Dynamic Web Service Composition Using Google API Crawling

A dissertation Presented by

Maria Allauddin

(2009-NUST-MS PhD-CSE(E)-02)

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

Dr. Farooque Azam

College of Electrical & Mechanical Engineering

National University of Sciences and Technology

2012


بسم الله الرحمن الرحيم

THE COMMITTEE

Dynamic Web Service Composition Using Google API Crawling

A dissertation Presented by

Maria Allauddin

Approved as to style and content by:

Dr. Farooque Azam , *Supervisor*

Dr. Aasia Khanam, *Member*

Dr. Arsalan Shaukat, *Member*

Dr. Usman Qamar, *Member*

DEDICATION

Dedicated to My Parents, Teachers, Friends and Family

ACKNOWLEDGEMENTS

This thesis would not have been conceived without the help of many, whom I owe a great deal.

First and foremost is Allah Almighty, The Most Gracious and Most Merciful who has given me the strength to read and write. Truly, we plan and He plans. And Allah is the Best of All Planners.

I would like to record my sincere gratitude to my supervisor **Dr. Farooque Azam**, whose guidance, careful analysis and productive comments were valuable. I am grateful that he allowed me to work with him for this thesis.

I would like to thank my committee members **Dr. Aasia Khanam, Dr. Arsalan Shaukat and Dr. Usman qamar** for providing me the due guidance.

I thank my parents for their lots of prayers, for allowing me to follow my ambitions and for being patient with my endless years of study.

ABSTRACT

Dynamic Web Service Composition Using Google API Crawling

Service Oriented applications are becoming very popular due to ease of Web services Usage. The area of Web Service Discovery (WSD) is a primary area of research today. It has fundamental importance in web services utilization for personal or organizational needs. However the users of web service are yet facing challenges to find the desired web service due to rapid growth of web services available on internet. There is a need of a strategy to locate web services with issues covering like performance, flexibility and reliability across multiple heterogeneous registries, which is a challenging task yet. Our proposed framework covers these issues; it actively obtains user required web service by crawling among different repositories. Google Custom Search API is used for this purpose. Verification and validation checks are performed to confirm that the retrieved document is a web service and is currently available.

One use of Web Services in computer applications is its automated Composition. Our framework presents a Service Composition through user interaction. To resolve compositional complexity parameters matchmaking is performed. Overall the framework provides a reliable and trust-worthy composition.

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
JAX-RPC	Java API for XML-based RPC
OWL-S	Ontology Web Language Semantic
QoS	Quality of Service
SAAJ	SOAP with Attachments API for Java
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
WS	Web Service
WSCI	Web Service Choreography Interface

WSDB	Web Service DataBase
WSDL	Web Service Description Language
WSDL4J	Web Service Description Language for Java
XSLT	EXtensible Stylesheet Language Transformations
XSRL	XML Service Request Language

Chapter 1

INTRODUCTION

1.1 Overview of Web Services

Services are small components present on internet that can cooperatively make a complete application environment. Web services are made to be served on internet, so the basic requirement is that they should be platform independent. They are flexible enough to be integrated with other services as well as can function in total isolation. Each web service as a whole is responsible for its own operation. [1]

Web Services are applications that can be found on internet, identified by a URI which is then invoked to give result of the operation defined in it. Some relevant examples are

- Finding Currency Exchange Rates.
- Getting Weather News of certain area.
- Booking for a Flight Ticket

1.1.1 Definition

An official definition by WebServices.org is:
“Web Services are encapsulated, loosely coupled contracted functions offered via standard protocols” where:

- Encapsulated is a term used for such implementation of a function which can never be perceived by an external program.
- Loosely coupled means functions are not inter-dependent.
- Contracted means that it is already defined how to make use of function what will be its behavior and what will be its input and output parameters. [2]

1.1.2 Architecture

Usually in internet world, there is a client and a server, the server offers a functionality, the client is supposed to use it via some agent.

Web services are same as distributed applications so they have almost the same components.

- A service broker that searches a service provider for the client
- A service provider that publishes its services to the service broker.
- A service requester that asks the service broker to find the required service and use it.

Following is diagram of web service components

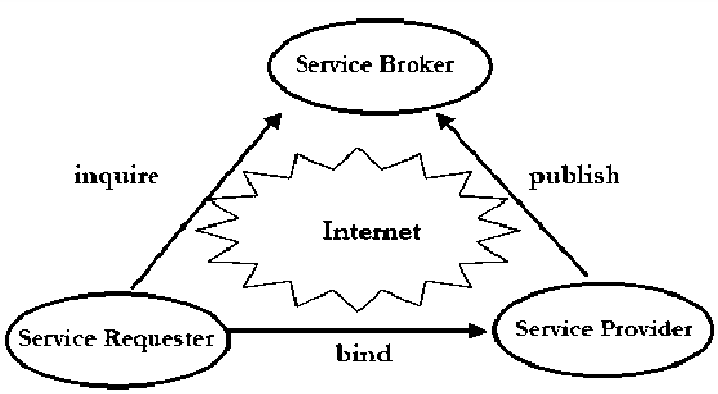


Figure 1-1 Web Service Components

Web services consist of the following:

- XML (Extensible Markup Language).
- SOAP (Simple Object Access Protocol).
- WSDL (Web Services Definition Language).

Web Services use XML on top of HTTP. For exchange of data XML is widely accepted format. For communication messages between service providers and clients, Web Service Description Language (WSDL) is used. There is no typical Graphical User Interface of a web service. This is also a reason for application independence. On client side it can be added to GUI e.g. web page.

A Web service can be an application component like: currency convertor, weather reports, or even dictionary as service. They also solve interoperability problems by providing a way to exchange data between different applications with different platforms. At present, most organizations and business companies prefer to implement only basic structure and use rest of functional components from web service. So the capability to efficiently find and add services to core components has turned on the importance of web service. It happens sometimes that a single service cannot fulfill the requirements of the organization. At that time there is a need to find out more than one service and integrate them in a proper way to get the final results as required.

1.2 Web Services Composition

To accomplish certain required functionality which can't be fulfilled by single web service there is a need to combine more than one service. In this case there is a need to use services together. The procedure of combining different services is called service composition. It can either be performed by composing elementary or composite services. A lot of work has been done to find out different efficient and effective ways of service composition. Composition of services has reduced the time of new application development.

Regardless of all struggles for better composition, it is still an extremely difficult task. With time numerous web services are available on internet and the figure is increasing day by day. So while searching for required web service there is a need to find it in huge repository. As web services can be created and updated very often and fast, so during composition there is a need to find out updated information to make different decisions.

Web services can be composed in following two ways:

1) Static Web Service Composition

2) Dynamic/ Automated Web Service Composition

In static web service composition, services are invoked one by one to achieve the required result. In automated service composition integration of web services is done by agents such that user is not fully involved in getting the results one by one and calling the other service. Further, in static

composition the sequence of web service composition is defined at start. In Dynamic composition the sequence of composition can be specified by user at run time.

1.3 Dynamic Web Services Composition

In automated/ dynamic web service composition the services are executed sequentially without much user involvement. In Dynamic service composition the request is given to the agent who performs the composition steps. First the translator component performs translation for the system, then the service is selected from database/ repository and then process generator generates the services. If multiple services of same functionality are selected Evaluator evaluates and the Execution Engine returns the results to clients.

A Generalized dynamic web services composition framework is shown in Fig 1.2

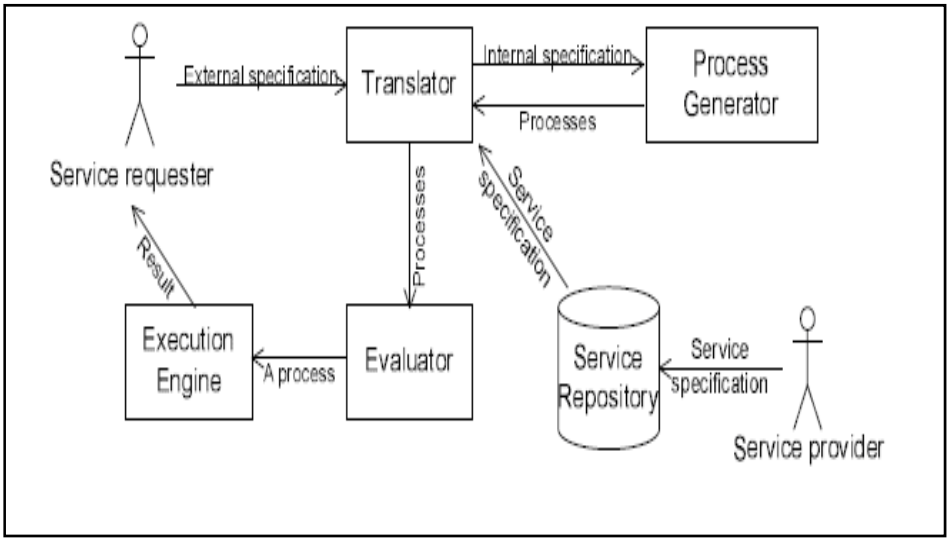


Figure 1-2 Generalized framework for dynamic web services composition

1.4 Problem Statement

Current approaches for web service discovery have some of the following limitations:-

a) Querying Heterogeneous registries at a time i.e., for service search, the user can query only one UDDI at a time.

b) Retrieving up to date information on user's request.
c) In case of searching from web there is a need of in time response.
d) One time consuming task is that the users have to search whole registry each time they need a service. It requires a lot of effort.
e) Majority of current approaches lack a reliable stable and trust-worthy discovery.
f) Services are themselves heterogeneous i.e. they have different formats for exchanging data.
g) The published web services are tagged with a lots of information that makes it difficult for a program to find the required web service on given attribute.[2]
h) Keywords are used to discover web services in UDDI. Ranking services and filtering them is the main advantage of UDDI. Main drawback is that search can only be made on basis of metadata so it limits the search criteria.

Few problems faced in composition are given below:-

a) Dynamic composition needs very little user involvement which makes it difficult to find out an exact required service on huge repository of internet.
b) Not all the services on internet are public. So there is a need to select service of user interest.
c) Transactional support can be very small in fully automated composition as different service providers may have different conceptual models. (transactional support means support for exchange of data between different services)
d) Compositional correctness cannot be guaranteed as automation cannot verify middle stages of composition.
e) Full automation is possible for specific infrastructures. If there is need of a general application in which requirements change every time. A little user intervention is helpful.

1.5 Contribution

We have proposed a framework to overcome following web service discovery and composition problems: - It provides the flexibility to search the user query on more than one heterogeneous registry at a time using Google Custom Search API. We have programmed to retrieve only relevant WSDL files that are valid and available. The limitation of UDDI search has also been overcome as whole web is searched for the user query word, so there is no more search specification of service name/category. It takes less time as compared to a usual open source crawler which reads every word of each child link one by one in desktop application. It provides a reliable and trust-worthy service discovery. And further it provides up to date information.

By using simple xml messaging, we have provided a flexible communication for integration. Also it's helpful for complex type services where most frameworks fail to integrate services. There is no need to worry about data types and hence complex data types. It provides a reliable and trust-worthy composition. User can select service of his own interest and hence overcomes the problem of availability and updates. Compositional correctness is guaranteed by matchmaking and final input decision by user.

This thesis proposed a solution for researchers who are facing the problems of web service composition due to constant changes in input/output parameters and independent nature of different web services. We have given generic implementation of proposed model to prove the correctness of algorithm.

For service search and composition our frameworks are accepted and published by IJCA Foundation of computer sciences New York.

- Maria Allauddin, Farooque Azam "*Service Crawling using Google Custom Search API*", International Journal of Computer Applications, Volume 34 - Number 7 , 2011 (Published).
- Maria Allauddin, Farooque Azam *"Dynamic Web Service Composition and Parameters Matchmaking"* International Journal of Computer Applications, Volume 36 - Number 9, 2011 (Published).

Another paper is accepted in IACSIT conference

- Maria Allauddin, Farooque Azam "QOS Based Service Search and Composition Algorithm" 2012 International Conference on Network and Computer Science, IACSIT(Accepted)

1.6 Thesis Structure

Chapter 1 This chapter comprises of an overview of Web Services and Dynamic Web Services composition with brief explanation. Also the Problem statement and contributions to our work are briefly stated.

Chapter 2 This chapter describes the related research work and commercially available APIs and techniques.

Chapter 3 This chapter presents the proposed Methodology.

Chapter 4 This chapter presents the implementation of proposed methodology.

Chapter 5 This chapter is concerned with Analysis and Results.

Chapter 6 This chapter includes the summary and conclusion.

Chapter 2

RELATED WORK

2.1 Literature Review

Holger Lausen and Thomas Haselwanter [3] described that centrally maintained repositories are not enough for service search and does not provide full matching requirements for user query. They have considered the information provided in WSDL documentation. In a next step they have refined the results on user's explicit feedback from users. They used HeritrixWeb crawler by adding some rules to crawl only relevant pages. Subsequently, they removed duplicate results. However they could not achieve a relative accuracy in the retrieval.

Mydhili K Nair Dr. V.Gopalakrishna [4] stated different methods of service discovery. As today WSDLs are abundant and scattered across the WWW so the count of Web Services already deployed with similar functionality are large in number. There is an increasing need to develop Service Discovery Methods that help the Consumer to find the right kind of services for their requirements.

They have put forth survey results of the work conducted by researchers across the globe on the WS Discovery techniques based on User Requirements as their input. They concluded that the functional requirements of the WS are more or less handled by the WSDL. There is an analysis of the various techniques used by search engines such as Google, Yahoo AlltheWeb and Web Crawlers such as WebSPHINX to fish-out the relevant WSDLs.

They have given the list of problems which need to be looked into and investigated. They declared open-ended unresolved issues in a novel way by providing its *Cause-Effect Analysis*. In Figure 2.1

Sl#	Problem	Cause / Reason of the Problem	Effect of the Problem
	WSDL Problem		
1	**WSDL** is inherently designed to give descriptions detailing its functional aspects like Service Type, Implementation interface details such as the port to bind to, the type of parameters etc. It is ***not designed*** to publish the non-functional aspects[2,4,7,8,9].	WSDL is ***not designed*** to take the ***"semantic descriptions"*** of the service. It is used to *Publish* a WS in terms of its ***ports***, ***port types*** and ***bindings***[2,4,7,8,9].	This makes it difficult to store the non-functional aspects of the service such as its ***Quality of Service*** (***QoS***). Parameters such as reliability, availability, response time, throughput, mean time before failure, price, etc. ***Several techniques*** have been formulated to solve this problem[23,25,32,33,14,35,34]
	UDDI Problem		
2	Current **UDDI** implementations are limited in scope. It is ***not innately designed*** to publish and store the QoS requirements and other non-functional requirements of a service[31,18,19].	UDDI allows search on ***limited attributes*** of a service, namely, ***Service Name*** (selected by the Service Provider), ***keyReference*** (unique for a service), or on a ***categoryBag*** (listing all the business categories[31,18,19].	This problem makes it difficult to store within the UDDI, the ***run-time performance parameters*** of the service capturing its QoS parameters. It is also difficult to capture the ***Customer Feedback*** about the service and store it to analyze and improve on these valuable metrics[31,18,19,20,15,16].
3	Public UDDI registries, that were run by IBM, Microsoft, SAP and NTT Com. have been ***shut down*** in the beginning of 2006[15].	There was ***no consensus*** regarding ownership of the root UDDI rsegistries. UBRs used to contain listings of businesses that ***no longer existed*** and sites that were ***no longer active*** [19,15].	There is ***no Universal Registry*** where all Web Services are published. This makes it difficult to check the performance, scalability and statistical gathering of data. An earlier work carried out by Su Myeon Kim and Marcel-Catalin Rosu[19] reports that only one-third of the 1200 registered services referenced a valid WSDL.

Figure 2-1 Cause Affect Analysis[4]

Woogle [5] is a web service search engine. It does extraction of information about WSDL functionality descriptions, inputs and outputs. They used clustering of parameters, matching of inputs outputs and operations, and stored the results in a database. They compared their method with Func and Comb. Comparison of words with operation names is done by Func method. Whereas in Comb method web service names, parameters names and descriptions are also used for matching; in contrast to Woogle, both of the mentioned keywords are used.

In multi-registry environments [6] provides foundation for web service discovery. It also provides reliability to some extent. A responsibility of crawler is that it actively seeks Web services; they made a registry monitor to track any changes of the provided registries. Further there is a Term Probing (TB) component which is responsible to extract words from WSDL descriptions, at end they provide web service storage to enable web service search. However there is no semantic support for service UDDI. They have used the specific registries such as MUBR, MUTR, SUBR and SUTR and they go around among them. So the framework is not flexible to be scaled.

The architecture in [7] extends SOA with Quality of service support for web services. In addition, it verifies, certifies, confirms, and monitors QoS properties. The architecture contains these major roles: - UDDI with QoS Information, Verifier and Certifier, Discovery Agent, QoS

Matching, Ranking and Selection Algorithm. The discovery agent discovers functionally similar web service from provided UDDI registry when it receives request from the user.

In [8] the researchers described main features required for a QoS based agent. Response Time, Availability, Throughput and Price are considered. Their approach is dynamic which keep cover on actual systems complexity. However their architecture is theoretical so there is no performance test. They argue that their framework will enable a more flexible, and trustable architecture.

Web services are presented as XML based software components in [9], that can be discovered on the basis of signature and interface matching. So the search process completely depends on actual components of the service. WSDL is an XML based format which not only defines it's functionality but also abstract operations and network bindings.

In [10], keyword matching is used for service discovery using UDDI. This work matches XML schema with various comparisons using intelligent algorithms. Suffix, prefix and infix can be used for string matching.

Liang-Jie Zhang, Qun Zhou's [11] framework solves the problem of linked documents. WSIL is used to search the chain services and results are returned to the users after aggregation. So they solved the problem of manual links document search. The chains of the documents are retrieved by re exploring the links in history using some calculations and caching.

Paul Palathingal [12] gave an agent based approach. The agent acts dynamically to discover, invoke and then execute the web services. Using agents it is possible that the sender never knows the receivers address. The agent who sends request for the service gets results from then the next agent; composition agent composes the web service.

Schahram Dustdar and Wolfgang Schreiner in [13] performed composition frameworks analysis, modules and supporting features are given by them for composition design and development. They say that the current web service technology is quite limited as seen from results of different research papers. This is due to dependency of this technology on standards as SOAP, WSDL or UDDI. They compared different composition strategies by finding similar features. Finally they

concluded that interactions among services with different specifications have to be considered with more attention.

Freddy L´ecu´e1, Eduardo Silva, and Lu´ıs Ferreira Pires[14] have given a framework for automated composition. They used SPICE ACE for automated composition. In their framework web services and their requests are distinguish by their functional and non-functional properties. Composition Factory is a part that figure out the non-functional properties of service compositions each time a new service is added. They used a causal link matrix, to guarantee that the obtained compositions have valid semantic connections. Finally, if a composition does not match the non-functional properties of the service request, it is neglected.

Faisal Mustafa, T. L. McCluskey [15] made a sketch of major challenges tackled by automated web services composition. The problems are associated with distributed, dynamic and unsure disposition of web service. Their model is semi-automatic but fixes few problems of fully automated service composition. They pointed out that internet has huge repository of services it is not possible to automatically analyze them. And hence integration is difficult. They said that second difficulty for automated service composition is that web services are updated frequently. So there is a need to have current information and decision of composition should be based on that. Their technique has few drawbacks. If server is not working input output problems occur. Also their repository does not contain updated information. The given model is semi-automatic static composition model and fixes some issues as: - Repository has huge collection of web services and it is almost impossible to analyze them from repository. Second when there is a need to complete a task most recent information is required. Input/output issue arises if the server goes down in the proposed technique. Also updated information is not present in the database as it does not update its contents. The framework is shown in Fig.2-2

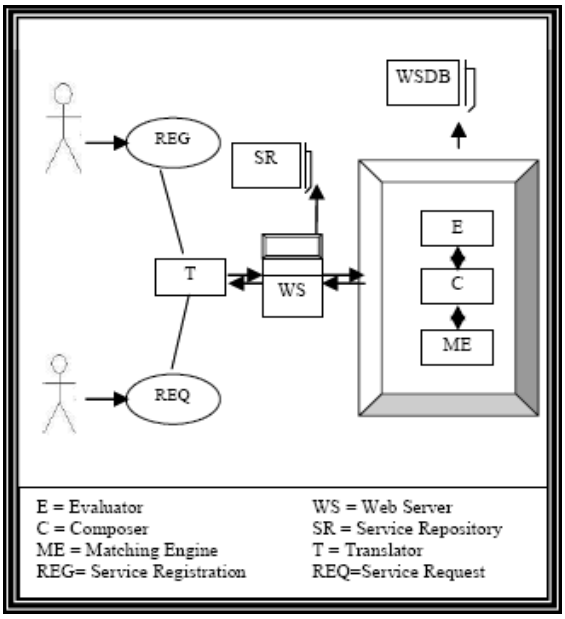


Figure 2-2 Composition framework by Faisal & McClusky [15]

Pat. P. W. Chan and Michael R. Lyu in [16] presented Dynamic Web Service Composition using Nversion programming. This method expands the reliability for planning among web services. If a server fails, there is another to provide the required service. Their composed service is free of deadlock consumes less time for composition. Since the frame work is dynamic it uses updated versions without the need of rewriting the specifications. To verify the correctness of algorithm experimental evaluation and results are given at the end. The framework is shown in Fig 2.3

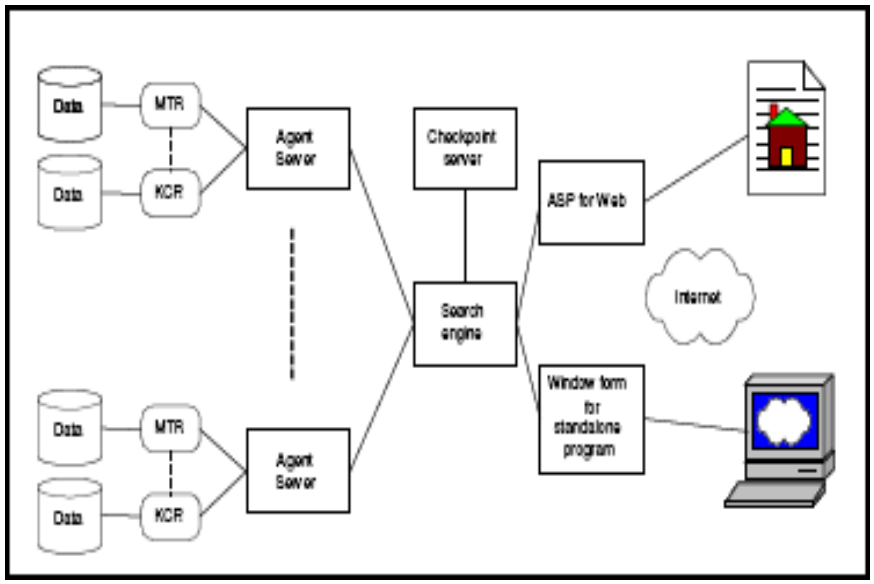


Figure 2-3 Best Route finding system architecture [16]

LIU AnFeng et al. [17] proposed a technique that supports web services interface. They used peer to peer ontology with overlay network to provide composition. The composition uses domain ontology and DHT Distributed Hash Table for composition and discovery. Their analysis shows that separation of details of interface from underlying details makes it easy to understand. The composition is fast and fault tolerant and hence provides QoS Based execution.

Kazuto Nakamura, Mikio Aoyama in [18] presents a structure for dynamic web service composition. They used value based composition and provided QoS. A Meta model is used together with the VSDL. To compose the services Value added service broker architecture is used and to define relationship among values the value-meta model is employed. This paper provides dynamic web services composition framework which is automated and fault tolerant. The framework is shown in Fig 2.4

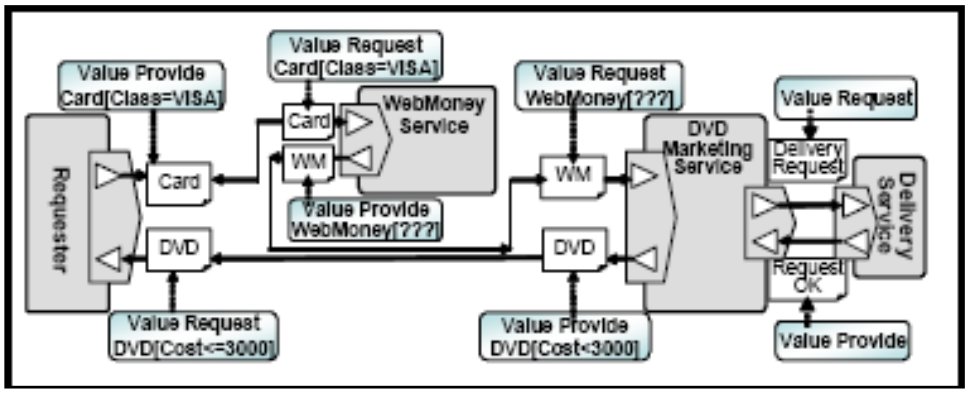


Figure 2-4 Value based dynamic composition [18]

R. Jaya prakash, R. Vimal [19] evaluated different web service composition methods using Business Application. They considered basic perspectives like QoS, scalability, and correctness to analyze different methods. They performed the analysis by making an online book shop. The Web service composition approaches are compared with each other based on connectivity, exception handling and compensations. They indicated the results as good, average and low. They describe that different methods provide different automation level and that they cannot conclude that higher automation is better due to high complexity of web service environment. So full automation cannot be provided. Though highly automated methods are appropriate for making implementation structures that are requirement specific it is almost impossible to implement in highly fluctuating environments.

Freddy L´ecu´e, Alain L´ in [20] outlined major challenges of Semantic Web Services. They used xCLM for automated service composition which provides formal model to face the challenges. They described that advantage of functional level composition is use of OWL-S. They analyzed different proposals and concluded that no formal model is helpful for automation of composition. Their framework is robust, secure and verifiable. Matchmaking of input output parameter is performed at functional level. Matchmaking functions summary is provided that gives details of how different comparisons are made. Overall the Service composition is viewed as fundamental link composition.

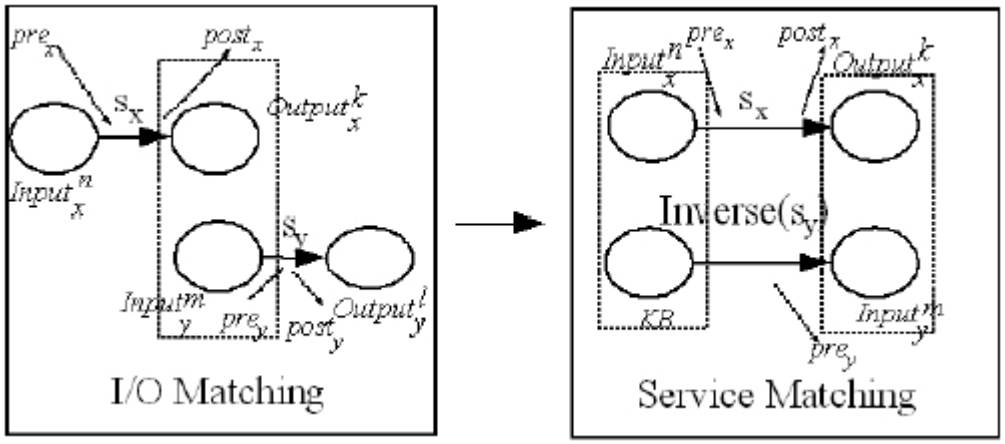


Figure 2-5 Problems Mapping [20]

Different matching functions are defined they do not allow only to value similarity between Web services but also to value a composition. There is still a need to take into account non functional properties of Web services in order to solve a problem of multi-objective (e.g., Semantic connection, QoS...) optimization.

San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, and Cheng-Hung Chen in [21] formulated the dynamic WS selection procedure in a dynamic environment that is failure prone. They proposed FSM usage to invoke operations of service in an order. They defined parameter to find a probability of execution of services weather they will terminate successfully or not. They used Eigen vector to show aggregated reliabilities. The approach can be used only in industrial applications and hence is environment specific.

Liping Liu , Anfeng Liu , Ya Gao [22] used Particle Swarm Optimization for Service Composition. PSO is meaningful for the composition of complex services spread on internet. If there is requirement of multiobjective composition only PSO can do so. A non inferior pareto solution is provided by PSO group search. The solution meets all the required constraints. They used general service overlay model. They say that full automation of service composition is complex rather unachievable task that is why most of algorithms are semi automated. They said that their algorithm can be applied to specific compositions.

Zhang Hai-tao, Gu Qing-rui [23] presented a dynamic process of domain ontology-based method. Their method considered semantics of service for composition. They verified their work by experimental examples. Their module for service composition makes a portfolio as soon as a service request is received. When all services are determined for integration it starts calling webs services. An OWL-Agent is used to mark functions of the service by forming OWL-S documents that call the services. Shortcomings of this approach is it must have a fixed field of experts in the field service portfolio template firstly, and then selected the user needs to match the template in the service of specific Web services. The limitation of the method is the construction of the field of service composition templates requires human intervention, so degree of automation is reduced.

Yujie Yao, Haopeng Chen [24] solved the rule based web service composition problem. They gave a framework in which selection engine executes business rules. Pereto optimal solution is obtained by an algorithm called NSGA-II. This is a selection algorithm. They said that the work can be extended by a large scale implementation. There is still a space to research about the communication between selection engine and composition engine.

Farhan Hassan Khan, M.Younus Javed, Saba Bashir [25] presented a framework for dyamic web service composition and execution. At first they discussed major problems of dynamic composition, then proposed an algorithm for dynamic web service composition. They mentioned composition issues like reliability, availability, data distribution. They introduced the concept of multiple repositories for system reliability. Availability is also guaranteed by this concept. An aging factor is used to retrieve up-to-date information. They claim that their system is reliable, fault tolerant and performs fats data retrieval. They said that due to limitations of UDDI there is a need to extend this framework to service crawling. It is also essential to look into more details of every component of the framework to ensure better and efficient composition.

Kaouthar Boumhamdi and Zahi Jarir [26] research contains a contribution in dynamic composition of Web services. First they stated some approaches of Dynamic Web Service Composition which are of vital importance. They made a comparison of those approaches and on bases of final analyses they proposed a new architecture. They argue that the architecture is

flexible and modular. They gave the idea of dynamic service adaption. The adaptation is according to user requirements and sometimes according to the availability of resources. They used a BPEL selection manager and hence their approach can be applied to specific scenario.

Jinghai Rao and Xiaomeng Su et al. [27] presented different web service composition methods which include workflow and AI planning. They proposed a five step composition model. Five layers are Presentation, Translation, Process generation, Evaluation and Execution. The author also concludes that although different automatic web services composition techniques are available, it is not true that more automation is better. Service composition environment is highly complex its not feasible to generate every thing automatically. Highly automated methods are only suitable for generating the implementation of formal specification skeletons. The framework is shown in Fig 2.6

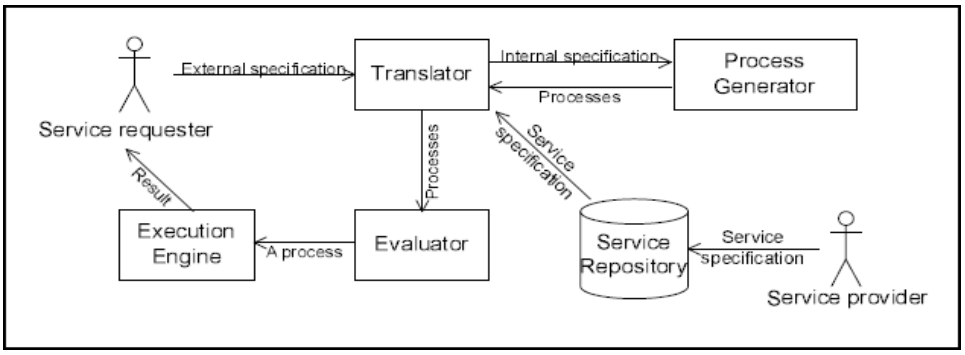


Figure 2-6 A framework of service composition system [27]

Biplav Srivastava, Jana Koehler et al. [28] explored problems of web service composition and analyzed two other approaches and compared them. 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. 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.

2.2 Commercial APIs to Support Service Search and Composition

2.2.1 Google Custom Search API

Google Custom Search facilitates us to generate a search engine for website, blog, or application. Google Custom search provides us the facility to create and engine that focuses on a particular topic. Certain web sites can be added to be searched, prioritized or ignored. In simple the search engine can be tailored to our interest.

As Google SOAP Search API has been depreciated, Custom Search can be used and adapted for Service search. While using Google Custom Search API and searching for WSDL documents, it can be seen that the working looks like a crawler. As a crawler application takes some certain URL and searches the user query on different links of that parent URL, custom search API does same. But it is flexible and scalable enough that user can add more than one parent links at a time. An example, search for an “add wsdl” document on Google Custom search, while some parent links are already added on the search engine. The search results are shown in figure 2-7.

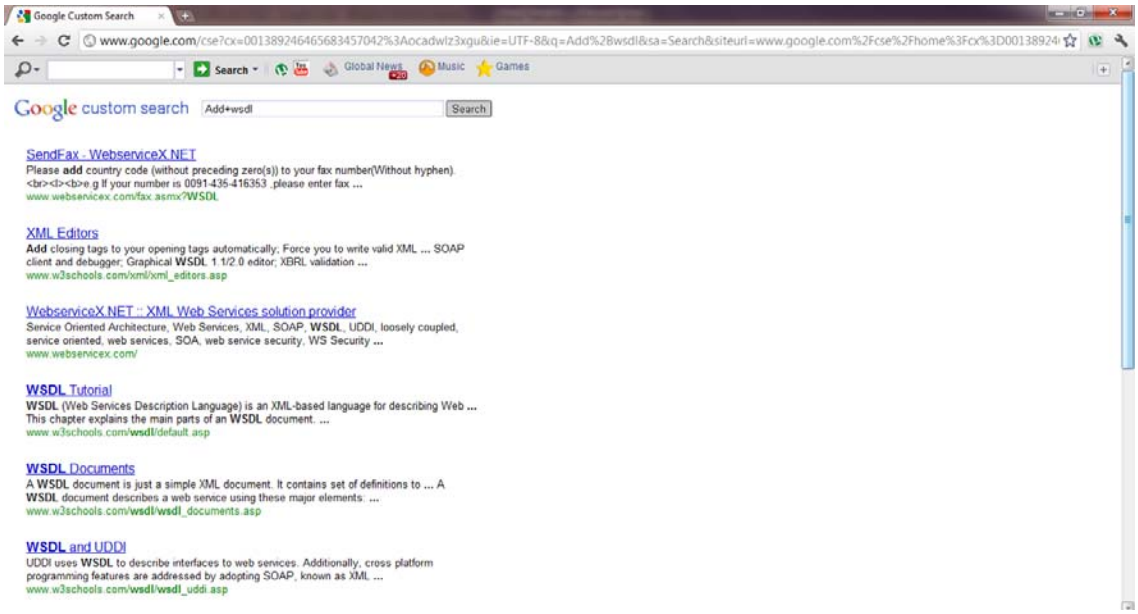


Figure 2-7 Results returned by Google Custom Search API

2.2.2 Google Custom Search Features

With Google Custom Search, the power of Google can be used to create a tailored engine for searching WSDL documents. It: -

- Include one or more websites/ parent links, or specific WebPages.
- Crawls all child links of given link to find out user query word.
- Provides its API to be used in desktop application.
- Provide fast and relevant search results.
- Contains rich results formats.
- Provides facility to use wild card patterns for parent URL.
- User query single word or combination of words.
- User and Programmer Documentation are provided.

2.2.3 Programmatically Creating Custom Search

Creating an engine programmatically needs requests such as: GET, POST, and DELETE. All of them need a header which is for authorization. The header contains and authentication token. Custom Search server sends an HTTP status code in response of that request. The response reflects the result of each request. Any programming language can be used that enables to issue HTTP requests and parse XML documents in response. Following are some steps for programmatically creating search engine.

2.2.3.1 Authentication:

Like every other Google service, custom search is protected. It needs a Google account. The account provides an authentication token in order to interact. Once authentication token acquired it can be used to create Authorization header for each request. i.e.

“Authorization: GoogleLogin auth=yourAuthToken”

Here is an example of an authenticated GET request:

GET http://www.google.com/cse/api/default/cse/ Authorization: GoogleLogin auth=IM6F7Cx2fo0TAiwlhNVdSE8Ov8hw6aHV

2.2.3.2 Request Methods

Following are tables that describe different operations and corresponding URLS

1) Creating updating and Deleting Search Engine

If you want to...	Send this HTTP request method...	To this URL....	For more information about the method and the required message body, see the section on....
Create a new search engine or modify an existing one (by submitting a context XML)	POST	http://www.google.com/cse/api/default/cse/<CSE_ID>	Creating and Updating the Search Engine Specification
Create or modify the list of sites to search (by sending a message with annotations XML). You can add or delete individual annotations.	POST	http://www.google.com/cse/api/default/annotations/	Creating and Updating Annotations
Create or modify the list of promotions (by sending a message with promotions XML). You can add or delete individual promotions.	POST	http://www.google.com/cse/api/default/promotions/<CSE_ID>	Creating and Updating Promotions
Create or modify the list of synonyms (by sending a message with synonyms XML). You can add or delete individual synonyms.	POST	http://www.google.com/cse/api/default/synonyms/<CSE_ID>	Creating and Updating Synonyms
Delete a search engine	DELETE	http://www.google.com/cse/api/default/cse/<CSE_ID>	Deleting a Search Engine

Figure 2-8 Creating updating and Deleting Search Engine

2) Retrieving Search Engines, Promotions, Synonyms, and Search Results

If you want to...	Send this HTTP request method...	To this URL....	For more information, see....
Retrieve a list of search engines under an account	GET	http://www.google.com/cse/api/default/cse/	Retrieving a List of Search Engines
Retrieve the context file of a specific search engine	GET	http://www.google.com/cse/api/default/cse/<CSE_ID>	Retrieving the Specifications of a Search Engine
Retrieve all annotations for all search engines	GET	http://www.google.com/cse/api/default/annotations/	Retrieving All Annotations
Retrieve promotions	GET	http://www.google.com/cse/api/default/promotions/<CSE_ID>	Retrieving Promotions
Retrieve synonyms	GET	http://www.google.com/cse/api/default/synonyms/<CSE_ID>	Retrieving Synonyms

Figure 2-9 Retrieving Search Engines, Promotions, Synonyms, and Search Results

2.2.3.3 JSON

JSON is used for data exchange. It is a lightweight format. It is in human readable form. Also machines can understand it easily to parse. It is language independent but uses standard conventions. For example programmers of C, C#, Java, Perl and Python can easily understand JSON format.

JSON has two structures:

- Name/value pairs. As in other languages there is an object, record, struct or an associative array.
- Array, vector or list; an orders list.

2.2.3.4 JSON Object in java

> A JSONObject is an unordered collection of name/value pairs. Its external form is a string wrapped in curly braces with colons between the names and values, and commas between values and names. The internal form is an object having get() and opt methods for accessing the values by name, an put() methods for adding or replacing values by name. the values can be any of these types: Boolean, JSONArray, JSONObject,Number and String, or the JSONObject.NULL object

Figure 2-10 JSON Object Java

External form JSON needs to be converted to internal form for this purpose JSON Object constructor is used. Value is returned if found by get method, if it is not found and exception is thrown. If an opt method is used instead of get method it returns default value in case no result if found. It never throws an exception. Put and toString Methods are used to convert values into JSON text.

2.2.3.5 Custom Search concepts

The JSON/Atom Custom Search API permits to add the power of Google Custom Search to desktop application.

2.2.3.5.1 API data model

Custom Search Engine API query result is a JSON or Atom object that includes three types of data:

- Metadata describing the requested search (and, possibly, related search requests)
- Metadata describing the custom search engine
- Search results

The JSON/Atom Custom Search API defines three custom properties and two custom query roles:

- Custom properties
  - cx: The identifier of the custom search engine.
  - cref: A URL pointing to the definition of a custom search engine.
  - safe: A description of the safe search level for filtering the returned results.

- Custom query roles
  - nextPage: A role that indicates the query can be used to access the next logical page of results, if any.
  - previousPage: A role that indicates the query can be used to access the previous logical page of results, if any.

2.2.3.5.2 Calling styles

To invoke the API there is more than one way:

Using REST directly

Using REST from JavaScript (no server-side code required)

2.2.3.6 REST

JSON/Atom REST is different from traditional REST, i.e. it provides access to service rather providing access to resources. So the API provides one URI which is a service endpoint.

JSON/Atom API URI format is

“https://www.googleapis.com/customsearch/v1?*parameters”*
This is specific for Single JSON/Atom API URI. Where parameters are specific to single query.

Here is a working example of JSON/Atom Custom Search API, which searches a test Custom Search Engine for **lectures**:

GET *https://www.googleapis.com/customsearch/v1?key=INSERT-YOUR-KEY&cx=017576662512468239146:omuauf_lfve&q=lectures*

2.2.3.7 Using REST to Invoke the API

2.2.3.7.1 Working with Search Results

To put a query and Get results back HTTP GET is used. The format for the request URI is:-

“https://www.googleapis.com/customsearch/v1?*parameters”*

Each request has three query parameters:-

- **API key.** Used to identify an entity. i.e. user application
- **Custom search engine identifier.** To specify the custom search engine an identifier is used.
  - Use *cx* identifier for a search engine created with the Google Custom Search page.
  - Use *cref* for a linked custom search engine.
  - If both are specified, *cx* is used.
- **Search query.** q Query parameter is used to specify the query.

Search request example is:

“GET https://www.googleapis.com/customsearch/v1?key=INSERT-YOUR-KEY&cx=013036536707430787589:_pqjad5hr1a&q=flowers&alt=json”

If requested query is succeeded the server sends response data along with 200 OK HTTP status code.

2.2.3.7.2 Response data

The response data, which is output of JSON query has three classes:

- Search metadata
- Custom search engine metadata
- Search results

These are described below

2.2.3.7.3 Search metadata

It describes characteristics of searches that are possible. These characteristics are in the form of array of objects. Usually and array contains a single element because each query role object is a separate array.

Below are possible query objects:

- Request: it describes set of current results.
- NextPage: query for next page of results.
- PreviousPage: query for the previous page of results.

2.2.3.7.4 Custom Search Engine Metadata

The metadata describes a specific search engine. This description is in context property. The information includes engine's name and any public object for refinement of search query.

2.2.3.8 Standard Query Parameters

Query parameters for JSON/Atom Custom Search API operations are shown in the table below. All parameters are optional.

Table 2-1 Query Parameters

Parameter	Meaning	Notes
alt	Data format for the	Valid values: json, atom

	response.	Default value: json
callback	Callback function.	Name of the JavaScript callback function that handles the response. Used in JavaScript JSON-P requests.
fields	Selector specifying a subset of fields to include in the response.	Use for better performance.
key	API key. (REQUIRED*)	*Required unless you provide an OAuth 2.0 token. Your API key identifies your project and provides you with API access, quota, and reports. Obtain your project's API key from the APIs Console.
access_token	OAuth 2.0 token for the current user.	One possible way to provide an OAuth 2.0 token.
prettyPrint	Returns response with indentations and line breaks.	Returns the response in a human-readable format if true. Default value: true. When this is false, it can reduce the response payload size, which might lead to better performance in some environments.
userIp	IP address of the site where the request originates.	Use this if you want to enforce per-user limits.

2.2.3.9 API-Specific Query Parameters

Request parameters that apply specifically to the JSON/Atom Custom Search API are summarized in the following table.

Table 2-2 Request Parameters JSON/Atom

Parameter	Meaning	Notes
cr	Country restrict(s)	The cr parameter restricts search results to documents originating in a particular country. You may use Boolean operators in

		the cr parameter's value. Google WebSearch determines the country of a document by analyzing: the top-level domain (TLD) of the document's URL the geographic location of the Web server's IP address
cref	The URL of a linked custom search engine	The url of a linked custom search engine specification to use for this request (e.g., cref=http%3A%2F%2Fwww.google.com%2Fcse%2Fsamples%2Fvegetarian.xml). If both cx and cref are specified, the cx value is used.
cx	The custom search engine ID to scope this search query	The unique ID for the custom search engine to use for this request (e.g., cx=000455696194071821846:reviews). If both cx and cref are supplied, the cx value is used.
filter	Controls turning on or off the duplicate content filter	The filter parameter activates or deactivates the automatic filtering of Google search results. Valid values for the parameter are: filter=0 - Turns off the duplicate content filter filter=1 - Turns on the duplicate content filter (default) By default, Google applies filtering to all search results to improve the quality of those results.
gl	Geolocation of end user	The gl parameter value is a two-letter country code. The gl parameter boosts search results whose country of origin matches the parameter value.

		Specifying a gl parameter value should lead to increased relevance of results. This is particularly true for international customers and, even more specifically, for customers in English-speaking countries other than the United States.
lr	The language restriction for the search results	You can restrict the search to documents written in a particular language (e.g., lr=lang_ja). This list contains the permissible set of values.
num	Number of search results to return	You can specify the how many results to return for the current search. Valid values are integers between 1 and 10, inclusive. If num is not used, a value of 10 is assumed.
q	Query	The search expression.
safe	Search safety level	You can specify the search safety level. Possible values are: high - enables highest level of safe search filtering. medium - enables moderate safe search filtering. off - disables safe search filtering. If safe is not specified, a value of off is assumed.
start	The index of the first result to return	You can set the start index of the first search result returned. Valid values are integers between 1 and (101 - num). If start is not used, a value of 1 is assumed.

2.3 JAX-RPC

It is Java API for XML-based RPC. It provides many benefits to Java developers, including:

- Support for open standards: XML, SOAP, WSDL
- Processing model of SOAP message and extensions.
- Web services Security

JAX-RPC has built-in feature of mapping between Java and WSDL. No other environment provides such functionality within single tool. JAX-RPC client has different programming models, such as dynamic invocation interface, dynamic proxy and stub-based. Any of these can be used to invoke a heterogeneous web service endpoint.

2.3.1 SOAP and Other Messaging

SOAP is required by JAX-RPC for interoperability. The requirement is over HTTP. SAAJ API provides this support for message handling. Construction and manipulation of SOAP messages with attachments is provides by SAAJ standard Java API.

Document based messaging service is also provided by JAX-RPC. A MIME-encoded content can be part of SOAP message using JAX-RPC. SSL-based security mechanisms and HTTP-level session management is supported by JAX-RPC for security needs.

2.4 WSDL4J

WSDL documents creation, representation and manipulation are done by Java Toolkit Web Service Description Language (WSDL4J). JSR110 ‘JWSDL’ provides reference implementation for WSDL4J.

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.

2.5 SAAJ

2.5.1 Overview of SAAJ

The section below gives a brief view of SAAJ messaging. Working mechanism and concepts explanation are stated.
Two main perspectives are kept in notice for overview. These are i) messages ii) connections.

2.5.2 SAAJ Messages

SOAP standard give format for messages, SAAJ follows this standard. SOAP also specifies things that are required, optional or not allowed at all. SOAP 1.1 and 1.2 specifications messages can be created with SAAJ API. These are XML messages. The message also conforms to WS-I Profile 1.1 specification. All these are done my making Java API calls.

2.5.3 The Structure of an XML Document

The structure of XML document is hierarchical such that it has elements than sub elements than elements inside sub elements which are called sub sub elements and the hierarchy goes on. Almost all SAAJ classes and their interfaces are represented by SOAP messages and XML elements they have word element or SOAP or both in their names. A single element is a node. There is an interface node in SAAJ API. This node is base class for other classes and interfaces.

2.5.4 What Is in a Message?

There are two types of messages. One is with attachments, other is without attachments.

2.5.5 Messages with No Attachments

Structure of SOAP message with no attachments is shown below.
I. SOAP message A. SOAP part 1. SOAP envelope a. SOAP header (optional) b. SOAP body

Figure 2-11 illustrates the structure of a SOAP message with no attachments.

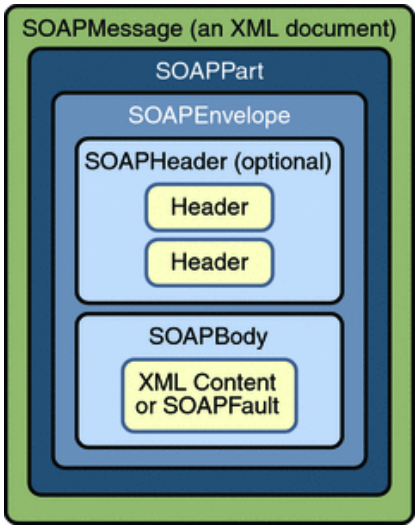


Figure 2-11 SOAPMessage Object with No Attachments

When `SOAPMessage` object is created it contains parts that are required to be in a SOAP message automatically. Which means a new `SOAPMessage` object contains an object of `SOAPPart` which has `SOAPEnvelope` Object. `SOAPHeader` object and `SOAPBody` object are already in `SOAPEnvelope` object. The `SOAPBody` object is empty at start. `SOAPHeader` object can be deleted if not required.

There can be one or more than one headers in a `SOAPHeader`. They contain metadata of the actual message. For example they may contain information of who is receiving party. `SOAPBody` object contains the actual SOAP message. Also it contains `SOAPFault` object.

2.5.6 SAAJ and DOM

SAAJ APIs counterparts are below:

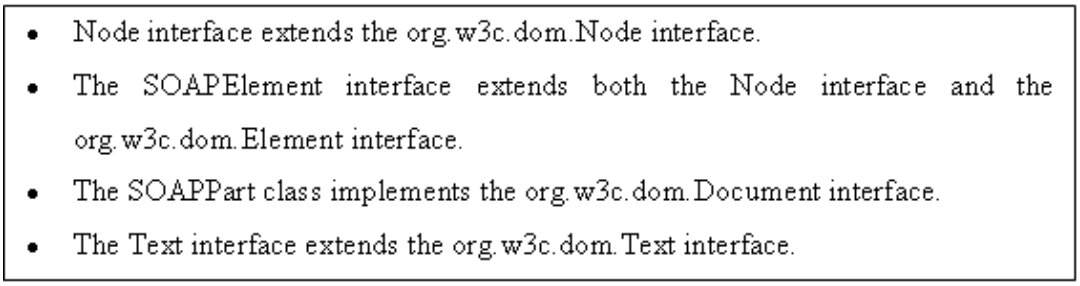


Figure 2-10 API Counter Parts

`SOAPPart` of a `SOAPMessage`, can be manipulated by applications, tools and libraries that use DOM, as it is also a DOM Level 2 Document.

2.5.7 SAAJ Connections

If there is a connection than SOAP message can be sent or received. In SAAJ, `SOAPConnection` object represents a connection. The connection is direct from sender to destination. The connection is between two endpoints so it is called point-to-point connection. Messages through these are called request response messages.

2.5.8 SOAP Connection Objects

`SOAPConnection` Object is given in following code fragment. Call method is used to send SOAP connection messages. When the message is sent it is blocked until a response is received.

The request parameter is the message being sent; endpoint represents where it is being sent.

```
SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();

SOAPConnection connection = factory.createConnection();

. . .// create a request message and give it content

java.net.URL endpoint = new URL("http://fabulous.com/gizmo/order");

SOAPMessage response = connection.call(request, endpoint);
```

The second argument to the call method identifies where message is sent it can be URL or String. So the last two lines of above code can be:-

```
String endp = "http://fablous.com/order";

SOAPMessage respo = connection.call(request, endpoint);
```

Web service returns a response for the request sent. This response is actual `SOAPMessage` object. Some requests may not require any response message at all. Still some response is required to confirm that service has been successfully called.

2.6 Castor

Castor XML helps binding java classes to XML document by mapping. Data in java object model is transformed into/from XML.

Castor can marshal and unmarshal Java objects by default. However sometimes there may be need to have control over this. An example is that: - Suppose Java object exists already, Castor Mapping need to be used as a bridge between Java and XML.

Mapping file is used to specify marshalling/unmarshalling behavior of Castor. Basic information of how and XML document and Java Objects relate is provided by this file explicitly.

XML document is written from point of view of Java and gives mapping information. It describes the properties of Java Objects which are to be translated. It describes field of each object so that there is information of each field to be mapped.

Property of object is represented by field. Field is theoretical correspondence of public class variable or property, where public class variable has direct and property has indirect accessor method. When Castor cannot find mapping file information and needs to handle an XML object it requires conjunction of mapping and Castor default behavior. Java Reflection API utilizes Java Objects to identify what to do. This is how Castor conjunction works.

In rare cases Castor cannot perform all mappings. Then it is essential to employ combination of XSL transformation and Castor. Due to this XML document is transformed in an understandable format.

2.7 XSLT

XSLT is a language which converts XML documents in to other XML documents. XSL includes XML and XSLT. XML terms are included for format specification. XSLT used these formatting terms for document conversion.

XSLT-defined elements are distinguished by namespaces of XML, so they are called XSLT namespace. To generate text and for conditional processing, XSLT uses expression language. The language is defined by XPath and is used for elements selection for processing.

2.8 Chapter Summary

In this chapter we have discussed related work to analyze how much work has been done in the field of Web Service Discovery and Composition. After that we gave information about Commercial APIs that are available to Support Service Search and Composition. We have used Google Custom Search API for Service Search and WSDL4J, SAAJ, Castor, XSLT for service invocation and composition.

Chapter 3

PROPOSED APPROACH

3.1 Problem Statement

Current approaches for service discovery have some of the following limitations: -

a) Querying heterogeneous registries at a time i.e. user can query only one registry at a time.
b) Retrieving up to date information on user's request.
c) In case of searching from web timely response is needed.
d) One time consuming task is that the users have to search whole registry each time whenever they need a service.
e) Majority of current approaches, lack a reliable, stable and trust-worthy discovery.
f) Services are themselves heterogeneous i.e. they have different formats for exchanging data.
g) The published web services are tagged with a lots of information that makes it difficult for a program to find the required web service on given attribute.[2]
h) Keywords are used to discover web services in UDDI. Ranking services and filtering them is the main advantage of UDDI. The most important drawback is that search can only be made on basis of metadata so it limits the search criteria.

Primary purpose of service composition is to enhance the functionality of web services and to get automated results. Also there should be flexibility as well as agility. Few problems faced in composition are given below:-

a) Dynamic composition needs very little user involvement which makes it difficult to find out an exact required service on huge repository of internet.
b) Secondly, all the services on internet are not public. So, there is a need to select service of user interest. Not only to functionality but also its accessibility, .i.e., if a service provides required functionality, whether it is available to be called publically or not.

c) Transactional support can be very small in fully automated composition as different service providers may have different conceptual models. (transactional support means support for exchange of data between different services)

d) Compositional correctness cannot be guaranteed as automation cannot verify middle stages of composition.

e) Full automation is possible for static infrastructures. If there is need of an application in which requirements change too often (i.e dynamic needs), a little user intervention is helpful.

3.2 Proposed Framework

Proposed framework of web services composition includes the following components:-

3.2.1 Service Provider

Main purpose of a web service is to provide user's required functionality. This functionality is provided by some individual or organization. Service providers register their services in registries to make them accessible to clients. There are many registries provided by different companies which are synchronized after regular period.

3.2.2 Service Requester

Client wants to use web service provided, and is called requestor. 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. The requestor initiates the message exchange most of the time.

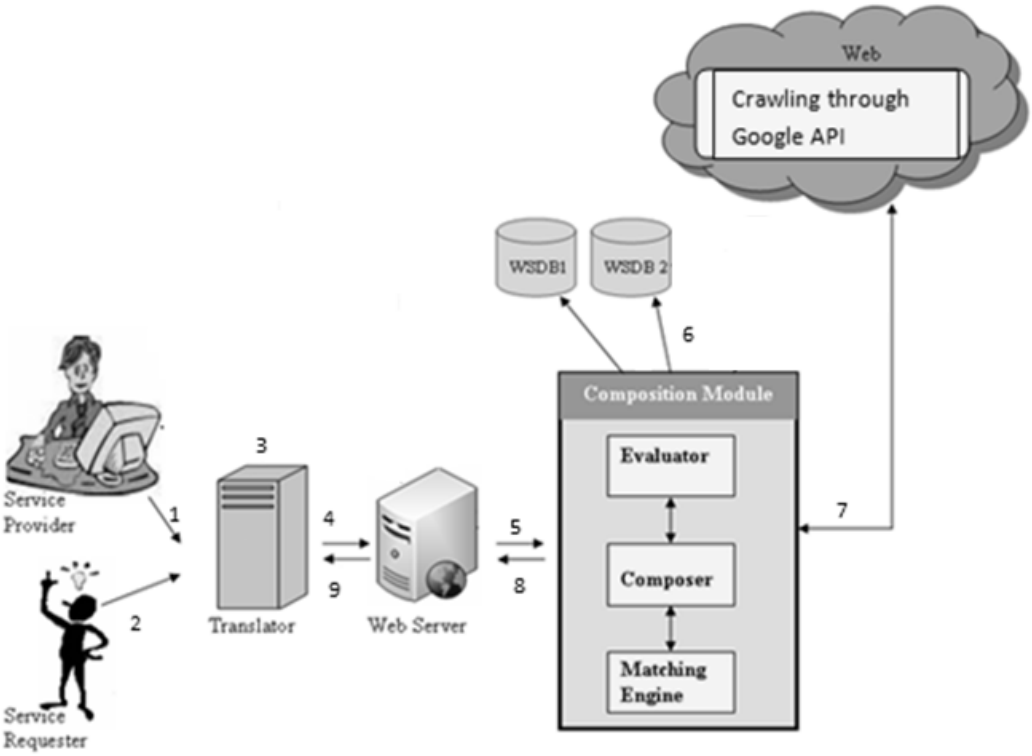


Figure 3-1 Proposed Framework for Dynamic Web Services Composition

3.2.3 Web Server

A place where service is actually hosted and performs its functionality.

3.2.4 Translator

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.

3.2.5 Evaluator

Evaluator evaluates weather the service is valid and is available at current time.

3.2.6 Composer

Composer composes the selected web services.

3.2.6.1 Framework of Composition

The framework shown in Fig 3.1 is self explanatory. It involves following workflow.

1. User queries the system for required service.

2. Service is selected.

3. Information of selected service is retrieved.

4. User enters required input parameters for first service.

5. User adds the service to composition module and selects next service.

6. On basis of information of service matchmaking is performed.

7. If number of parameters match.

    i) Composition is performed.

    ii) Results are displayed.

8. If number of parameters do not match. User is prompted to take action.

9. Composition is performed after user's action.

10. Results are displayed.

All communication with web server is done by XML messaging.

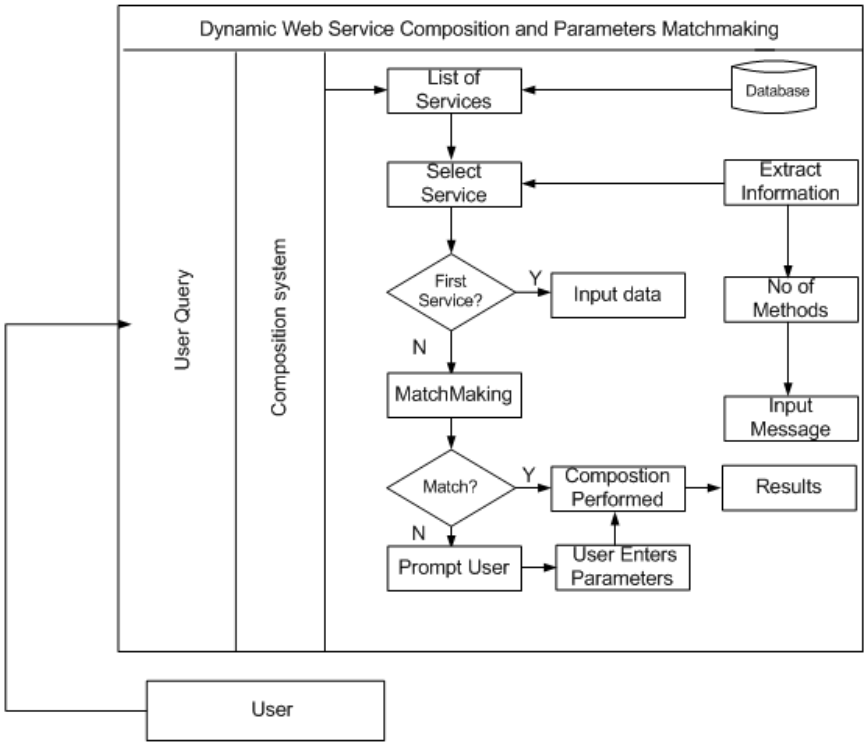


Figure 3-2 Workflow for Dynamic Web Service Composition and Parameters Matchmaking

The algorithm of Composition process is given as:

Algorithm: Web Service Composition
Input: Request for Web service
Output: Composed Service Results
User requests a desired service from Database;
User selects a service from list;
For each Selected Service
    Information about service is retrieved.
    The information includes number of methods and input message.
For First selected service user enters input data in XML message.
For remaining services selected Matchmaking is performed.
    If number of parameters match composition is performed and results are displayed.
    If number of output parameters are less than the number of required input parameters user enters the remaining parameters.
    If the number of output parameters are greater than the required input parameters. User selects some from previous result and/or enters other parameters.
Composition is performed and results are returned to interface.

Figure 3.3 Algorithm for Dynamic Web Service Composition and Parameters Matchmaking

During composition user selects more than one service at a time, only additional step is that matchmaking is performed and output of first service is sent as input to other.

In matchmaking number of input output parameters are matched. If the number of input and output parameters are same. The composition is performed without any interruption. If number of output parameters of first service is more than the number of required input parameters for second, user is prompted to enter the required parameters. While if number of parameters of first are more than the required input parameters for second user is again prompted to select some from the intermediate output and/or enter from text box. Final result is displayed to the user when response of all services invocation is successful.

3.2.7 Matching Engine

It performs matchmaking during composition. The numbers of input and output parameters are matched.

3.2.8 WSDB

A database of around 5000 WSDL links is maintained. User can query database if no other option is available.

3.2.9 Crawling Process

1. User queries the system. The input can be any word in users mind. System matches the query word not only with service interface but also with its methods.

2. The request goes to Google Custom Search Engine through Google Custom Search API.

3. The engine has been scaled to the desired links to crawl. It can be scaled any time by adding more links on the engine's control panel.

4. Engine crawls on all the available links and produces the results.

5. Results produced are in a format infeasible for ~~manual~~ processing. So the system parses the results produced.

6. System Extracts the WSDL files from the set of results.

7. Results are displayed to the Client.

8. To ensure that the service is available at given time, validity check is performed.

9. Results are displayed and sent to local database.

10. A backup database is maintained to ensure reliability.

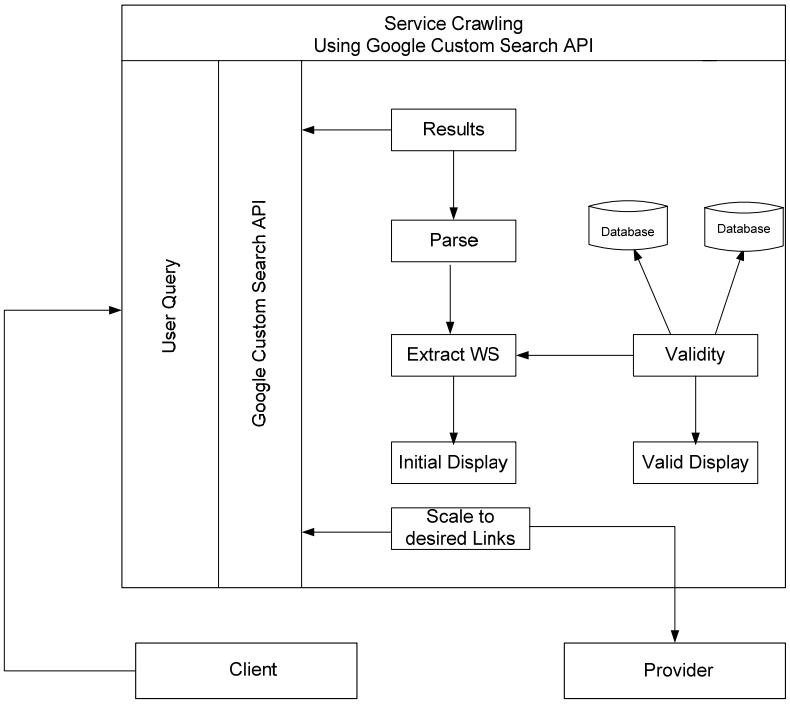


Figure 3.4 Framework for Service Crawling using Google Custom search API

The algorithm of crawling portion is given as below:

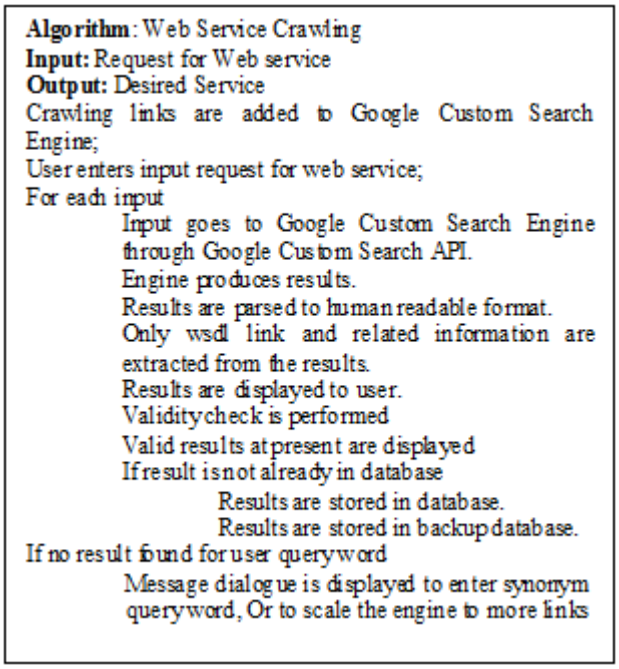


Figure 3.5 Algorithm for Service Crawling using Google Custom search API

Since more parent links can be added to Google Custom engine, the user has more chances of getting the required service which is updated and exact. So our framework is scalable and flexible. Crawling procedure is same as compared to other open source crawlers. The user query is matched on all the available child links of the provided link. But the engine response is much efficient than the application crawlers.

Further those crawlers can crawl only one domain at a time. The custom search engine crawls all the provided links at once.

3.3 Methodology

The methodology used is as under: -

1. Services are registered in service repositories on internet.

2. User queries the system and translator converts it into language used by internal system.

3. The request arrives, application searches for the requested service from its Database. If it gets the required service it sends results back to requester. Moreover, multiple databases are introduced, so that there is a backup if one goes down. If required service is not found from database the user then requests to crawl through Google Custom Search API.

4. Evaluator checks validity of the service and user selects the desired service.

5. In composition module matching engine matches the number of parameters. After resolution it sends selected services to composition module. Composer integrates these services.

6. Services are composed via web server request, and application returns the results back to client.

3.4 Chapter Summary

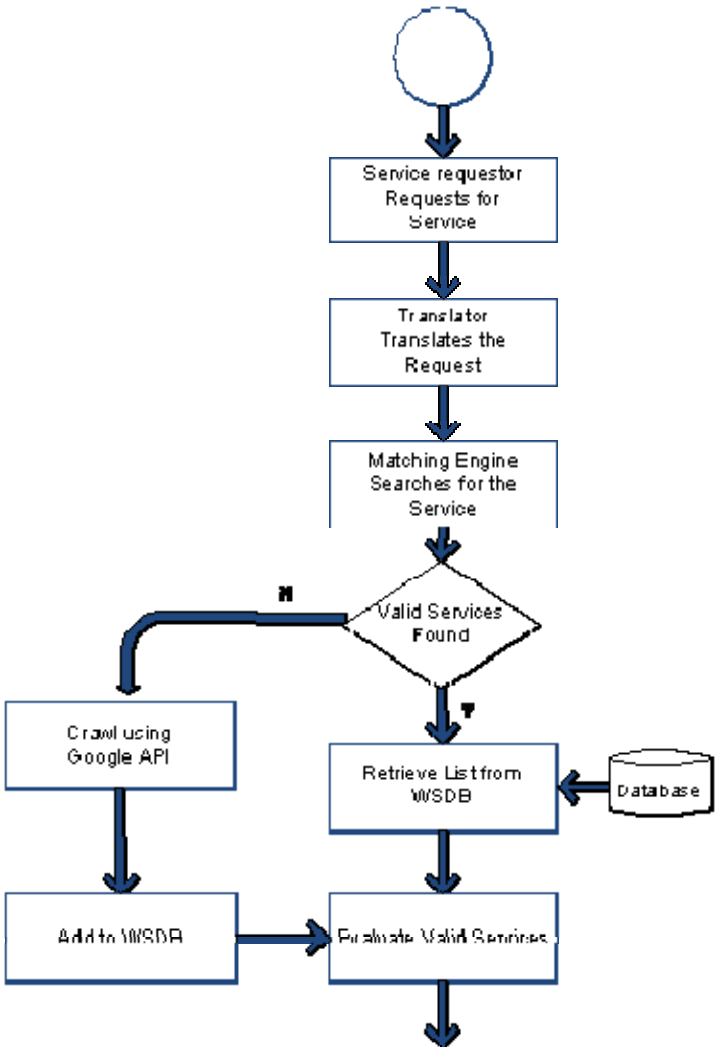
In this chapter we first defined the problem statement then proposed a framework to overcome current problems faced during Web Service Discovery and Composition. Next we discussed all the components needed for web service discovery and composition. Two sub-parts, framework of composition and framework for service search are described in detail. They are discussed along their workflow and complete algorithm. At the end we gave complete methodology to implement this framework.

Chapter 4

SYSTEM DESIGN

4.1 Data Flow Diagram:

Figure 4-1 shows the flow of data for web services composition. The flow begins by entering the user request and then searching the desired services. To fulfill the user request the desired service is discovered and composed. The composed service is returned to service requestor through translator.



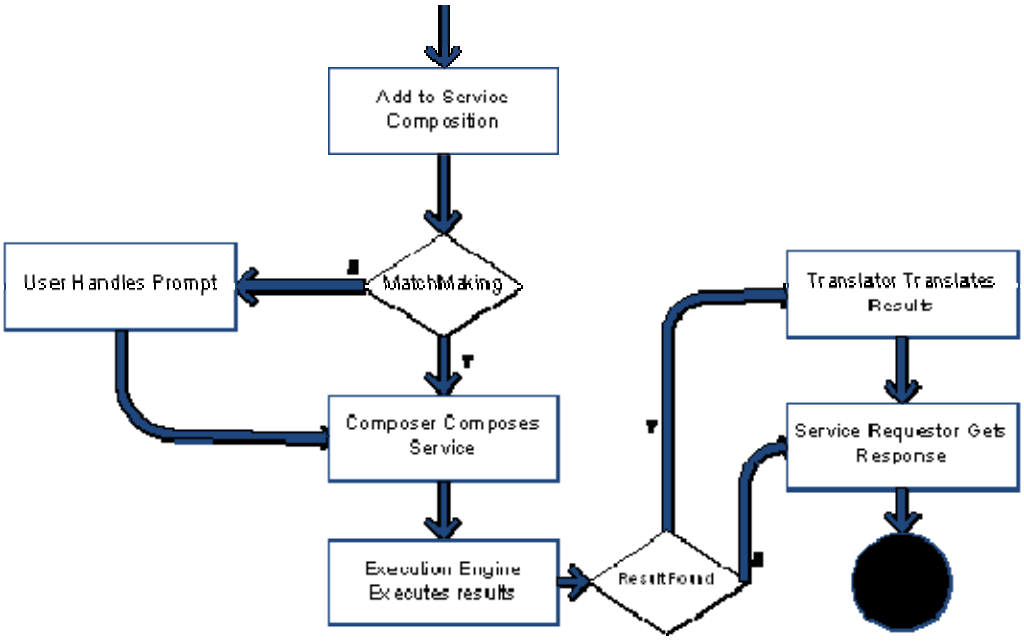


Figure 4-1 Dynamic Web services Composition Using Google API Crawling DFD

4.2 Sequence Diagram

The sequence diagram in figure 4-2 shows different processes and objects that work simultaneously. Processes and objects are shown as parallel vertical lines. Message exchanges are shown by the horizontal arrows. The diagram shows the complete sequence of steps starting from service request to web service composition. Matching engine matches the requested service and Execution engine executes the discovered service. Finally composer composes and returns the desired result to user.

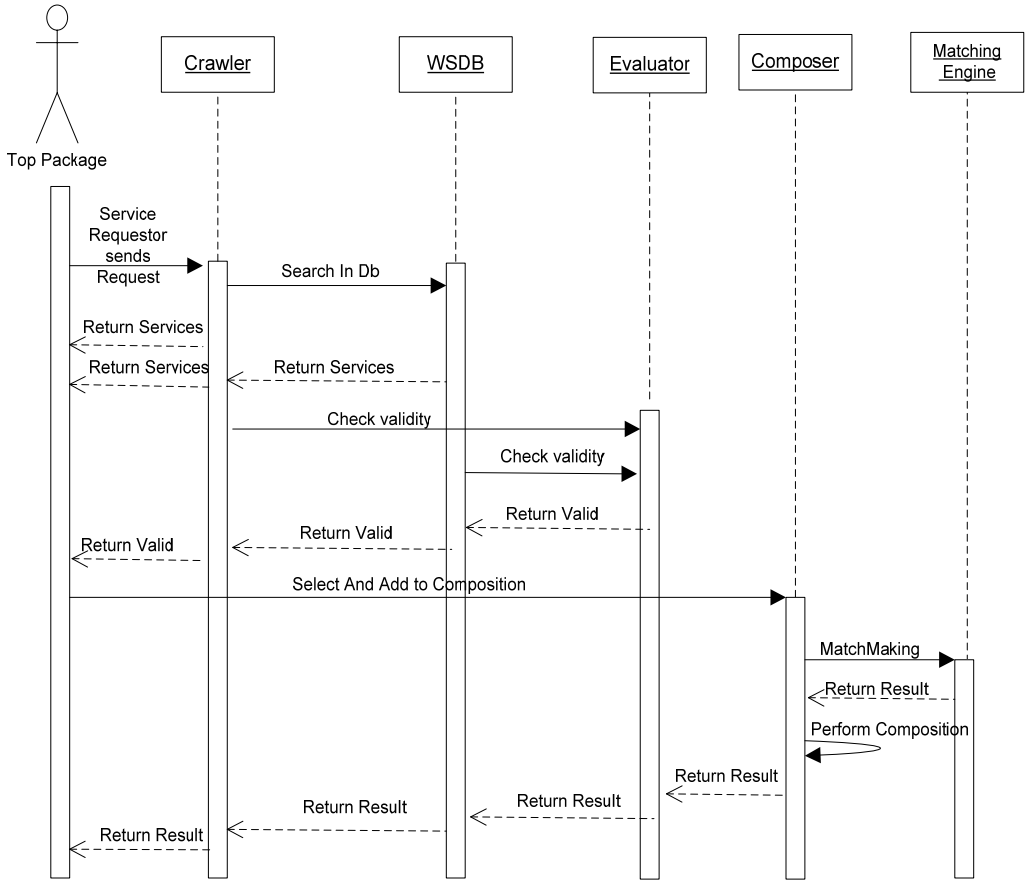


Figure 4-2 Sequence diagram of proposed framework

4.3 Use Cases

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: Search DB, Crawl, Validity, Add to Composition, Invocation and Composition.

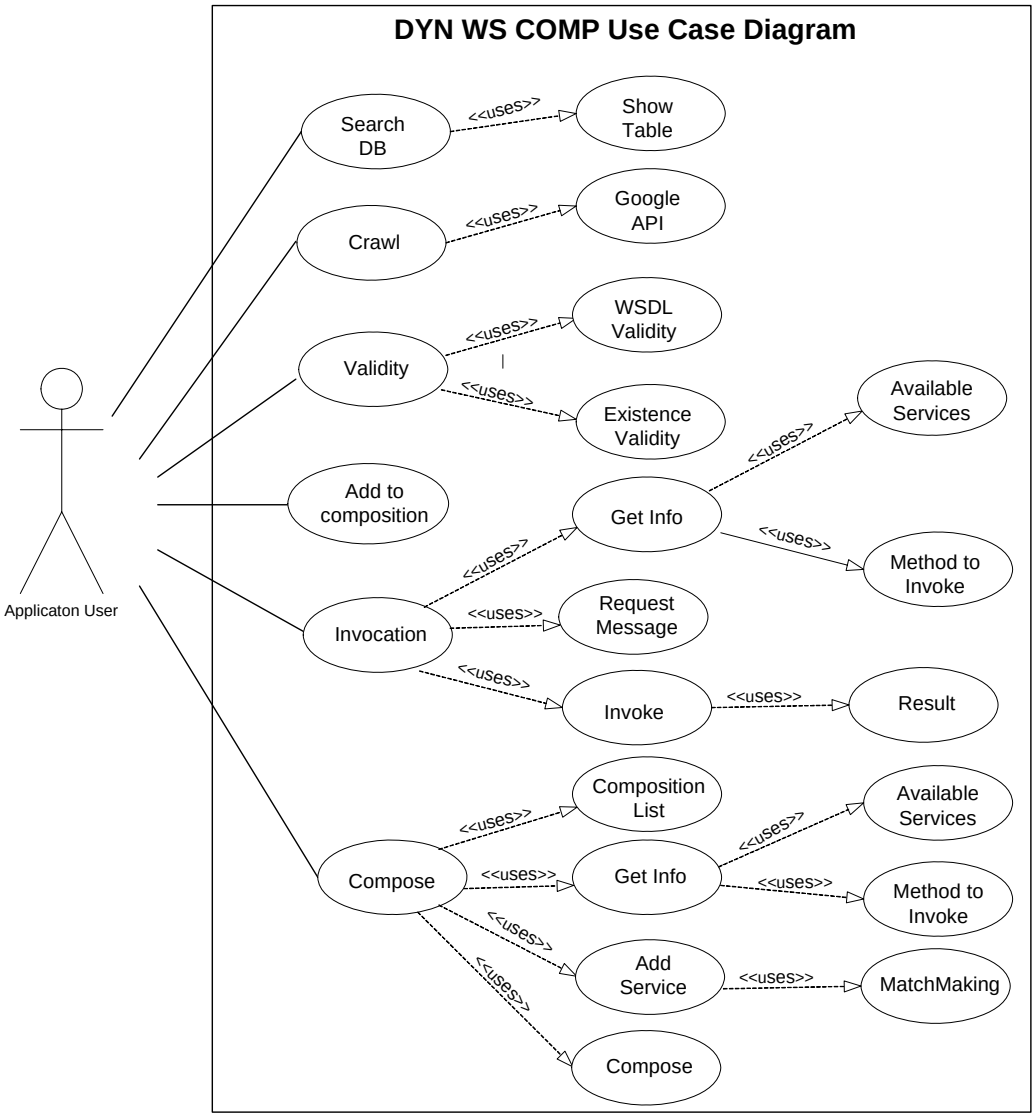


Figure 4-3 Dynamic Web Services Composition Using API Crawling Use Case Diagram

4.3.2 Extended Use cases for Interface

4.3.2.1 Search Database:

Use case: Search Database

Actors: APPLICATION USER

Pre Condition: APPLICATION USER request for service.

Post Condition: Services are listed.

Description: APPLICATION USER queries for service by clicking Database Button.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER clicks on Database button.	
	2. System shows the available services and its related information.

Alternative courses

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

4.3.2.2 Crawl*:*

Use case: Crawl

Actors: APPLICATION USER

Pre Condition: APPLICATION USER requests for Crawl.

Post Condition: List the search Result.

Description: APPLICATION USER searches for service by clicking on crawl button.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER requests for service search.	
	2. System shows results returned.

Alternative courses

Line 2: Service not found, indicate an error.

4.3.2.3 Validity*:*

Use case: Validity

Actors: APPLICATION USER

Pre Condition: APPLICATION USER enters Valid Tab.

Post Condition: Valid links displayed.

Description: APPLICATION USER enters into Valid Tab. Validity checks are performed.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER clicks on Valid Tab.	
	2. System shows the Valid Links.

Alternative courses

Line 2: No Valid Link found, indicate an error.

4.3.2.4 Add to Composition:

Use case: Add to Composition

Actors: APPLICATION USER

Pre Condition: APPLICATION USER adds service to Composition Tab.

Post Condition: Service Added.

Description: APPLICATION USER adds service from application interface by selecting the particular Service and clicking on Add to composition button.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when APPLICATION USER selects a service.	
2. USER than clicks on Add to Composition button.	
	3. Service Added to Composition.

4.3.2.5 Invocation:

Use case: Invocation

Actors: APPLICATION USER

Pre Condition: APPLICATION USER enters Invocation Tab.

Post Condition: Service Invoked.

Description: APPLICATION USER selects a service from Valid tab and enters into Invoke tab.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when User Selects service from Valid Tab and enters into Invoke Tab.	
2. User than clicks get info button	
	3. List of available Services and Methods are shown.

4. User Selects Required Method	
	5. Request Message Structure is shown.
6. User edits input parameters	
7. User clicks invoke Button	
	8. Results are shown in Table

Alternative courses

Line 2: No Result Found, indicate an error.

4.3.2.6 Compose Services:

Use case: Get Info

Actors: APPLICATION USER

Pre Condition: APPLICATION USER enters composition tab.

Post Condition: Information of selected service is displayed.

Description: APPLICATION USER views all the Services listed in composition tab and selects one.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER selects a service from list.	
2. User clicks Get Info button	
	3. Information of service is displayed

Alternative courses

Line 2: No Service is selected from composition list, indicate an error.

4.3.2.7 Compose Services:

Use case: Add

Actors: APPLICATION USER

Pre Condition: APPLICATION USER Adds the Services.

Post Condition: Services added to be composed.

Description: APPLICATION USER adds all the services to be composed and gives required input.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER gives first input parameters.	
2. User then Clicks on Add button.	
3. User then Selects next service to be added	
	4. Services are added to be composed.

Alternative courses

Line 2: No first input given by user, indicate an error.

4.3.2.8 Compose Services:

Use case: Matchmaking

Actors: APPLICATION USER

Pre Condition: APPLICATION USER clicks Add button.

Post Condition: Matchmaking is performed.

Description: APPLICATION USER adds different services to be composed matchmaking is performed.

Typical Course of Events

ACTIONS:	**SYSTEM RESPONSE:**
1. This use case begins when USER clicks on Add button.	
	2. System performs Matchmaking for input output parameters.
	3. System Prompts user if numbers of parameters are not same.
4. User gives the required input.	
	5. System shows the complete detail of selected service.

Alternative courses

Line 2: No Service is added composition list, indicate an error.

4.3.2.9 Compose Services:

Use case: Compose service

Actors: APPLICATION USER

Pre Condition: APPLICATION USER clicks Compose button.

Post Condition: Services are composed.

Description: APPLICATION USER wants the composed service and systems returns final result after composition.

Typical Course of Events

ACTIONS:	SYSTEM RESPONSE:
1. This use case begins when USER clicks on compose web services button.	
	2. System shows the required result

Alternative courses

Line 2: No Service Result found, indicate an error.

4.3.2.10 Clear All:

Use case: Clear All

Actors: APPLICATION USER

Pre Condition: APPLICATION USER clears composition information.

Post Condition: All information cleared.

Description: APPLICATION USER wants to clear and composition steps by clicking on clear all button.

Typical Course of Events

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

4.4 Chapter Summary

In this chapter we illustrated working of framework with the help of dataflow diagram and sequence diagram. Then we have given a generic use case diagram to make it more understandable which includes Search DB, Crawl, Validity, and Add to Composition, Invocation and Composition. Individual use cases include Actors, Pre Condition, Post Condition, Description, Typical Course of Events and Alternative course where required.

Chapter 5

IMPLEMENTATION

5.1 Accessing Google Custom Search

5.1.1 JSON/Atom Custom Search API

To retrieve and display results programmatically from Google Custom Search, JSON/Atom Custom Search API is used. Requests are made to get the results in either JSON or Atom format.

An API key is required for using JSON/Atom Custom Search. This key is available at Google APIs console. There is a limitation that one can query only 100 times per day. However if one needs to query more than 100 times billing option is available.

Following steps are required for setting up a Google custom search engine.

1) Get a Google account

A Google account is created to use Google Custom Search API.

2) Set up a custom search engine

There is a need to setup the engine first. To set up and customize the engine selected sites are included in the search and other options are configured. By Clicking on "control panel" one can note the Search engine's unique ID. This is the cx parameter used by the API.
Our cx is "*cx=001389246465683457042:ocadwlz3xgu*"

3) Identifying application to Google

Application needs to identify itself every time it uses API to send request to JSON/Atom. An API Key is included with each request.

4) Acquiring and using an API key

In the API consoles Services pane, activate the JSON/Atom Custom Search API; the section "Simple API Access" contains particular API key.

The query parameter key=*yourAPIKey* can be appended to all requests.

Our key is "key=AIzaSyDL9RdFrNh-x4fXVdwGCVfN98QxkJollBw"

5.1.2 Java Access

Following code reveals the request to API request from java. JSON library is used. In the example below the users own key has to be given. The request also includes the parameter userip. The userip parameter tells the server from which IP this request came from.

Here is Example Code Snippet:-

```
URL url = new URL(
   " https://www.googleapis.com/customsearch/v1?&key=AIzaSyDL9RdFrNh-
x4fXVdwGCVfN98QxkJollBw&cx=001389246465683457042:ocadwlz3xgu&callback=proces
sResults&filter=1&q="+value+");
URLConnection connection = url.openConnection();
connection.addRequestProperty("Referer", /* Enter the URL of your site here */);

String line;
StringBuilder builder = new StringBuilder();
BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
while((line = reader.readLine()) != null) {
 builder.append(line);
}
```

Figure 5-1 Request to Google API from Java

5.1.3 Search results

The actual search results are contained in an items array. The results contain URL, main title and actual text snippet. If results are on more than one page they also contain page map information.

5.2 Functions used for Crawling

Following functions are used for crawling:

5.2.1 public void crawls(int ab)

This function actually calls the Google API and returns the search results.

Sample Code Snippet :-

```
public void crawls(int ab)

 {

URL pageUrl = new
URL("https://www.googleapis.com/customsearch/v1?&key=AIzaSyDL9RdFrNh-
x4fXVdwGCVfN98QxkJollBw&cx=001389246465683457042:ocadwlz3xgu&callback=
processResults&filter=1&start="+sta+"&q="+value+"asmx?wsdl");

 URLConnection getConn = pageUrl.openConnection();

 getConn.connect();

BufferedReader dis = new BufferedReader(new
InputStreamReader(getConn.getInputStream()));

 while ((urlData = dis.readLine()) != null) {

 System.out.println(urlData);

}

}
```

Figure 5-2 Function call Google API

The above code takes input variable "`value`" to search web service in given links. Buffered reader variable reads the result returned by the google search query.

5.2.2 public void splitfunc()

This function is used to parse the results to user friendly format. The `match ()` and `split()` functions are used to parse the returned data into the desired form.

Sample Code Snippet:-

```
if(urlData.matches(".*totalResults.*"))
                                    {
                                      tr1=urlData.split(":")[1];
                                      tr1=tr1.split(" ")[1];
                                      tr1=tr1.split(",")[0];
                                      tr1=tr1.split("\"")[1];
                                      tr=Integer.parseInt(tr1);
                                    }
                                    //to split the required data
                                    if (urlData.matches(".*items.*")) {
                                       i = 1;
                                    }
                                    if (i == 1 &&
            urlData.matches(".*title.*")) {
                                       info1 = urlData;
                                       info1 = info1.split(":")[1];
                                       info1 = info1.split("\"")[1];
                                       info1=info1.split("-")[0];
                                       // System.out.println(info1); ……
                                }
```

Figure 5-3 Function to Parse Search Results

5.2.3 public void valid()

This function checks weather the returned result is a valid wsdl or not.

Sample Code Snippet:-

```
public void valid()
        {
        if((infolink.matches(".*/?wsdl.*"))||(infolink.matches(".*/?WS
        DL.*")))
        {
        countv++;
         }
        else

        try {
         infolink= vc.parvalid(infolink);
         if (!infolink.equals("0"))
        {
        countv++;
          }

        }
```

Figure 5-4 Function to Validate Search Results

This function checks the extension of the returned links, whether they are wsdl links or not. It also uses the parvalid function for returned links that are in different format.

Where parvalid function is:

```
public String parvalid(String infolink)

{
        if(infolink.matches(".*asmx"))
     {
         infolink=infolink+"?wsdl";
     }
    if(infolink.matches(".*op.*"))
     {
          infolink=infolink.split("op")[0];
         infolink=infolink+"wsdl";
      }
    else
     if(infolink.matches(".*CATID.*"))
     {
        ……
       }
       ……
}
```

Figure 5-5 Function to Validate Search Results of Different Format

5.3 Simple WS Client

Before going for composition of web services there is a need to make a simple client to invoke a web service. The client is general such that any sort of service with parameters of any number or type can be invoked.

User has to select one most suitable service out of search results, provide required input and invoke the service to view results. The application uses industry standard tools and Java APIs which are readily available e.g. WSDL4J, SAAJ, Castor, JDOM, and XSLT.

When the user selects a WSDL file, it is loaded such that all operations associated with it are displayed. Operations can be for example GetQuote, GetHistoricalQuote etc. Again when user selects a single operation application displays dummy message of corresponding input in XML

format. User can fill in the input values in that message and click invoke button to run the service. Original SOAP message will be sent to retrieve results. Result message can contain an actual service result or SOAP fault which portrays failure.
The procedure from loading WSDL file to service response is described below:-

5.3.1 WSDL Parsing and Analysis

Web Service Description Language describes an XML grammar to define Web Services. Concrete data formats and deployments are separated from definitions of messages and endpoints by WSDL.

WSDL provides details of the communication requirements of Web service. This description is necessary for service invocation by client. WSDL provides messages to be exchanged between client and service provider. These messages are in XML model.

TABLE 1: Defining a Web Service

WSDL Element	What the Element Describes
Types	Describes the data types used by a Web service using a type system such as XML schema.
Message	The abstract format of a particular message that a Web service sends or receives
Operation	The abstract description of an action supported by a Web service.
Port Type	A named abstract collection of operations.
Binding	Concretely defines the protocol and data format specification for a specific port type.
Port	A single endpoint defined by a combination of a binding and a network location
Service	A collection of ports that a Web service provides

Figure 5-6 A WSDL document uses the elements shown for defining a Web service.

All elements shown in above diagram supply Web Service invocation information. Our Application uses WSDL4J to analyze the structure of the WSDL programmatically and to identify the operations available for consumption.

5.3.2 Finding the Services and Operations

`ComponentBuilder` is a class in our application. It uses WSDL4J, the Castor Schema Object Model and JDOM. Together they all make local model of web service and analyze it. To load

the WSDL definition our class creates an instance of WSDL4J. This is the instance of WSDLReader and it loads the WSDL definition. User sends URI of WSDL document to `readWSDL()` method that returns instance of Definition interface.

Due to definition interface user is able to know the methods needed. These methods include how to analyze the WSDL definition, how to programmatically discover the defined services, operations of services, their data types and URI of service's endpoint. At this point we have definition of WSDL document in systems memory. We can call the services that are defined. A method named `getServices()` returns Service instances.

5.3.3 Finding the Parts

To retrieve parts of SOAP defined for current message, a method `getParts()` is defined by Message interface. An instance of the Part interface is message part. To retrieve part's element's and data type's names methods are defined by Part interface. This information is used to obtain related schema for individual message part. This enables us to define a request message. This message is sent to Web service to process the result generated by invocation.

5.3.4 Creating Sample XML Input

To invoke a Web service we need to create XML message. This message is used as request data. The message is created on basis of information of message parts. In case of complex message part, XML message will be passed as part of SOAP. If a complex type response is expected a reference to schema is required to process the result.

A method `buildMessageText()` takes input as an instance of WSDL4J message. `ComponentBuilder` class defines this method. List of parts defining a message is obtained from the Message object. A sample input text is built after iteration of message parts. There is a need to check whether a complex type is defined for each part processed. This complex type must be defined in Castor Schema Object Model.

XML initial instance is generated for each message part to invoke Web service. An example input message is shown below:-

```
<GetQuotesHistorical xmlns="http://www.xignite.com/services/">
<Symbol>SUNW</Symbol>
<Month>12</Month>
<Year>2002</Year>
</GetQuotesHistorical>
```

Figure 5-7 Input Message Schema

The message above is saved to an instance named `OperationInfo`. Then this is used to be sent as initial message to invoke service.

5.3.5 SOAP Invocation

To invoke a Web service our application uses Java SAAJ API. SAAJ provides straightforward and flexible way to consume a service. The elements necessary to invoke Web Service using SAAJ are:

- Create a connection
- Create the message
- Add message content
- Send the message to the destination
- Process the response

5.3.5.1 Creating a Connection and Message

A connection is created by calling `newInstance()` method of `SOAPConnectionFactory`. A class of SAAJ named `MessageFactory` is used to create new `SOAPMessage` instance. This `SOAPMessage` instance comes with `SOAPEnvelope` which is contained in `SOAPPart`. Further `SOAPEnvelope` comes with `SOAPHeader` and `SOAPBody`. Both of them are empty at start. `SOAPHeader` part is not mandatory. It can be removed if not needed.

5.3.5.2 Adding Content

Actual content is added to `SOAPBody` using instances of `SOAPElement`. We add the name of the service we are calling to `SOAPElement`. Input message parts are then added as `SOAPElements` childs.

5.3.5.3 Invoking the Web Service

The sample input message gives the request content. This message was built during analysis of WSDL. This content is only an initial value. It must be edited by user. GUI is used to fill out the request message. We can invoke the web service when we have added the input content to the initial request message. The response is a `SOAPMessage` in XML.

5.3.5.4 Making Sense Out of the Response

Responses of a web service can be long and confusing. They contain data that is irrelevant to the client of the service. Our tool generates structure for particular web service. The structure each time knows the specifications how to traverse the WSDL and extract the actual response. Common XML schema elements are of no difficulty for example ComplexType and Sequence. So it can infer where to display the result element and when to create table.

5.4 Functions used for WSDL Analysis and Invocation

5.4.1 private void analyzeWsdl(String purl)

This uses `showServiceInfo` and `showOperationInfo` to analyze wsdl. This function returns name of the service and its all methods. Also it returns the dummy input message which is used for sending actual input later.

```
private void analyzeWsdl(String purl)
  {
       // Create the in memory model of services and operations
       // defined in the current WSDL
       ComponentBuilder builder = new ComponentBuilder();
       List services = builder.buildComponents(url);
      // List all the services defined in the current WSDL
       Iterator iter = services.iterator();
       while(iter.hasNext())
       {
         // Load each service into the services combobox model
         ServiceInfo serviceInfo = (ServiceInfo)iter.next();
         serviceModel.addElement(serviceInfo);
       }
  …….
     }
```

Figure 5-8 Analyze WSDL Function

5.4.2 private void showServiceInfo(ServiceInfo serviceInfo)

It is used to show information of service. Information of service is actual name of the service on provided link.

```
private void showServiceInfo(ServiceInfo serviceInfo)
  {
    // Clear UI components
    operationModel.removeAllElements();
    if(serviceInfo == null)
    {
      return;
    }
    // Load the operations model with operations defined for this service
    Iterator iter = serviceInfo.getOperations();
    while(iter.hasNext())
    {
      // Load each service into the appropriate combo box model
      OperationInfo operInfo = (OperationInfo)iter.next();
      operationModel.addElement(operInfo);
    }
  }
```

Figure 5-9 Service Info Function

5.4.3 private void showOperationInfo(OperationInfo operationInfo)

It is used to show operations of the service, total number of methods and their names. It also displays the input dummy message.

```
private void showOperationInfo(OperationInfo operationInfo)
  {
        if(operationInfo != null)
    {
           outputpar[ioc]= operationInfo.getOutputMessageText();
 inputpar= operationInfo.getInputMessageText();
       if(checkc==1)
             {
       opinc=operationInfo.getInputMessageText();

       System.out.println(opinc);
       }
      else
       messageText.setText(operationInfo.getInputMessageText());
    }
    else
    {
      messageText.setText("");
    }
   }
```

Figure 5-10 Operation Info Function

5.4.4 public String[] sendRequest(OperationInfo operationInfo1)

It is used to send input request message to the service. This function creates Object of WSClient class to invoke operation of the selected service. It takes OperationInfo1 variable as input. OperationInfo1 contains the information of selected method to be invoked. Function sends the input message and gets the response. Further it also parses the result to human readable format.

```
    public String[] sendRequest(OperationInfo operationInfo1)
  {
check1=0;
     // Send the request and get the response
      WSClient_1 ab= new WSClient_1();
     String response = ab.invokeOperation(operationInfo1);
while(cot<MyStringArray.length)
{   if(MyStringArray[cot].matches(".*lt.*")||MyStringArray[0].matches(".*gt.*"))
       {
   if( newarray[cot].matches(".*Table.*"))
     {
          head[cot1]=newarray[cot].split("&lt;")[1];
      head[cot1]=head[cot1].split("&gt;")[0];
          checkonly[cot1]=0;
          if( newarray[cot].matches(".*/.*"))
               {
…….
     }
    else
      if(!(newarray[cot].matches( ".*xml version.*")))
      {
          checkonly[cot1]=0;

          if( newarray[cot].matches(".*/.*"))
               {
             result[count2]=newarray[cot].split("&gt;")[1];
             result[count2]=result[count2].split("&lt;")[0];
            head[cot1]=head[cot1];
            checkonly[cot1]=1;
……..
}
```

Figure 5-11 Send Request Function

5.5 Composition

We have used all the above procedure for integrating Web services. User can select more than one service for composition. We have configured the input XML message to take input for first service. Also we have parsed the output to get results in required format. During composition user selects more service, only additional step is matchmaking is performed and output of first service is sent as input to other. In matchmaking only number of input output parameters are

matched. If these are same composition is performed without any interruption. If number of output parameters of first service is more than the number of required input parameters for second. User is prompted to enter the required parameters. While if number of parameters of first are more than the required input parameters for second user is again prompted to select some from the intermediate output and/or enter from text box. At the end of all services invocation response final result is displayed to the user.

5.6 Functions used for Composition

For invocation during composition all functions of 5.4 are used.

5.6.1 public void countparam(String param)

It is used to perform parameters match making. It counts the number of parameters of service.

```
 public void countparam(String param)
   {
String [] myarray = new String[100];
StringTokenizer tokens = new StringTokenizer(param,"\n");
i=0;
while(tokens.hasMoreTokens()) {
myarray[i] = tokens.nextToken();
myarray[i]=myarray[i].trim();
if (myarray[i].matches(".*0.*"))
{
   countpar++;
}
i++;
}
System.out.println("out"+countpar);
   }
```

Figure 5-12 Count Parameter Function

5.6.2 public String matchmake (int countco2)

It is used to display the matchmaking decision. This function actually checks whether the number of output parameters of first service and number of input parameters of next matches or not.

```
public String matchmake(int countco2)
   {
   if (countpar==countinpar)
   {
System.out.println("matched");
return "matched";
   }
   else if(countpar>countinpar)
   {
System.out.println("greator");
return "greator";
   }
      else if(countpar<countinpar)
   {
System.out.println("less");
abe=mdob.coux;
if(abe==0||countco2==2)
{
mdob.show();
      }
 return "less";
      }
   return "null";
   }
```

Figure 5-13 Parameter Matchmake Function

5.7 Chapter Summary

In this chapter we have briefly described the actual implementation of the framework. We explained how we have done web service discovery by Accessing Google Custom Search via JSON/Atom Custom Search API and Java Access. The Search results are parsed in readable format. Then we discussed Functions used for Crawling along with sample code snippet. Complete procedure of service analysis and invocation is discussed in Simple WS Client. Functions used for WSDL Analysis and Invocation also contain description with sample code snippet. Composition is performed by number of parameters matchmaking. Functions used for Composition gave details of composition.

Chapter 6

RESULTS

Measuring the performance of web service composition 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 and Fallout. Static Dynamic and Statistical factors are fundamental for evaluation of quality of Web Service Discovery and Composition. Static factors remain constant dynamic factors changes according to certain situation and statistical factors are calculated by actual statistical data of the web service. The hardware environment used is Intel Core i5 CPU with 4GB RAM and Windows 7 Ultimate operating system.

6.1 Definitions

6.1.1 Precision

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

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

6.1.2 Fallout

It is the proportion of all non-relevant services retrieved out of all retrieved services.

6.1.3 Static Factors

Following constant factors are considered.

Table 6-1Static 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?

6.1.4 Dynamic Factors

Following dynamic factors are taken in account

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?

6.1.5 Statistical Factors

Statistical factors are stated in table.

Table 6-3Statistical 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?

6.2 Dataset

The framework is implemented in Java 6 using Netbeans 6.9 integrated development environment. WSDL4J (Web Service Description Language for Java) is used to parse the WSDL files. For the evaluation of our framework, we have setup Database with a total of around 5000 web services WSDL references present. The actual services are hosted by the service providers on their web servers.

6.3 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. 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.

6.3.1 Average Precision

We took various sets of services and for each set we made 25 readings and then compute an average for that set. Following is the average precision of our framework.

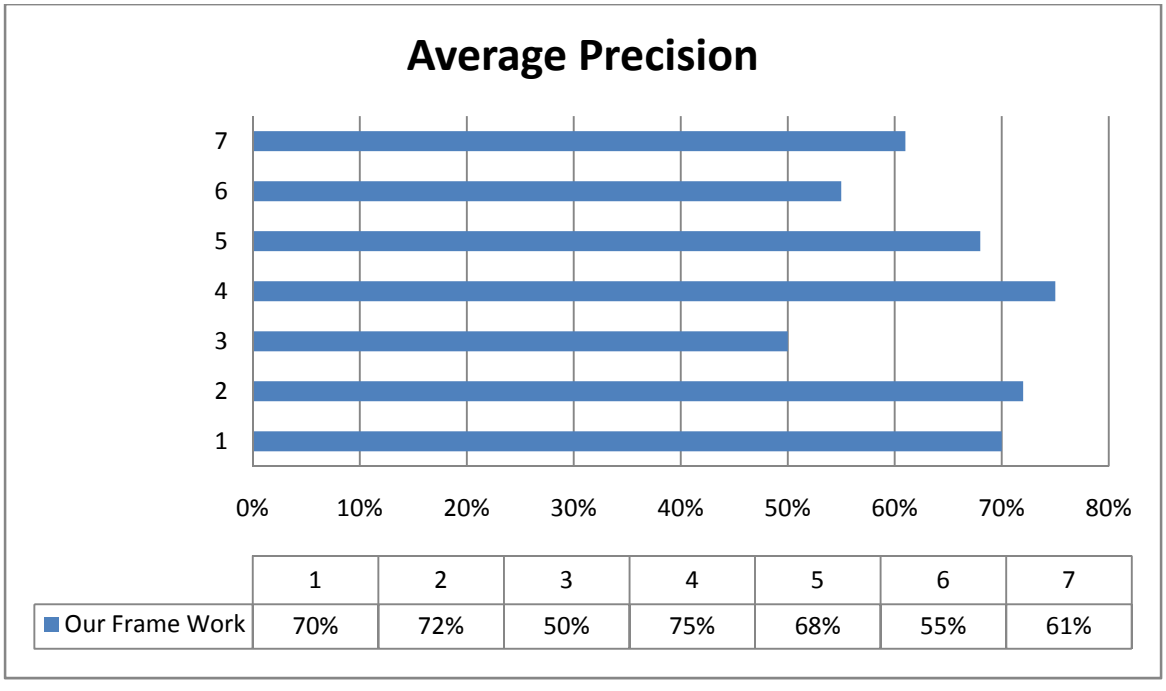
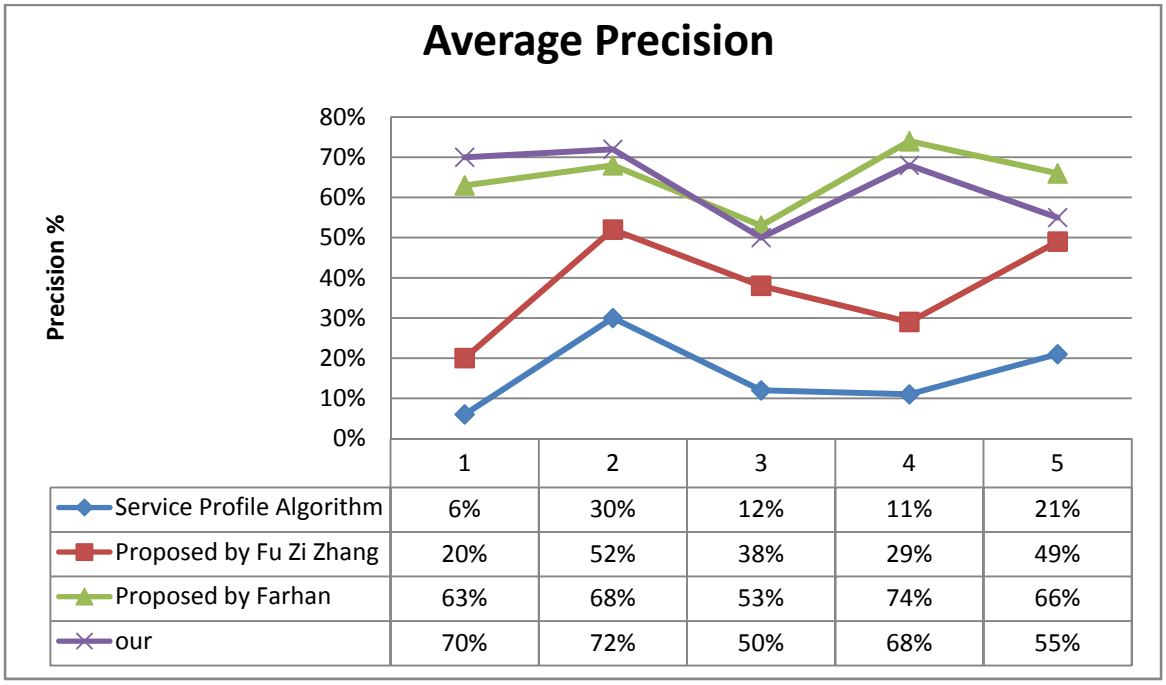


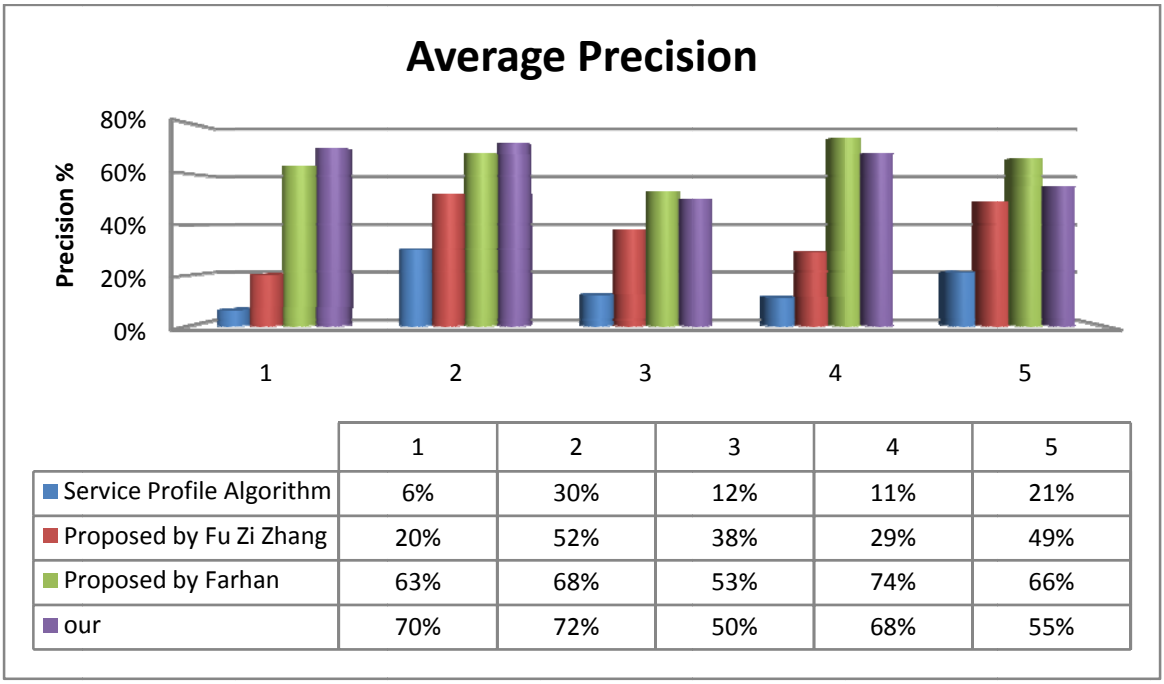
Figure 6-1Average Precision

We compared our framework with couple of other techniques. First technique is proposed by Farhan Hassan et al [25], second one is proposed by Fu Zhi Zhang et al. [29] and third one proposed by Lei Li and Ian Horricks [30]. Figure 6-2a shows the results. The graph shows better performance throughout the test results. At start the local database was populated. The range from 1-5 shows 5 datasets with gradual increase in database e.g. 1000, 1500, up to 3000 web services information in database. It can be seen that with increase in dataset precision of our framework is decreased as compared to framework by Farhan et al [25]. Still it lies at better percentage as compared to other two frameworks. Better results were achieved because of exact string matching in the search results listed. This may reduce the number of results being found but the precision percentage is high.



	1	2	3	4	5
Service Profile Algorithm	6%	30%	12%	11%	21%
Proposed by Fu Zi Zhang	20%	52%	38%	29%	49%
Proposed by Farhan	63%	68%	53%	74%	66%
our	70%	72%	50%	68%	55%

Figure 6-2a Average Precision Comparison



	1	2	3	4	5
Service Profile Algorithm	6%	30%	12%	11%	21%
Proposed by Fu Zi Zhang	20%	52%	38%	29%	49%
Proposed by Farhan	63%	68%	53%	74%	66%
our	70%	72%	50%	68%	55%

Figure 6-3b Average Precision Comparison

Also we performed comparison of Top-K Precision with "Woogle" by Xin Dong et al[5] . The formula we used is:-

Where k is total number of results retrieved and $retriverel_n$ is total number of relevant results.

In 6-3 Top K Precision is shown. Top 2 precision means out of all retrieved results only top two are considered to find precision. Similarly Top 5 means out of all retrieved results only first five are considered to find out the precision. From the graph we can clearly see that at Top 2 we have got precision=1, which means top 2 results are relevant all time. Out of Top 5 results we got less precision because few retrieved results out of Top 5 are not relevant all time. The graph descends as Top k is increased; still it shows better precision as compared to other frameworks shown in graph.

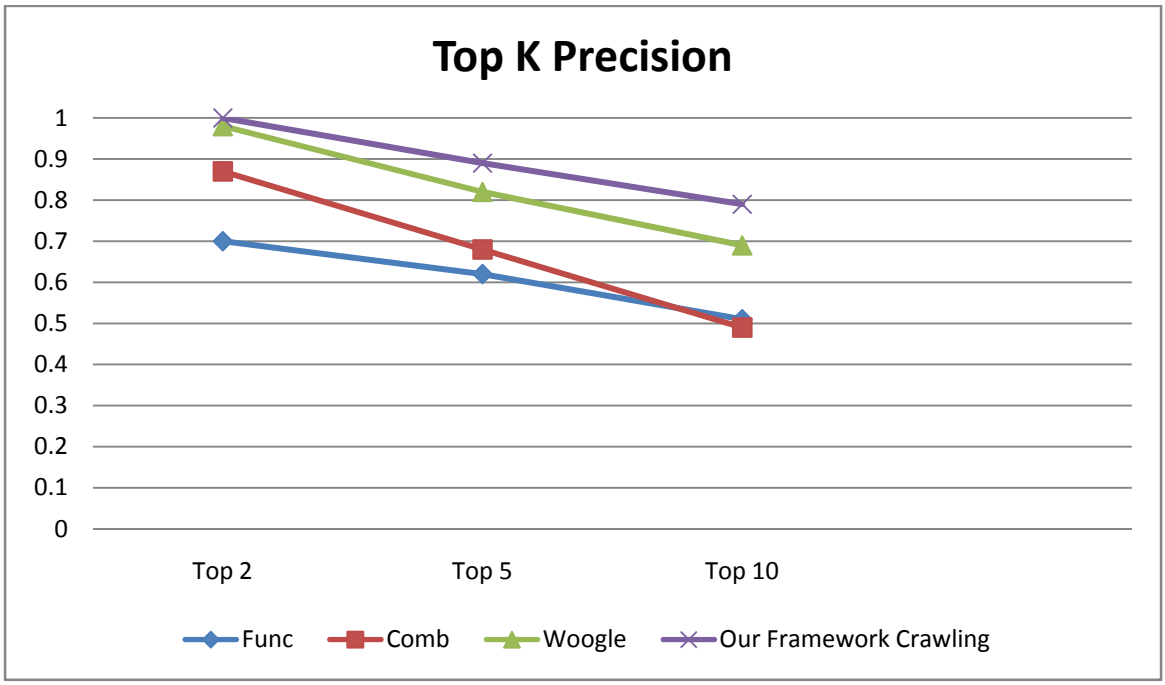


Figure 6-4 Comparison of Top K Precision with Other Techniques

6.3.2 Average Fall-out

Fallout is the proportion of all non-relevant services retrieved out of all retrieved services. We took various sets of services and for each set we made 25 readings and then computed an average for that set. Figure 6.4 shows results for Average Fallout.

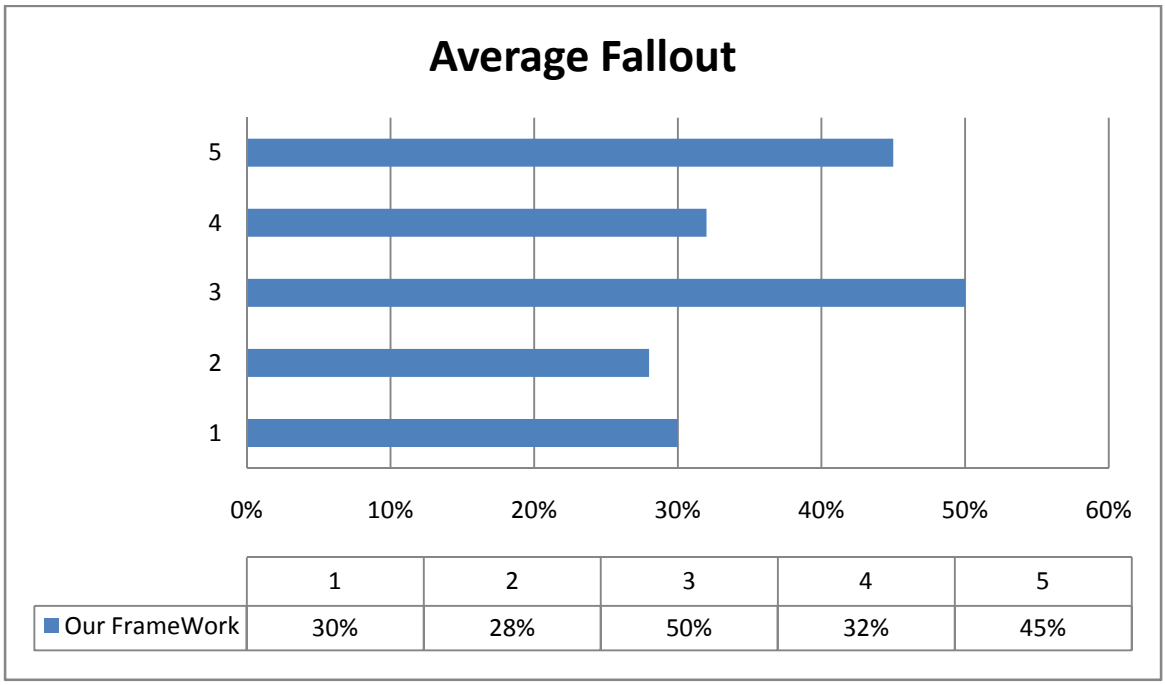
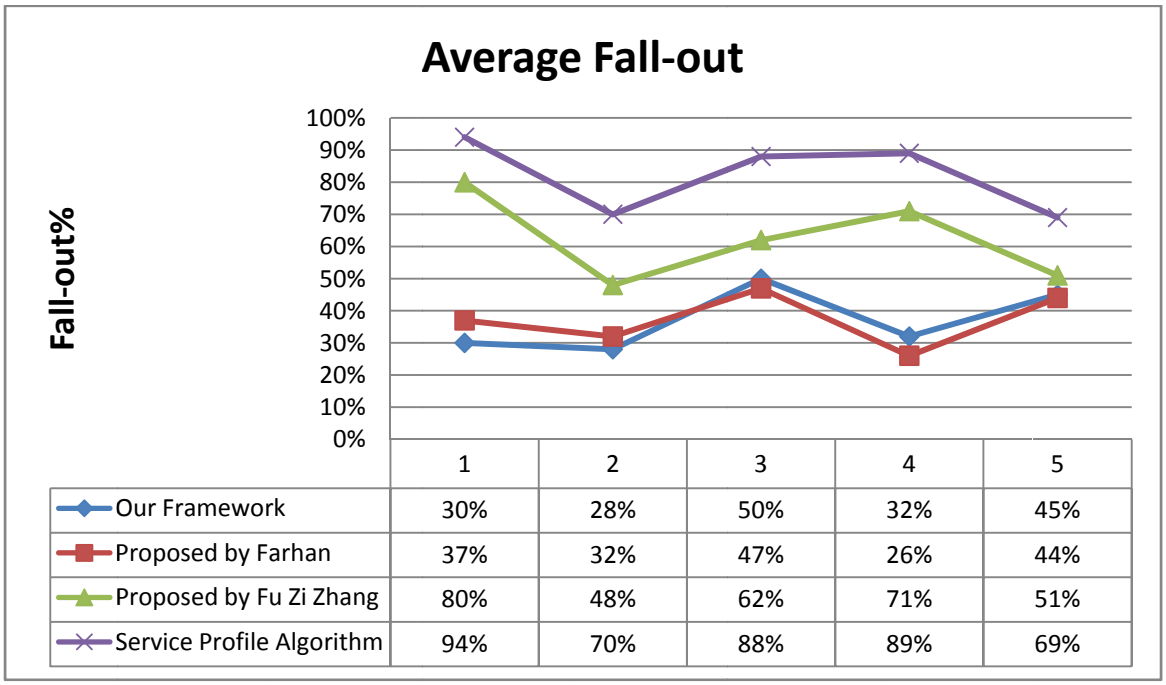


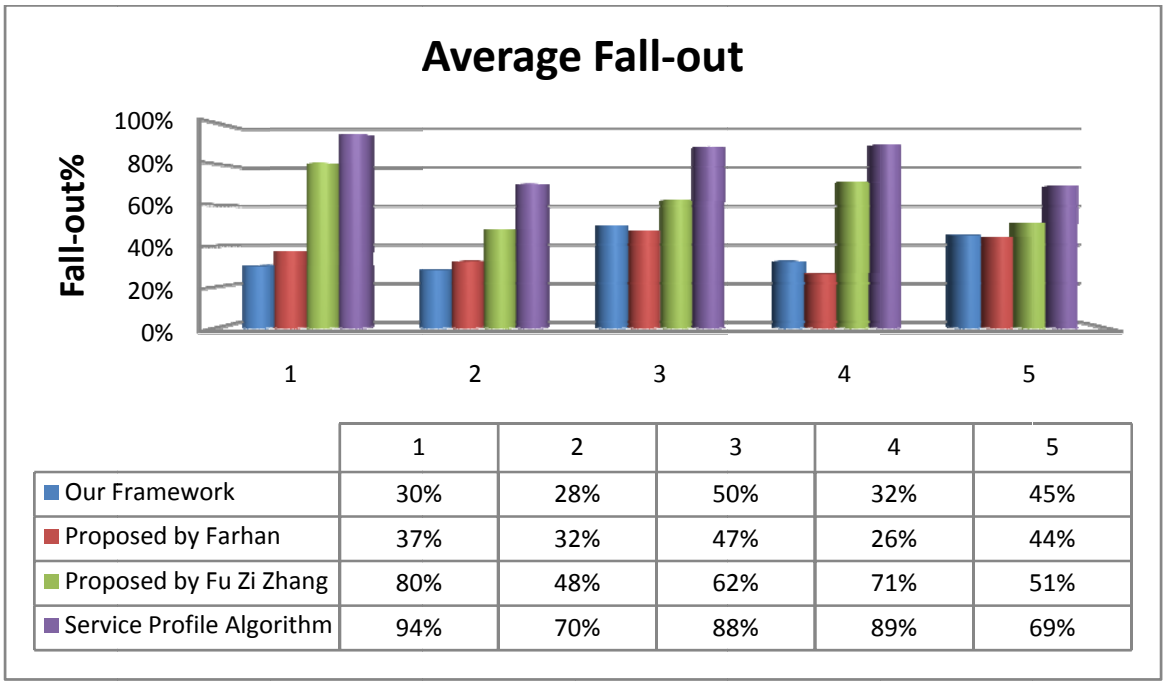
Figure 6-5Average Fall-out

We also compared these values with the other techniques. The graphs 6-5a and 6-5b show that proposed framework has a low fallout rate as compared to the other techniques. Low fall-out rate verified that the proposed framework discovered lesser services that were not relevant to the desired services. With increase in dataset the fall out is little increased but still its better as compared to the framework by Fu Zi Zhang [29] and Service Profile Algorithm [30].



	1	2	3	4	5
Our Framework	30%	28%	50%	32%	45%
Proposed by Farhan	37%	32%	47%	26%	44%
Proposed by Fu Zi Zhang	80%	48%	62%	71%	51%
Service Profile Algorithm	94%	70%	88%	89%	69%

Figure 6-6a Comparison of Average Fall-out with Other Techniques



	1	2	3	4	5
Our Framework	30%	28%	50%	32%	45%
Proposed by Farhan	37%	32%	47%	26%	44%
Proposed by Fu Zi Zhang	80%	48%	62%	71%	51%
Service Profile Algorithm	94%	70%	88%	89%	69%

Figure 6-7b Comparison of Average Fall-out with Other Techniques

6.3.3 Evaluation Time of Services

WSDL4J is used to parse the WSDL file of the web service. Once a service is selected the method it provides must be known. The evaluation time of web service was noted for various methods exposed by the web service. The web services were randomly evaluated and the timings were noted.

Figure 6-6 shows evaluation time of different services having 1 to 5 numbers of methods. This graph is at 100 kbps internet connection. From graph we can see that time is not dependent on increasing number of methods of web services. For example when number of methods of sample services is 1 the average time recorded is 1000 ms where as when this number was increased to 4 methods it took an average time of about 900 ms.

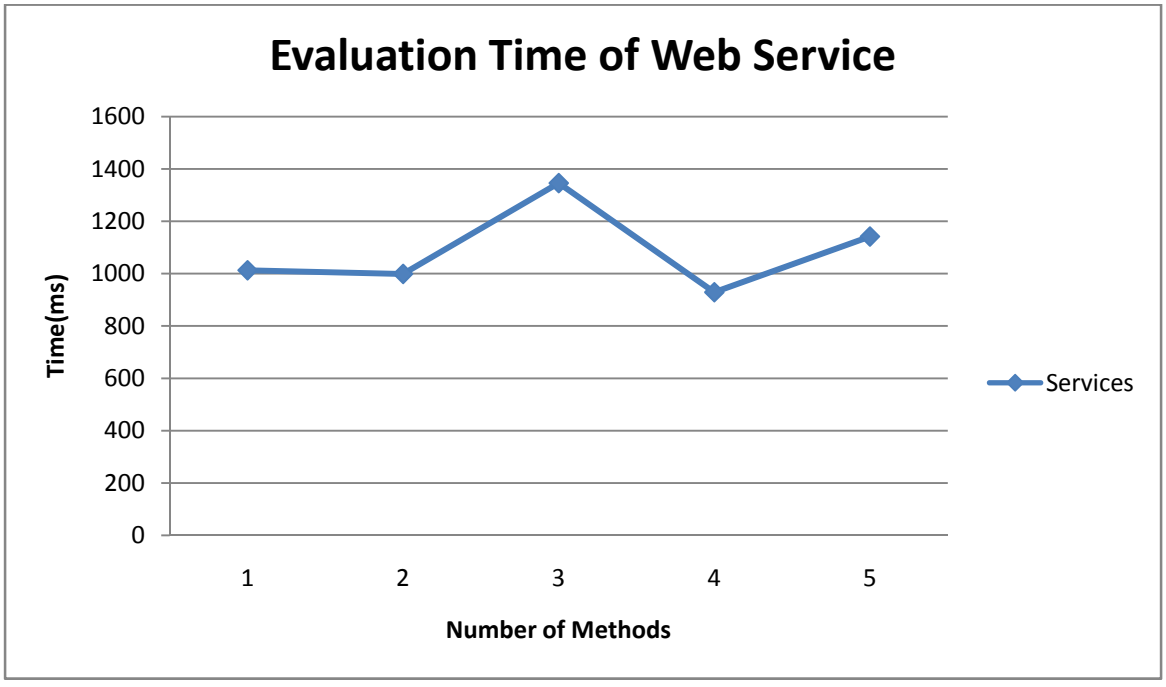


Figure 6-8Evaluation Time of Web Service

The comparison in 6-7 uses web services with number of methods from 1 to 5. It can be seen that evaluation of services doesn't depend on number of methods it contain. Time consumed by our

frameworks is very close to Farhans [25] framework. More over when the service is evaluated first time it will span more time than second time evaluation due to local cache information storage.

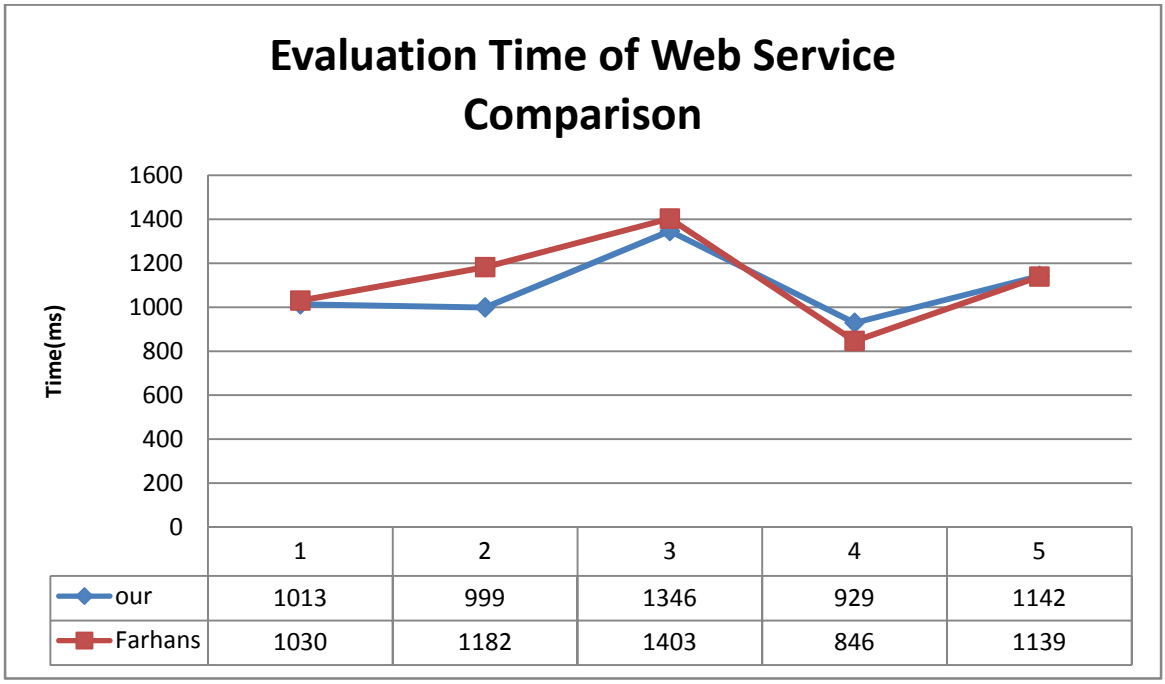


Figure 6-9Evaluation Time of Web Service Comparison

After getting basic information different web services were invoked in isolation to find out how much time an Xml message and SAAJ takes to invoke an individual service. We found that service invocation time is less than the time utilized for basic information capturing. Also using SAAJ it takes less time as compared to other methods used for service invocation. Figure 6.8 to 6.11 shows a bar graph for invocation of single web service at a time using SAAJ API at different rates of connection of internet.

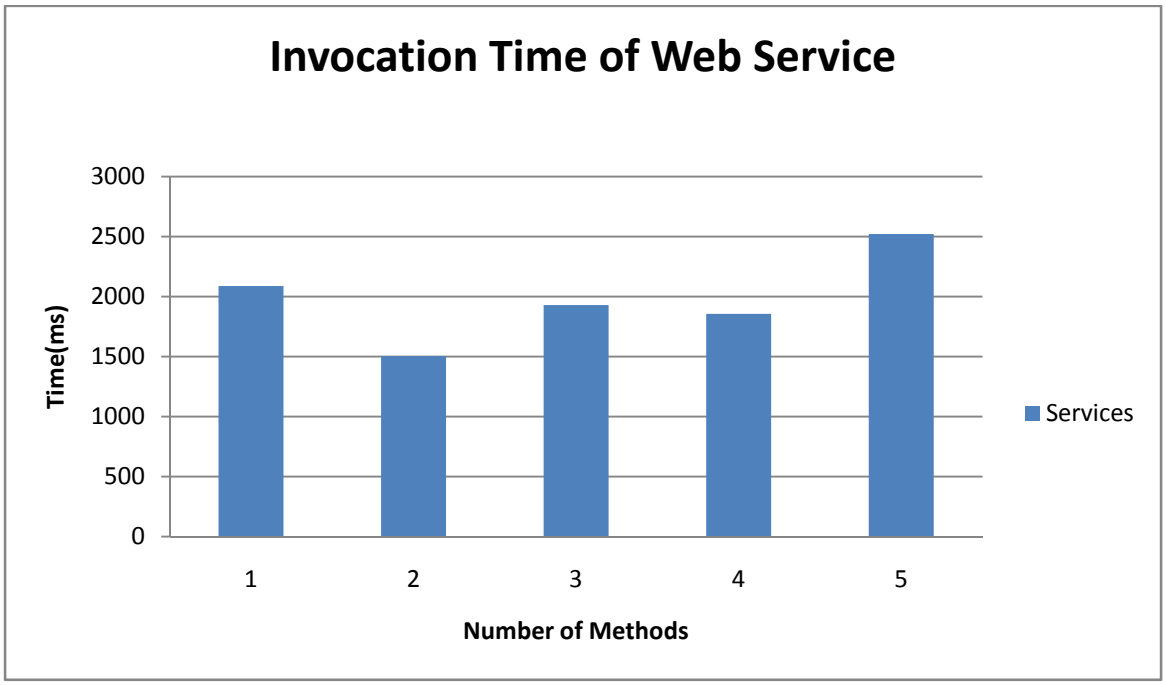


Figure 6-10Invocation Time of Web Service at 30kbps

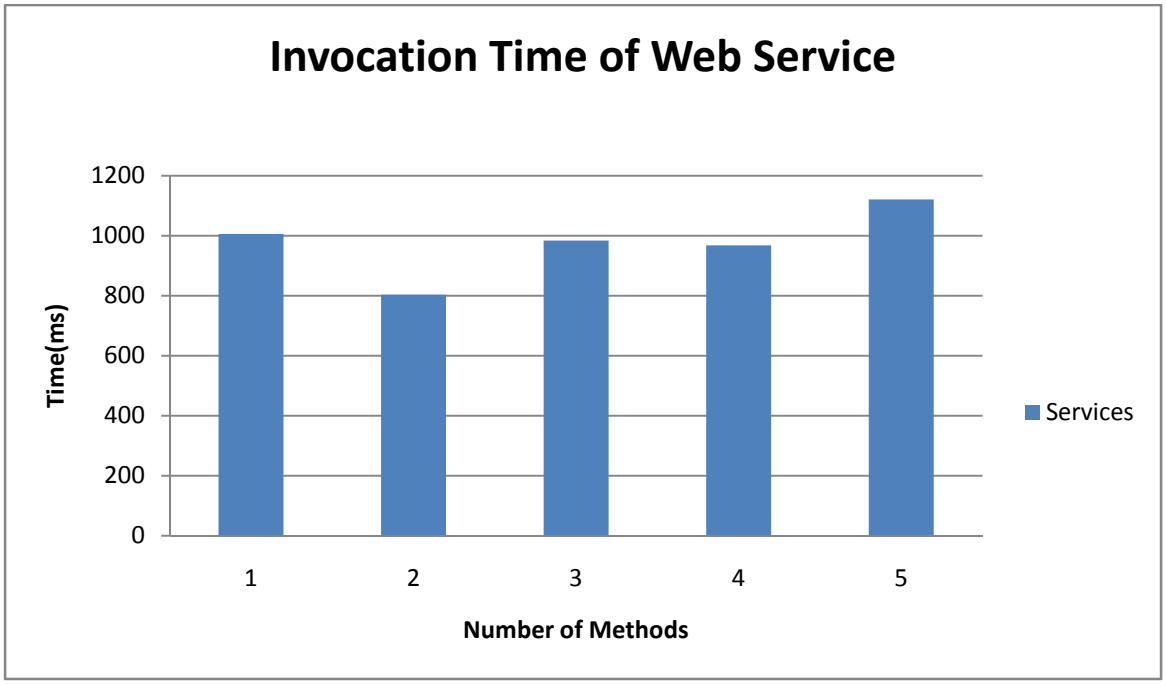


Figure 6-11Invocation Time of Web Service at 100kbps

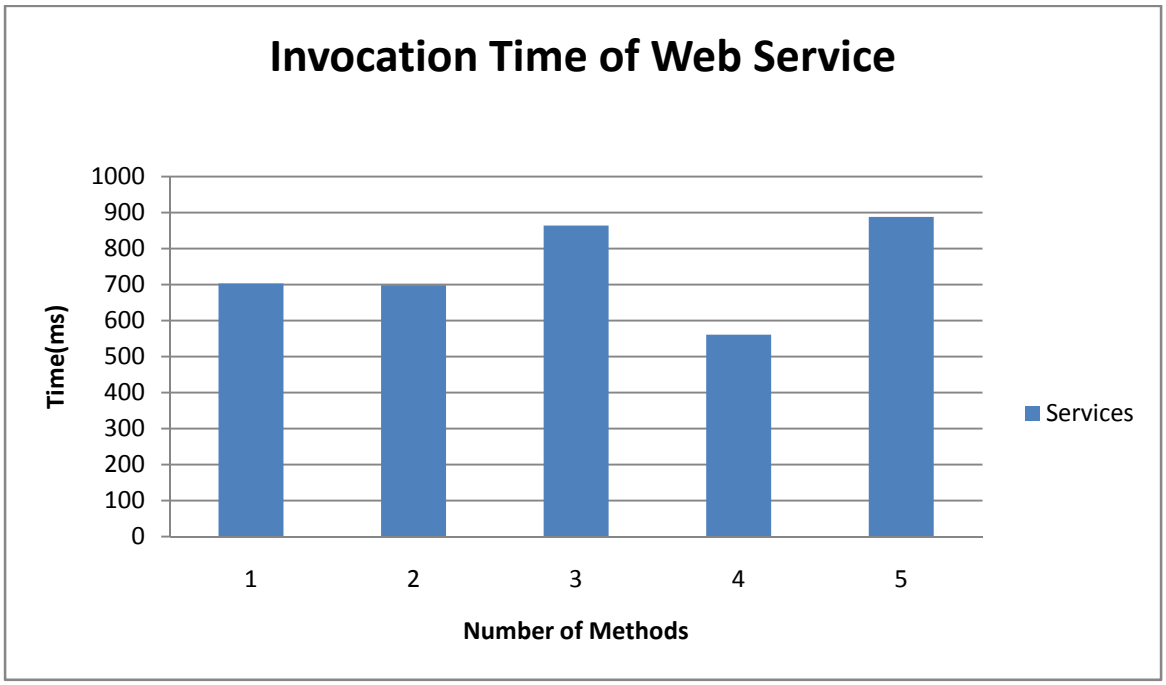


Figure 6-12Invocation Time of Web Service at 300kbps

In 6-11 comparison clearly shows that invocation time span is dependent on connection speed not on increasing number of methods in a service, when connection speed is good less time is taken by the application to invoke the service. For example a service with five methods took 2521 ms at internet connection of 30kbps while the same service took 888 ms when at connection of 300kbps.

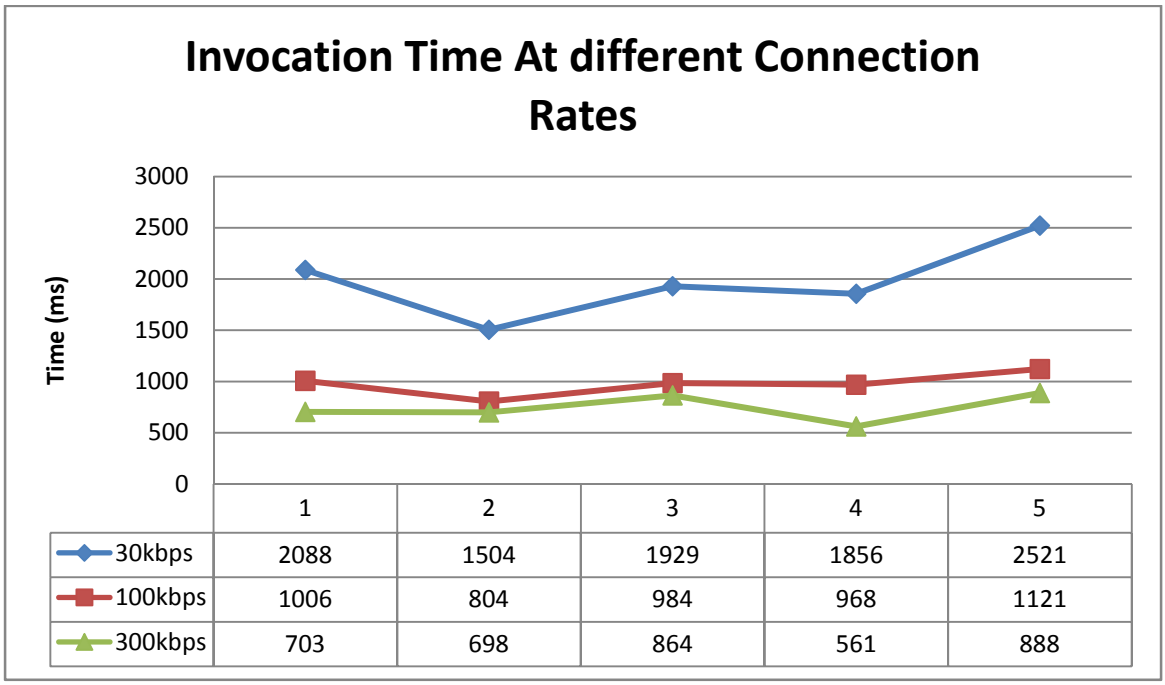


Figure 6-13 Invocation Time of Web Services Differet Connection Rates

6.3.4 Execution Time for Web Service Composition

The web services were randomly composed to get fruitful output. The time for the execution of composite web service was logged to make graphical analysis. This was repeated for web services composed of 2, 3, 4 and 5 services. The graphical analysis of execution time of web service composition is shown in Figure 6-12. The graph shows that time increases when number of composition services is increased. This time may also vary on different internet connection but gradual increase with increasing number of services will remain same.

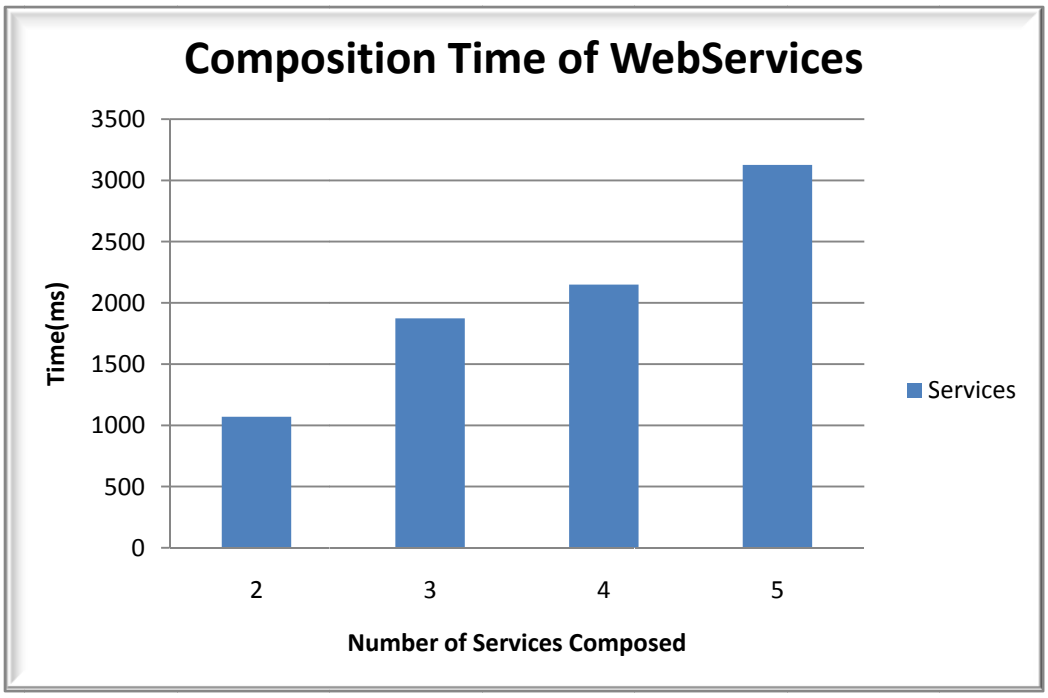


Figure 6-14 Execution Time for Web Service Composition

6.4 Comparison with other Framework

We compare our framework results with those of the framework described by Farhan Hassan Khan et, al. [5] From graph of evaluation of services it can be noticed that the information retrieval is not dependent on number of methods exposed. The time shown includes the information retrieval of methods that service provides and creating a dummy message that includes input parameters. Once the message is created and input is entered by the user. Less time is consumed for invocation of single service.

Composition time of services rises with increasing the number of services to be integrated. Though the individual service takes less time we added all the times consumed to find out final time span. Time taken by user for input after matchmaking decision is not included in the composition time graph. When we compared our composition time with another framework [25] we saw results shown in Figure 6-13. Our framework took less time for composition as compared to [25] this is due to use of SAAJ for exchange of message that made easy exchange of message for service invocation and hence composition.

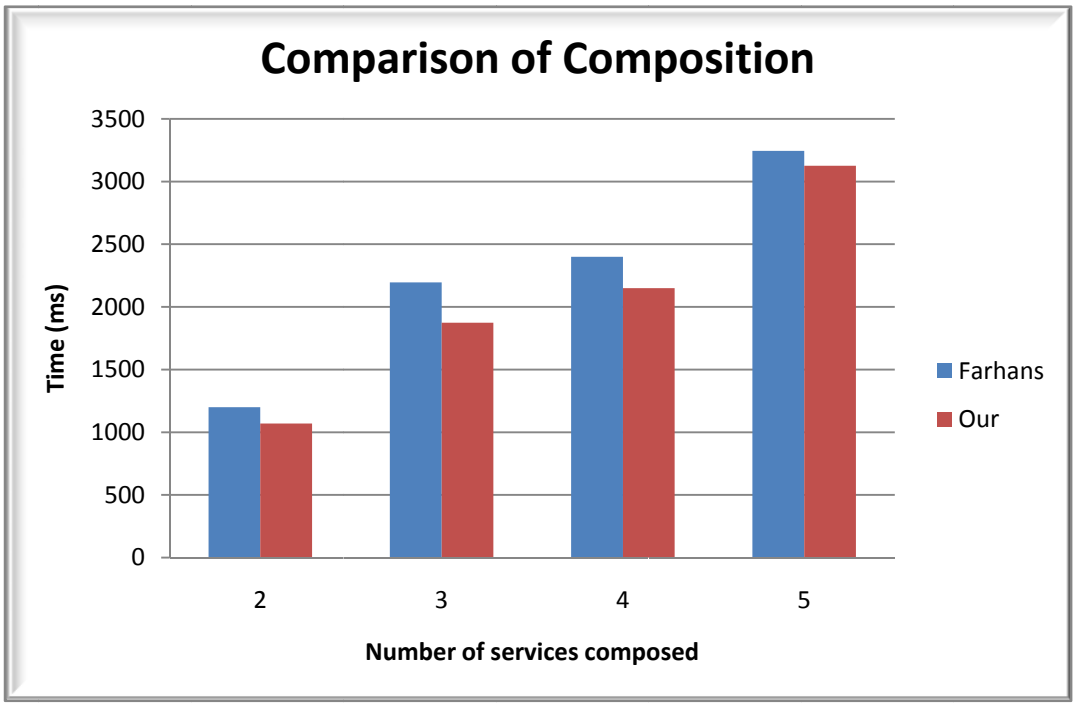


Figure 6-15 Composition Time Comparison with Other Technique

6.5 Static Dynamic and Statistical Factors

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

Table 6.4 Comparison of Static Factors

Factor	Our Framework	Proposed by Farhan et al.	Proposed by Faisal et al.
Regulatory	JAX-RPC ,WSDL4J, SAAJ, Castor, XSLT, Google Custom Search	UDDIv2, UDDIv3, SOAP, AXIS	UDDI v2, SOAP
Security	Yes	Yes	Yes

Table 6-5 Comparison of Dynamic Factors

Factor	Our Framework	Proposed by Farhan et al.	Proposed by Faisal et al.

Service Availability	Service may be hosted on multiple servers to improve 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	Multiple paths available which allows efficient network utilization	Only one path available. If this is congested, then the system fails.
Execution Duration	Normal	Normal	Normal

Table 6-6 Comparison of Statistical Factors

Factor	**Our Framework**	**Proposed by Farhan et al.**	**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	Service may be hosted on multiple servers to improve service reliability
Execution Reliability	Normal	Normal	Normal
Reputation	Average	Average	Average

Tables 6-4, 6-5 and 6-6 show different static dynamic and statistical factors. The tables show that our framework has similar factors as in Farhan's framework [25]. It can be observed that the framework provides average or normal quality in terms of reliability availability and execution. Also there are

multiple paths available to connect to internet and services used can be hosted on different service providers.

6.7 Chapter Summary

In this chapter we narrated different statistical and dynamic factors for web service discovery and composition comparisons. We evaluated the performance by formula for precision and fallout. We have made comparisons of service discovery precision, service evaluation time, service invocation time and composition time with other frameworks. Service invocation time is logged at different internet connections speeds. All comparisons are shown with help of line and column graphs.

Chapter 7

SUMMARY

7.1 Overview of Research

The area of Web Service Discovery (WSD) is a primary area of research today. It has pivotal importance for utilizing web services for personal or organizational needs. However the users of web service are yet facing a challenge to find the desired web service due to rapid growth of web services available on internet. There is a need of a mechanism to locate web services with issues covering performance, flexibility and reliability across multiple heterogeneous registries, which is a challenging task yet. Our proposed framework actively obtains user required web service by crawling among different repositories. One use of Web Services in computer applications is its automated Composition. We have tried to fix main dynamic composition problems. In previous chapters, we have provided the implementation of proposed algorithm and compared the performance with existing approaches and presented the results. The analysis shows that proposed approach has better results to some extent, than existing ones.

7.2 Achievements

The framework for service crawling using Google Custom Search API is flexible, scalable, efficient and reliable. In our approach the requester always gets up to date services, the retrieval is fast and efficient. Also the client is able to add more repositories from where the services can be crawled. Our framework covered the limitations of formal UDDI search by searching the whole page for user query. So user is not limited to give only the service name or category.

Also it covers the limitation of usual crawlers in which the crawling for service can be done on only one domain at a time. Though there are many web service crawlers available online but our framework is for those clients who want to crawl and invoke services from a desktop applications. To provide reliability we have made a database to store the crawled services. To prevent duplication the system only adds those services which are not already present in the

database. The updated information retrieval means the system checks weather the service is available at present or not. Also the results give better precision as compared to online engines for service search. Thus the proposed algorithm fix current issues of web services discovery.

We discussed main problems faced by dynamic service composition. Among which are transactional support and compositional correctness. We made the system to be flexible in terms of automation hence we include user involvement at few steps for example selection of service and matchmaking decision. We have used SAAJ API and XML messaging that helps invoke complex services without hectic job of finding data types of any input output parameter. The values are passed as XML messages and hence consume less space for data type's inspection and declaration. Matchmaking of input and output parameters guarantees compositional correctness and transactional support. Although at this stage we have only performed matchmaking of number of parameters latter we will try to find match of type of input and output parameters.

Due to matchmaking step we are able to provide compositional correctness and transactional support also little user intervention of service selection guarantees the reliability of required services to be composed and the framework shows flexibility towards general varying requirements of service composition.

- Maria Allauddin, Farooque Azam "*Service Crawling using Google Custom Search API*", International Journal of Computer Applications, Volume 34 - Number 7 , 2011 (Published).
- Maria Allauddin, Farooque Azam *"Dynamic Web Service Composition and Parameters Matchmaking"*, International Journal of Computer Applications, Volume 36 - Number 9, 2011 (Published).
- Maria Allauddin, Farooque Azam "*QOS Based Service Search and Composition Algorithm*", is accepted in 2012 International Conference on Network and Computer Science, sponsored by IACSIT (Accepted)

7.3 Limitations

In this thesis our discussion is around service search, execute-ability issues, data distribution, and matchmaking and QoS issues. Currently 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 work but still it can be further optimized and improved providing more functionality.

For example, in future the framework can be extended by making use of AI algorithms for discovery process. An Indexer discovery algorithm [31] can be merged with our framework. For example indexer enhances the search capability. Services stored in the database can be indexed or categorized as: Value Manipulation, Convertors, and Commerce etc. These categories will enhance the query results returned from local database. Indexer can also be used for Google API search results, as few links provide only specific services, results from those links can be categorized at initial search results returned. So that the user will have information of each result category and query will find the results based on the indexed structure. The returned results will be more accurate and enhance the search capability.

Ranking mechanism [32] can be added to index the links such that more trusted ones can be prioritized. According to [32] "http://ws.strikeiron.com/" has Page Rank=6 and Trust Score 29.92%, http://www.webservicex.com/ has Page Rank=5 and Trust Score=23.16% and http://ws.cdyne.com/ has Page Rank=2 and Trust Score=27.61% etc. The higher Page Rank and more Trust Score gives better Services. This information can be used to rank the results returned by Google API. This Ranking will provide more trustworthy Service Search and hence reliable composition.

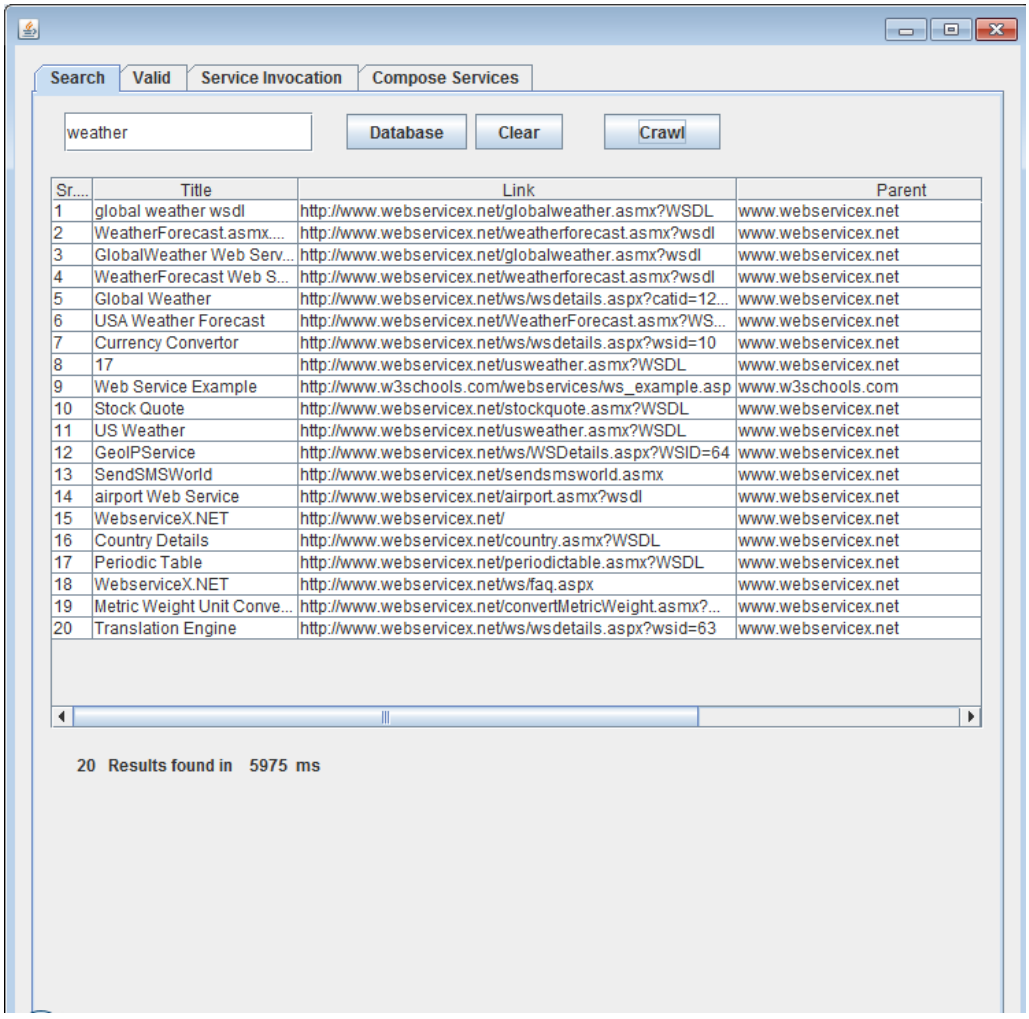
Although at this stage we have only performed matchmaking of number of parameters for composition, latter type of parameters can also be matched to make an accurate composition. When type of input output parameters will be known along with number of parameters more exact matchmaking will be performed and hence compositional correctness will be guaranteed at higher rate. At Matchmaking step an additional action will be needed that is to parse the type of first service's output according to next input parameter type.

APPENDIX A

User Manual

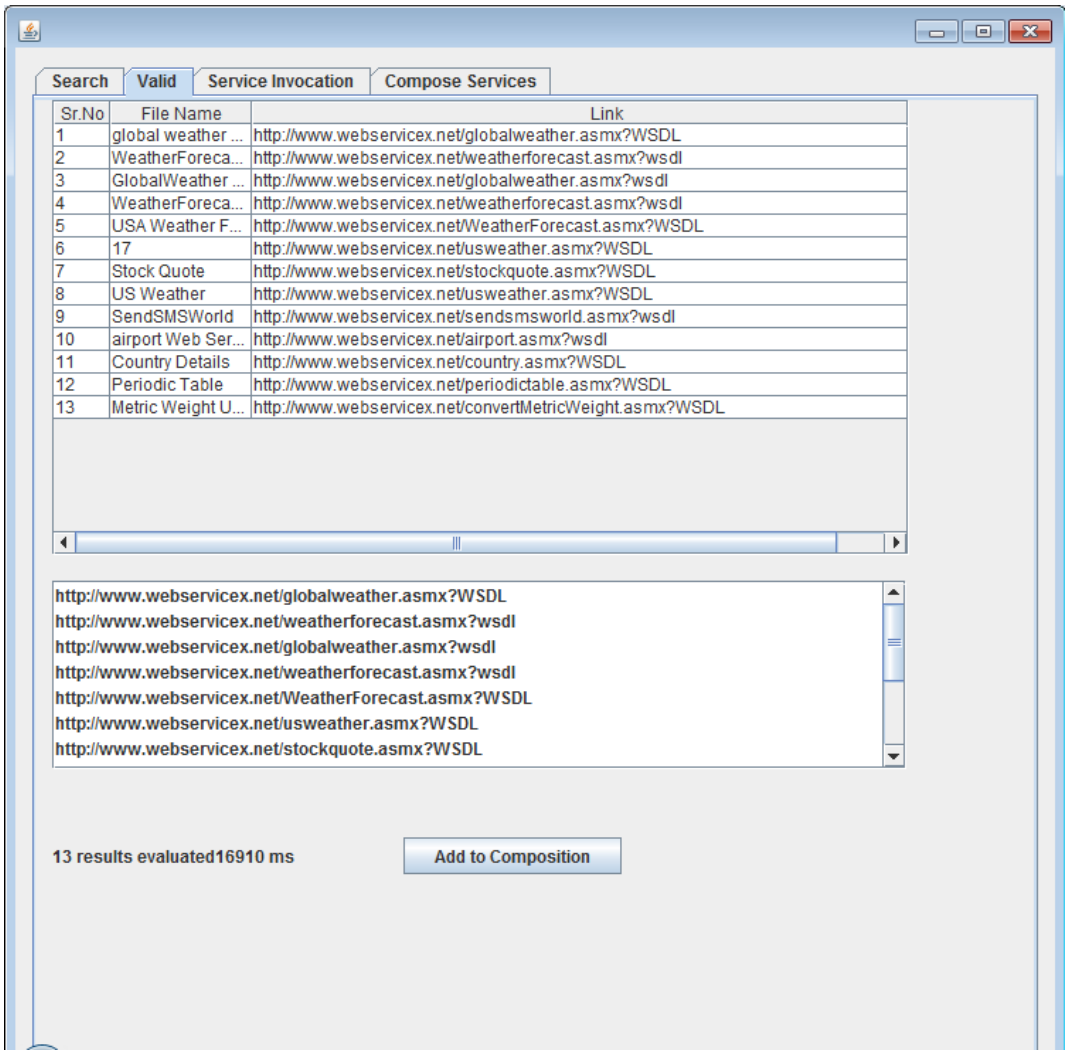
Main Screen

Double click the executable jar file to run the application. The main page of the application appears as seen below. Menu tabs are used to partition working steps. First tab is used for service search by crawling or from database.



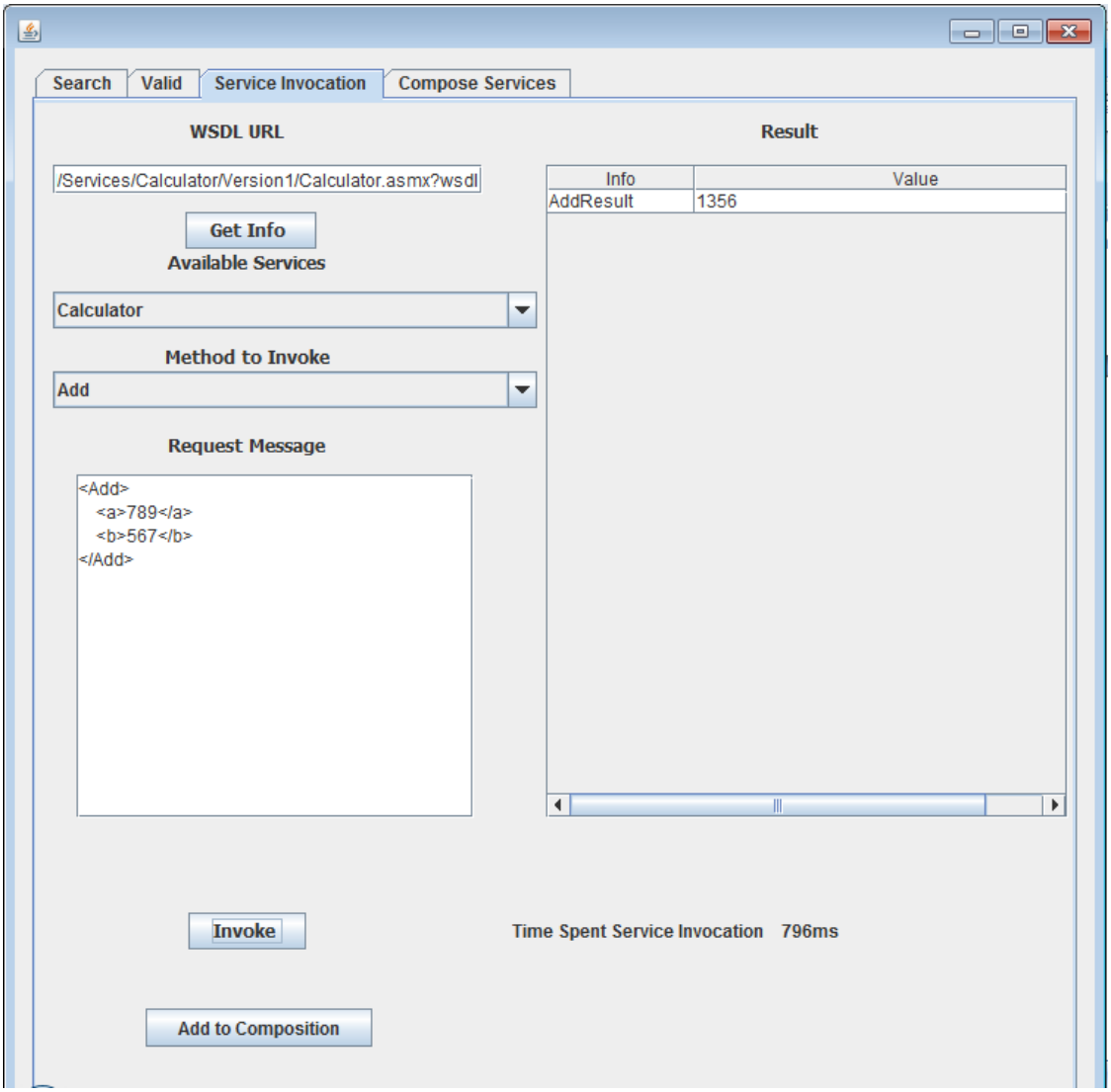
Valid Services

Valid panel performs availability check of the service. And checks weather the resulting document is a service or not.



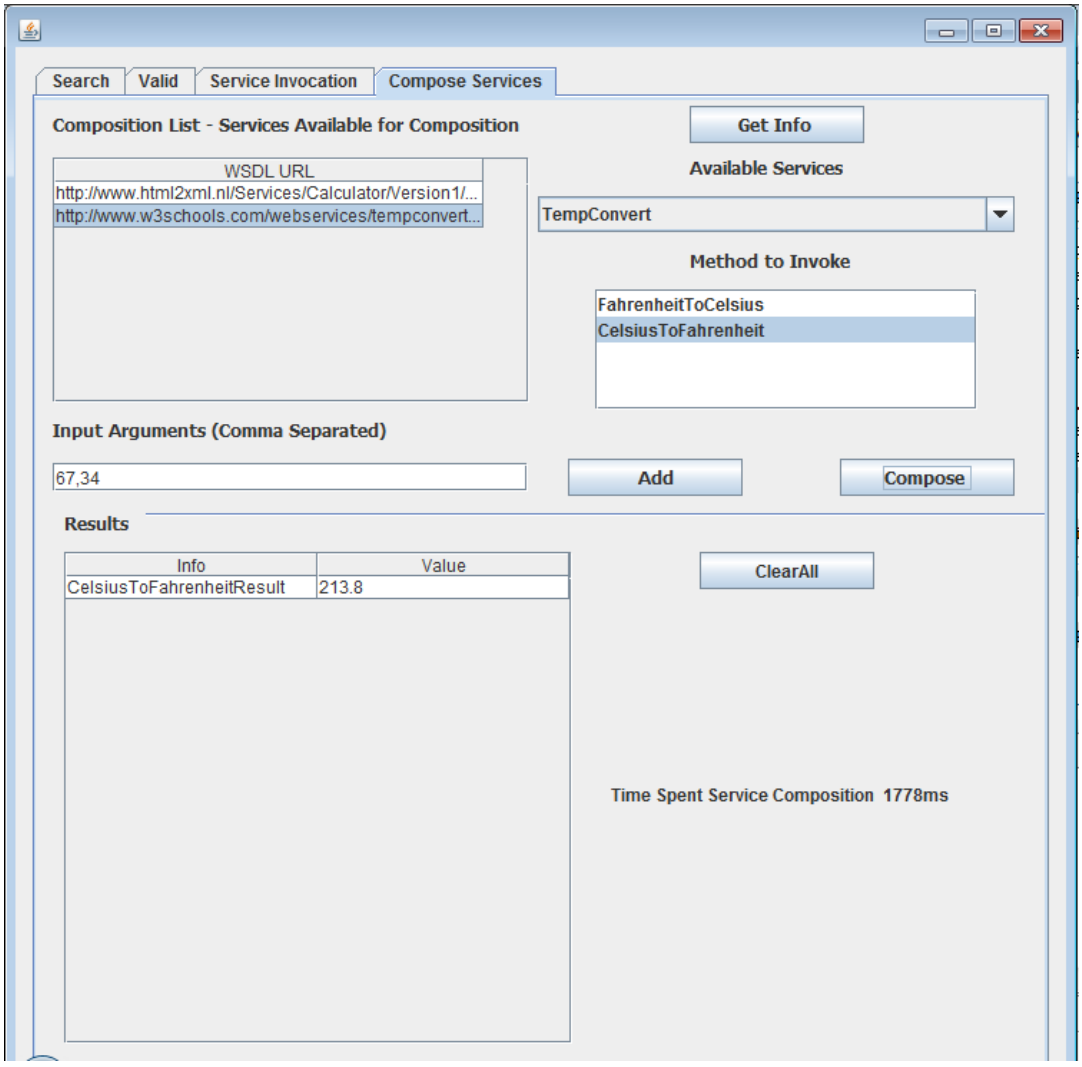
Service Invocation

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 arguments and click Invoke button. The web service is 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 by pressing Add button. When Compose button is pressed the services are executed in the order specified and the results are displayed in the results section.



REFERENCES

[1] Introduction to Web services technologies, http://ptgmedia.pearsoncmg.com/images/0131428985/samplechapter/0131428985_ch03.pdf

[2] Introduction to Web Services *by Hartwig Gunzer, March 2002, http://www.daimi.au.dk/~thomasr/Wearable/intro_to_web_services_wp.pdf*

[3] Holger Lausen and Thomas Haselwanter, "Finding Web Services" 2007.

[4] Mydhili K Nair, Dr. V.Gopalakrishna, "Look Before You Leap: A Survey of Web Service Discovery" International Journal of Computer Applications (0975 – 8887) Volume 7– No.5, September 2010

[5] Xin Dong Alon Halevy Jayant Madhavan Ema Nemes Jun Zhang, "Similarity Search for Web Services" Proceedings of the 30th VLDB Conference,Toronto, Canada, 2004

[6] Eyhab Al-Masri and Qusay H. Mahmoud, " Framework for Efficient Discovery of WebServices across Heterogeneous Registries", (NSERC), 2007

[7] T. Rajendran, Dr.P. Balasubramanie "An Optimal Agent-Based Architecture for Dynamic Web Service Discovery with QoS", 2010 Second International conference on Computing, Communication and Networking Technologies

[8] Eyhab Al-Masri and Qusay H. Mahmoud, "WSCE: A Crawler Engine for Large-Scale Discovery of Web Services" (ICWS 2007)

[9] Karastoyanova and A. Buchmann, "Components, Middleware and Web Services," Technische Universität Darmstadt, 2003

[10] E. Christensen, F. Curbera, G. Meredith, and S.Weerawarana, "Web Services Description Language (WSDL) 1.1," 2001.

[11] Liang-Jie Zhang, Qun Zhou, Tian Chao "A Dynamic Services Discovery Framework for Traversing Web Services Representation Chain", Proceedings of the IEEE International Conference on Web Services

[12] Paul Palathingal “Agent Approach for Service Discovery and Utilization”, Proceedings of the 37th Hawaii International Conference – 2004.

[13] Schahram Dustdar and Wolfgang Schreiner “A survey on web services composition”, *Int. J. Web and Grid Services, Vol. 1, No. 1, 2005*

[14] Freddy L´ecu´e, Eduardo Silva, and Lu´ıs Ferreira Pires, “A Framework for Dynamic Web Services Composition”.

[15] Faisal Mustafa, T. L. McCluskey “Dynamic Web Service Composition” 2009 International Conference on Computer Engineering and Technology

[16] 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

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

[18] Kazuto Nakamura Mikio Aoyama “Value-Based Dynamic Composition of Web Services” xiii asia pacific software engineering conference (APSEC'06)

[19] R. Jaya prakash, r. Vimal raja “evaluating web service composition methods with the help of a business application” R. JayaPrakash et. al. / International Journal of Engineering Science and Technology Vol. 2(7), 2010, 2931-2935

[20] Freddy L´ecu´e, Alain L´eger “Semantic Web Service Composition through Matchmaking of Domain”.

[21] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, and Cheng-Hung Chen ,“Dynamic Web Service Selection for Reliable Web Service Composition” Ieee Transactions On Services Computing, Vol. 1, No. 2, April-June 2008

[22] Liping Liu , Anfeng Liu , Ya Gao , “Improved Algorithm for Dynamic Web Services Composition”, The 9th International Conference for Young Computer Scientists

[23] Zhang Hai-tao, Gu Qing-rui, “A Dynamic Web Services Composition and Realization on the Base of Semantic”, 2010 IEEE

[24] Yujie Yao, Haopeng Chen, "A Rule-based Web Service Composition Approach",2010 Sixth International Conference on Autonomic and Autonomous Systems

[25] Farhan Hassan Khan, M.Younus Javed, Saba Bashir, "QoS Based Dynamic Web Services Composition & Execution", (IJCSIS) International Journal of Computer Science and Information Security,Vol. 7, No. 2, February 2010

[26] Kaouthar Boumhamdi, Zahi Jarir , "A Flexible Approach to Compose Web Services in Dynamic Environment", International Journal of Digital Society (IJDS), Volume 1, Issue 2, June 2010

[27] Jinghai Rao and Xiaomeng Su et al. *"A Survey of Automated Web Services Composition Methods"*

[28] Biplav Srivastava, Jana Koehler "*Web Service Composition - Current Solutions and Open Problems"*, ICAPS 2003

[29] Fu zhi Zhang, Yan Wang, Lin Wang "*A Web service discovery algorithm based on dynamic composition*" , SNPD '07 Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing - Volume 02 IEEE Computer Society Washington, DC, USA ©2007

[30] Lei Li, Ian Horrocks *"A software framework for matchmaking based on semantic web technology*" WWW '03 Proceedings of the 12th international conference on World Wide WebACM New York, NY, USA ©2003

[31] Saba Bashir, Farhan Hassan Khan, M.Younus Javed, "*Indexer Based Dynamic Web Services Discovery*", IJCSIS, Vol. 7 No. 2, February 2010, USA

[32] Gopinath Ganapathy and Chellammal Surianarayanan "*Identification of Candidate Services for Optimization of Web Service Composition*", Proceedings of the World Congress on Engineering 2010 Vol I WCE 2010, June 30 - July 2, 2010, London, U.K.