works in synergy with related Java APIs for XML, such as Java API for XML Processing (JAXP), Java Architecture for XML Binding (JAXB), Java API for XML-based RPC (JAX-RPC), and SOAP with Attachments API for Java (SAAJ), to enable Web services within the Java 2 Platform, Enterprise Edition (J2EE).

5.3.1 JAXR Goals

Following functionalities are supported by JAXR API.

1. Support for industry-standard XML registry functionality
2. Support for registration of member organizations and enterprises
3. Support for submission and storing of arbitrary registry content
4. Support for lifecycle management of XML and non-XML registry content
5. Support for user-defined associations between registry content
6. Support for user-defined multi-level classification of registry content along multiple user defined facets
7. Support for registry content querying based on defined classification schemes
8. Support for registry content querying based on complex ad hoc queries
9. Support for registry content querying based on keyword based search
10. Support for sharing of Web services
11. Support for sharing of business process between partners
12. Support for sharing of schemas between partners

13. Support for sharing of business documents between partners
14. Support for trading partner agreement assembly and negotiation
15. Support for schema assembly
16. Support for heterogeneous distributed registries
17. Support for enabling publish/subscribe XML Messaging between parties

JAXR is expected to support not only UDDI, but other similar registry standards (such as ebXML) as well.

5.3.2 JAXR architecture

The JAXR architecture defines three important architectural roles:

- A **registry provider** implements an existing registry standard, such as the OASIS (Organization for the Advancement of Structured Information)/ebXML Registry Services Specification 2.0.
- A **JAXR provider** offers an implementation of the JAXR specification approved by the Java Community Process (JCP) in May 2002. Currently, the JAXR reference implementation 1.0 offers a JAXR UDDI provider implementation.
- A **JAXR client** is a Java program that uses JAXR to access the registry provider via a JAXR provider. A JAXR client can be either a standalone J2SE (Java 2 Platform, Standard Edition) application or J2EE components, such as EJBs (Enterprise JavaBeans), Java Servlets,

or JSPs (JavaServer Pages). The JAXR reference implementation also supplies one form of a JAXR client, a Swing-based registry browser application.

Figure 1 illustrates how diverse JAXR clients interoperate with diverse registries using JAXR. Architecturally, JAXR clients use the API to perform registry operations, while JAXR providers implement the API. Since JAXR offers a standard API for accessing diverse registry providers and a unified information model to describe registry contents, JAXR clients, whether HTML browsers, J2EE components, or standalone J2SE applications, can uniformly perform registry operations over various registry providers.

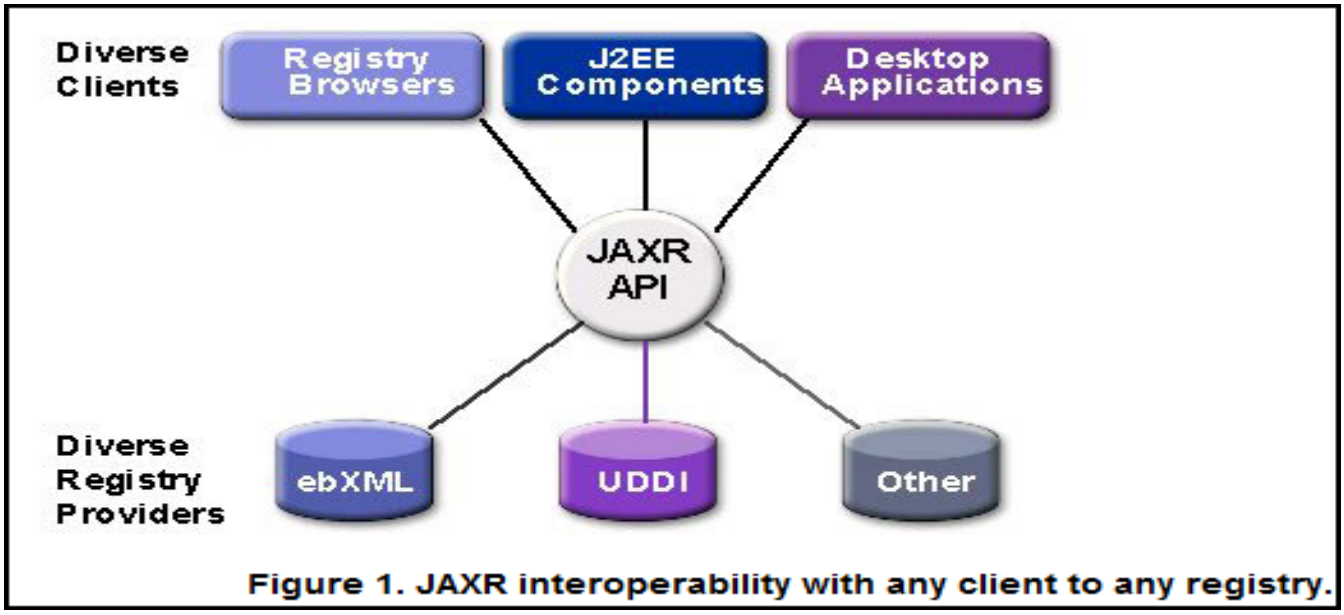


Figure 1. JAXR interoperability with any client to any registry.

Figure 2 shows a high-level view of the JAXR architecture. The JAXR provider shown is a JAXR pluggable provider with underlying implementations of a UDDI-specific JAXR provider and an ebXML-specific provider. The JAXR provider exposes capability-specific methods to the

JAXR client via the RegistryService interface. The JAXR client queries the RegistryService and discovers the provider capability level via the CapabilityProfile interface.

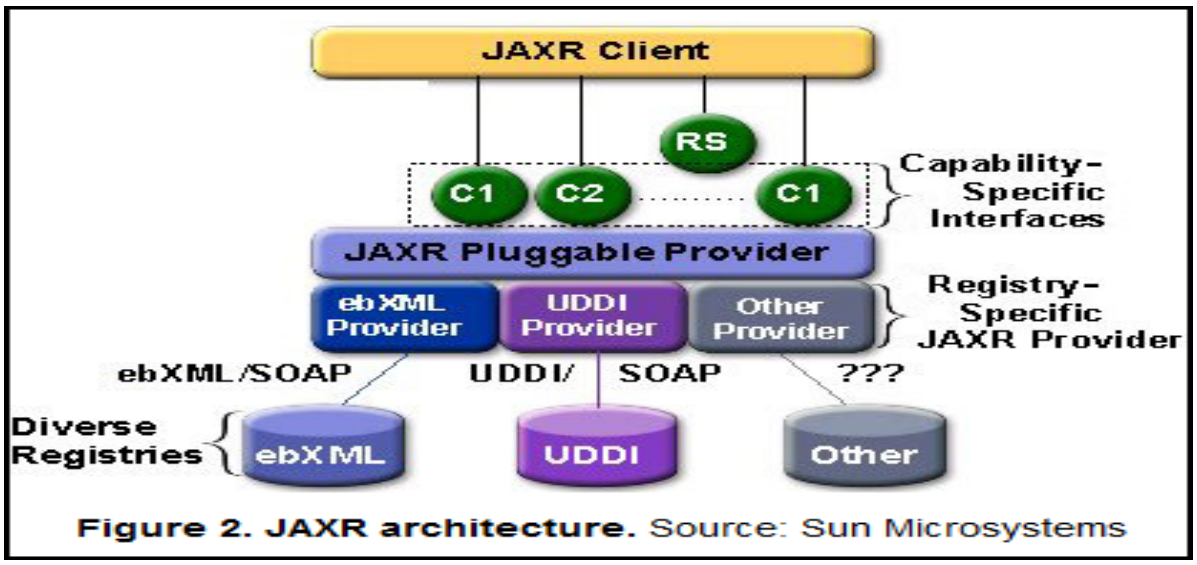


Figure 2. JAXR architecture. Source: Sun Microsystems

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.

5.5 WSIF

The Web Services Invocation Framework (WSIF) is a simple Java API for invoking Web services, no matter how or where the services are provided. Please refer to the release notes before you proceed with using WSIF.

WSIF enables developers to interact with abstract representations of Web services through their WSDL descriptions instead of working directly with the Simple Object Access Protocol (SOAP) APIs, which is the usual programming model. With WSIF, developers can work with the same programming model regardless of how the Web service is implemented and accessed.

WSIF allows stubless or completely dynamic invocation of a Web service, based upon examination of the meta-data about the service at runtime. It also allows updated implementations of a binding to be plugged into WSIF at runtime, and it allows the calling service to defer choosing a binding until runtime.

Finally, WSIF is closely based upon WSDL, so it can invoke any service that can be described in WSDL.

5.5.1 Overview

WSIF stands for the Web Services Invocation Framework. It supports a simple Java API for invoking Web services, no matter how or where the services are provided. The framework allows maximum flexibility for the invocation of any WSDL-described service.

In the WSDL specification, Web service binding descriptions are *extensions* to the specification. So the SOAP binding, for example, is one way to expose the abstract functionality (*and there could be others*). Since WSIF mirrors WSDL very closely, it also views SOAP as just one of several ways you might wish to expose your software's functionality. WSDL thus becomes a normalized description of software, and WSIF is the natural client programming model.

The WSIF API allows clients to invoke services focusing on the abstract service description - the portion of WSDL that covers the port types, operations and message exchanges without referring to real protocols. The *abstract invocations* work because they are backed up by protocol-specific pieces of code called *providers*. A provider is what conducts the actual message exchanges according to the specifics of a particular protocol - for example, the SOAP provider that is packaged with WSIF uses a specific SOAP engine like Axis to do the real work.

The decoupling of the abstract invocation from the real provider that does the work results in a flexible programming model that allows dynamic invocation, late binding, clients being unaware of large scale changes to services - such as service migration, change of protocols, etc. WSIF also allows new providers to be registered dynamically, so you could enhance your client's capability without ever having to recompile its code or redeploy it.

Using WSIF, WSDL can become the centerpiece of an integration framework for accessing software running on diverse platforms and using widely varying protocols. The only precondition is that you need to describe your software using WSDL, and include in its description a binding that your client's WSIF framework has a provider for. WSIF defines and comes packaged with providers for local java, EJB, JMS, and JCA protocols. That means you

can define an EJB or a JMS-accessible service directly as a WSDL binding and access it transparently using WSIF, using the same API you would for a SOAP service or even a local java class.

5.5.2 WSIF Structure

In WSDL a binding defines how to map between the abstract PortType and a real service format and protocol. For example, the SOAP binding defines the encoding style, the SOAPAction header, the namespace of the body (the targetURI), and so forth.

WSDL allows there to be multiple implementations for a Web Service, and multiple Ports that share the same PortType. In other words, WSDL allows the same interface to have bindings to for example, SOAP and IIOP.

WSIF provides an API to allow the same client code to access any available binding. As the client code can then be written to the PortType it can be a deployment or configuration setting (or a code choice) which port and binding it uses.

WSIF uses 'providers' to support these multiple WSDL bindings. A provider is a piece of code that supports a WSDL extension and allows invocation of the service through that particular implementation. WSIF providers use the J2SE JAR service provider specification making them discoverable at runtime.

Clients can then utilize any new implementations and can delegate the choice of port to the infrastructure and runtime, which allows the implementation to be chosen on the basis of quality of service characteristics or business policy.

5.5.3 Reason for using WSIF

Imagine complicated Enterprise software system consisting of various pieces of software, developed over a period of tens of years - EJBs, legacy apps accessed using Java's connector architecture, SOAP services hosted on external servers, old code accessed through messaging middleware. You need to write software applications that use all these pieces to do useful things; yet the differences in protocols, mobility of software, etc. comes in the way.

The software you use moves to a different server, so your code breaks. The SOAP libraries you use change - say for example you moved from using Apache SOAP to Apache Axis - so your code breaks since it uses a now deprecated SOAP API. Something that was previously accessible as an EJB is now available through messaging middleware via JMS - again, you need to fix the code that uses the software. Or lets suppose you have an EJB which is offered as a SOAP service to external clients. Using SOAP obviously results in a performance penalty as compared to accessing the EJB directly. Of course, SOAP is a great baseline protocol for platform and language independence, but shouldn't java clients be able to take advantage of the fact that the software they are accessing is really an EJB? So your java customers pay a performance penalty since you have to use SOAP for to accommodate you non-java clients.

WSIF fixes these problems by letting you use WSDL as a *normalized description* of disparate software, and allows you to access this software in a manner that is independent of protocol or location. So whether it is SOAP, an EJB, JMS (or potentially .NET and other software frameworks), you have an API centered around WSDL which you use to access the functionality. This lets you write code that adapts to changes easily. The separation of the API from the actual

protocol also means you have flexibility - you can switch protocols, location, etc. without having to even recompile your client code. So if your an externally available SOAP service becomes available as an EJB, you can switch to using RMI/IIOP by just changing the service description (the WSDL), without having to make any modification in applications that use the service. You can exploit WSDL's extensibility, its capability to offer multiple bindings for the same service, deciding on a binding at runtime, etc.

Chapter 6

6 Results and Discussion

Measuring the performance of web service composition and execution framework is non-trivial. Generally a framework is evaluated by implementing the framework and then using a dataset to test the web services discovery, composition and execution based on calculating Precision, Recall and Fallout. 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. The hardware environment used is Intel Pentium IV 2.0GHz Core2Duo CPU with 4GB RAM and Windows Vista Home Premium operating system.

6.1 Precision

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

$$Precision = \frac{|\{Relevant\ Web\ Services\} \cap \{Retrieved\ Web\ Services\}|}{|\{Retrieved\ Web\ Services\}|}$$

6.2 Recall

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

$$Recall = \frac{|\{Relevant\ Web\ Services\} \cap \{Retrieved\ Web\ Services\}|}{|\{Relevant\ Web\ Services\}|}$$

6.3 Fall-Out

The proportion of non-relevant web services that are retrieved out of all non-relevant services available.

$$Fall - out = \frac{|\{Non \quad Relevant\ Web\ Services\} \cap \{Retrieved\ Web\ Services\}|}{|\{Non - Relevant\ Web\ Services\}|}$$

Table 6.1 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 6.2 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 6.3 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?

6.4 Dataset

The framework is implemented in Java 6 using Netbeans 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. WSIF (Web Service Invocation Framework) is used to invoke and execute the discovered services after composition. For the evaluation of our framework, we have setup 50 UDDIs, on which 500 businesses are registered with a total of around 4500 web services WSDL references present. The actual services are hosted by the service providers on their web servers.

6.5 Performance Evaluation

The performance of the proposed approach is evaluated using all of the factors discussed above. We test the framework for web service discovery and log the values for Precision, Recall and

Fall-Out. Also, we compare these values with the existing frameworks and show where our framework has improved the discovery, composition and execution. After discovery, the services are available for evaluation. We log the timings for different type of services having various number of methods exposed. Later, we log the composition time depending on the number of services being composed and the size of the service space. At the end, we present comparison with an existing technique to present the improvements of our framework.

6.5.1 Average Precision

We take various sets of services and for each set we make 25 readings and then compute an average for that set. We start with a service set of 1000 web services and then keep on increasing the number of web services to 1500, 2000, 2500, and finally 3000. Following is the average precision of our framework.

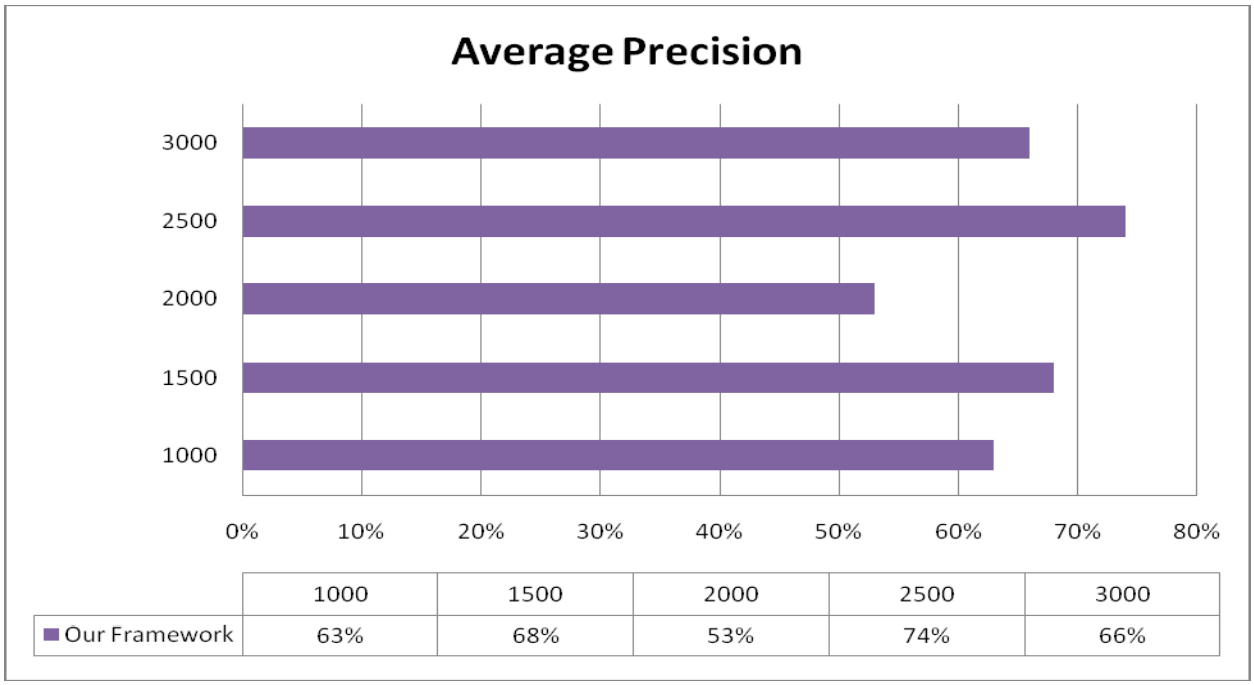
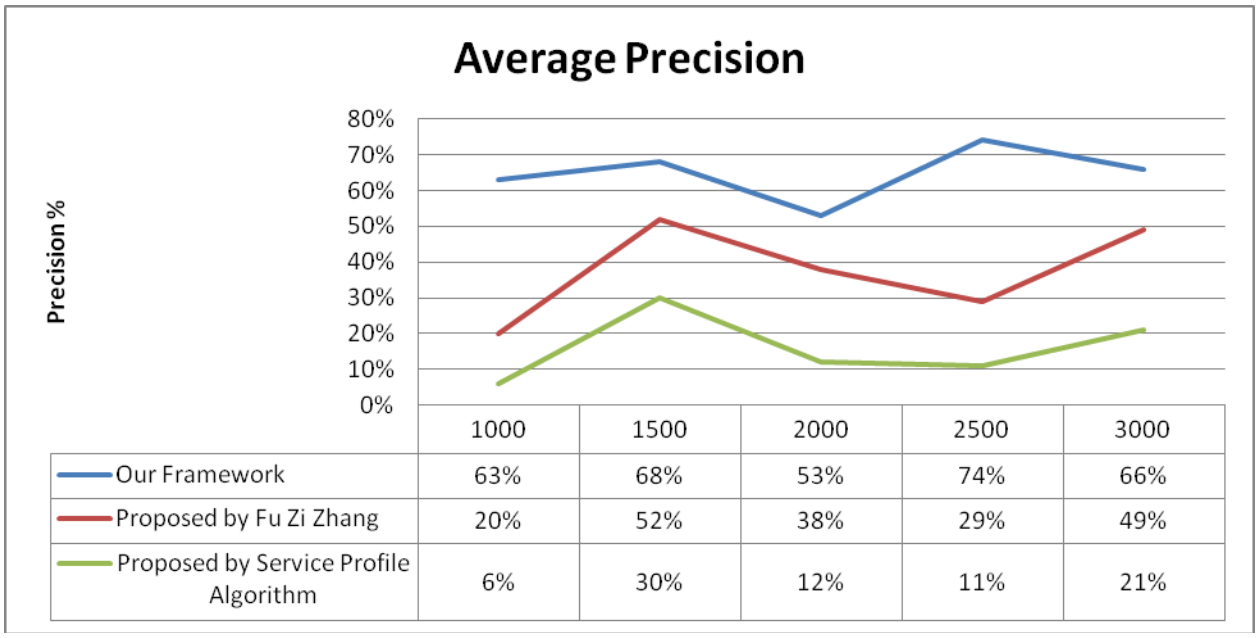


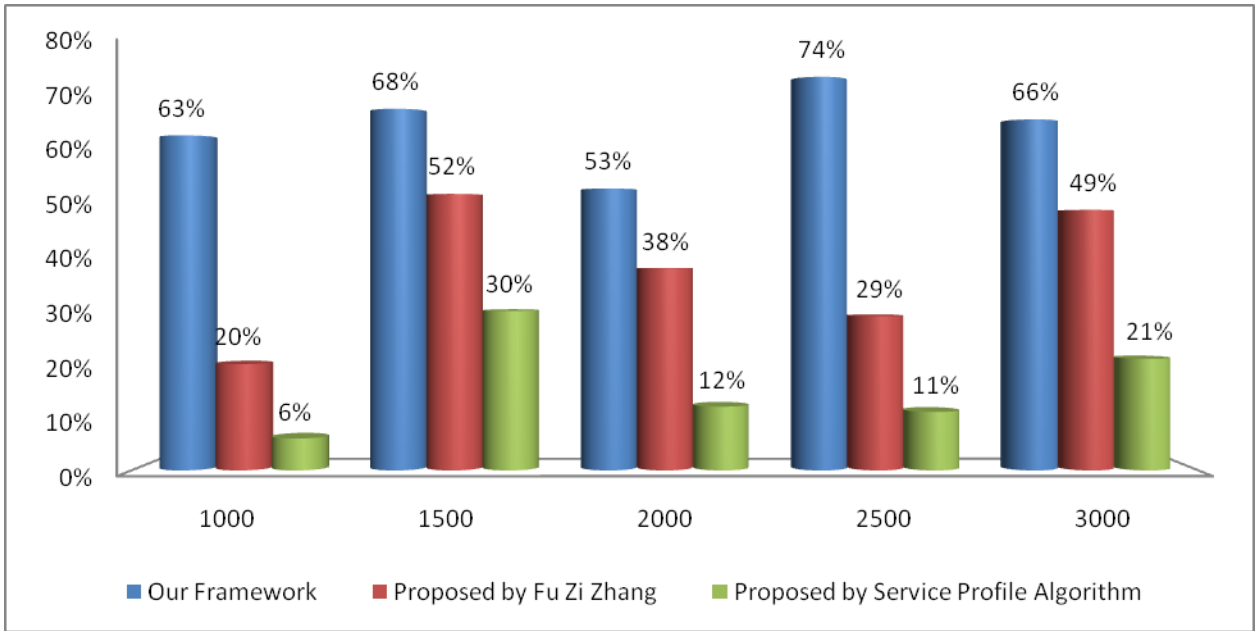
Fig 6.1: Average Precision

We compare our framework with couple of other techniques. First technique "A Web service discovery algorithm based on dynamic composition" is proposed by Fu Zhi Zhang et al. The other technique "A Software Framework for Matchmaking Based on Semantic Web Technology" is proposed by Lei Li and Ian Horricks. The analysis of the results clearly shows that our technique has greatly improved the precision.



	1000	1500	2000	2500	3000
Our Framework	63%	68%	53%	74%	66%
Proposed by Fu Zi Zhang	20%	52%	38%	29%	49%
Proposed by Service Profile Algorithm	6%	30%	12%	11%	21%

Figure 6.2: Comparison of Precision with Other Techniques



6.5.2 Average Recall

We take various sets of services and for each set we make 25 readings and then compute an average for that set. We start with a service set of 1000 web services and then keep on increasing the number of web services to 1500, 2000, 2500, and finally 3000.

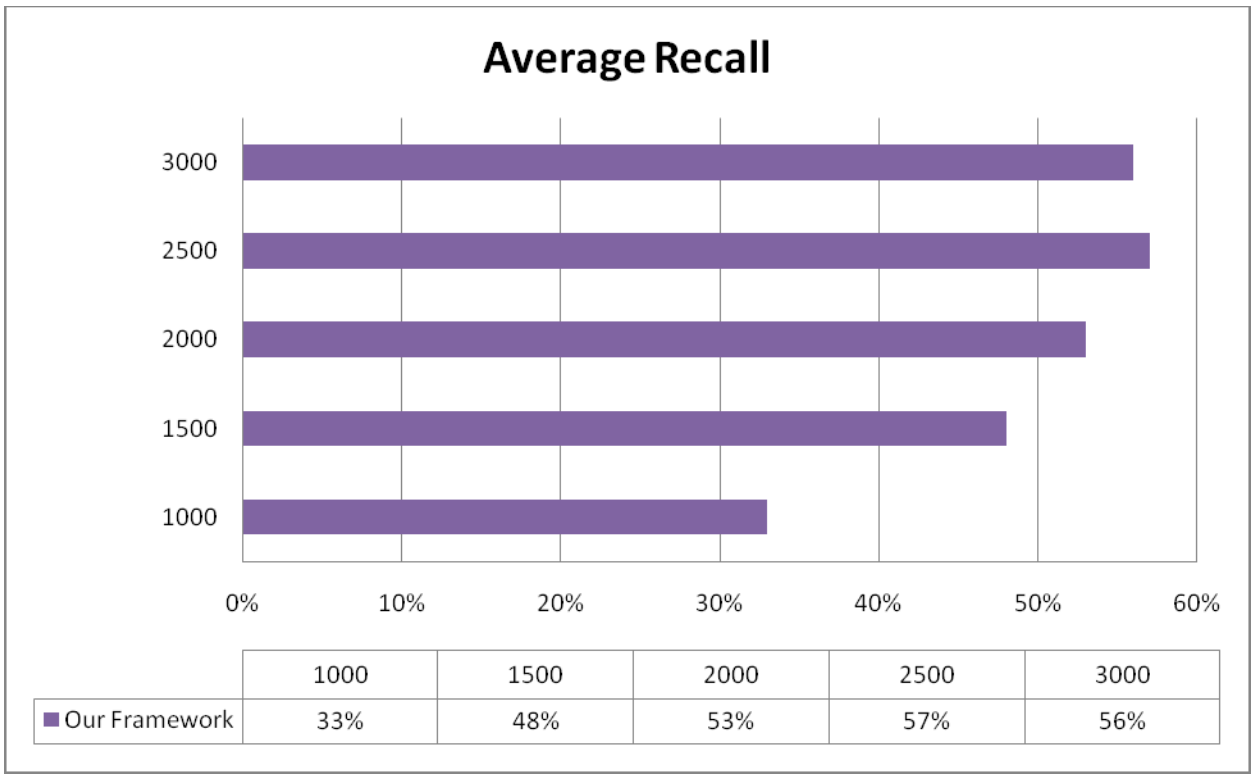
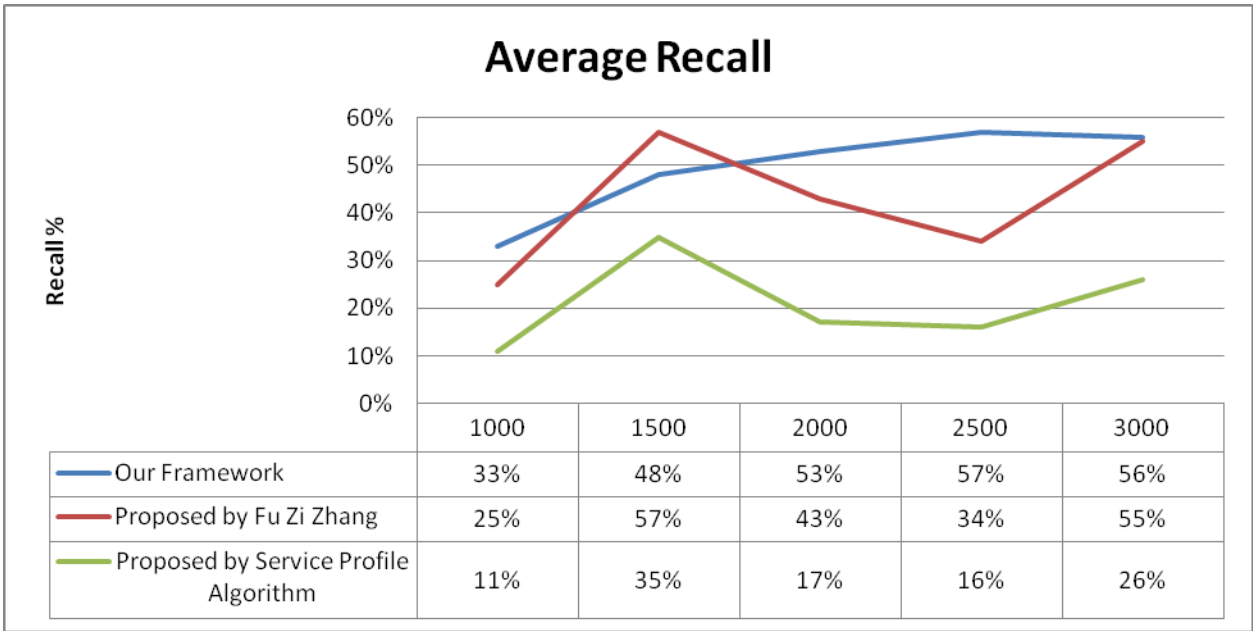


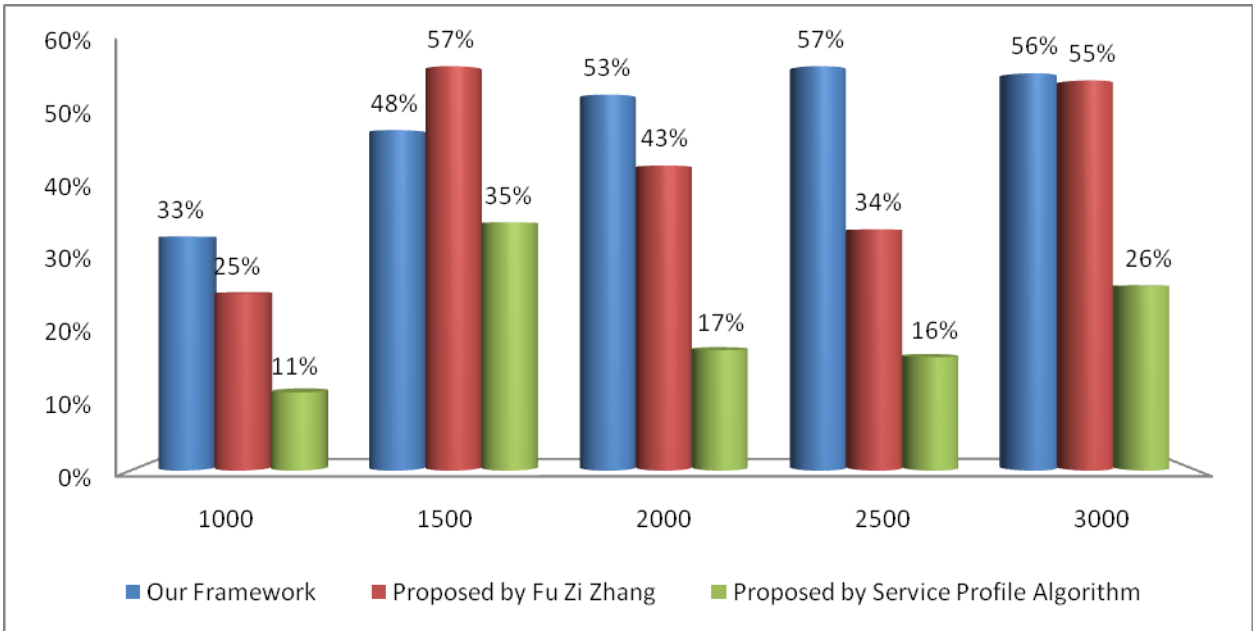
Fig 6.3: Average Recall

Similar to the precision, we compare our framework with the other two techniques.



	1000	1500	2000	2500	3000
Our Framework	33%	48%	53%	57%	56%
Proposed by Fu Zi Zhang	25%	57%	43%	34%	55%
Proposed by Service Profile Algorithm	11%	35%	17%	16%	26%

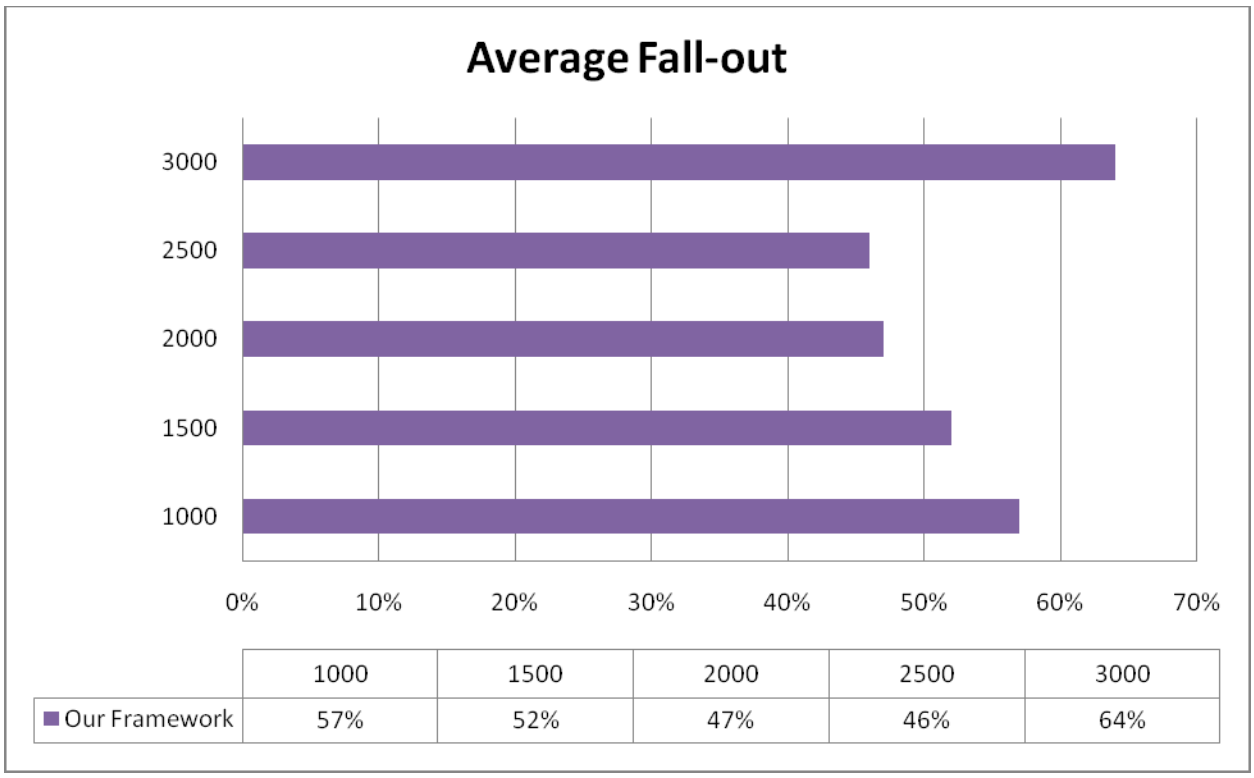
Fig 6.4: Comparison of Recall with Other Techniques



The analysis of the results clearly shows that our technique has greatly improved the recall.

6.5.3 Average Fall-out

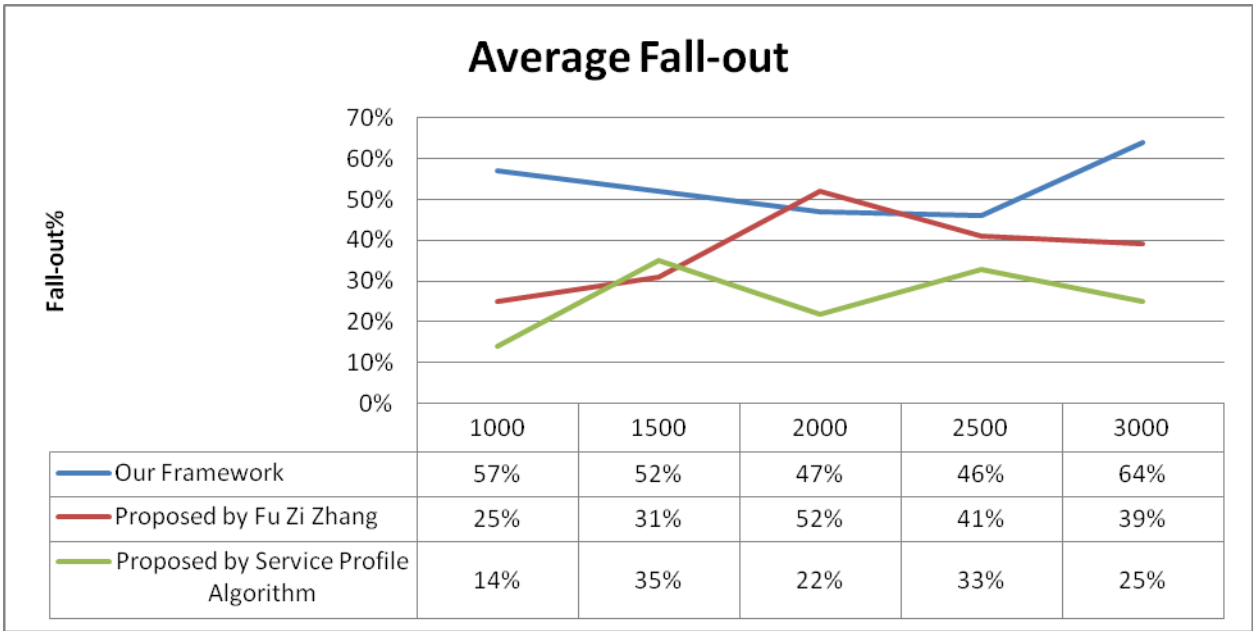
We take various sets of services and for each set we make 25 readings and then compute an average for that set. We start with a service set of 1000 web services and then keep on increasing the number of web services to 1500, 2000, 2500, and finally 3000.



	1000	1500	2000	2500	3000
Our Framework	57%	52%	47%	46%	64%

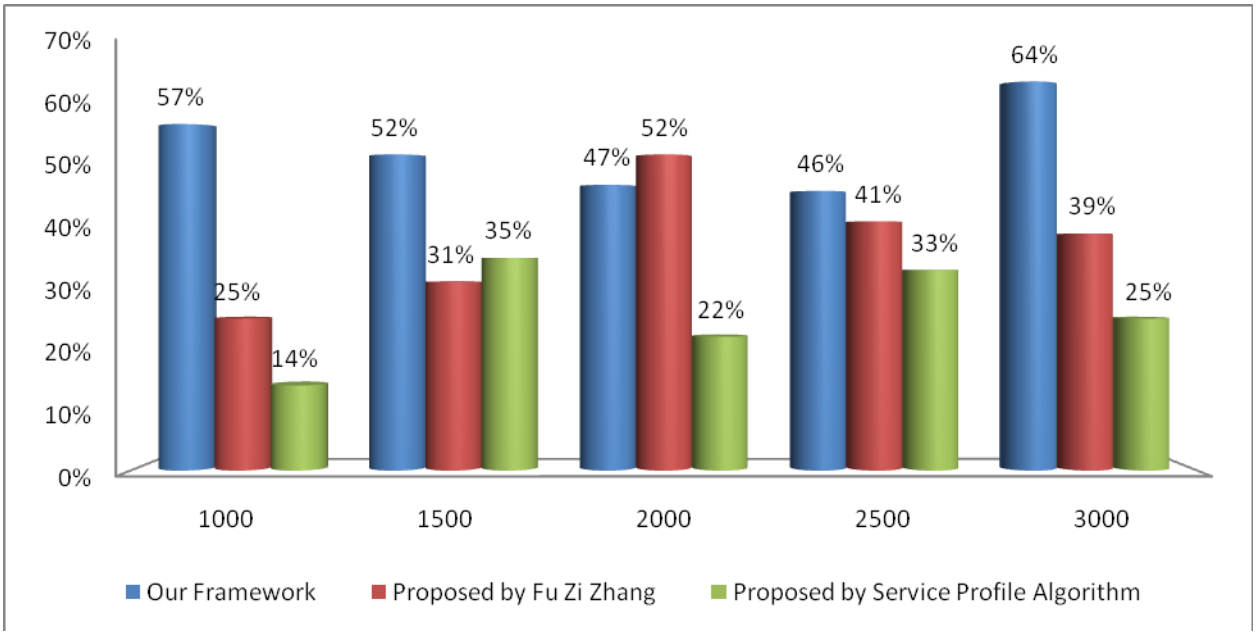
Figure 6.5: Average Fall-out

We also compare these values with the other two techniques.



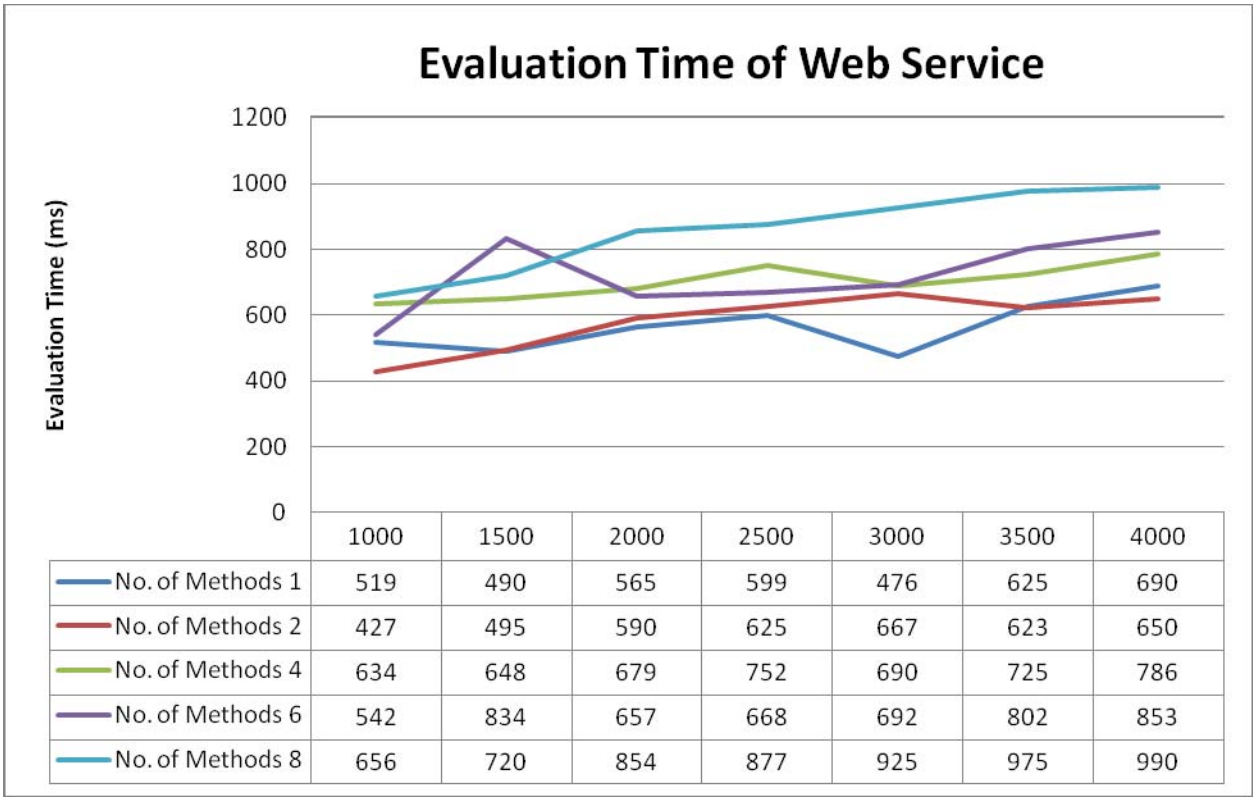
	1000	1500	2000	2500	3000
Our Framework	57%	52%	47%	46%	64%
Proposed by Fu Zi Zhang	25%	31%	52%	41%	39%
Proposed by Service Profile Algorithm	14%	35%	22%	33%	25%

Figure 6.6: Comparison of Average Fall-out with Other Techniques



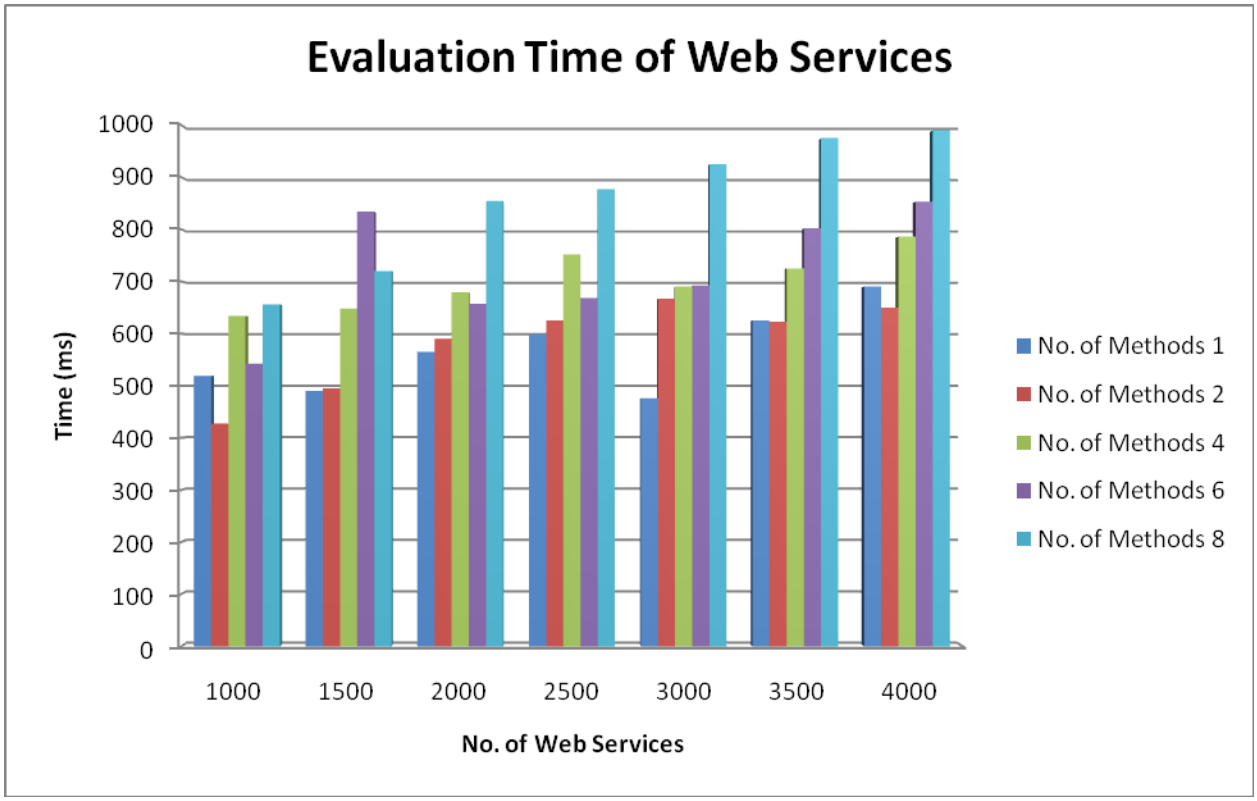
6.5.4 Evaluation Time of Services

WSDL4J is used to parse the WSDL file of the web service. Once the service is discovered, we must know the methods that it presents to be used from outside world. We log the evaluation time of web service for various number of methods exposed by the web service. Once again we randomly evaluate the web service and then log the timings. Following is the analysis of the evaluation time.



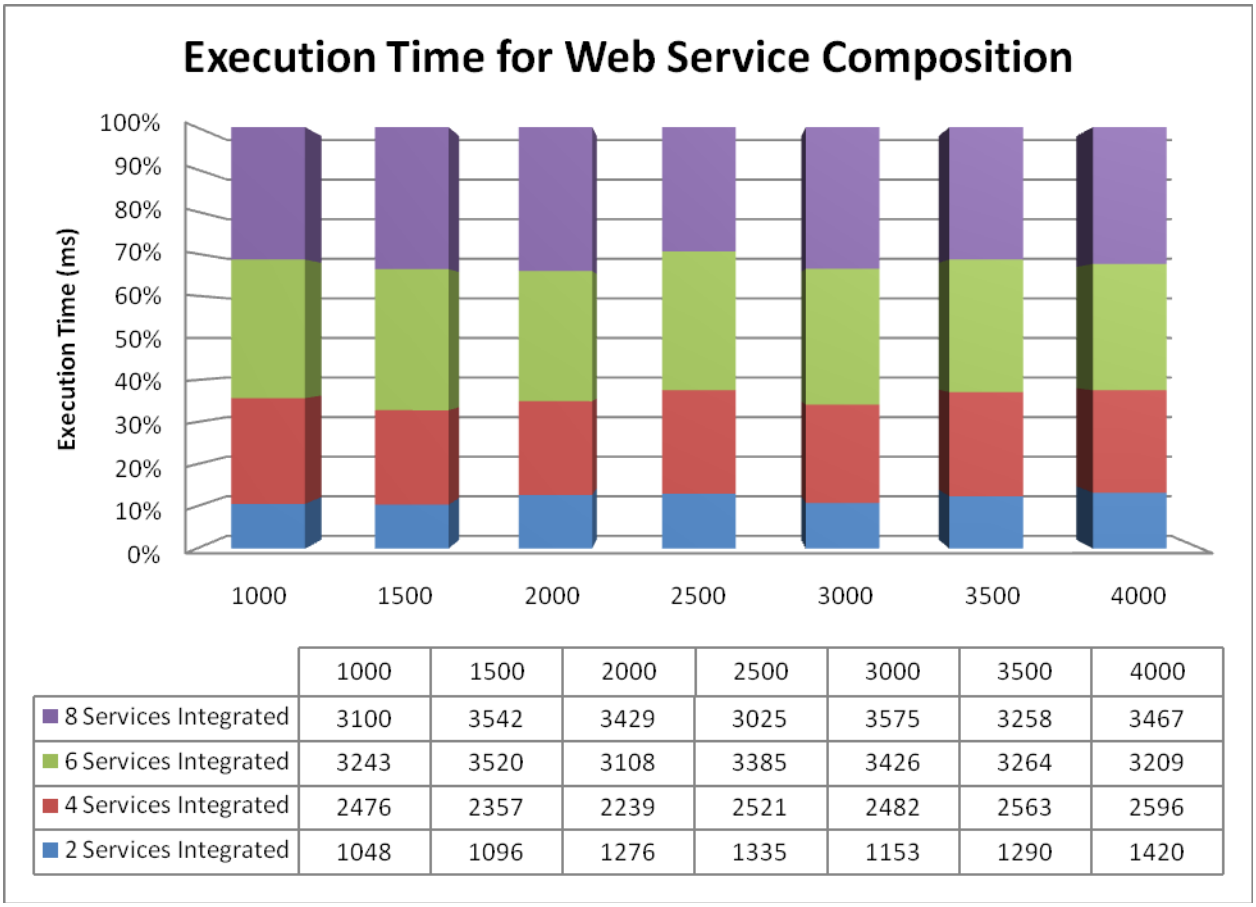
	1000	1500	2000	2500	3000	3500	4000
No. of Methods 1	519	490	565	599	476	625	690
No. of Methods 2	427	495	590	625	667	623	650
No. of Methods 4	634	648	679	752	690	725	786
No. of Methods 6	542	834	657	668	692	802	853
No. of Methods 8	656	720	854	877	925	975	990

Fig 6.7: Evaluation Time of Web Service



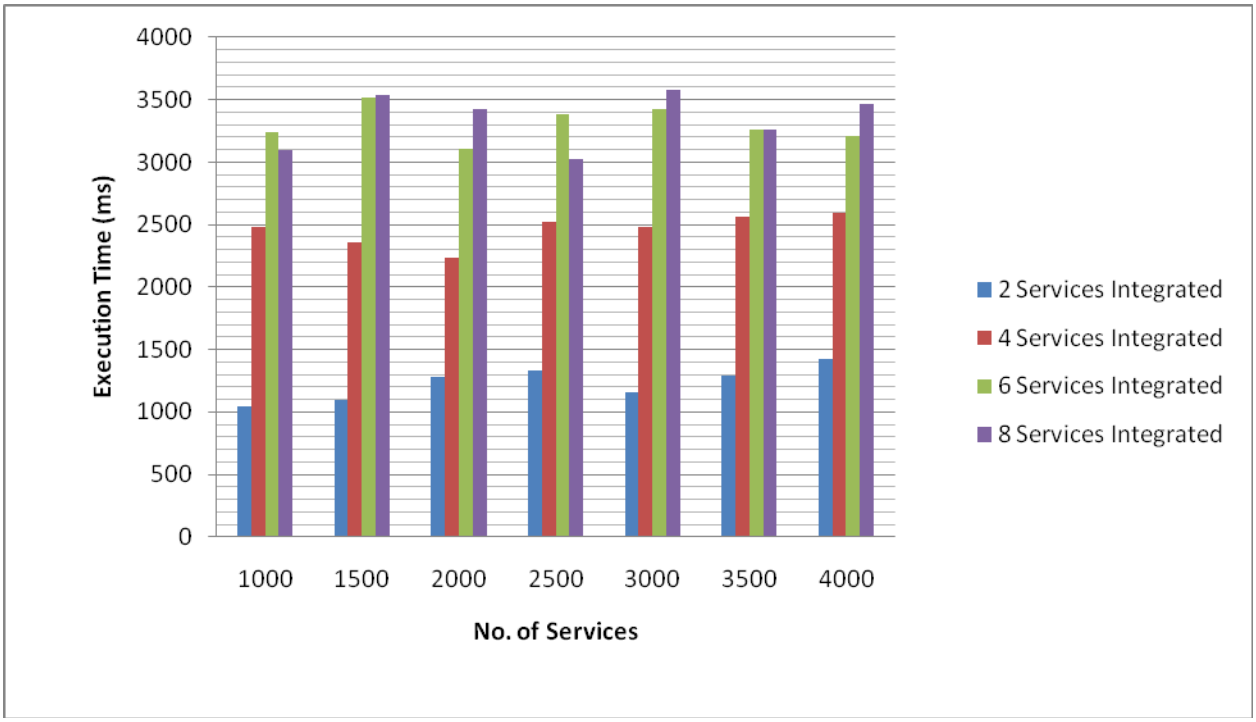
6.5.5 Execution Time for Web Service Composition

Web Service Invocation Framework (WSIF) is used to invoke the method of web service after discovery and evaluation. We randomly compose web services to get fruitful output. Then, we note the time for the execution of composite web service. We log the times for web services composed of 2, 4, 6 and 8 services. Following is the graphical analysis of execution time of web service composition.



	1000	1500	2000	2500	3000	3500	4000
8 Services Integrated	3100	3542	3429	3025	3575	3258	3467
6 Services Integrated	3243	3520	3108	3385	3426	3264	3209
4 Services Integrated	2476	2357	2239	2521	2482	2563	2596
2 Services Integrated	1048	1096	1276	1335	1153	1290	1420

Fig 6.8: Execution Time for Web Service Composition



6.6 Comparison of Various Factors

We compare our framework results with those of the framework described by Faisal Mustafa et al. in “Dynamic Web Service Composition”.

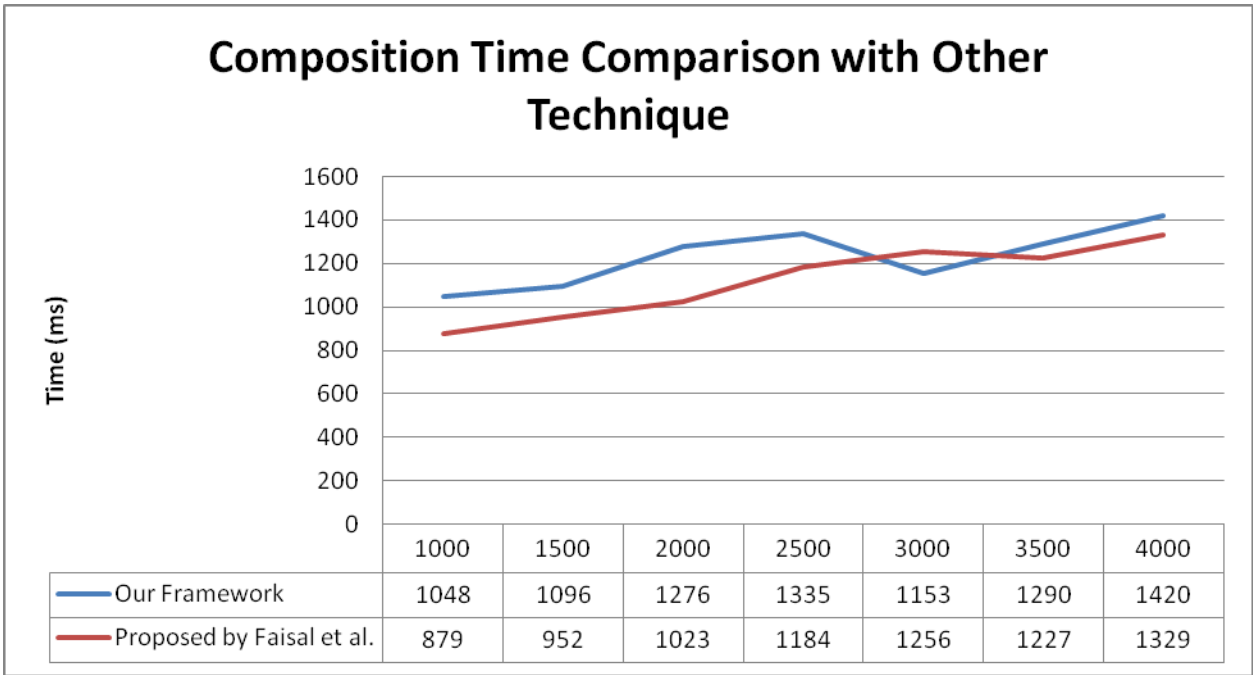


Fig 6.9: Composition Time Comparison with Other Technique

We use the static, dynamic and statistical factors to make a detailed comparison.

Table 6.4 Comparison of Static Factors

Factor	Our Framework	Proposed by Faisal et al.
Regulatory	UDDIv2, UDDIv3, SOAP, AXIS	UDDI v2, SOAP
Security	Yes	Yes

Table 6.5 Comparison of Dynamic Factors

Factor	Our Framework	Proposed by Faisal et al.
Service Availability	Service may be hosted on multiple servers to improve service reliability	Service may be hosted on multiple servers to improve service reliability

Network Availability	Multiple paths available which allows efficient network utilization	Only one path available. If this is congested, then the system fails.
Execution Duration	Normal	Normal

Table 6.6 Comparison of Statistical Factors

Factor	Our Framework	Proposed by Faisal et al.
Service Reliability	Service may be hosted on multiple servers to improve service reliability	Service may be hosted on multiple servers to improve service reliability
Network Reliability	Multiple registries make it possible to reach the service from multiple paths which increases network reliability	Only one path available, if it fails, network reliability cannot be guaranteed.
Execution Reliability	Normal	Normal
Reputation	Average	Average

6.7 Comparison with Other Factors

There are a few other factors that are of interest. Following is the comparison based on those factors.

Table 6.7 Comparison with Other Factors

Factor	Our Framework	Proposed by Faisal et al.

Decentralized Approach	Multiple web servers are available. If one fails, the framework is still active using some other web server.	Single web server is present which presents a centralized approach. If this fails, the system collapses.
Multiple Databases	Database replication allows the system to keep operating even if a database crashes	Single database. The system fails if it crashes.
New Services	Having a timestamp allows to use new services	No technique to use new services.

Chapter 7

7 Summary and Conclusion

7.1 Overview of Research

The research lies in the field of dynamic web services composition selection. At this stage, automated dynamic web service composition development process is still under development, although some automated tools and proposals are available. The full automation of this dynamic process is still an ongoing research activity. In this thesis, we have discussed the main problems faced by dynamic web services composition, such as execute ability, data distribution and its effect on QoS. We also tried to elaborate the main differences and advantages of web services over distributed application development. Based on an analysis of current problems, we have introduced a model of dynamic services. In the proposed model we try to fix current issues for dynamic composition. In last chapter, we have provided the implementation of proposed algorithm and compare the performance with existing approaches and presented the results. The analysis shows that proposed approach has better results then existing ones.

7.2 Achievements

In this thesis we have proposed the dynamic web services composition algorithm to solve the composition issues related to data distribution, reliability, availability and QOS. A framework is proposed by combination of interface based and functionality 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, our algorithm eliminates the dynamic service composition and execution issues, supports web service composition considering QOS(Quality of Services), efficient data retrieval and updation, 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 our 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 dynamic web service composition development process is still under development, although some automated tools and proposals are available. The full automation of this dynamic process is still an ongoing research activity.

7.4 Future Work

Nothing is perfect in this world and no work is ever perfect, there is always room for improvement. Similarly, in this, although we did a lot of hard work 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. We will also be looking into deeper details of every component of the framework to ensure better and efficient composition.

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, dynamic web services discovery and utilization. The framework is flexible, scalable and new services can easily be inserted/updated in local cache and UDDI registries. Through the proposed approach requester always retrieve up to date services because a timestamp is attached with each repository and when it expires, services are updated. Also through local cache there is fast retrieval of services as requester doesn't need to go to web each time for discovery of services. If they are present in local repositories then in less time services can easily be discovered. Interoperability between service providers and requesters is achieved through Translator. CSP solver selects the service satisfied specified constraints, which is a part of matching engine. Thus the proposed algorithm fix current issues of dynamic web services discovery. At last, we have provided the implementation of proposed

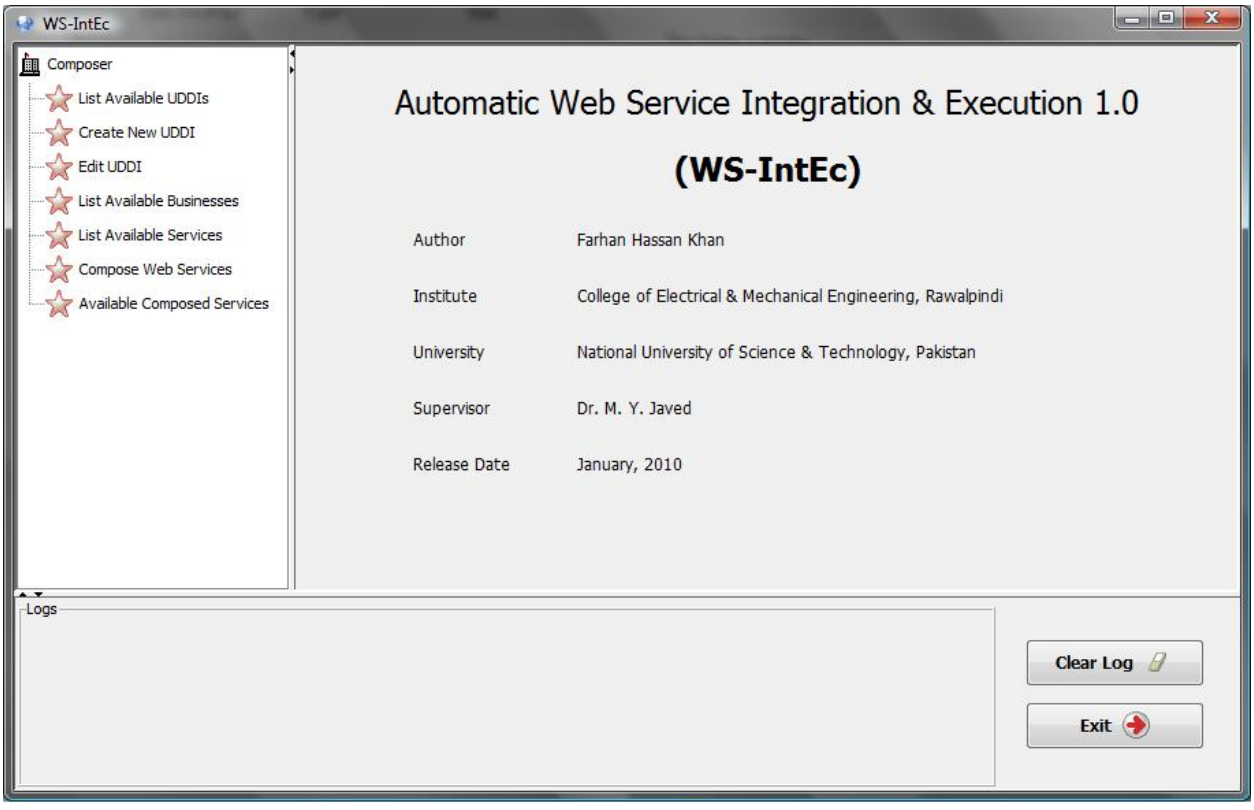
framework and the results shows greater performance and accuracy in dynamic discovery mechanism of web services resolving the existing issues of flexibility, scalability, based on quality of services, and discovers updated and most relevant services with ease of usage.

8 APPENDIX A

User Manual

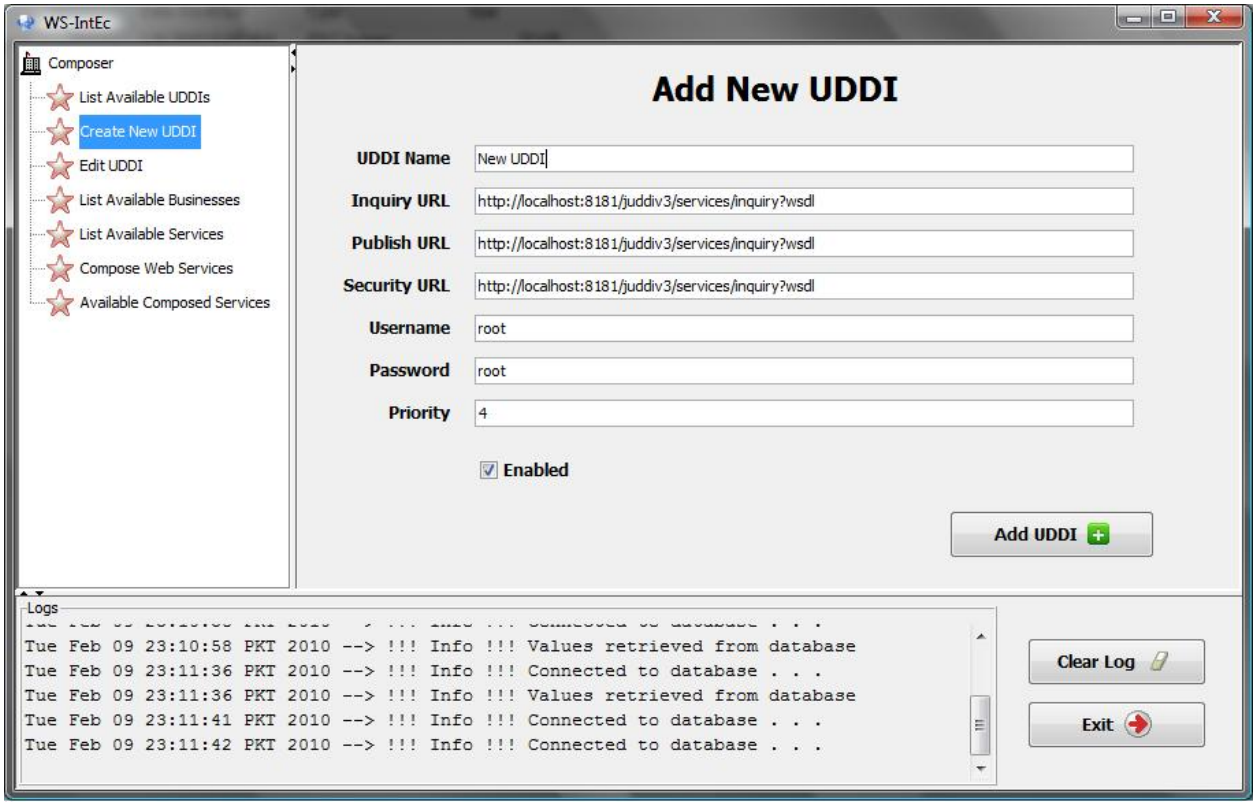
Main Screen

Double click the executable jar file to run the application. 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. On the bottom of the main page, we have a log panel that displays the logs about the current system events happening. There are also buttons to Clear the Log panel and Exit the application.



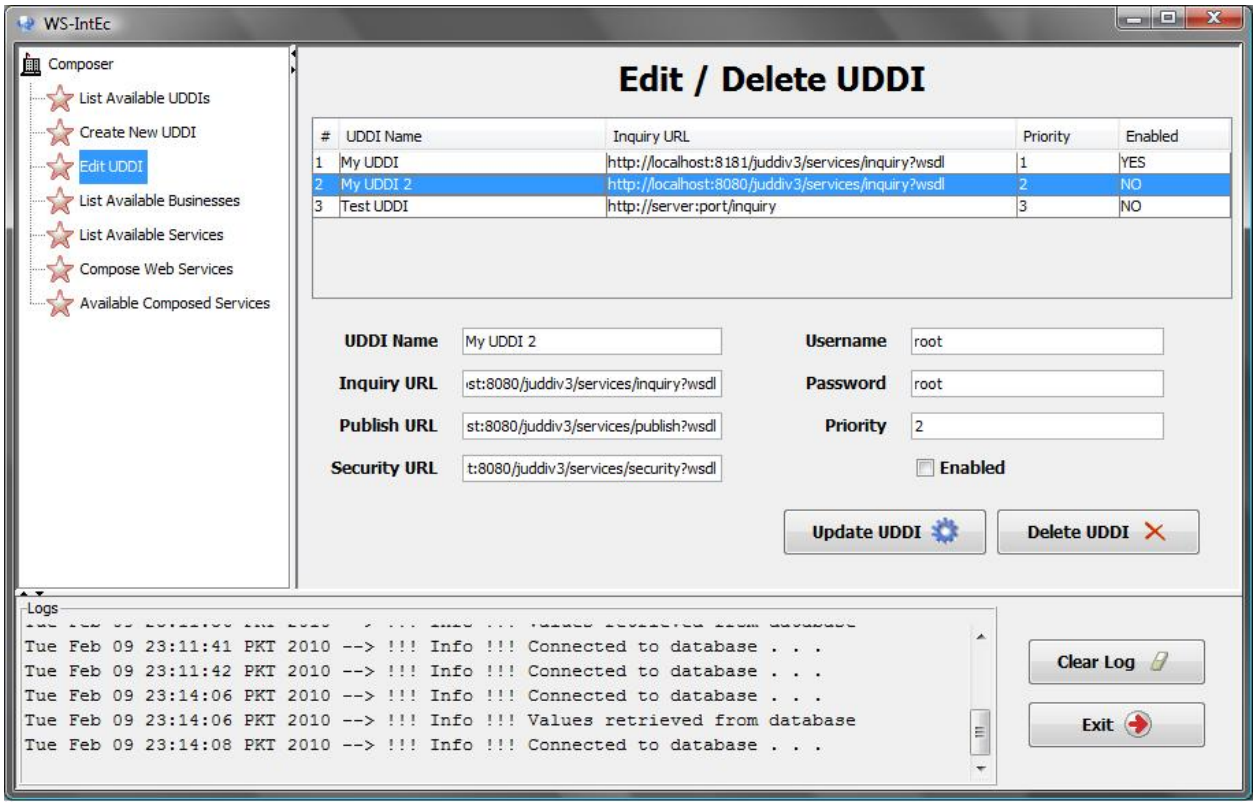
Add New UDDI

We can add a new UDDI to the system. We have to provide the name, inquiry, publish and security URLs with username/password (if applicable). We also have to set a unique priority for this UDDI. There is also an option to enable/disable this UDDI. After entering all these values, click on Add UDDI buttons which saves this UDDI information to the database.



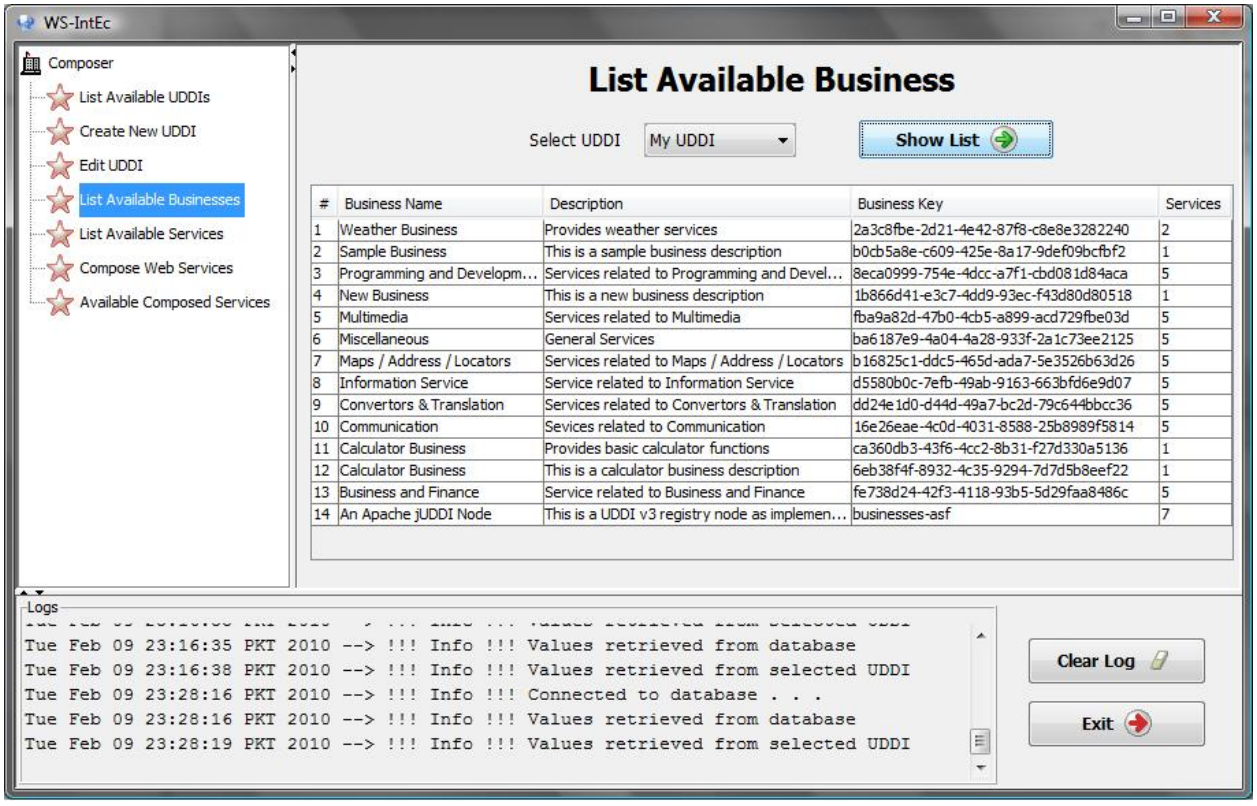
Edit / Delete UDDI

We can edit the UDDIs that have been added to the system. We are also able to delete as required. When we click on “Edit UDDI” in the main menu, the detail panel displays the list of available UDDIs. To select a UDDI, we click on any row in the table being displayed. We can change any or all of the values for this UDDI and then press Update. Or we can also delete this UDDI by pressing the Delete button.



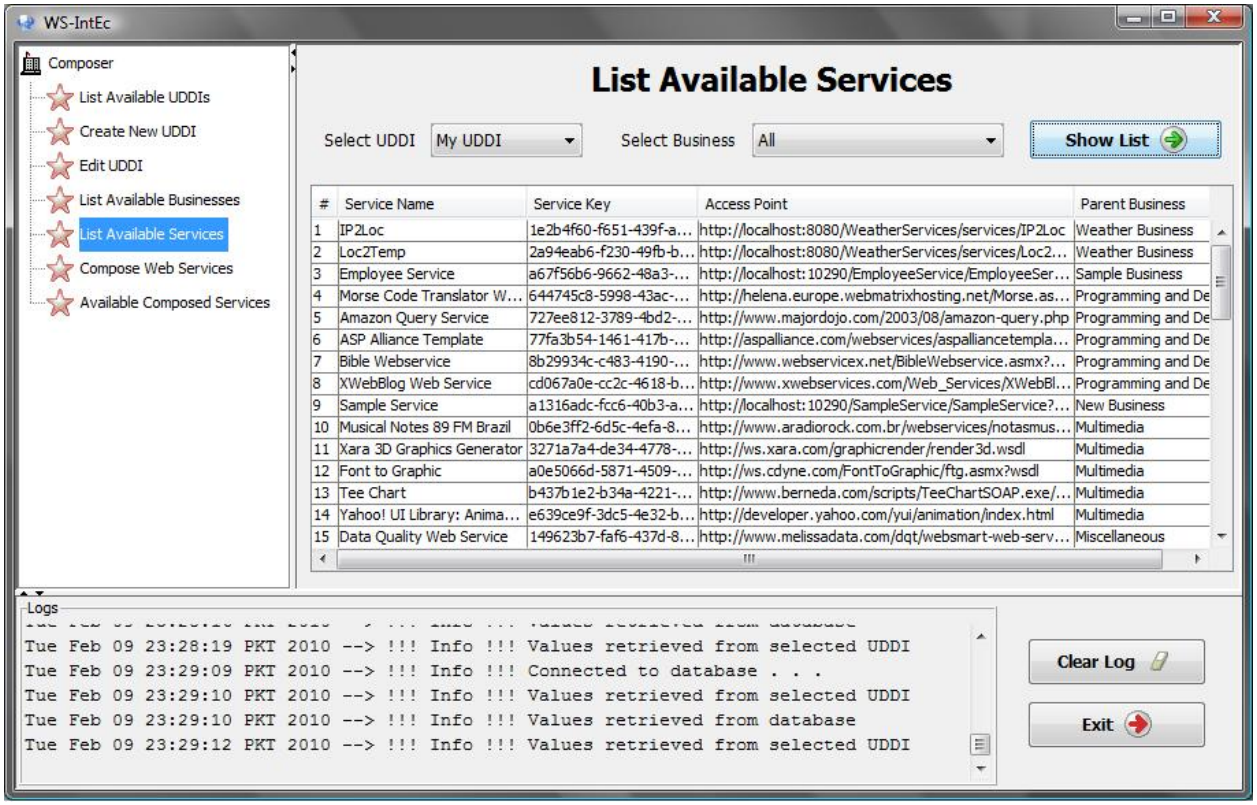
List Available Business

All the UDDIs which have been enabled are shown in the drop down. We can select any of the UDDI and then press the Show List button which displays all the available businesses in the table below.



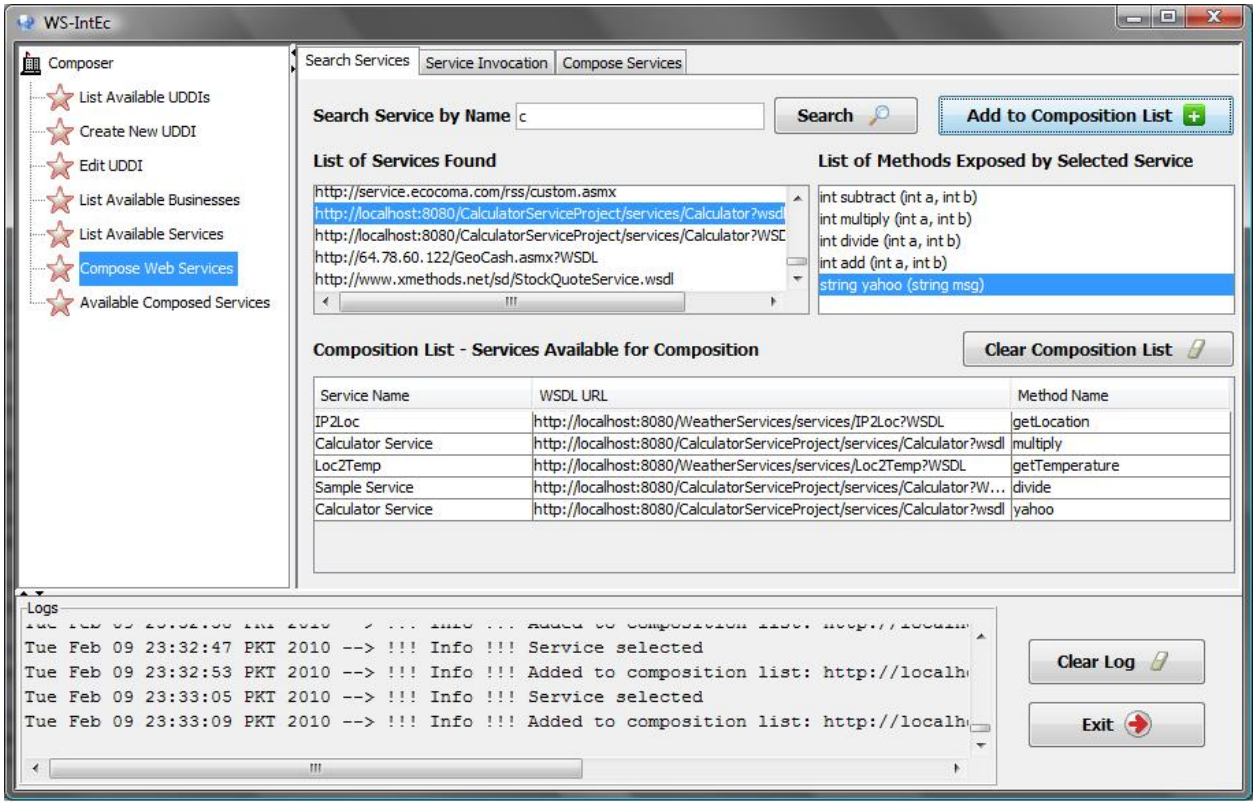
List Available Services

All the UDDIs which have been enabled are shown in the drop down. We can select any of the UDDI, list of businesses registered on that UDDI are populated in the business dropdown. We can either select a business name or select All option to display the related services. Then press the Show List button which displays all the available services in the table below.



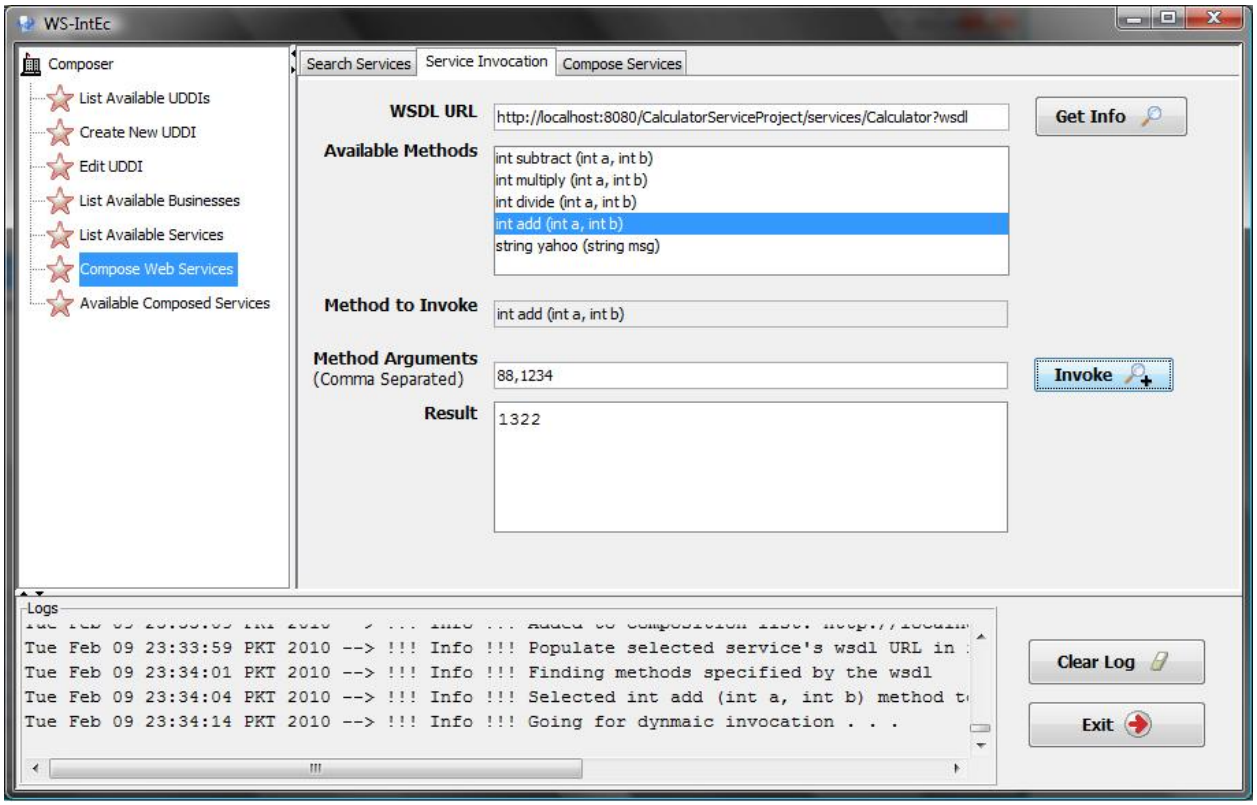
Compose Web Services

We can search the UDDIs or in other words discover web services by name. We can type the service name to be searched in the search box and then press Search button which displays all the found services in the associated table. When we click on any of the services found, it displays the list of methods exposed by that web service. We can select any method and then press “Add to Composition List” button by which these services and methods become available for composition. We also have a button for “Clear Composition List” which is self explanatory.



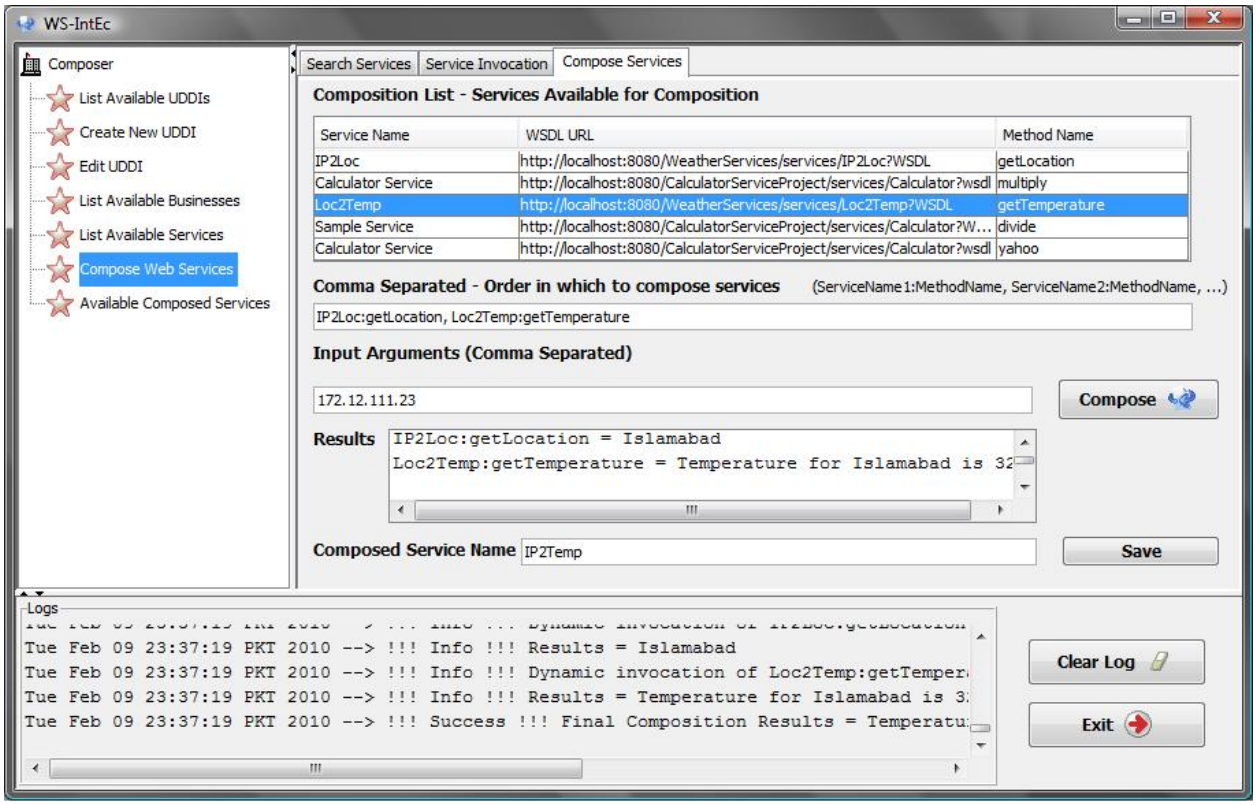
Service Invocation

We can dynamically invoke a web service by providing its WSDL address. We specify the WSDL URL and then press Get Info button which displays the methods exposed by the web service. We can select any method and then provide comma separated arguments and click Invoke button. The web service is dynamically invoked and executed and the results are displayed in the Result section.



Compose Services

List of services available for composition are displayed. We can add the services to be composed in comma separated order and then provide the input arguments. When Compose button is pressed the services are executed in the order specified and the results are displayed in the results section. We can also save this composed service in the database by providing a name and press Save. This composed service is available for execution when needed later.



9 References

[1] http://en.wikipedia.org/wiki/Web_service

[2] Incheon Paik*, Daisuke Maruyama* *"Automatic Web Services Composition Using Combining HTN and CSP"* Seventh International Conference on Computer and Information Technology

[3] Biplav Srivastava, Jana Koehler *"Web Service Composition - Current Solutions and Open Problems"*

[4] Yilan Gu and Mikhail Soutchanski. " A Logic For Decidable Reasoning About Services" 2006

[5] Annapaola Marconi, Marco Pistore and Paolo Traverso. *"Implicit vs. Explicit Data-Flow Requirements in Web Services Composition Goals"*.

[6] Michael Hu, Howard Foster *"Using a Rigorous Approach for Engineering Web Services Compositions: A Case Study"*.

[7] Daniela Barreirs clars *"selecting web services for optimal composition"* 2006.

[8] Jinghai Rao and Xiaomeng *Su "A Survey of Automated Web Service Composition Methods"*

[9] Faisal Mustafa, T. L. McCluskey *"Dynamic Web Service Composition"* 2009 International Conference on Computer Engineering and Technology

[10] Pat. P. W. Chan and Michael R. Lyu *"Dynamic Web Service Composition: A New Approach in Building Reliable WebService"* 22nd International Conference on Advanced Information Networking and Applications

[11] LIU AnFeng, CHEN ZhiGang, HE Hui, GUI WeiHua *"Treenet:A Web Services Composition Model Based on Spanning tree"* IEEE 2007

[12] Kazuto Nakamura Mikio Aoyama *"Value-Based Dynamic Composition of Web Services"* XIII ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC'06)