

Increasing Loose Coupling in SOA through Implementation

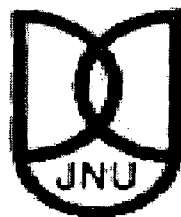
(Using Java EE and Web Services)

A dissertation submitted to the
School of Computer & Systems Sciences,
Jawaharlal Nehru University, New Delhi
in partial fulfillment of the requirement
for the award of the degree of

Master of Technology
in
Computer Sciences and Technology

by
Ali Alwasouf

Under supervision of
Prof. Pramila N.

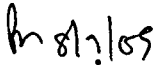


School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi-110067 – India
July 2009

Certificate

This is to certify that the dissertation entitled: **“Increasing Loose Coupling Through Implementation (Using Java EE and Web Services)”** being submitted by **Mr. Ali Alwasouf** to the **School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi-110067** in partial fulfillment of the requirements for the award of the degree of **Master Of Technology in Computer Sciences and Technology** is a record of bona fide work carried out by him under the supervision of Prof. Parimala N.

This work has not been submitted in part or in full to any university or institution for the award of any degree or diploma.


Supervisor

Prof. Parimala N.

Candidate

Mr. Ali Alwasouf




Dean

Prof. Parimala N.
School of Computer & Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

Being grateful to somebody can not be expressed by words or few lines, but at least by these words they will be able to know that I am really thankful to them.

"Whatever I'll achieve in my life, it will be signed by your imprint".

*My Supervisor
prof. Parimala N.*

"Words which can pay you back, have not been created yet".

*My parents
Prof. Ahmad Alwassouf
Amal Ahmad*

"You'll be my little toys forever".

Soamia & Alaa

"Hope that you'll be united soul forever".

Sulaf & Majd

"Your words were always my last resort".

Amjad Akel

"It is always easy to smile when I remember any of you".

*All my Syrian friends
(Muhanad, Jaffar, Majd, Jawdat, Nisressn, Raida, Rima)*

"I have never known the feeling of being lonely whenever I was with any of you".

*All friends who I meet at India
(Munir, Ali Barakat, Saleem, Shipra, Dipesh, Maher and Alia, Mandana, Shailendra)*

*Ali Alwasouf
22nd of June 2009
New Delhi-India*

Table of Contents

<u>CERTIFICATE.....</u>	<u>1</u>
<u>TABLE OF CONTENTS</u>	<u>3</u>
<u>LIST OF FIGURES.....</u>	<u>5</u>
<u>CHAPTER – 1 INTRODUCTION OF SOA.....</u>	<u>6</u>
1.1. INTRODUCTION	7
1.2. CHARACTERISTICS OF LARGE DISTRIBUTED SYSTEMS	7
1.3 SOA DEFINITIONS.....	8
1.4. SOA THE WHOLE PICTURE	10
1.4.1. SERVICES	10
1.4.1.1. Service definition	10
1.4.1.2 Service Attributes.....	11
1.4.1.3 Service Classification	13
1.4.2 ENTERPRISE SERVICE BUS (ESB)	13
1.4.3 LOOSE COUPLING	14
1.5. SERVICE LIFE CYCLE.....	15
1.6 COMMON QUESTIONS ABOUT SAO	17
1.6.1. ARE WEB SERVICES THE BEST TO IMPLEMENT SOA?.....	17
1.6.2. IS SOA BETTER THAN DISTRIBUTED OBJECTS?	17
1.6.3 IS SOA SOMETHING NEW?.....	18
1.6.4. DOES SOA INCREASE COMPLEXITY?.....	18
1.6.6. DOES SOA REPLACE OOP?.....	19
1.8. SUMMARY	20
<u>CHAPTER – 2 WEB SERVICES.....</u>	<u>21</u>
2.1. WEB SERVICE ARCHITECTURE.....	22
2.2. WEB SERVICES PROTOCOL STUCK	22
2.3. WEB SERVICES STANDARDS	23
2.3.1. WEB SERVICE DESCRIPTION LANGUAGE (WSDL)	23
2.3.1.1. The WSDL Document format	23
2.3.2 SIMPLE OBJECT ACCESS PROTOCOL (SOAP).....	25
2.3.3 UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI)	27
2.5. WEB SERVICES IMPLEMENTATION USING JAVA EE	29
2.5.1. BUILDING WEB SERVICES WITH JAX-WS	29
2.5.2.1. Coding the Service Provider	31
2.5.3.2 Coding the Service Client	32
2.6. SUMMARY	33

CHAPTER – 3 LOOSE COUPLING	34
3.1. INTRODUCTION	35
3.2. FORMS OF LOOSE COUPLING	36
3.2.1. ASYNCHRONOUS COMMUNICATIONS	36
3.2.2. HETEROGENEOUS DATA TYPES	37
3.2.3. MEDIATOR	37
3.2.4. WEAK TYPE CHECKING	38
3.3. SERVICE COUPLING	38
3.3.1. LOGIC-TO-CONTRACT COUPLING	39
3.3.2. CONTRACT-TO-LOGIC COUPLING	39
3.3.3. CONTRACT-TO-IMPLEMENTATION COUPLING	39
3.7. SUMMARY	40
CHAPTER – 4 CONTRACT-CONSUMER COUPLING PROBLEM(SUGGESTED SOLUTION)	41
4.1 CONSUMER-CONTRACT COUPLING PROBLEM	42
4.2. THE SOLUTION	44
4.2.1. GENERATING PROXY	46
4.2.2. UPDATING A SERVICE	46
4.3. EXAMPLE	47
4.4. GUI TOOL EXAMPLE	54
4.4.1. MAINFRAME CLASS	55
4.4.2. RESULTFRAME CLASS	65
CHAPTER – 5 CONCLUSION AND FUTURE PLAN	69
REFERENCES.....	72
BIBLIOGRAPHY	73
APPENDIX A.....	74

List of Figures

Figure 1-1 A Service Lifecycle under development.....	16
Figure 1-2 A service Lifecycle in production.....	16
Figure 2-1 Web Service.....	28
Figure 4-1 Tight coupling between and the consumer and the contract.....	43
Figure 4-2 The role of the suggesting tool.....	44
Figure 4-3 Extracting service out of WSDL file.....	45
Figure 4-4 generate proxy.....	46
Figure 4-5 Update a Service.....	47
Figure 4-6 The Main window of the tool.....	54
Figure 4-7 Result Window of the tool.....	55

Chapter - 1

Introduction of SOA

- ✓ *Introduction*
- ✓ *Characteristics of Large Distributed Systems*
- ✓ *SOA definitions*
- ✓ *SOA the Whole Picture*
 - *Services*
 - ❖ *Service definitions.*
 - ❖ *Service Attributes.*
 - ❖ *Service Classification.*
 - *Enterprise Service Bus.*
 - *Loose Coupling.*
- ✓ *Service Lifecycle*
- ✓ *Common Questions about SOA*
 - *Are Web Services The Best to Implement SOA?*
 - *Is SOA Better than Distributed Objects?*
 - *Is SOA something new?*
 - *Does SOA Increase Complexity?*
 - *Does SOA Replace OOP?*
- ✓ *What Is Next?*
- ✓ *Summary*

1.1. Introduction

The fast changes in the economic market have forced big changes in IT. The systems which were automated individual systems went towards being one large distributed system with different owners and users. Here, the need for software to control, test, debug and allot recourses and policies to those owners arose. Many solutions have been given to achieve previous concepts and meet the new challenges which arose with the new large systems.

The biggest challenge was how to keep those systems flexible as much as possible to follow the fast changes in the market and being able to maintain them easily. Many solutions have succeeded to achieve part of the goals of distributed systems but as these systems are growing up and becoming more and more complicated, they started to be difficult to scale and maintain. Here, we can say the need or motivation for service oriented architecture SOA became necessary. As a result, SOA came to deal with large distributed systems.

In order to keep such systems scalable and flexible as far as SOA can, SOA brought many existing concepts together like loose coupling. Loose coupling helps our systems to be flexible and scalable and have fault tolerance; but still achieving loosely coupled systems is a difficult mission for developers. In order to improve this concept many researches have suggested some solutions to decrease coupling in large systems as well as this work will be about.

In order to get better understanding for such concept we introduce characteristics of large distributed systems.

1.2. Characteristics of Large Distributed Systems

There are many characteristics of large distributed systems. The following four are the major common ones that will help us to get better understanding for the motivation behind SOA.

First, large distributed systems must deal with legacies, which means they must accept those systems which are already in use. We can't build everything from scratch; we have to accept the systems in use which needed a lot of money, effort and time to be like what they are. It is not acceptable to throw them away and start again to build new ones.

Second, by nature such systems are heterogeneous. They have different purposes, times of implementation and ages. Also, such systems may have been written in different programming languages by different programmers and developers.

Of course, heterogeneity prevents such systems to scale, there were many attempts in the past to solve such problems by using harmonization. The solution helped but was not a fully integrated one.

Third, they are complex. That is why maintenance is an impossible option in some cases, because finding the right place for modification is tough and it needs a lot of effort, although most of the large distributed systems have many strategies to debug, test and develop.

Fourth, large systems have a certain amount of redundancy. Thus, the issues of managing redundancy, determining which parts must be eliminated and which ones must be kept and be managed must be taken into account.

1.3 SOA Definitions

“In computing, the term Service-Oriented Architecture (SOA) expresses a perspective of software architecture that defines the use of services to support the requirements of software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation” [1].

Thus, SOA is expected to be applied for big distributed systems which have many users and many resources. Service forms the base for SOA, every order, allotment, query and so on, will be available as a service for the users. Let’s take a look at another definition of SOA:

“Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.”

We can see from the two definitions that both agree SOA is a Paradigm. SOA is not software or something we can buy from the market. It is a concept that already exists. By applying this concept to a specific case or situation leads us to make decisions those are appropriate for our circumstances. SOA came to control resources, allot it to many different owners and ensure that everything is going correctly with them. Thus, policies and security became a challenge for SOA.

“A Service Oriented Architecture (SOA) is a style of design that guides all aspects of creating and using business services throughout their lifecycle (from conception to retirement), as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes regardless of the operating systems or programming languages underlying those applications.”[1]

SOA uses services as the basic concept in its architecture; business services can help us out to decouple systems. Services help us to deliver solutions with high interoperability and loose coupling. SOA has appeared to keep large distributed systems flexible. Thus, the key of using SOA is helping people in their business and to deliver solutions on time. SOA has a lot to do with companies or organizations, roles, processes and so on.

Although, the definitions of SOA have common points but they still have many differences. Whatever the differences are the question would be here, why there are many differences among those definitions?

Actually, the point here is how do you look at SOA and what do you need from it. The people who feel insecure, find a sword is good for protection, the same sword can be seen good for sale for a merchant, while others can see it as a beautiful thing to show off by hanging on the home's wall and so on. We can see SOA from different points of view depending on our need.

Now, Let us summarize the previous points as a definition of SOA:

SOA is a paradigm or a concept of large distributed systems (not standalone applications), offers a good way to manage resources, allot it to different users and ensure secure systems. It depends on existing three major Components: services, loose coupling and interoperability. SOA tries to deliver systems with high interoperability and loose coupling using services and enterprise service bus balances between them by avoiding applying them blindly in such a way that will not increase complexity of the systems and remain flexible. So, SOA helps organizations to make decisions suitable for their own case or circumstances, but it's not a silver bullet which can solve everything. There are many ways to implement SOA, one of the best is Web Services but it's not the only one.
[1, 2]

1.4. SOA the Whole Picture

There are three concepts on which SOA is based services, interoperability and loose coupling.

*Service: “is a piece of self contained business functionality” [1].

The service can be *simple*, composed which is composed from more than one simple service.

There are many classifications for services we can mention right here. We can classify them up to their work or contents or the systems they are connecting with.

*ESB: Enterprise Service Bus or ESB is the infrastructure that provides high interoperability among distributed systems for services. Also ESB offers some kind of loose coupling for the systems.

*Loose Coupling: “is a concept that defines how to reduce the dependencies in the systems” [1]. It has many forms like data types, infrastructure and so on.

1.4.1. Services

1.4.1.1. Service Definition

“Service is a business function” [1], the primary goal of a service is to represent a natural step of business functionality. In other words it must represent a “self-contained business functionality corresponding to real word activity” [1].

Technically a service is an “interface that can receive and respond to multiple requests” [2]. Each service has its signature and contract. Signature is which define the service by its name and the parameters so it must be unique and be used to call the service while the contract is “all the information that must be known by a specific consumer to be able to call the service” [1].

It must meet the primary features as being stateless, discoverable, reusable and many others.

We can summarize service definition as follows:

A service is a self contained interface that represents business functionality. Each service has its own signature that can be used to call it, and its contract that it contains all the information

needed to call it by a specific consumer. The service must achieve the primary attributes as being stateless and discoverable, and many other attributes [1, 2].

This definition has been deliberately made short by ignoring many attributes, which if included could have made the definition longer and confusing. So, we can list the whole attributes of the service below to explore the term ‘*service*’ in the best way.

1.4.1.2 Service Attributes

We give an overview of service attributes here and go through it in detail whenever we need to know more about any of them. The list of all attributes of a service can be given as:

1st Self Contained: All definitions of SOA agree that it’s a design goal that services are self contained, that mean they must be independent and autonomous. However, always there are some dependencies that can not be avoided.

2nd Coarse Grained: Usually, in the services it is better to have one service call instead of multiple ones to complete the operation of a task. The reason behind this is the cost we pay for the abstraction (which aims to hide the implementation details) is slower running times. Additional advantage for coarse grained services is that it helps to separate the internal data structure of a services provider from its external interface, because having one service for each access to a service attribute would result in distributed objects which increase the dependencies among distributed systems.

One problem of coarse granularity is that processing a lot of data takes time while the data may be unneeded for the customer. As a result, time is being wasted.

3rd Visible/ Discoverable: To be able to call a service, we have to know if the service exists. There is a public place where we can search for the service. As an example, the repository is the place in Web Services.

4th Stateless: being a stateless service is a kind of ideal thought. Always there is some state data, which makes state management of the service one of the most important issues in SOA (*note:* lot of complexity is involved in this issue).

5th Idempotent: The meaning of this term in a few words is the “ability to redo an operation if you are not sure whether it was completed or not” [1]. The service must always be idempotent. But, of course still there are some cases in which it is difficult to achieve that.

6th Reusable: Of course, reusability must be a goal but not a rule. It’s good to achieve it, but still it has some cost. As an example, high reusability decreases the performance of the service.

7th Composable: The service must be able to use or call the other services: actually this discussion of composing and decomposing leads to business process modeling.

8th Technical: The service can be used for exchanging technical data, “in this case it’s not representing self contained business functionality” [1].

9th Quality of Service (QoS) and Service Level Agreement (SLA)-Capable: for performance and reliability reasons, we have to specify some non-functional attributes such as QoS and SLA which deal with previous issues.

10th Pre- and Post-Conditions: the pre and post conditions help us to specify the semantic behavior of services. The pre-conditions define the specific service rules the consumer has to meet when calling the service (constraints). Post-conditions define the output when the service runs successfully (benefits).

11th Vendor-Diverse & 12th Interoperable: Actually these two attributes are SOA attributes more than being service attributes. Even though, we see most of the books and articles specify them as attributes of a Service.

13th Implemented as Web Services: This attribute can be considered to be an option to implement SOA more than being an attribute for the service. Web services are one option to implement SOA and there are many other options as well.

1.4.1.3 Service Classification

Services can have different attributes. They can differ within the same system or landscape; they serve different purposes and play very different roles. Some of them read data, some write and others read and write. So, we can classify services in many ways. The most common way to classify services is Fundamental Service Classification which classifies services into:

* ***Basic Services:*** presents a basic business functionality, which means there is no way or benefit to divide it into one more service. They are “usually stateless and short-term running” [1].

* ***Composed Services:*** composing services out of existing services is called orchestration and this name came to express that we are trying to collect services together as orchestra to perform one melody. Composed services are the higher level of basic ones by comparing them with basic services they share the same points as they are both stateless, short-term running but they differ by their tasks and concepts.

* ***Process Services:*** represent a long-term workflows or business processes. In other words, in business point of view it represents a macro flow which it can be interruptible by a human.

1.4.2 Enterprise Service Bus (ESB)

Some organizations have faced a problem in their system landscape called “mess” or “lack of interoperability”. Actually, the organizations had a mess of systems and protocols and that is why they had to create an individual solution for each kind of connection. At the beginning the solution has been called the magic bus by definition, the magic bus is “a piece of software that reduces the number of connections and interfaces in our system” [1].

Magic bus needed only one connection for each system and interface. But the solution didn't live for a long time, after they switched to the magic bus and their systems began to grow easily, a new problem happened to their system. It turned out that with this technique nobody could understand the dependencies among the systems. Thus, modifying one of them may cause some problems in the others. We can realize that high interoperability must be accompanied by a well defined structures, architecture and process; otherwise it will cause problems more than helping us to solve ours.

After the magic bus failed to solve the problems for enterprises, something new has appeared to take the benefits of magic bus advantages and develop the idea of finding some balance between the interoperability and well-defined structure.

It is ESB or Enterprise service bus which is a “technical part of SOA that enables high interoperability” [1] and which became the infrastructure of SOA. The main reason for depending SOA on ESB is that ESB enables us to call services among heterogeneous systems, data transformation, routing, dealing with security and reliability, service management, monitoring and logging.

ESBs differ widely from the technical point of view and conceptual point of view. We can classify ESB into the type of connection they implement (point-to-point ESB versus mediator ESB) or regarding where the responsibility of an ESB begins for each of consumer or provider (protocol driven versus API driven ESBs).

1.4.3 Loose Coupling

The term “Coupling” is involved in every part of IT world. IT vocabulary introduces coupling in simple words “anything that connects has coupling and coupled things can form dependencies on each other” [2]. However, we can qualify coupling with loose or tight.

“The term couple itself implies that two of something exist and have a relationship” [2]. The most common way of explaining coupling is to compare it to dependencies. As we saw in ESB section above, ESB came to reduce dependencies between systems. Also, eliminating objects in object oriented distributed systems and replace them with services decoupled the system components and reduced dependencies.

Everything is revolving to somehow reduce dependencies among systems as well as among the components of each system. The dream of being able to update system components and have fault tolerance in such systems in high percentage is considered as ideal thought. Having loosely coupled systems or components is a relative thing; nobody can claim that his system is loosely coupled one hundred percent. For example, web services are considered as loosely coupled services although web services can not guarantee loose coupled systems, they have many types of tight coupling forms. Services have different forms of coupling involved among provider and the contract from one side and the consumer and the contract from the other side.

Coupling can be measured in many ways using many factors. For example, the ability for updating without effecting the other components in the system.

Web services is the most suitable example to study coupling, therefore we will study forms of loose coupling in web services later on in third chapter in order to solve a form of tight coupling between the service consumer and the service contract in the fourth chapter.

1.5. Service Life Cycle

Service is a piece of software, so the same lifecycle for software can be applied here, of course there are some differences between lifecycles of software under development and the software under maintenance. Service has the usual software lifecycle which at its core consists of phases of design, implementation, integration and running or in other words bringing into production (waterfall approach), see Figure 1-1. Service development usually should be an iterative process; a service is part of a more general business process. Thus, any modifications of a service's design or implementation might impact other systems. For this reason, we must think when it is appropriate to modify a service. If a modification isn't backward compatible, a new service (or a new service version) is necessary. A new service version is typically considered to be, technically, a new service. A service under production which is doing some critical business for the company has different rules to be modified. Because modification became more critical than before, so we have to search for the suitable moment to withdraw or modify the service. The best way to modify a service is to issue a new version of the service, leave the existing one under production and try to identify a new version. Of course, there is an exception for this rule whenever we find a bug in the

existing service, then we have to stop it and modify it, see Figure 1-2. It is easy to see the effect of coupling for updating and modifying the service under production. If the service are basic one, then some other services are calling it (basic services never work alone), changing the service contract will force other changes in the calling services. Thus, loosely coupled services do not mean they are loosely coupled for modifying or updating.

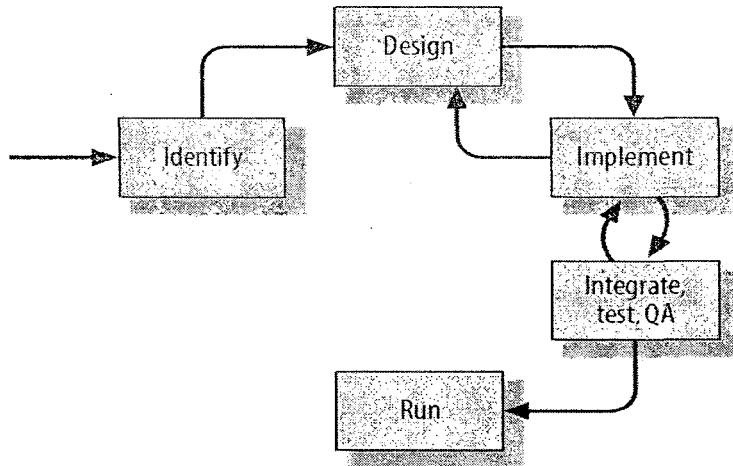


Figure 1-1 A Service Lifecycle under development.

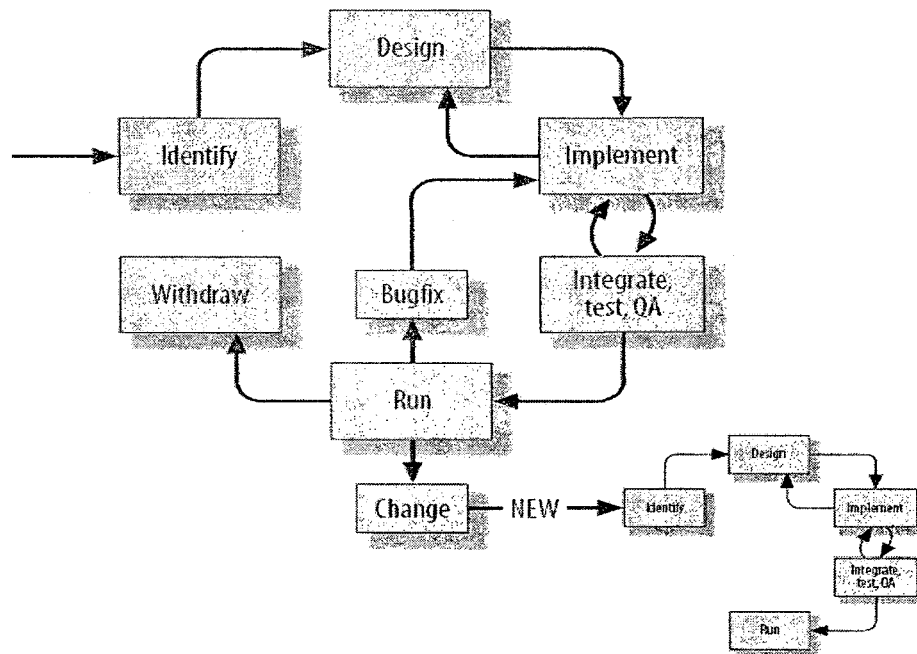


Figure 1-2 A service Lifecycle in production.

Updating composed service is easier because it will affect nothing in the basic services that are called within it, but still that will impact the other services which they are calling it. However, loosely coupled services for updating the service under production are required whether they are basic, composed or process services. Still nobody can claim that we can design a service which can be updated in any case while it is under production without making the necessary changes in the calling service or the consumer.

1.6 Common Questions about SAO

1.6.1. Are Web Services the Best to Implement SOA?

Web services are one option to implement SOA based on three major components -Consumer, Provider and Repository. Web Services is wide-spread because it introduced many solutions better than other distributed systems technique. We have to know that “Web services alone will not solve any of our problems because it introduces some problems”. For example Web Services can not guarantee interoperability and loose coupling. So we can say that we have to use Web Services as an option to apply SOA but not as the main solution for our problems. For this reason we should use Web Services only when we feel that it can play a good role in our case. Web services can't guarantee loose coupling but still they are loosely coupled services when they compared with other technologies. They are interoperable, language independent as well as they are using XML to exchange the data which is considered as the most common way to decouple systems and exchange data with high interoperability. Interoperability decouples the dependency between the software and the landscape or the system.

1.6.2. Is SOA Better than Distributed Objects?

There have been many different options to deal with distributed systems like RMI, CORBA and DCOM. CORBA was to use distributed objects. CORBA enables remote access to objects of external systems. It was fine grained kind of interface to remote systems. The difference between

CORBA and SOA is CORBA “has one business object model spanning distributed systems” [1] while “SOA is the exact opposite of the concept of distributed objects” [1], so data can be exchanged between different systems and each one has its local copy with its local methods and procedures. As a consequence, distributed objects didn’t scale in practice while SOA approach decouples the systems and let them to scale. Also, having one object to present the business functionality will result in more tight coupled components.

1.6.3 Is SOA Something New?

SOA doesn’t introduce any new concept. As been mentioned before it brings together existing concepts and practices for a specific set of requirements.

Those existing concepts have been applied to large distributed systems individually. So, the question here we can ask why we are pretending that we are doing something new by introducing SOA? Or in other words, what is new about SOA to get all that attention? The answer simply is, SOA has gathered all those existing concepts and made them clear as never before, but it is still not enough reason to have all this reputation. The main reason is that SOA has introduced some improvements through web Services, the most important one being the new standards for interoperability. In addition, those existing systems were fighting with heterogeneity, meanwhile SOA accepted it and dealt with it, this issue was never seen before.

1.6.4 Does SOA Increase Complexity?

SOA is a concept of large distributed systems. As we know, large systems are more complex than simple automated individual systems because distributed systems have different owners, collaboration and loose coupling are also required.

Debugging and testing need more effort in such systems, where there is nothing under central control. For these reasons, SOA can not be an end in itself. We have to apply it whenever we need it. It doesn’t help in case of simple systems, and at the same time applying it to distributed systems

can be so useful. Thus, in such a discussion, the next question that can be asked is where SOA is appropriate? And where it is not?

Of course the previous discussion clarifies the idea that SOA has some limitations, the requirements might be so high, SOA in some cases can lead to big problems, more than it solves. Database replication, mass data processing and local client as examples are not appropriate to apply SOA. Here, we can find out that the question must not be what is suitable for SOA, but the question is whether the solution we are taking is suitable for our problem and requirements or not.

1.6.6. Does SOA Replace OOP?

This question is too confused for many people who are trying to find out what is SOA. Actually, this discussion is helpless to get better knowledge about SOA.

OOP is a concept of standalone applications that is written in the same programming language, while web services provide a way to communicate among services which may be written in different programming languages.

Therefore, we can realize that there is no sense of comparing the two concepts “OOP is programming paradigm for application” [1] while “SOA is an architectural paradigm for system landscapes” [1].

SOA is the approach to connect systems written in object-oriented or other paradigms, we can say that we need both in some cases.

1.7. What Is Next?

In this chapter, we introduced SOA, services, ESB and loose coupling. The discussion about such concepts will help us out to get better understanding for the next chapters. Next chapter will give an introduction about web services which are an instance of services, its standards as well as how to implement them using Java EE. Fourth chapter will discuss some forms of loose coupling in general then service coupling forms. The problem of tight coupling between the consumer and the contract will be discussed in the fifth chapter as well as suggest some solution for it through implementation.

1.8. Summary

- ✓ SOA is a paradigm, or existing ideas that have been put together.
- ✓ SOA is concept of large distributed systems that can solve some problems for it.
- ✓ SOA is not a silver bullet it's something that can raise some problems instead of solving them, if we don't know where and how to apply it.
- ✓ A service is self-contained business functionality.
- ✓ The service has many attributes, which must be achieved.
- ✓ We can classify services based on many factors and many viewpoints.
- ✓ ESB, service parts and other components of SOA have main role to achieve loosely coupled systems.
- ✓ Loose coupling is an aim of SOA.
- ✓ Loose coupling is a relative concept which can be measured using many factors. These factors differ from one system or landscape to another.
- ✓ SOA works with OOP and doesn't replace it.
- ✓ SOA infrastructure is ESB which can be classified into many types based on the type of the connection (point-to-point or mediator ESB) or up to the where the responsibility of an ESB begins for the provider and the consumer (protocol driven ESB or API's driven protocol).
- ✓ This work is revolving around increasing loose coupling using web services through implementation.

Chapter - 2

Web Services

- ✓ *Web Service Architecture*
- ✓ *Web Services Protocol Stack*
- ✓ *Web Services Standards*
 - *Web Service Description Language (WSDL)*
 - ❖ *The WSDL Document format*
 - ❖ *Simple Object Access Protocol (SOAP)*
 - ❖ *Universal Description, Discovery and Integration (UDDI)*
- ✓ *Web Services, the Whole picture*
- ✓ *Web Services Implementation Using Java EE*
 - *Building Web Services with JAX-WS*
 - ❖ *Coding the Service Provider*
 - ❖ *Coding the Service Client*
- ✓ *Summary*

006.76

AL93

In

TH-17499



Web services are easy to be understood after introducing services. Web Services is one option to implement SOA, as has been mentioned before in the first chapter.

They have many features that make them the best recommended option by all companies and all IT people to implement SOA. Before going through this chapter, we should have the knowledge of the basics of XML and Java languages.

2.1. Web Service Architecture

Web Services in definition is “application components”. They use open protocol Simple Object Access Protocol (SOAP) to communicate with each other. Like all other services, web service must be self-contained, self- describing and discoverable. They use standards to achieve all their attributes. WSDL or Web Service Description Language is being used by web services to describe service provider. UDDI or Universal Description, Discovery and Integration are used by service providers to make the service discoverable and by service consumers to discover the service.

“Don’t expect too much, too soon from Web Services” [3]. The Web Services platform is a simple, interoperable, messaging framework. It still misses many important features like security and routing. But, these features will be available as soon as SOAP becomes more advanced.

Web-applications are simple applications run on the web. These are built around the Web browser standards and can mostly be used by any browser on any platform. Web Services take Web-applications to the next level. Using Web services, application can publish its function or message to the rest of the world. Web services help to solve the interoperability problem by giving different applications a way to link their data.

2.2. Web Services Protocol Stuck

In order to get better understanding how web services work, it is better to take a look at web services protocol stuck which consists of four layers as described below:

1. Service Transport: responsible for transporting messages between applications. It includes HTTP, SMTP, FTP and BEEP protocols.

2. XML Messaging: this layer is responsible for encoding messages in a common XML format. It includes XML-RPC and SOAP.
3. Service Description: this layer is responsible for describing the public interface to a specific web service. It is handled via WSDL.
4. Service discovery: this layer is responsible to register services into a public registry and providing easy ways to publish and find the service. It's handled by UDDI.

2.3. Web Services Standards

There are more than thirteen standards of Web Services. The most important one are those which play a direct role in web service architecture. So we are going to introduce those standards in detail right here.

2.3.1. Web Service Description Language (WSDL)

WSDL stands for Web Service Description Language, which it is recommended by World Wide Web Consortium(W3C)[4]. It is written in XML or in other words we can say that it is XML document. The main role that WSDL plays with web services is describing the service provider and also it is used to locate the services.

2.3.1.1. The WSDL Document Format

A WSDL document describes a web service using major elements which can be listed as [4]:

Element	Defines
<code><portType></code>	The operations performed by the web service
<code><message></code>	The messages used by the web service
<code><types></code>	The data types used by the web service
<code><binding></code>	The communication protocols used by the web service

The main structure of a WSDL document looks like this:


```

<definitions>
    <types> definition of types..... </types>
    <message> definition of a message.... </message>
    <portType> definition of a port.....</portType>
    <binding> definition of a binding.... </binding>
    <service> definition of a service .... </service>
</definitions>

```

A WSDL document contains other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document. Let's take a look at the elements of WSDL document:

1- **WSDL Ports:** The <portType> element is the most important WSDL element. It describes a web service, the operations that can be performed, and the messages that are involved. The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language. This section has child Element called <operation>. The <operation> element has <name> attribute and three sub elements which can be used to specify which messages will be for input, output or fault. The order and existence of these elements reflect the type of the operation whether it is input operation (one way) , input and output operation (request response), output input operation (solicit response) or only output operation (notification). Additionally, <operation> element can have <fault> element which can be used to specify exceptions. All these three elements have a <name> attribute and <message> attribute to specify the messages which can be used for input, output and fault.

The request-response type is the most common operation type, but WSDL defines four types:

Type	Definition
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

2- WSDL Messages: The <message> element defines the data elements of an operation. Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language. The <part> element has a name attribute which present the parameter name. It has also two attributes <element> and <type>. Both the attributes can be used to specify the type of the parameter. <Element> attribute can be used in document oriented files and <type> attribute can be used in RPC oriented files. The difference between the two is <element> refers to another element in XML schema which been specified by <type> section, while name attribute refers to a type directly in XML schema whether the type is simple or complex. The <message> element has <name> attribute which accepts a simple string value.

3- WSDL Types: The <types> element defines the data types that are used by the web service. For maximum platform neutrality, WSDL uses XML Schema syntax to define data types [to know more about XML Schema].

4- WSDL Bindings: The <binding> element defines the message format and protocol details for each port. It has two attributes - the name attribute and the type attribute. We can use any name we want for the attribute's name defines the name of the binding, and the type attribute points to the port for the binding. The <soap:binding> element has two attributes - the style attribute and the transport attribute. The style attribute can be "rpc" or "document". The transport attribute defines the SOAP protocol to use. The <operation> element defines each operation that the port exposes. For each operation the corresponding SOAP action has to be defined. We must also specify how the input and output are encoded.

2.3.2 Simple Object Access Protocol (SOAP)

SOAP stands for Simple Object Access Protocol. It is recommended by W3C[4]. It is a protocol for communication among applications, or in other words we can say it is a format to send messages over internet. The advantages of using this protocol are that it is platform independent, language independent (because it's based on XML) that is why SOAP is extendable and simple.

There are some rules that must be followed in message syntax:

- 1- It must be encoded using XML.
- 2- It must use the SOAP envelope namespace.
- 3- It must use the SOAP encoding namespace.
- 4- It must not contain a DTD reference.
- 5- It must not contain XML processing instructions.

We take a look at SOAP message syntax below:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:header> . . . . </soap:Header>
    <soap:body>
        <soap:fault> . . . . </soap:Fault>
    </soap:body>
</soap:envelope>
```

The previous syntax contains the following elements:

- 1- The SOAP `<Envelope>` Element: It's the root of the SOAP message. It defines the XML document as a SOAP message. Actually anyone knows XML can recognize its attribute immediately.
- 2- The SOAP `<Header>` Element: The optional SOAP Header element contains application-specific information about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element.

Note: All immediate child elements of the `<Header>` element must be namespace-qualified.

- 3- The SOAP `<Body>` Element: The required SOAP `<Body>` element contains the actual SOAP message intended for the ultimate endpoint of the message. Immediate child elements of the SOAP Body element may be namespace-qualified.

- 4- The SOAP <Fault> Element: The optional SOAP Fault element is used to indicate error messages. If a <Fault> element is present, it must appear as a child element of the <Body> element. A <Fault> element can only appear once in a SOAP message. This element has many sub elements.

2.3.3 Universal Description, Discovery and Integration (UDDI)

UDDI stands for Universal Description, Discovery and Integration. It is a directory service where businesses can be registered and search for Web services as well. It is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet. Thus, in other words we can say UDDI is a directory for storing information about web services or a directory of web service interfaces described by WSDL which communicates via SOAP. It uses Internet standards such as XML, HTTP, and DNS protocols. There are many benefits of using UDDI. Actually, any industry or businesses of all sizes can take the benefits from UDDI. Before UDDI, there was no Internet standard for businesses to reach their customers and partners with information about their products and services. Nor was there a method of how to integrate with each other's systems and processes.

Problems which UDDI can help to solve are as following:

- 1- It offers a way to discover right business among many businesses which are currently online.
- 2- It enables to describe services and business processes in a single, open and secure environment.
- 3- It enables to expand the internet capability to add new users and manage access of existing ones.

2.4. Web Services, the Whole Picture

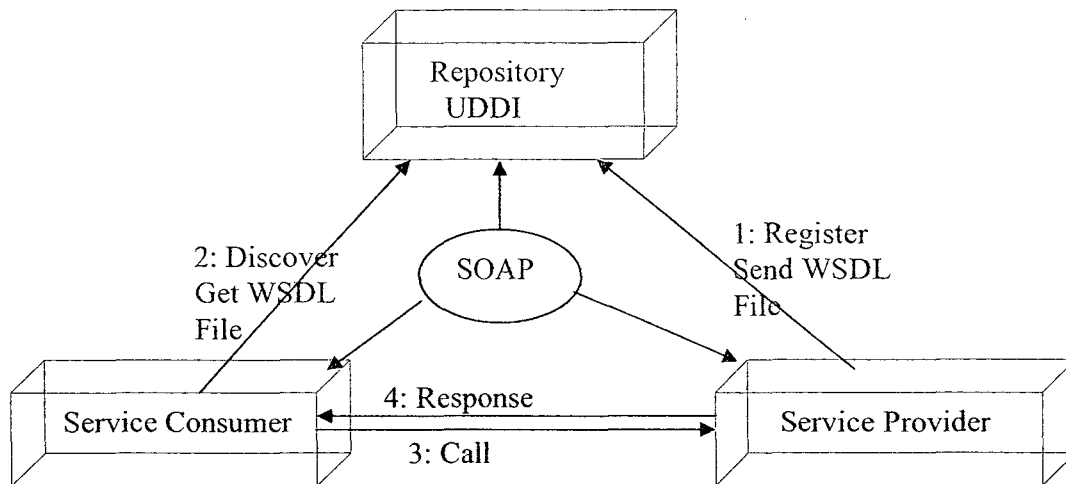


Figure 2-1 Web Service

Let us summarize the way the web services work in few steps as we can see in the Figure 2-1.

- 1- Register a Service: Service provider registers itself in the UDDI repository, to enable other to discover it by sending WSDL file which contains the service interface description.
- 2- Discover a Service: Service consumer can discover a service by searching UDDI Repository.
- 3- Make a call: At any time, the service consumer can make a direct call to a service provider after it gets the information necessary to call the service by discovering the service, using SOAP message.
- 4- Get a response: In return the provider must give back a response of the call as a SOAP message as well.

Note: All messages can be sent only using SOAP Protocol.

2.5. Web Services Implementation Using Java EE

The Java EE platform provides the XML API and tools which are needed to quickly design, develop, test, and deploy web services and clients that fully interoperate with other web services and clients running on Java-based or non-Java-based platforms. To write web services and clients with the Java EE XML API, all we need is passing parameter data to the method calls and process the data returned; or for document-oriented web services, we send documents containing the service data back and forth. No low-level programming is needed because the XML API implementations do the work of translating the application data to and from an XML-based data stream that is sent over the standardized XML-based transport protocols. The translation of data to a standardized XML-based data stream is what makes web services and clients written with the Java EE XML APIs fully interoperable. This does not necessarily mean that the data being transported includes XML tags because the transported data can itself be plain text, XML data or any kind of binary data such as audio, video, maps, program files and so on.

2.5.1. Building Web Services with JAX-WS

JAX-WS stands for Java API for XML Web Services. It allows developers to write message-oriented as well as RPC-oriented web services. In JAX-WS, a web service operation invocation is represented by SOAP. Although, SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods.

A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages. With JAX-WS clients and web services have a big advantage. The platform independence of the Java programming language. In addition, JAX-WS is not restrictive. A JAX-WS client can access a web service that is not running on the Java platform, and vice versa.

The starting point for developing a JAX-WS web service is a Java class annotated with the `javax.jws.WebService` annotation.

The `@WebService` annotation defines the class as a web service endpoint.

A service endpoint interface or service endpoint implementation (SEI) is a Java interface or class that declares the methods that a client can invoke on the service. An interface is not required when building a JAX-WS endpoint. The web service implementation class implicitly defines an SEI. One may specify an explicit interface by adding the `endpointInterface` element to the `@WebService` annotation in the implementation class. One must then provide an interface that defines the public methods made available in the endpoint implementation class. Now, we can list the basic steps for creating the web service and client:

1. First, we have to code the implementation class.
2. Then, we have to compile the implementation class.
3. Using `wsgen` tool we can generate the artifacts required to deploy the service.
4. The next step is packaging the files into a WAR file.
5. Then, we have to deploy the WAR file. The web service artifacts (which are used to communicate with clients) are generated by the Application Server during deployment.
6. Finally, we can code the client class.
7. Using `wsimport` we can generate and compile the web service artifacts needed to connect to the service.
8. Again, we have to compile the client class.

JAX-WS endpoints have some strict rules to be followed. We can list them as follows:

- The implementing class must be annotated with either the `javax.jws.WebService` or `javax.jws.WebServiceProvider` annotation.
- The implementing class may have explicit reference to a SEI through the `endpointInterface` element of the `@WebService` annotation, but is not required to do so. If no `endpointInterface` is specified in `@WebService`, an SEI is implicitly defined for the implementing class.
- The business methods of the implementing class must be public, and must not be declared static or final.

- Business methods that are exposed to web service clients must be annotated with `javax.jws.WebMethod`.
- Business methods that are exposed to web service clients must have JAXB-compatible parameters and return types.
- The implementing class must not be declared final and must not be abstract.
- The implementing class must have a default public constructor.
- The implementing class must not define the `finalize` method.
- The implementing class may use the `javax.annotation.PostConstruct` or `javax.annotation.PreDestroy` annotations on its methods for life cycle event callbacks. The `@PostConstruct` method is called by the container before the implementing class begins responding to web service clients. The `@PreDestroy` method is called by the container before the endpoint is removed from operation.

2.5.2.1. Coding the Service Provider

The following example is a simple hello example which is recommended by all programming languages as a standard example to start with:

```

package package_name;
import javax.jws.WebService;
@WebService
public class Class_Name {
    //some code
    public Class_Name() {} // the constructor
    @WebMethod
    public String the_Methode_Name(list_of_parameters) {
        // implementation details
        return some result;
    }
}

```


2.5.3.2 Coding the Service Client

When invoking the remote methods on the port, the client performs these steps:

1. Uses the `javax.xml.ws.WebServiceRef` annotation to declare a reference to a web service.

`@WebServiceRef` uses the `wsdlLocation` element to specify the URI of the deployed service's WSDL file.

```
@WebServiceRef(wsdlLocation="http://localhost:8080/package_Name/implementation_c  
lass?wsdl").
```

2. Retrieves a proxy to the service, also known as a port, by invoking `getClass_NamePort` on the service.

```
Class_Name port = service.getClass_NamePort();
```

The port implements the SEI defined by the service.

3. Invokes the port's `sayHello` method, passing to the service a name.

```
Return_Type response = port.the_Method_Name(name);
```

Here, the complete code for the client :

```
package simpleclient;  
import javax.xml.ws.WebServiceRef;  
import package_Name;  
import package_Name.Class_Name;  
  
public class Client_Name {  
  
    @WebServiceRef(wsdlLocation="http://localhost:8080/  
package_Name/Class_Name?wsdl");  
    static class_Service service;  
  
    public static void main(String[] args) {  
        try {  
            Client_Name client = new Client_Name();  
            client.doTest(args);  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

public void doTest(String[] args) {
    try {
        System.out.println("Retrieving the port from
            the following service: " + service);
        Class_Name port = service.getClass_NamePort();
        //some code
        Return_Type response =
        port.the_Methode_Name(list_of_parameters);
    } catch(Exception e) {
        e.printStackTrace();
    }
}
}

```

2.6. Summary

- ✓ Web Services have many standards as WSDL, SOAP and UDDI.
- ✓ WSDL file plays the role of the contract in web services
- ✓ SOAP is the protocol which is used by web services to exchange data and messages.
- ✓ UDDI is the standard to manage and control the repository of web services.
- ✓ Java EE is an easy Option to implement Web Service.

Chapter - 3

Loose Coupling

- ✓ *Introduction*
- ✓ *Forms of Loose Coupling*
 - *Asynchronous Communications*
 - *Heterogeneous Data Types*
 - *Mediator*
 - *Weak Type Checking*
- ✓ *Service Coupling*
 - *Logic-to-Contract Coupling*
 - *Contract-to-Logic Coupling*
 - *Contract-to-Implementation coupling*
 - *Consumer-to-Contract and Consumer-to-Implementation Coupling*
- ✓ *The Solution*
 - *Generating Proxy*
 - *Updating a Service*
- ✓ *Example*
- ✓ *Summary*

3.1. Introduction

As we know, SOA is applied to large distributed systems. The most important keys for such systems are scalability and fault tolerance. Therefore, decreasing the impact of the modifications and failures on the system is a goal right here. Thus, loose coupling is a key concept of SOA.

The ability to grow fast for such systems keeps them in the market. If they are not growing with enough speed and are not cheap enough that means they will be out of business sooner or later.

“Loose coupling is the concept which is used to deal with the requirements of scalability, flexibility and fault tolerance” [1]. Loose coupling minimizes dependencies. Thus modification or fault in one system will have fewer results in other systems.

Loose coupling is not a tool or something that can be applied in certain degree, or can be measured by a tool. Loose coupling degree can differ from one system to another, it is up to us how much we want to make our systems scalable, flexible and have a fault tolerance.

In the past, many custom applications were developed with certain types and levels of coupling. As examples consider the following:

- 1- In traditional two-tier client-server architecture, the clients were developed specifically to interact with a designated database. Proprietary commands were embedded within the client programs and changes to this binding affected all client installations.
- 2- In a typical multi-tier component-based architecture, components were often developed to work with other specific components. Even shared components that became more popular after Object Oriented principles were applied, still required tight levels of coupling when made part of inheritance structures.
- 3- When Web services emerged they were often mistakenly perceived to automatically establish a looser form of coupling within distributed architectures. While Web services can naturally decouple clients from proprietary technology, they can just as easily couple client programs to many other service implementation details.

3.2. Forms of Loose Coupling

There are many forms of loose coupling. Starting with the type of communication, landscape, and some designing issues we are going to apply whenever we are implementing our system.

3.2.1. Asynchronous Communications

To explain this type of communication, imagine that someone has sent you a mail to do some work for him. After you received the mail the sender will not wait for answer, he will continue his work till you answer him that you have done it for him or delivering the results to him.

This example is the best for loose coupling that is because both the sender and receiver will not affect each other whenever one of them is not available immediately or is out of the service.

But, such type of communication has a drawback. When the sender needs a reply, in asynchronous communication, the sender does not get replies to its messages immediately. So when it gets the reply it must associate the answer with the original request, for example, "correlation ID". In addition, it has to process the reply, which usually requires some knowledge of some of the initial state and context when the request was sent. Both correlating the response to the request and transforming the state from the request to response need some effort. Now, imagine that the sender sends a lot of messages to one or many destinations. Thus, the order of responses becomes an issue here. In addition, what will happen if some replies didn't arrive at all? Programming, testing and debugging have to take into account all possibilities which can be very complicated and time consuming.

Here, we can summarize the previous discussion in two points:

- The advantage is that the systems exchanging service message don't have to be online at the same time. In addition, we can get the benefits of non-blocking service providers and consumers.
- The drawback is service consumer gets much more complicated.

3.2.2. Heterogeneous Data Types

Of course, having harmonized data types among different systems will make such systems easier to deal with each other. For this reason, harmonizing data type became usual approach for those systems.

In fact, when Object-Oriented came into play, having a common business object or BOM became a goal. But it turned out that this approach caused a crisis for large distributed systems. The first reason for the disaster was that it is not possible to make an agreement among all systems, because they have different interests about the same topic. Thus, we will reach one of the two ends. Either we will not be able to fulfill all interests or our model will be more complicated.

As we saw, sooner or later the price of harmonization will become too high. For this reason we have to accept heterogeneity among distributed systems.

However, this approach proves that heterogeneity decouples components of large systems. The service provider will introduce new types which will be consumed by the service consumer.

Anyway, having no BOM offer us some advantage and drawback as well:

- The advantage is that the systems can modify their data types without affecting other systems.
- The drawback is that we have to map data types from one system to another.

Note: Of course we need some kind of harmonization for primitive data types among all systems.

3.2.3. Mediator

A third example of loose coupling is how a consumer finds the provider that has to process its request.

One option is, sending the request to one specific system using its physical address. This approach is tightly coupling one, because whenever the receiver moves or changes his address the consumer will not be able to find it again before finding its address somehow. This kind of mediator is called a broker.

Second type of mediator chooses the right endpoint for the request; the consumer sends the request to a symbolic name and the infrastructure (network, middleware, ESB) routes the call to the appropriate system according to routing rules.

Note: Web Services use a Point-to-Point communications.

3.2.4. Weak Type Checking

Usually, in most programming languages, we use checking type to detect errors early. The languages which have this feature are better than the other in sense of avoiding disasters. But, such a way to detect errors takes time and needs effort, so it doesn't work with large distributed systems.

In addition, in order to check types in SOA by ESB it needs to know some information about those types. As has been mentioned before, we also have a problem to introduce types in advance. That will increase complexity of services.

3.3. Service Coupling

All points discussed in this chapter are too general and applicable to any distributed software. Service coupling types can be discussed from another point of view.

Services are loosely coupled. Service contract imposes low consumer coupling requirements and themselves are decoupled from their environment or landscape.

The service contract is the core element of the service which most coupling design issues revolve around it.

We can design the contract with dependencies on the underlying service Logic or we can design the service logic with dependencies on the service contract. Thus, from the relationships and dependencies between service logic and contract we can extract a set of coupling types those are related to a service design directly:

- Logic-to-Contract Coupling

- Contract-to-Logic coupling
- Contract-to-Implementation Coupling

3.3.1. Logic-to-Contract Coupling

This type of coupling result from approach which is known as “contract first” process. This approach allows us to tune the underlying logic in support of service contract which can optimize runtime performance and reliability.

Following such approach can result in the service logic being tightly coupled to the service contract, because it has been created specifically in support of independent designed contract. But, still it is considered as a positive type of coupling.

3.3.2. Contract-to-Logic Coupling

This type of coupling appears whenever we derive contracts from existing logic. An example of this type of coupling in web services is when we derive the WSDL file from the implementation of the service provider using an automated tool.

3.3.3. Contract-to-Implementation Coupling

Sometimes, technology-specific characteristics force the contract to be coupled to them, as being implemented in a specific technology which needs to use a specific driver to communicate to database or being implemented in a specific programming language. Thus, consumer has to do as well. Using Web Services can help to eliminate a portion of this type of coupling by using WSDL files to describe or define the service. Service contract is written in XML language, therefore, consumer can have its own choice to be implemented in any language.

There are many other types of service coupling, but still our interest is about those types which are related to the contract and the consumer. We discuss another type of service coupling in the next

chapter which we will suggest some solution in order to reduce coupling between the consumer and the contract.

3.7. Summary

- ✓ The key of scalable, flexible large distributed systems is fault tolerance which can be achieved by applying loose coupling concepts.
- ✓ There are many forms of coupling as heterogeneous data types, mediator, and asynchronous connections.
- ✓ There are many forms of service coupling.

Chapter - 4

Contract-Consumer Coupling Problem

(Suggested Solution)

- ✓ *Consumer-Contract Coupling Problem*
- ✓ *The Solution*
 - *Generating Proxy*
 - *Updating a Service*
- ✓ *Example*
- ✓ *GUI Tool Example*
 - *MainFrame Class*
 - *ResultFrame Class*

4.1 Consumer-Contract Coupling Problem

After this overview about coupling type between the service contract and its logic or implementation, we have to take a look at the other side of the mirror, which means the coupling type but from the consumer point of view, we can name it as consumer-to- contract and implementation coupling.

To understand the consumer dependencies, let us recall the way that web services have been implemented, as we saw in the chapter 3.

Starting point to implement web service is coding the service provider then extracting the contract which is WSDL file here, then publish it at the repository.

The consumer using a tool can extract types and messages from the contract to know how to call the service provider. Coding the consumer must be accompanied with reference to the location of the contract. First question that comes into the mind here is:

Q: suppose the location of the contract has been changed or moved. What will be the result at the consumer side? Is not it true that we will be forced to modify the consumer code? Thus, is not this a type of tight coupling between the consumer and the contract up to location of contract (repository)? see Figure 4-1.

Second point that can be discussed here is that there are many strategies to update a service provider. One of them to mark a service as a depreciated one and publish a new service, then delete the old one whenever it becomes out of use. This strategy can work easily whenever the old service was working properly and was in use for short time so there are a few consumers are using it. Also, this strategy does not work to fix the bugs which are necessary for any new service.

Suppose we want to update the service provider for fixing some bugs. There are two probabilities right here from the technical point of view.

First, the bug will not modify the contract, which means we just can modify the body of the method without modifying the interface or technically API. Therefore, no modification will result in the consumer code. The second possibility is, the modification will result in modifying the contract as we change the name of the service, the location of the service, name of any of its

methods or its API. Thus, the modification will force us to republish the contract, and make corresponding updates to the consumer code as well.

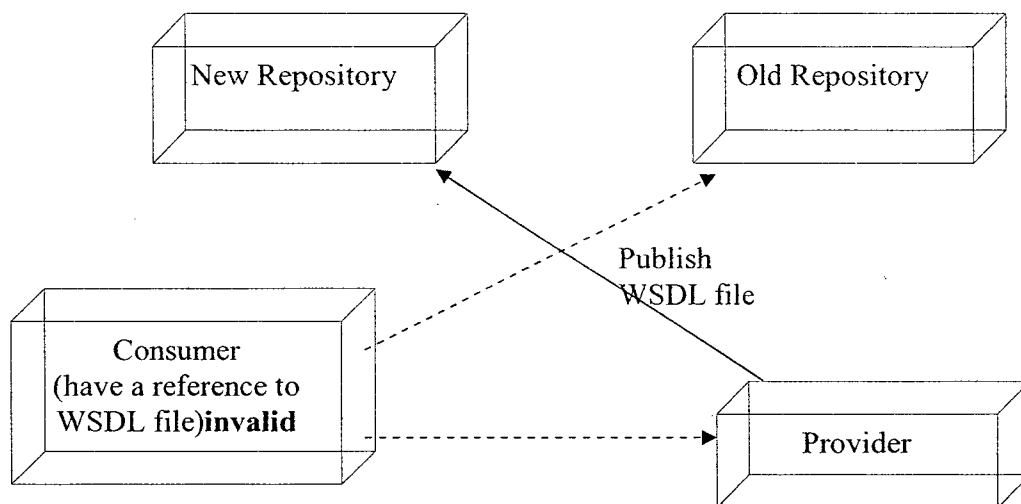


Figure 4-1 Tight coupling between and the consumer and the contract

Thus, up to the previous discussion, the second question raises up.

Q: Isn't this a kind of tight coupling between the consumer and contract for updating the contract? In other words, isn't it a kind of tight coupling if we can not update the contract and meanwhile the consumer can not resume working without modifying its code?

As a summary for the discussion above, we can say that contract (WSDL file) can increase loose coupling from the consumer point of view, but also create other types of coupling as those between the consumer and the contract.

4.2. The Solution

This section is specified to suggest a solution for the problem of tight coupling between the consumer and the contract through a set of classes (organized in packages) which are implemented in Java. These classes can be used to build a tool which can increase loose coupling between the consumer and the contract.

Let us start first to explain the idea behind these classes, then give a simulation how they work together to achieve the aim. We can find in appendix A the complete code of the classes. Additionally, we can find a working example of how to build a tool using these classes at the end of this chapter as well as another example about how the actions will be performed by the tool.

The idea depends on creating a proxy which plays the role of a broker between the consumer and the provider. This proxy will contain all the information about the WSDL file and how to call the service. The picture will be like this now, first the consumer will call the service through the proxy which has a reference to WSDL file and some methods which have the same name and parameters and return type of the service provider methods. If, the location of the contract (WSDL file) has been updated or the contract itself has been updated then the developer using the tool has to update the proxy as well. Thus, consumer code will not be changed or detect any problem with the new service.

The proxy also provides a way to map between the old and new services (updated Service). Take a look at Figure 4-2 to get a better understanding of the role of the proxy and the tool:

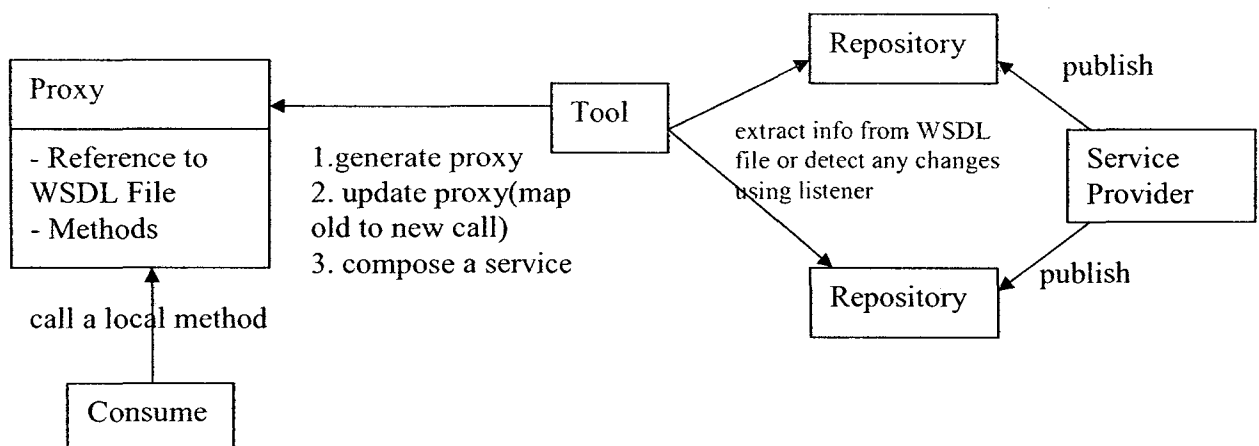


Figure 4-2 The role of the suggesting tool.

All that has to be done now is that the developer has to use the tool to reflect the changes in the proxy, and then the consumer will be able to call the new service without changing its code. First step to achieve our solution is illustrated in Figure 4-3.

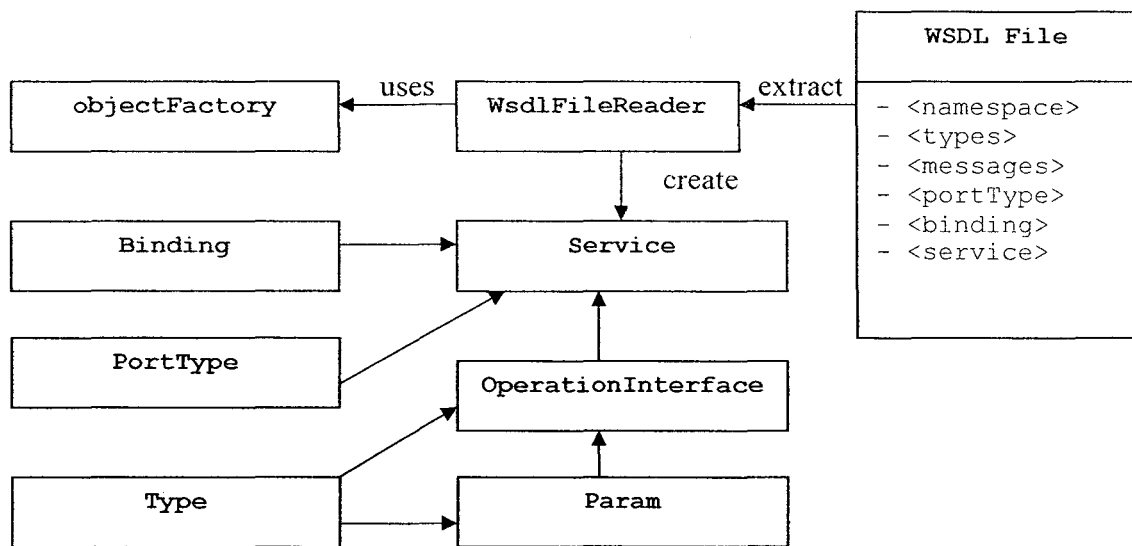


Figure 4-3 Extracting service out of WSDL file.

It the figure we can see that WsdlFileReader class use ObjectFactory class to create a Service object using the information which has been extracted from WSDL file. The tool has to support the following actions:

- 1- Generate a proxy for a specific service.
- 2- Update a service.

Next section will discuss the way to implement the basic actions of the tool and will be followed by an example [see the whole code of classes, Appendix A].

4.2.1. Generating Proxy

To generate a proxy for a specific service, there are some classes which can help to accomplish this mission. Look at Figure 4-4. The tool uses `ProxyCodeCreator` class to generate proxy written in java code which uses a `Service` object as input. The generated code can be compiled using the `Compiler` class.

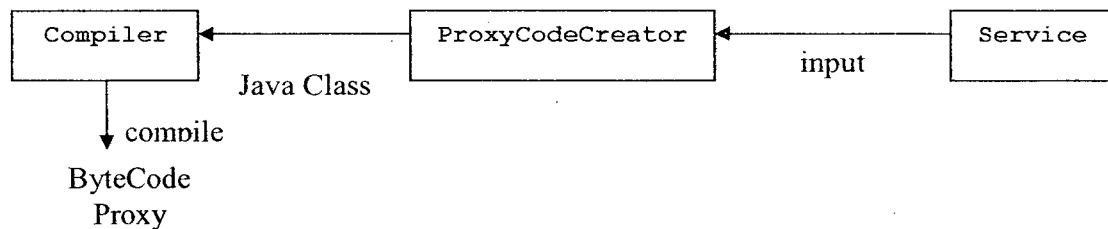


Figure 4-4 generate proxy

4.2.2. Updating a Service

To update a service (updating the contract), we need a way to map between the old and new services. This can be done using XML, the developer must write an XML code to express how to map new service to the old one. The XML code can be checked using a schema with the help of a class.

Let's take a look at the figure below. The XML updating code can be checked by `UpdateServiceSchema.xsd` to ensure that it is a well formed XML code. Then it will be checked by `UpdateServiceCodeChecker` which checks names of the service, operations, parameters and so on. Additionally, the class will extract all information related to mapping.

After extracting all classes we need to create the proxy as can be seen from the Figure 4-5. These classes will be used as an input to the class named by `UpdateServiceCodeCreator` which creates java class proxy which will be compiled by the `Compiler` class.

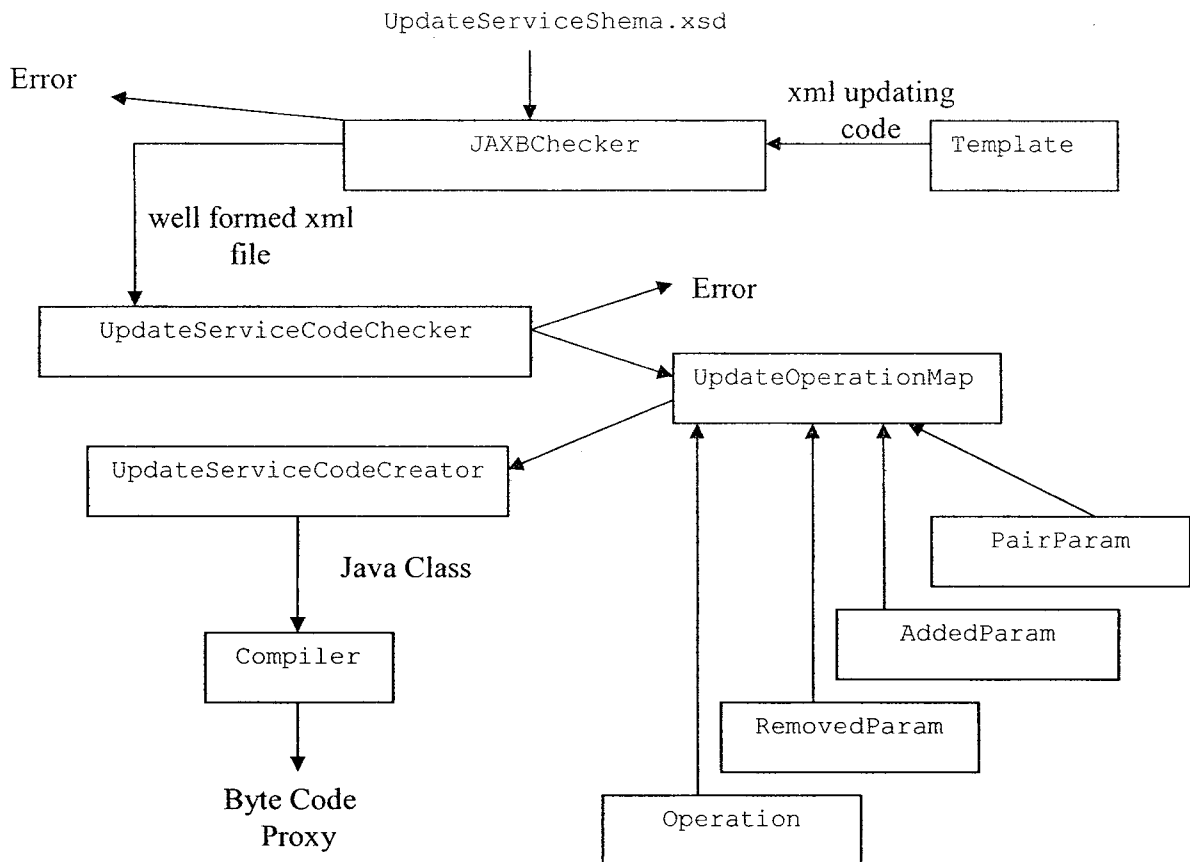


Figure 4-5 Update a Service

4.3. Example

We have seen pseudo code how to implement a web service at chapter three. Let us take an example and apply the given idea to it. Even though the example itself is too simple, the idea is not related to the logic of the service, it is related to the service contract. Therefore, we are going to apply the idea to Hello service.

The service provider code looks as follows:

```

package endpoint;
import javax.jws.WebService;
@WebService
public class Hello{
    private final String message = "Hello";

```



```

    public Hello() {
    } // the constructor
    @WebMethod
    public String getHello(String name) {
        return message + name;
    }
}

```

The generated WSDL file forms the input data for the tool. You can take a look at the WSDL file of Hello service example:

```

<definitions targetNamespace="http://endpoint/"
    name="HelloService">
    <types>
    <xsd:schema>
        <xsd:import namespace = "http://endpoint"
            schemaLocation="http://localhost:8080/Hello/
                HelloService?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="getHello">
        <part name="parameters" element="tns:getHello"/>
    </message>
    <message name="getHelloResponse">
        <part name="parameters" element="tns:getHelloResponse"/>
    </message>

    <portType name="Hello">
        <operation name="getHello">
            <input message="tns:getHello"/>
            <output message="tns:getHelloResponse"/>
        </operation>
    </portType>
    <binding name="HelloPortBinding" type="tns:Hello">
    <soap:binding
        transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
        <operation name="getHello">

```

```

    <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
<service name="HelloService">
    <port name="HelloPort" binding="tns:HelloPortBinding">
        <soap:address location="http://localhost:8080/Hello/HelloService"/>
    </port>
</service>
</definitions>

```

As has been mentioned in this chapter, the tool will extract information from WSDL file using WsdLFileReader class. Each instance of the previous class read only one WSDL file and the generated service by this class keeps a reference to it.

The WsdLFileReader gets some help from ObjectFactory to create a Service object. Let us see the result will be out from the previous class when it applies to our example:

(1)Service Object
<ul style="list-style-type: none"> - serviceName = "HelloService" - operations = vector of OperationInterface (2) - reader = instance of WsdLFileReader which created the service - Port : PortType (3) -packages = vector of String.(4)

(2)Vecotr of OperationInterface Object
item(0)
<ul style="list-style-type: none"> - name = "getHello" - params = vector of Param(5) - returnType = Param(6)

(3)PortType Object
<ul style="list-style-type: none"> - name = "Hello" - Operations = set of the service operations(2)

(4) Vector of String
item(0)
- package = "endpoint"

(5) Vector of Param
item(0)
<ul style="list-style-type: none"> - name = "name" - type :Type(7) - in = ture

(6) Param
<ul style="list-style-type: none"> - name = null - type : Type(7) - out = true

(7) Type Object
<ul style="list-style-type: none"> - name = "String" - fields = null - primitive = true

These objects will be used by ProxyCodeCreator to create the proxy as a java file which will be as follows:

```
import endpoint;
import javax.xml.ws.WebServiceRef;
public class HelloProxy {
    @WebServiceRef(wsdlLocation="http://enpoint/hello?wsdl")
    private static HelloService service;
    public void getHello(String name) {
        String response = null;
        try{
            Hello port = service.getHelloPort();
            response = port.getHello(name);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

The file above will be compiled by the Compiler class to generate the proxy Byte Code. Now, the consumer has to call the service through the generated proxy locally without referring to WSDL explicitly as follows:

```
package simpleclient;
import javax.xml.ws.WebServiceRef;
public class Client {
    private HelloProxy hello;
    private String name;
    public Client(String name) {
        this.name = name;
    }
}
```

```

        hello = new HelloProxy();
    }

    public void doTest(String name) {
        try {
            System.out.println(hello.getHello(name));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Suppose now that we want to update the service to include the login time and other information. The updated code for Hello service will look as follows:

```

package endpoint;
import javax.jws.WebService;
@WebService
public class Hello {
    private final String message = "Hello"
    public Hello() {} // the constructor

    @WebMethod
    public String getHello(User user, Time time) {
        return message + " " +user.getName()+"at"+time.getTime("yy,mm,dd");
    }
}

```

To reflect the changes in the service provider the developer has to update the precious proxy. All he has to do is to use the tool to map between the old and new service using Template class which gives a general form of updating code written in XML language as follows:

```

<? xml version="1.0">
<service oldServiceName="Hello" newServiceName="Hello">
    <operation oldName ="getHello"
        oldReturnType="String"
        newName ="getHello"

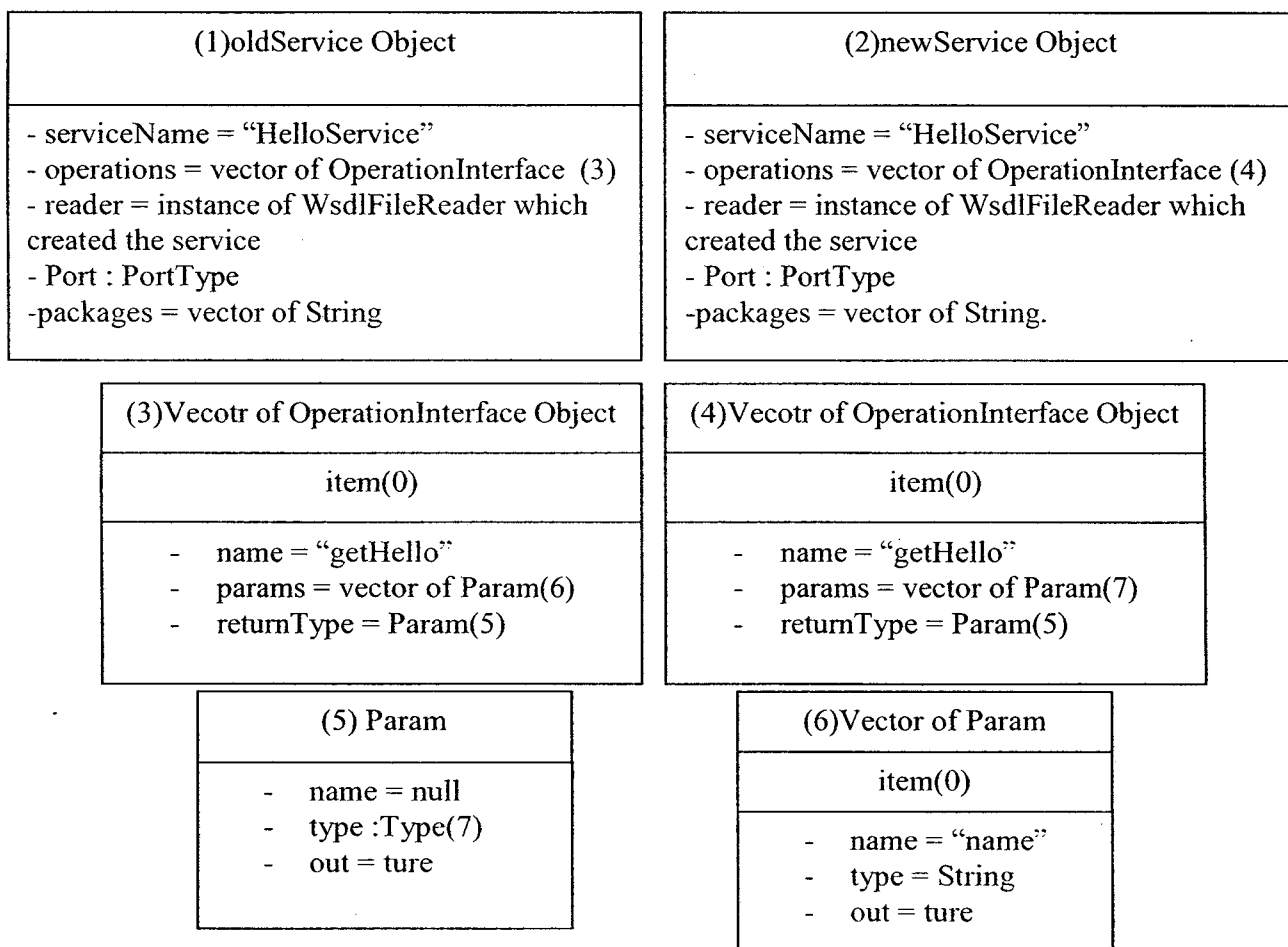
```

```

        newReturnType="String">
<addedParams>
<param name="time" type="Time"
        defaultValue="time.now()"/>
</addedParams>
<updatedParams>
        <param oldName="name"
                type ="string"
                newName="user"
                type = "User"
                map = "user.setName(name)"/>
</updatedParams>
</operation>
</service>

```

This code will be checked using UpdateServiceSchema.xsd and the object of class UpdateServiceCodeChecker will extract the following objects out of the code:



(7) Vector of Param	
item(0)	item(1)
- name = "user" - type = User - in = true	- name = "name" - type = Time - in = ture

(8) Vector Of UpdateOperationMap
item(0)
- oldOprName = "getHello" - newOprName = "getHello" - pairs : vector of PairParam (9) - addedParams : vector of AddedParam(10) - removedParams : vector of RemovedParam = null

(9) Vector of PairParam
item(0)
- oldParam : (6) -item(0) - newParam:(7)- item(0) - map : "user.setName(name)"

(10) Vector of PairParam
item(0)
param : (7)-item(1)

We can notice from the previous objects that all we need are objects (1, 8) to create a new proxy as the follows:

```
import endpoint;
import javax.xml.ws.WebServiceRef;
import endpoint.Time;
public class HelloProxy {
    @WebServiceRef(wsdlLocation="http://enpoint/hello?wsdl")
    private static HelloService service;
    public void getHello(String name) {
        String response = null;
        try{
            Hello port = service.getHelloPort();
            User user = new User();
            user.setName(name);
            response = port.getHello(user, Time.now());
        } catch(Exception e) {
            e.printStackTrace(); }
    }
}
```

4.4. GUI Tool Example

Here, we can find a simple example about how to use the Classes which we mentioned above to build the tool. We have the choice to build any Graphical interface we want. Two pictures of the Interface of the tool (Figures 4-6 and 4-7) as well as some portion of the programming code written in Java can be found next. This interface is made to enable easy adaptation.

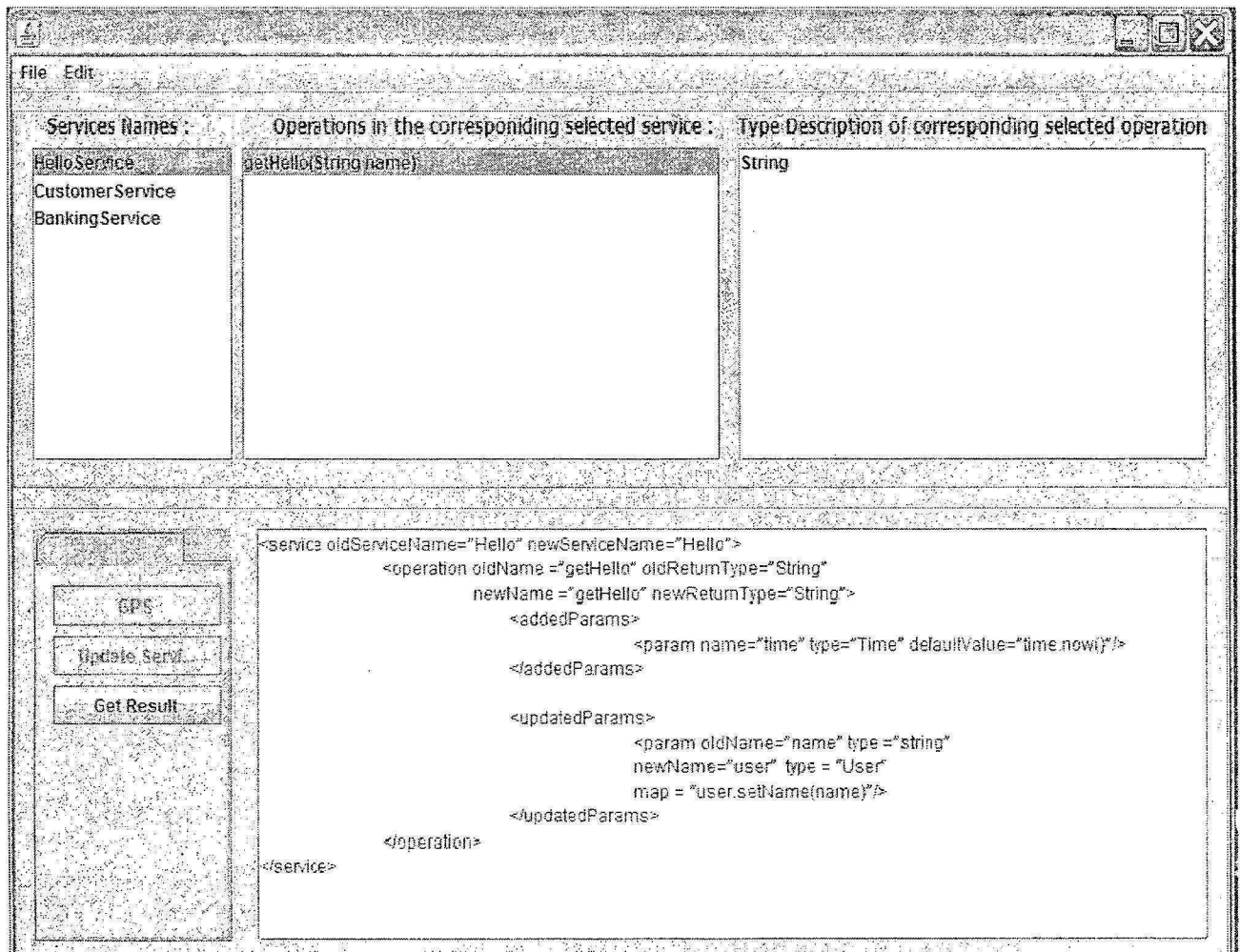


Figure 4-6 The Main window of the tool.

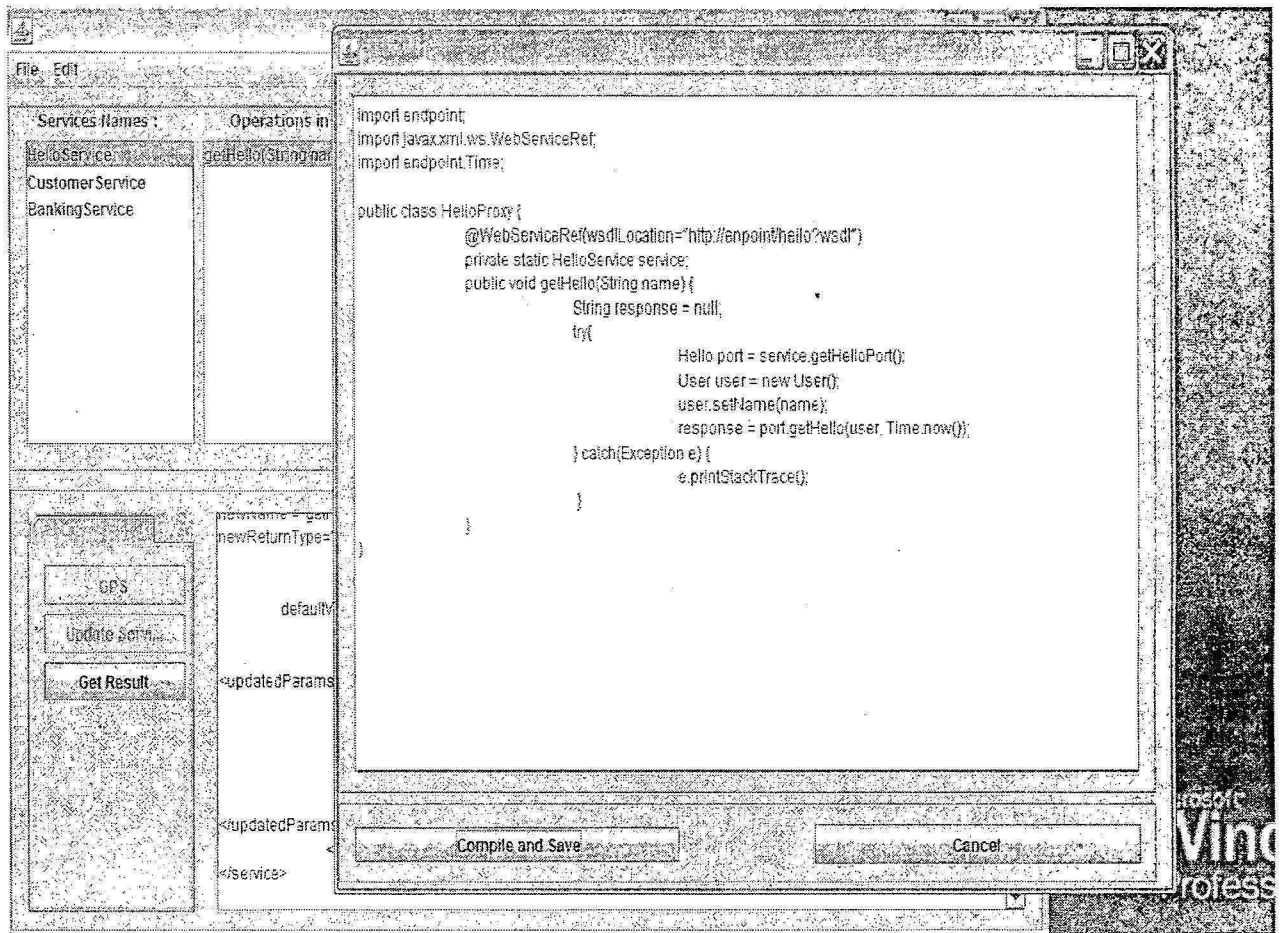


Figure 4-7 Result Window of the tool.

4.4.1. MainFrame Class

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.GUI;
//import section has been ignored
public class MainFrame extends JFrame {
    private JTabbedPane actionsTabPanel;
    private JPanel downPanel;
    private JMenu editMenu;
    private JMenuItem exitMenuItem;
    private JMenu fileMenu;
    private JPanel jPanel1;
    private JScrollPane jScrollPane1;
}

```



```

private JScrollPane jScrollPane2;
private JScrollPane jScrollPane3;
private JScrollPane jScrollPane4;
private JMenuBar mainMenu;
private JMenuItem openMenuItem;
private JLabel operationLabel;
private JList operationsList;
private JTextArea scriptTextArea;
private JLabel serviceLabel;
private JList servicesList;
private JLabel typeLabel;
private JList typeList;
private JPanel uperPanel;
private JFileChooser wsdlFileChooser;
// data
private String url;
private String[] wsdlFilesNames;
private Vector<WsdFileReader> inputBuilders;
private Vector<Service> services;
private String selectedServiceName;
private Service selectedService;
private OperationInterface selectedOperation;
private String selectedOperationName;
private File directory;
// Actions Buttons
private JButton createProxyButton;
private JButton updateServiceButton;
private JButton cancelButton;
private JButton getResultButton;
// result form
private ResultFrame resultFrame;
// construcotr: creating all fields objects and call others
//initialization methods as initComponents and initlisteners.
public MainFrame() {
    uperPanel = createPanel();
    servicesList = creatList("List Of Services Names");
    jScrollPanePanel = creatScrollPane(servicesList);
    operationsList = creatList("List of Operations");

```

```

jScrollPane2 = creatScrollPane(operationsList);
typeList = creatList("List of Types Descriptions");
jScrollPane3 = creatScrollPane(typeList);
serviceLabel = creatJLable("Services Names :");
operationLabel = creatJLable("Operations in the
                               corresponiding selected service :");
typeLabel = creatJLable("Type Description of
                               corresponding selected operation :");
wsdlFileChooser = creatFileChooser();
jPanell = createPanel();
actionsTabPanel = creatTabbedPane();
downPanel = createPanel();
createProxyButton = createButton("GPS",
                                "Generate Proxy for a Service", false);
cancelButton = createButton("cancel", "cancel the
                                operation", false);
getResultButton = createButton("Get Result", "show up
                                the output", false);
updateServiceButton = createButton("update Service",
                                "update existing service", false);
scriptTextArea = createTextArea();
jScrollPane4 = creatScrollPane(scriptTextArea);
mainMenu = createMenuBar();
fileMenu = createMenu(mainMenu, "File");
openMenuItem = createMenuItem(fileMenu, "Open WSDL
                                File", KeyEvent.VK_O, InputEvent.CTRL_MASK);
exitMenuItem = createMenuItem(fileMenu,
                                "Exit", KeyEvent.VK_F4, InputEvent.ALT_MASK);
editMenu = createMenu(mainMenu, "Edit");
setJMenuBar(mainMenu);
// data
inputBuilders = new Vector<WsdlFileReader>();
services = new Vector<Service>();

setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
initComponentsLayout();
initListeners();
pack();

```

```

}
// method to create an object of JPanel
private JPanel createPanel() {
    JPanel panel = new JPanel();
    panel.setBorder(BorderFactory.createEtchedBorder());
    return panel;
}
// method to create an object of JScrollPane
private JScrollPane creatScrollPane(Component comp) {
    JScrollPane jScrollPane = new JScrollPane();
    jScrollPane.setViewportView(comp);
    return jScrollPane;
}
// method to create an object of JLabel
private JLabel creatJLabel(String text) {
    JLabel label = new JLabel();
    label.setFont(new java.awt.Font("Tahoma", 1, 12));
    label.setAlignmentY(0.0F);
    label.setText(text);
    return label;
}
// method to create an object of JList
private JList creatList(String tip) {
    JList list = new JList();
    list.setBorder(new MatteBorder(null));
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    list.setToolTipText(tip);
    return null;
}
// method to create an object of JFileChooser
private JFileChooser creatFileChooser() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Open WSDL File");
    fileChooser.setAutoscrolls(true);
    fileChooser.setInheritsPopupMenu(true);
    fileChooser.setDialogType(JFileChooser.OPEN_DIALOG);
    fileChooser.setSelectedFiles(null);
    fileChooser.setFileSelectionMode(JfileChooser.DIRECTORIES_ONLY);
}

```

```

        return fileChooser;
    }
// method to create an object of JTabbedPane
private JTabbedPane creatTabbedPane() {
    JTabbedPane tabbedPane = new JTabbedPane();
    tabbedPane.setBackground(new Color(204, 204, 204));
    tabbedPane.setForeground(new Color(204, 204, 204));
    tabbedPane.setAlignmentX(0.0F);
    tabbedPane.setAlignmentY(0.0F);
    tabbedPane.setFont(new java.awt.Font("Tahoma", 1, 12));
    tabbedPane.setOpaque(true);
    tabbedPane.addTab("Actions Panel", downPanel);
    return tabbedPane;
}
// method to create an object of JButton
private JButton createButton(String text, String tip, boolean enabled) {
    JButton button = new JButton();
    button.setText(text);
    button.setToolTipText(tip);
    button.setEnabled(enabled);
    return button;
}
// method to create an object of JTextArea
private JTextArea createTextArea() {
    JTextArea textArea = new JTextArea();
    textArea.setColumns(20);
    textArea.setRows(5);
    return textArea;
}
// method to create an object of JMenuBar
private JMenuBar createMenuBar() {
    JMenuBar menuBar = new JMenuBar();
    return menuBar;
}
///// method to create an object of JMenu
private JMenu createMenu(JMenuBar menuBar, String text) {
    JMenu menu = new JMenu();
    menu.setText(text);
}

```

```

        menuBar.add(menu);
        return menu;
    }
    // method to create an object of JMenuItem
    private JMenuItem createMenuItem(JMenu menu, String text, int
                                     key1, int Key2) {
        JMenuItem menuItem = new JMenuItem();
        menuItem.setText(text);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(key1, Key2));
        menu.add(menuItem);
        return menuItem;
    }
    //Action which will be performed by open menu item.
    private void openMenuActionPerformed(ActionEvent evt) {

        if (evt.getSource() == openMenuItem) {
            int returnVal = wsdlFileChooser.showOpenDialog(this);
            if (returnVal == JFileChooser.APPROVE_OPTION)
                directory = wsdlFileChooser.getSelectedFile();
        }
    }
    //Action which will be performed by exit menu item
    private void exitMenuActionPerformed(ActionEvent evt) {
        if (evt.getSource() == exitMenuItem) {
            System.exit(1);
        }
    }
    //Action which will be performed when we choose a file
    private void fileChooserActionPerformed(ActionEvent evt) {
        url = wsdlFileChooser.getSelectedFile().getPath();
        if (directory.isDirectory()) {
            wsdlFilesNames = new String[directory.list().length];
            wsdlFilesNames = directory.list();
        }
        for (int i = 0; i < wsdlFilesNames.length; i++) {
            inputBuilders.addElement(new
                                     WsdlFileReader(wsdlFilesNames[i]));
        }
    }

```

```

Iterator<Wsd1FileReader> buildersIterator=inputBuilders.iterator();
while (buildersIterator.hasNext()) {
    for (int i = 0; i <
        buildersIterator.next().getServices().size(); i++) {
        services.addElement(buildersIterator.next()
            .getServices().get(i));
    }
}
Iterator<Service> servicesIterator = services.iterator();
Vector<String> servicesNames = new Vector<String>();
while (servicesIterator.hasNext()) {
    servicesNames.addElement(servicesIterator.next()
        .getServiceName());
}
servicesList.setModel((ListModel) servicesNames);
}
//Action which will be performed by selecting an item of the
// service list
private void serviceListActionPerformed(ListSelectionEvent e) {
    selectedServiceName = (String) servicesList.getSelectedValue();
    selectedService = inputBuilders.get(0).getServiceByName(
        selectedServiceName, inputBuilders);
    Vector<String> operationsData = new Vector<String>();
    for (int i = 0; i < selectedService.getOpetaions().size(); i++) {
        operationsData.addElement(selectedService.getOpetaions()
            .get(i).toString());
    }
    operationsList.setModel((ListModel) operationsData);
    Vector<String> typesData = new Vector<String>();
    for (int i = 0; i < selectedService.getOpetaions().size(); i++) {
        for (int j = 0; j < selectedService.getOpetaions().get(i)
            .getParams().size(); j++) {
            typesData.addElement(selectedService
                .getOpetaions().get(i)
                .getParams().get(j)
                .toString());
        }
    }
}

```

```

    }

    typeList.setModel((ListModel) typesData);
    createProxyButton.setEnabled(true);
    updateServiceButton.setEnabled(true);
}
//Action which will be performed by selecting an item of
//operation list
private void operationListActionPerformed(ListSelectionEvent evt) {
    selectedOperation = selectedService.getOperationByName((String)
        operationsList.getSelectedValue());
}
//Action which will be performed by pressing create proxy
//Button
private void createProxyActionPerformed(ActionEvent evt) {
    String javaCode=ProxyCodeCreator.getProxyJavaFile(selectedService);
    resultFrame = new ResultFrame("Java Code for Selected Event",
        UpdateServiceCodeChecker.getOldService()
            .getPortType().getName()
            +"ProxyA");

    resultFrame.getResultEditor().setText(javaCode);
    resultFrame.setAlwaysOnTop(true);
    resultFrame.setVisible(true);
}
//Action which will be performed by canceling the updating.
private void cancelActionPerformed(ActionEvent evt) {
    cancelButton.setEnabled(false);
    updateServiceButton.setEnabled(true);
    createProxyButton.setEnabled(true);
    getResultButton.setEnabled(false);
}
//Action which will be performed by pressing get result
//button
private void getResultActionPerformed(ActionEvent evt) {
    String error = UpdateServiceCodeChecker.checkCode(scriptTextArea
        .getText(), UpdateServiceCodeChecker.getOldService()
            .getWsdLFileReader());

    if (error != null) {

```

```

        final Dialog errorDialog = new Dialog(this);
        errorDialog.setTitle("XML Code Compiler : check
                                your xml code");
        errorDialog.add(new JLabel(error));
        JButton ok = createButton("Ok", "close the dialog", true);
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                errorDialog.setVisible(false);
            }
        });
        errorDialog.add(ok);
        errorDialog.setVisible(true);
        return;
    } else {
        String javaCode = UpdateServiceCodeCreator.getProxyJavaFile(
            UpdateServiceCodeChecker.getOldService(),
            UpdateServiceCodeChecker.getNewService(),
            UpdateServiceCodeChecker.getMap());
        resultFrame.getResultEditor().setText(javaCode);
        resultFrame.setVisible(true);
        resultFrame.setAlwaysOnTop(true);
    }
}

//Action which will be performed by pressing update service
//button
private void updateServiceButtonActionPerformed(ActionEvent evt) {
    createProxyButton.setEnabled(false);
    updateServiceButton.setEnabled(false);
    cancelButton.setEnabled(true);
    getResultButton.setEnabled(true);
    if (scriptTextArea.getText() == "")
scriptTextArea.setText(Template.getUpdateServiceTemplate(selectedService,
                                                            selectedOperation));
}

/**
 * initiate Listeners
 */
private void initListeners() {

```



```

// select directory
wsdlFileChooser.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        fileChooserActionPerformed(evt);
    }
});

openMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        openMenuActionPerformed(evt);
    }
});

// select an item from a service list
servicesList.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
        if (servicesList.getSelectedIndex() != -1)
            serviceListActionPerformed(e);
        else {
            createProxyButton.setEnabled(false);
            updateServiceButton.setEnabled(false);
        }
    }
});

operationsList.addListSelectionListener(new
                                ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
        if (operationsList.getSelectedIndex() != -1)
            operationListActionPerformed(e);
        else {
            createProxyButton.setEnabled(false);
            updateServiceButton.setEnabled(false);
        }
    }
});

exitMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        exitMenuActionPerformed(evt);
    }
});

```

```

openMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        exitMenuActionPerformed(evt);
    }
});
createProxyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        createProxyActionPerformed(evt);
    }
});
updateServiceButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        updateServiceButtonActionPerformed(evt);
    }
});
cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        cancelActionPerformed(evt);
    }
});
getResultButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        updateServiceButtonActionPerformed(evt);
    }
});
}
}

```

4.4.2. *ResultFrame Class*

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.GUI;
/*import section is ignored */
public class ResultFrame extends JFrame{
    private JButton closeButton;
    private JButton saveAndCompileButton;
    private JEditorPane resultEditor;

```

```

private JPanel jPanel1;
private JPanel jPanel2;
private JScrollPane jScrollPane1;
private JFileChooser saveDialog;
private File directory;
private MainFrame frame;
private String javaClassName;
// constructor which initialize all fields and call other
//initiating methods like initComponents and initListeners.
public ResultFrame(String title, String javaClassName) {
    jPanel1 = createPanel();
    jScrollPane1 = creatScrollPane(resultEditor);
    resultEditor = creatEditorPane();
    jPanel2 = createPanel();
    closeButton = createButton("Close", "Close this window", true);
    saveAndCompileButton = createButton("S&C",
                                        "Save and Compile Java Classes", true);
    saveDialog = creatFileChooser();
    this.setTitle(title);
    this.javaClassName = javaClassName;
    setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
    initComponentsLayout();
    initListeneres();
    pack();
}
// create JPanel Object
private JPanel createPanel() {
    JPanel panel = new JPanel();
    panel.setBorder(BorderFactory.createEtchedBorder());
    return panel;
}
// create JscrollPane object
private JScrollPane creatScrollPane(Component comp) {
    JScrollPane jScrollPane = new JScrollPane();
    jScrollPane.setViewportView(comp);
    return jScrollPane;
}
// create JfileChooser object

```

```

private JFileChooser creatFileChooser() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Save Java Class Proxy");
    fileChooser.setDialogType(JFileChooser.SAVE_DIALOG);
    fileChooser.setAutoscrolls(true);
    fileChooser.setInheritsPopupMenu(true);
    fileChooser.setSelectedFiles(null);
    fileChooser.setSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    return fileChooser;
}
// creat JButton Object
private JButton createButton(String text, String tip, boolean enabled) {
    JButton button = new JButton();
    button.setText(text);
    button.setToolTipText(tip);
    button.setEnabled(enabled);
    return button;
}
// create JeditorPane object.
private JEditorPane creatEditorPane() {
    JEditorPane editor = new JEditorPane();
    editor.setEditable(false);
    return editor;
}
// get access to editor
public JEditorPane getResultEditor() {
    return resultEditor;
}
// action will be performed bu save and compile button
private void saveAndCompileActionPerformed(ActionEvent evt) {
    if (evt.getSource() == saveAndCompileButton) {
        int returnVal = saveDialog.showSaveDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION)
            directory = saveDialog.getSelectedFile();
    }
}
// action will be performed by close button.
private void closeActionPerformed(ActionEvent evt) {

```

```

        if (evt.getSource() == closeButton) {
            this.setVisible(false);
        }
    }
}

// action will be performed when we choose save action.
private void saveDialogActionPerformed(ActionEvent evt){
    String javaCode = resultEditor.getText();
    File javaFile = new File(directory.getPath()+ javaClassName);
    Writer output;
    try {
        javaFile.createNewFile();
        output = new BufferedWriter(new FileWriter(javaFile));
        output.write(javaCode);
        output.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    Compiler compiler = new Compiler();
    File []array = new File[1];
    array[0] = javaFile;
    boolean res = compiler.compile(array, directory.getPath());
    if(res){
        closeActionPerformed(evt);
    }else {
        saveDialog.showSaveDialog(this);
    }
}

// initialize all listeners.
private void initListeners() {
    closeButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            closeActionPerformed(evt);
        }
    });
}
}

```

Chapter - 5

Conclusion And Future Plan

Large distributed systems must deal with legacies by accepting those systems they are in use and develop them. Developing such systems is not easy always, they have to be scalable and flexible, in other words they have to be loosely coupled systems as well as their components. Thus, they must have strategies for developing, testing and debugging taking into account that such systems are heterogeneous.

SOA came to deal with large distributed systems; it is a paradigm which brings existing concepts together. With the help of three concepts (services, interoperability and loose coupling) SOA could ensure high interoperability for large systems using ESB, it could decouple large system dependencies using services. SOA could also motivate these systems to scale up by applying loose coupling concepts.

Loose coupling concepts must be applied in a certain degree, it must not be an aim in itself. There are many forms which must be applied to the system according to our understanding of the system dependencies among it's components as well as among other systems.

In order to build loosely coupled systems we have to apply loose coupling forms to the services which form the components of the system. Understanding the dependencies among services and internals of the service itself will help us out to have loosely coupled services. Any service has a provider, consumer and contract. The role of the contract is providing the consumer with all the information which is necessary to call the provider. Handling service lifecycle has the same process as any usual software which can be managed easily whenever it is under development but that will be too complicated and must be handled carefully whenever it is under production.

There are many strategies to update or modify a service under production, but still it is difficult to apply any of them without affecting the caller or the other interacting services. These difficulties come from the dependencies between the consumer and the contract according to the contract modification and the contract location.

Using web services can help to solve the problem using simple common standards. They use WSDL file as a contract which is written in XML as well as they use SOAP to exchange data. Thus, using web services and Java EE we could suggest a solution for this kind of coupling in order to increase loose coupling between the consumer or the service caller and the contract of the service. The solution suggested a tool which has been implemented in Java EE. The tool explore the service repository looking for WSDL files and lists them at the tool interface with a description about the service accompanied with service's methods and used types.

The tool take the service's WSDL file as input and extract information from it, out of this information the tool builds a local proxy and places it at the consumer side. The proxy has all information how to call the service provider. All that has to be done now from the consumer side is to call the proxy which will call the service provider. Any updating for the contract location can be reflected by the service developer using the tool to update the reference to the WSDL file location, thus, location transparency will be achieved as well.

Another benefit of the tool is increasing loose coupling according to the contract modification. In case of contract modification the developer can use the tool to map between the old and the new services and reflect this mapping in the proxy. The consumer will not need any modification and it will be able to call the new service according to the mapping values. Thus, we could achieve our goal by increase loose coupling through implementation using web services. The tool is able to compile the generated Java source proxy.

This work can be the first step to find some ways to increase loose coupling through implementation. In fact, most types of coupling between the consumer and the contract and between the provider and the contract would be as a result of the need for information from both sides to be able to interact with each other. The consumer needs previously all information from the contract to be able to call the service. Also, all the information would be provided by provider will be reflected in the contract. Thus, consumer depend on the contract to extract some information about the provider, any changes in the contract will force to make the corresponding changes in the consumer. In the same way, the provider depends on the contract to publish its information. Any changes in the provider implementation will be reflected in the contract as well. Thus, finding a way to enable the consumer to call the service with less information will decrease the coupling. In the same way trying to keep some information from the provider side till run time will result in decreasing the loose coupling as well. Therefore, the next step of this work must be trying to shift some information which is necessary for both sides for calling or publishing till the run time will result in decreasing loose coupling.

The GUI tool must get improved to be usable easily. We can add some code assistant to the code editor and drag and drop options.

References

1. SOA in Practice, Nicolai M. Josuttis, O'Reilly, 2007.
2. SOA: Principles of Service Design, Thomas Erl, Prentice Hall, 2007.
3. The Java EE 5 Tutorial For Sun Java System Application Server 9.1, Sun Microsystems, Inc. , 2007.
4. w3school WSDL Tutorial, www.w3school.com , 2009.

Bibliography

1. Enterprise SOA, by *Dan Woods and Thomas Mattern (O'Reilly)*.
2. SOA in Practice, *Nicolai M. Josuttis, O'Reilly, 2007*.
3. SOA: Principles of Service Design, *Thomas Erl, Prentice Hall, 2007*.
4. The Java EE 5 Tutorial For Sun Java System Application Server 9.1, *Sun Microsystems, Inc. , 2007*.
5. w3school WSDL Tutorial, *www.w3school.com/wsdl/default.asp*.
6. w3school XML Schema Tutorial, *www.w3schools.com/schema/default.asp*.
7. Java SE 5 API documentation, *Sun Microsystems, Inc. , 2007*.
8. Understanding WSDL , *Aaron Skonnard
Northface University, MSDN Library, <http://msdn.microsoft.com>, October 2003*
9. XML Web Services Basics, *Roger Wolter, Microsoft Corporation, MSDN Library,
<http://msdn.microsoft.com>, December 2001*.

Appendix A

Classes Source Code

✓ *Classes Code*

Classes Code

1- Type Class

*** Fields :**

- name : the name of the type.
- fields : vector of types which present the fields of the type.
- primitive : a field which reflect if this type is a primitive or non-primitive type.

*** Constructors:**

- First one assign value to the name of the type, create a fields vector and set the type to primitive.
- Second one assign values to the name of the type and fields vector as well as set primitive field to false.

*** Getters and Setters**

*** Public Methods:**

- addField : simply add field to the vector of fields.
- toString : return a string description of type.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class Type {
    private String name;
    private Vector<Type> fields;
    private boolean primitive;
    public Type(String name) {
        this.name = name;
        fields = new Vector<Type>();
        primitive = true;
    }
    public Type(String name, Vector<Type> fields) {
        this.name = name;
        this.fields = fields;
        primitive = false;
    }
    public void addField(Type field) {
        fields.add(field);
    }
}
```

```

    }
    public Vector<Type> getFields() {
        return fields;
    }
    public void setFields(Vector<Type> fields) {
        this.fields = fields;
        setPrimitive(false);
    }
    public boolean isPrimitive() {
        return primitive;
    }
    public void setPrimitive(boolean primitive) {
        if (primitive)
            fields = null;
        this.primitive = primitive;
    }
    public String getName() {
        return name;
    }
    public String toStringDescription() {
        if (primitive)
            return name;
        String res = name + ":\n";
        for (int i = 0; i < fields.size(); i++) {
            res = res + "  " + fields.get(i).toString();
        }
        return res;
    }
}

===== End of Type class =====

```

2- Param Class

* Fields

- name: the name of the parameter.
- type: type of the parameter.
- in & out : boolean fields to express if the parameter is input or output.

* Constructors

- o There is one constructor which initialize the name and the type.

* Getters and Setters

* public Methods

- o toString : describe the parameter as String.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
public class Param {
    private String name;
    private Type type;
    private boolean in;
    private boolean out;
    public Param(String name, Type type) {
        this.name = name;
        this.type = type;
    }
    public String getName() {
        return name;
    }
    public Type getType() {
        return type;
    }
    public boolean isIn() {
        return in;
    }
    public void setIn(boolean in) {
        this.in = in;
    }
    public boolean isOut() {
        return out;
    }
    public void setOut(boolean out) {
        this.out = out;
    }
    public String toString() {
        String parambool = "[";
        if (in)
```

```

        parambool = parambool + "In ";
    if (out)
        parambool = parambool + "Out ";
    parambool = parambool + "]\n";

    return type + " "+name + ":" + parambool;
}
}

===== End of Param class =====

```

3- Message Class

* Fields

- name : name of the message.
- params : vector of Param class to express the params which are used with the message.
- in & out & fault fields : to express if the message are used as input , output or fault message.

* Constructors

- First constructor initialize the name and params fields.
- Second one initialize the name of the message only.

* Getters and Setters

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class Message {
    private String name;
    private Vector<Param> params;
    private boolean in;
    private boolean out;
    private boolean fault;

    public Message(String name, Vector<Param> params) {
        this.name = name;
        this.params = params;
    }
}

```

```

public Message(String name) {
    this.name = name;
    this.params = new Vector<Param>();
}
public String getName() {
    return name;
}
public Vector<Param> getParams() {
    return params;
}
public void setParams(Vector<Param> params) {
    this.params = params;
}
public void addParam(Param param) {
    this.params.add(param);
}
public boolean isIn() {
    return in;
}
public void setIn(boolean in) {
    this.in = in;
}
public boolean isOut() {
    return out;
}
public void setOut(boolean out) {
    this.out = out;
}
public boolean isFault() {
    return fault;
}
public void setFault(boolean fault) {
    this.fault = fault;
}
}

```

===== **End of Message class** =====

4- OperationInterface **Class**

*** Fields**

- name : name of the operation or the method.
- params : vector of the passed parameters to the operation.
- returnType : type of return value of the operation.

*** Constructors**

- First one initialize the name of the operation and the vector object.
- Second one initialize the name and the parameters of the operation.

*** Getters and Setters**

*** Public Methods**

- toString : describes the operation API as String.
- addParam : add a param object to vector of params.
- notAddedBefore : check if any parameter has been added to the operation parameters in the same name of passed one.
- getParamByName : search for param who has the same name as the passed name and return object of Param.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class OperationInterface {
    private String name;
    private Vector<Param> params;
    private Param returnParam;
    public OperationInterface(String name) {
        this.name = name;
        this.params = new Vector<Param>();
    }
    public OperationInterface(String name, Vector<Param> params) {
        this.name = name;
        this.params = params;
    }

    public String getName() {
```

```

        return name;
    }
    public Vector<Param> getParams() {
        return params;
    }
    public Param getReturnParam() {
        return returnParam;
    }
    public String toString() {
        String result = name + " ( ";
        for (int i = 0; i < params.size(); i++) {
            result = result + " " +params.get(i).toString();
        }
        result = result + ")";
        return result;
    }
    public boolean addParam(Param param) {
        int pos = notAddedBefore(param);
        if (pos == -1 && !param.isOut()) {
            params.add(param);
            return true;
        } else if (pos == -1) {
            if (param.isOut()) {
                params.get(pos).setOut(param.isOut());
                returnParam = param;
            }
            return false;
        }
        return false;
    }
}

public int notAddedBefore(Param param) {
    int pos = -1;
    for (int i = 0; i < params.size(); i++) {
        if (params.get(i).equals(param)) {
            pos = i;
            break;
        }
    }
}

```

```

        }
    }
    return pos;
}
public Param getParamByName(String name) {
    for (int i = 0; i < params.size(); i++) {
        if (name == params.get(i).getName())
            return params.get(i);
    }
    return null;
}
}

===== End of OperationInterface class =====

```

5- PortType Class

* Fields

- name : name of service portType.
- operations : vector of operations.

* Constructors

- First one assign value to the name and create object of vector of OperationInterface.
- Second one assing values to the name and and the vector of OperationInterface.

* Getters and Setters

* Public Mehtods

- addOperation : add a single operation to the vector.
- addOperations : append vector of operations to the vector.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class PortType {
    private String name;
    private Vector<OperationInterface> operations;
    public PortType(String name) {

```

```

        this.name = name;
        operations = new Vector<OperationInterface>();
    }
    public PortType(String name, Vector<OperationInterface> oprs)
    {
        this.name = name;
        this.operations = oprs;
    }
    public String getName() {
        return name;
    }
    public Vector<OperationInterface> getOperations() {
        return operations;
    }
    public void addOperation(OperationInterface opr) {
        operations.add(opr);
    }
    public void addOperations(Vector<OperationInterface> operations) {
        for (int i = 0; i < operations.size(); i++) {
            this.operations.add(operations.get(i));
        }
    }
}

```

===== **End of PortType class** =====

6- Binding Class

* Fields

- name : name of service binding.
- port : an reference to the port type of binding.

* Constructors

- initialize the name and the port.

* Getters and Setters

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
public class Binding {
    private String name;
    private PortType port;
}

```

```

public Binding(String name, PortType port) {
    this.name = name;
    this.port = port;
}
public String getName() {
    return name;
}
public PortType getPort() {
    return port;
}
}

===== End of Binding class =====

```

7- Service Class

* Fields

- serviceName : the name of the service.
- operations : vector of operations included in the service.
- reader : reference to a WsdlFileReader which has read the WSDL file of the service.
- port : portType which has been used by this service.
- packages : vector of String included all the packages of the service.

* Constructors

- First one assigning a value to the service name and create object of vectors of OperationInterface and String.
- Second one assign a values to service name, operations, reader, port as well as packages.

* Getters and Setters

* Public Methods

- toString : describe the service as a String(Service name only).
- addOperation : add the passed operation to the operations vector.
- addOperations : append a vector of operations to operations vector.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class Service {
    private String serviceName;
    private Vector<OperationInterface> operations;
    private WsdFileReader reader;
    private PortType port;
    private Vector<String> packages;
    public Service(String serviceName) {
        this.serviceName = serviceName;
        operations = new Vector<OperationInterface>();
        packages = new Vector<String>();
    }
    public Service(String serviceName, PortType port, Vector<String>
        packages, WsdFileReader reader) {
        this.serviceName = serviceName;
        this.operations = port.getOperations();
        this.port = port;
        this.reader = reader;
        this.packages = packages;
    }
    public String getServiceName() {
        return serviceName;
    }
    public WsdFileReader getWsdFileReader() {
        return reader;
    }
    public void setWsdFileReader(WsdFileReader reader) {
        this.reader = reader;
    }
    public Vector<OperationInterface> getOpetations() {
        return operations;
    }
    public void setOpetations(Vector<OperationInterface> operations) {
        this.operations = operations;
    }
    public PortType getPortType() {
        return port;
    }
}

```

```

    }
    public void setPortType(PortType port) {
        this.port = port;
    }
    public Vector<String> getPackages() {
        return packages;
    }
    public void setPackages(Vector<String> packages) {
        this.packages = packages;
    }
    public String toString() {
        return serviceName;
    }
    public boolean addOperation(OperationInterface opr) {
        for (int i = 0; i < operations.size(); i++) {
            if (operations.get(i).getName() == opr.getName()) {
                return false;
            }
        }
        operations.add(opr);
        return true;
    }
    public OperationInterface getOperationByName(String operationName) {
        for (int i = 0; i < operations.size(); i++) {
            if (operationName == operations.get(i).getName())
                return operations.get(i);
        }
        return null;
    }
}

```

===== End of Service class =====

8- JAXPChecker Class

* Fields

- o document : a refrence of string or url to the WSDL file.
- o factory : refrence to a factory which builds the xml document object.
- o parser : check if the xml document is well formed document.

* Constructors

- o passing url to document field and create object of factory.

* Getters and Setters

* public Methods

- o check : call the parser to check if the document is well formed or not.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
public class JAXPChecker {
    private String document;
    private DocumentBuilderFactory factory;
    private DocumentBuilder parser;
    public JAXPChecker(String docUrl) {
        if (docUrl == null) {
            System.out.println("Usage: java JAXPChecker URL");
            return;
        }
        document = docUrl;
        factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        factory.setValidating(true);
    }
    public DocumentBuilder getParser() {
        return parser;
    }
    public void check() {
        try {
            parser = factory.newDocumentBuilder();
            parser.parse(document);
        } catch (SAXException e) {
            System.out.println(document + "is not well-formed");
        } catch (IOException e) {
```



```

        System.out.println("Due to an IOException, the parser could
            not check " + document);
    } catch (FactoryConfigurationError e) {
        System.out.println("Could not locate factory");
    } catch (ParserConfigurationException e) {
        System.out.println("Could not locate JAXP parser");
    }
}
}

```

===== **End of JAXPChecker class** =====

9- ObjectFactory class

* Getters and Setters

* Public Methods

- o createType : create vector Type Object out of xml document.
- o createParameter : create a Param object out of xml node.
- o createMessage : create a Message Object out of xml node.
- o createOperation : create a operationInterface out of xml node.
- o createService : create a Service out of xml node.
- o getMsgByName : search a vector of Message for a Message using it's name.
- o getPortByName : search a vector of PortType for a port using it's name.

* Private Methods

- o addPackage : add a package to a srvice object.
- o extractPackage : extract package name out of namespace or url.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
import org.apache.xml.Schema;
import org.w3c.dom.Attr;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
public class ObjectFactory {
    public static Vector<Type> createType(Node typeNode) {
        Schema schema = new Schema(typeNode);
    }
}

```

```

Vector<Object> objs = schema.extractTypes();
Vector<Type> types = new Vector<Type>();
for (int i = 0; i < objs.size(); i++) {
    try {
        types.add((Type) objs.get(i));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return types;
}

public static Param createParameter(Node partNode, Schema schema) {
    String paramName;
    String paramType;
    Element nodeElement = (Element) partNode;
    Attr nameAttr = nodeElement.getAttributeNode("name");
    paramName = nameAttr.getValue();
    Attr typeAttr = nodeElement.getAttributeNode("type");
    if (typeAttr == null)
        typeAttr = nodeElement.getAttributeNode("element");
    paramType = typeAttr.getValue();
    return new Param(paramName, (Type)
        schema.extractType(paramType));
}

public static Message createMessage(Node msgNode, NodeList
    paramsNodes, Schema schema){
    Element msgElement = (Element) msgNode;
    Attr msgNameAttr = msgElement.getAttributeNode("name");
    String msgName = msgNameAttr.getValue();
    Vector<Param> params = new Vector<Param>();
    for (int i = 0; i < paramsNodes.getLength(); i++)
        params.add(ObjectFactory.createParameter(
            paramsNodes.item(i), schema));
    return new Message(msgName, params);
}

public static OperationInterface createOperation(Node
    operationNode, NodeList parts,
    Vector<Message> msgs,

```

```

        WsdlFileReader reader){

    Element oprElement = (Element) operationNode;
    Attr oprNameAttr = oprElement.getAttributeNode("name");
    String operationName = oprNameAttr.getValue();
    OperationInterface opr = new
        OperationInterface(operationName);
    for (int i = 0; i < parts.getLength(); i++) {
        Element partElement = (Element) parts.item(i);
        String partName = partElement.getTagName();
        Attr msgAttr= partElement
            .getAttributeNode("message");
        String msgName = msgAttr.getValue();
        Message msg = getMsgByName(msgName, msgs);

        if (partName == "input")
            msg.setIn(true);
        else if (partName == "output")
            msg.setOut(true);
        else
            msg.setFault(true);
        for (int j = 0; j < msg.getParams().size(); j++) {
            opr.addParam(msg.getParams().get(j));
        }
    }
    return opr;
}

public static Service createService(Node serviceNode,
        NodeList children, Vector<PortType> ports,
        Node types, WsdlFileReader reader) {
    Element serviceElement = (Element) serviceNode;
    Attr serviceNameAttr
        =serviceElement.getAttributeNode("name");
    String serviceName = serviceNameAttr.getValue();
    int i = 0;
    boolean found = false;
    while (!found) {
        if (children.item(i).getNodeName() == "port") {

```

```

        found = true;
        break;
    }
    i++;
}
Element portElement = (Element) children.item(i);
String portName =
    (portElement.getAttributeNode("type")).getValue();
PortType port = getPortByName(portName, ports);

NodeList typesSubNodes = types.getChildNodes();
Vector<String> packages = new Vector<String>();
for (int j = 0; j < typesSubNodes.getLength(); j++) {
    if (typesSubNodes.item(j).getNodeName() == "import"){
        Element importElement=(Element) typesSubNodes.item(j);
        Attr namespaceAttr=importElement
            .getAttributeNode("namespace");
        String namespaceUrl = namespaceAttr.getValue();
        addPackage(packages, namespaceUrl);
    }
}
return new Service(serviceName, port, packages, reader);
}

public static Message getMsgByName(String msgName,
    Vector<Message> msgs) {
    for (int i = 0; i < msgs.size(); i++)
        if (msgName.equals(msgs.get(i).getName()))
            return msgs.get(i);
    return null;
}

public static PortType getPortByName(String portName,
    Vector<PortType> ports) {
    for (int i = 0; i < ports.size(); i++)
        if (portName.equals(ports.get(i).getName()))
            return ports.get(i);
    return null;
}

```

```

private static void addPackage(Vector<String> packages, String url){
    String packageName;
    packageName = extractPackage(url);
    packages.add(packageName);
}

private static String extractPackage(String url) {
    int i = url.lastIndexOf("/");
    String temp = url.substring(i + 2);
    i = temp.lastIndexOf("/");
    while (i != -1) {
        temp = temp.substring(0, i);
        i = temp.lastIndexOf("/");
    }
    String temp2 = "";
    i = temp.lastIndexOf(".");
    while (i != -1) {
        temp2 = temp2 + temp.substring(i + 1, temp.length())
            + ".";
        temp = temp.substring(0, i);
        i = temp.lastIndexOf(".");
    }
    temp2 = temp2 + temp;
    return temp2;
}
}

===== End of ObjectFactory class =====

```

10 - WsdlFileReader **Class**

* **Fields**

- types : vector of types are listed in WSDL file.
- wsdlFileUrl: url to WSDL file.
- checker : JAXPChecker object to check xml file.
- rootNode : root node Of WSDL file.
- schema : schema object which can be used to read chema part of WSDL file.
- services : vector of services listed in WSDL file.
- operations : vector of operations are used on the file.

- bindings : vector of binding types.
- ports: vector of ports.
- msgs : vector of messages.

*** Constructors**

- Take a url to WSDL file and make sure that it's well formed and extract all information are needed to build the objects.

*** Getters and Setters**

*** Public Methods**

- getServiceByName: search for a service in one WSDL file using it's name.
- getOperationByName: search for an operation using it's name.
- getServiceByName: overrides the method above and search for the sevice in all WSDL files using it's name.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;
import org.apache.xml.Schema;
import org.w3c.dom.Attr;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class WsdlFileReader {
    private Vector<Type> types;
    private String wsdlFileUrl;
    private JAXPChecker checker;
    private Node rootNode;
    private Schema schema;
    private Vector<Service> services;
    private Vector<OperationInterface> operations;
    private Vector<Binding> bindings;
    private Vector<PortType> ports;
    private Vector<Message> msgs;

```

```

public WsdlFileReader(String wsdlFileUrl) {
    this.wsdlFileUrl = wsdlFileUrl;
    types = new Vector<Type>();
    services = new Vector<Service>();
    operations = new Vector<OperationInterface>();
    bindings = new Vector<Binding>();
    ports = new Vector<PortType>();
    msgs = new Vector<Message>();
    checker = new JAXPChecker(this.wsdlFileUrl);
    checker.check();
    try {
        rootNode = checker.getParser().parse(wsdlFileUrl);
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    NodeList rootChildNodes = rootNode.getChildNodes();
    for (int i = 0; i < rootChildNodes.getLength(); i++) {
        if (rootChildNodes.item(i).getNodeName() == "message") {
            NodeList msgChildNodes =
                rootChildNodes.item(i).getChildNodes();
            msgs.add(ObjectFactory.createMessage(
                rootChildNodes.item(i),
                msgChildNodes, schema));
        }
    }
    Node typesNode = null;
    for (int i = 0; i < rootChildNodes.getLength(); i++) {
        if (rootChildNodes.item(i).getNodeName()=="types")
            typesNode = rootChildNodes.item(i);
        schema = new Schema(rootChildNodes.item(i));
        break;
    }
    for (int i = 0; i < rootChildNodes.getLength(); i++) {
        if (rootChildNodes.item(i).getNodeName()=="portType"){
            NodeList operationsNodes = rootChildNodes.item(i)

```

```

                .getChildNodes();
Element portElement = (Element)rootChildNodes.item(i);
Attr portNameAttr =
    portElement.getAttributeNode("name");
String portName = portNameAttr.getValue();
for (int k = 0; k < operationsNodes.getLength();
    k++){
    NodeList oprChildNodes = operationsNodes.item(k)
        .getChildNodes();
    operations.add(
        ObjectFactory.createOperation
            (operationsNodes.item(i),
            oprChildNodes, msgs, this));
    }
ports.add(new PortType(portName, operations));
}
}
for (int i = 0; i < rootChildNodes.getLength(); i++) {
    if (rootChildNodes.item(i).getNodeName() == "binding"){
        Element bindingElement = (Element)
            rootChildNodes.item(i);
        Attr bindingNameAttr =
            bindingElement.getAttributeNode("name");
        Attr bindingPortAttr =
            bindingElement.getAttributeNode("type");
        String bindingName =
            bindingNameAttr.getValue();
        String bindingPortName =
            bindingPortAttr.getBaseURI();
        PortType port = ObjectFactory.getPortByName(
            bindingPortName, ports);
        bindings.add(new Binding(bindingName, port));
    }
}
for (int i = 0; i < rootChildNodes.getLength(); i++) {
    if (rootChildNodes.item(i).getNodeName() == "service") {
        NodeList serviceChildNodes = rootChildNodes.item(i)
            .getChildNodes();

```



```

        services.add(ObjectFactory.createService(
            rootChildNodes.item(i),
            serviceChildNodes, ports,
            typesNode, this));
    }
}

public Vector<Service> getServices() {
    return services;
}

public Vector<OperationInterface> getOperations() {
    return operations;
}

public Vector<Type> getTypes() {
    return types;
}

public Vector<Message> getMsgs() {
    return msgs;
}

public String getWsdFileUrl() {
    return wsdlFileUrl;
}

public Service getServiceByName(String serviceName) {
    for (int i = 0; i < services.size(); i++) {
        if(services.get(i).getServiceName()
            .equals(serviceName))
            return services.get(i);
    }
    return null;
}

public OperationInterface getOperationByName(String
    serviceName, String operationName) {
    for (int i = 0; i < services.size(); i++) {
        if (services.get(i).getServiceName().equals(serviceName))
            for (int j = 0; j <
                services.get(i).getOperations().size(); j++) {
                if (services.get(i).getOperations().get(j)
                    .getName().equals(operationName))

```

```

        return services.get(i).getOperations().get(j);
    }
}
return null;
}
public Service getServiceByName(String serviceName,
    Vector<WsdFileReader> inputBuilders) {
    Iterator<WsdFileReader> it = inputBuilders.iterator();
    Service service = null;
    while (it.hasNext()) {
        ss = it.next().getServiceByName(serviceName);
        if (service!= null)
            break;
    }
    return service;
}
}

```

===== **End of WsdFileReader class** =====

11- AddedParam Class

* Extends

- Param class.

* Fields

- defaultValue : default value of the added param.

* Constructors

- First one assign param name and type from super class and default value of the parameter.
- Second one assign name and type of parameter of super class.

* Getters and Setters.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
public class AddedParam extends Param {
    private String defaultValue;
    public AddedParam(Param param, String defaultValue) {
        super(param.getName(), param.getType());
        this.defaultValue = defaultValue;
    }
}

```

```

    }
    public AddedParam(Param param) {
        super(param.getName(), param.getType());
    }
    public String getDefaultValue() {
        return defaultValue;
    }
    public void setDefaultValue(String defaultValue) {
        this.defaultValue = defaultValue;
    }
}

===== End of AddedParam class =====

```

12- RemovedParam Class

*** extends**

- o Param class.

*** Constructors**

- o assign values to name and type of the super class.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
public class RemovedParam extends Param {
    public RemovedParam(Param param) {
        super(param.getName(), param.getType());
    }
}

===== End of RemovedParam class =====

```

13- PairParam Class

*** Fields**

- o oldParam : the parameter which we want to update.
- o newParam : the updated parameter.

*** Constructors**

- o Assign values to old and new parameters.

*** Getters and Setters**

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
public class PairParam {
    private Param oldParam;
    private Param newParam;
}

```

```

private String map;
public PairParam(Param oldParam, Param newParam) {
    this.oldParam = oldParam;
    this.newParam = newParam;
}
public Param getOldParam(){
    return oldParam;
}
public void setMap(String map){
    this.map = map;
}
public String getMap() {
    return this.map;
}
public Param getNewParam() {
    return newParam;
}
public void setNewParam(Param newParam) {
    this.newParam = newParam;
}
public void setOldParam(Param oldParam) {
    this.oldParam = oldParam;
}
}

===== End of PairParam class =====

```

14 - UpdateOperationMap **Class**

* **Fields**

- oldOprName: name of the operation which we want to update.
- newOprName: name of the operation which we want to call instead of the old one.
- currentPairParam, getPairCounter, currentAddedParam, getAddedCounter, currentRemoveParam and getRemovedCounter: all these parameters are used to refer to pair parameters , added ones and removed parameters.
- pairs: vector of updated parameters as pairs.

- removedParams: vector of removed parameters.
- addedParams: vector of added parameters.

*** Constructors**

- Assign values to each of old and new operations names and initialize the other fields.

• Public Methods

- addPair, addAddedParam and addRemovedParam : all these methods add param to the suitable vector.
- getNextPair, getNextAddedParam and getNextRemovedParam: all these methods give the next parameter of the suitable vector.
- getParamOperationType : get some parameter and check if this param has been added, removed or updated and return the related operation which will be applied to the parameter.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.module;
import java.util.Vector;
public class UpdateOperationMap {
    private String oldOprName;
    private String newOprName;
    private int currentPairParam;
    private int getPairCounter;
    private int currentAddedParam;
    private int getAddedCounter;
    private int currentRemoveParam;
    private int getRemovedCounter;
    private Vector<PairParam> pairs;
    private Vector<RemovedParam> removedParams;
    private Vector<AddedParam> addedParams;
    public UpdateOperationMap(String oldOprName, String newOprName) {
        this.oldOprName = oldOprName;
        this.newOprName = newOprName;
        currentPairParam = 0;
        pairs = new Vector<PairParam>();
        removedParams = new Vector<RemovedParam>();
    }
}
```

```

        addedParams = new Vector<AddedParam>();
        getPairCounter = -1;
        currentAddedParam = 0;
        getAddedCounter = 0;
        currentRemoveParam = 0;
        getRemovedCounter = 0;
    }
    public void addPair(Param oldParam, Param newParam, String map) {
        PairParam pair = new PairParam(oldParam, newParam);
        pair.setMap(map);
        pairs.add(currentPairParam, pair);
        currentPairParam++;
        if (getPairCounter == -1)
            getPairCounter = 0;
    }
    public PairParam getNextPair() {
        if (getPairCounter == -1)
            return null;
        getPairCounter--;
        return pairs.elementAt(getPairCounter + 1);
    }
    public void addRemovedParam(Param oldParam) {
        RemovedParam param = new RemovedParam(oldParam);
        removedParams.add(currentRemoveParam, param);
        currentRemoveParam++;
        if (getRemovedCounter == -1)
            getRemovedCounter = 0;
    }
    public RemovedParam getNextRemovedParam() {
        if (getRemovedCounter == -1)
            return null;
        getRemovedCounter--;
        return removedParams.elementAt(getRemovedCounter + 1);
    }
    public void addAddedParam(Param newParam, String value) {
        AddedParam param = new AddedParam(newParam, value);
        addedParams.add(currentAddedParam, param);
        currentAddedParam++;
    }

```

```

        if (getAddedCounter == -1)
            getAddedCounter = 0;
    }
    public AddedParam getNextAddedParam() {
        if (getAddedCounter == -1)
            return null;
        getAddedCounter--;
        return addedParams.elementAt(getAddedCounter + 1);
    }
    public String getOldOprName() {
        return oldOprName;
    }
    public String getNewOprName() {
        return newOprName;
    }
    public void getParamOperationType(String paramName, Param
        param, String type, String value) {
        for (int i = 0; i < pairs.size(); i++) {
            if (paramName ==
                pairs.get(i).getNewParam().getName()) {
                param = pairs.get(i).getOldParam();
                type = "pair";
                return;
            }
        }
        for (int i = 0; i < addedParams.size(); i++) {
            if (paramName == addedParams.get(i).getName()) {
                param = addedParams.get(i);
                type = "added";
                value = addedParams.get(i).getDefaultValue();
                return;
            }
        }
        for (int i = 0; i < removedParams.size(); i++) {
            if (paramName == removedParams.get(i).getName()) {
                param = removedParams.get(i);
                type = "removed";
                return;
            }
        }
    }

```

```

    }
}
}

===== End of UpdateOperationMap class =====

```

15- Compiler **Class**

* **Fields**

- verbose : java compiler parameter.
- memoryInitialSize: java compiler parameter.
- memoryMaximumSize: java compiler parameter.
- quiet: java compiler parameter.
- debug: java compiler parameter.
- cp: classpath environment parameter.
- javacPath: java compiler path.

* **Constructors**

- Initialize all fields of the class.

* **Getters and Setters**

* **Public Methods**

- compile: take list of java source files and output directory path and compile the files using javac compile

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.outputUtilities;
import java.io.File;
import java.util.List;
import java.util.Vector;
import org.apache.xmlbeans.impl.common.IOUtil;
import org.apache.xmlbeans.impl.tool.CodeGenUtil;
public class Compiler {
    private boolean verbose;
    private String memoryInitialSize;
    private String memoryMaximumSize;
    private boolean quiet;
    private boolean debug;

```



```

private File[] cp;
private String javacPath;
public Compiler() {
    verbose = false;
    memoryInitialSize = CodeGenUtil.DEFAULT_MEM_START;
    memoryMaximumSize = CodeGenUtil.DEFAULT_MEM_MAX;
    quiet = true;
    debug = false;
    File[] tempClasspath = CodeGenUtil.systemClasspath();
    cp = new File[tempClasspath.length + 1];
    System.arraycopy(tempClasspath, 0, cp, 0,
                     tempClasspath.length);
    javacPath = System.getenv("JAVA_HOME") + "\\lib\\javac.exe";
}
public boolean compile(File[] srcFiles, String outputDir) {
    File outputDirFile = IOUtil.createDir(new File("."),
                                         outputDir);

    List<File> srcFilesList = null;
    Vector<File> filesVector = new Vector<File>();
    for (int i = 0; i < srcFiles.length; i++) {
        filesVector.add(srcFiles[i]);
    }
    srcFilesList = filesVector;
    System.out.println("Compiling Java source files...");
    return CodeGenUtil.externalCompile(srcFilesList,
                                       outputDirFile, cp, debug, javacPath,
                                       memoryInitialSize, memoryMaximumSize,
                                       quiet, verbose);
}
}

```

===== **End of Compiler class** =====

16 - ProxyCodeCreator Class

* Public Methods

- creatProxyJavaFile: take a service as input and return a string which is java source code for the proxy.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.outputUtilities;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.Service;
public class ProxyCodeCreator {
    public static String getProxyJavaFile(Service service) {
        String code = "";
        // packages section
        for (int i = 0; i < service.getPackages().size(); i++) {
            code = code + "import " + service.getPackages().get(i)
                + ";\n";
        }
        code = code + "import javax.xml.ws.WebServiceRef;\n";
        // class section
        code = code + "public class"
            + service.getPortType().getName()
            + "Proxy {\n";
        code = code + "@WebServiceRef(wsdlLocation =\""
            + service.getWsdلFileReader().getWsdلFileUrl()
            + "\")\n";
        code = code + "private static" + service.getServiceName()
            + " service;\n";
        // operations
        for (int i = 0; i < service.getOperations().size(); i++) {
            String returnType;
            if (service.getOperations().get(i).
                getReturnParam().getType() == null)
                returnType = "void";
            else
                returnType =
                    service.getOperations().get(i)
                        .getReturnParam()
                        .getType().getName();
            code = code + "public " + returnType
                + service.getOperations().get(i).getName()
                + "(";
            for (int j = 0; j <
                service.getOperations().get(i).getParams()
                    .size(); j++) {
                if (j == service.getOperations()

```

```

        .get(i).getParams()
        .size() - 1)
        code = code +
            service.getOpetaions()
                .get(i)
                .getParams().get(j)
                .getType()
            + " "
        + service.getOpetaions().get(i)
            .getParams().get(j)
            .getName();

    else
        code = code + service.getOpetaions().
            get(i).getParams().get(j)
            .getType()
        + " "
        + service.getOpetaions().get(i)
            .getParams().get(j)
            .getName() + ",";
    }
    code = code + ") { \n";
    code = code + "try {\n";
    code = code + service.getPortType().getName()
        + " port = service."
        + service.getPortType().getName()
        + "Port();\n";
    String returnPart;
    if (returnType == "void")
        returnPart = "";
    else
        returnPart = "return ";
    code = code + returnPart + "port."
        + service.getOpetaions().get(i).getName()
        + "(";
    for (int k = 0; k <
        service.getOpetaions().get(i).getParams()
        .size(); i++) {

```

```

        if (k == service.getOperations().
            get(i).getParams().size() - 1)
            code = code + service.getOperations().get(i)
                .getParams().get(k)
                    .getName();
        else
            code = code + service.getOperations()
                .get(i).getParams()
                    .get(k)
                        .getName()
                            + ", ";
    }
    code = code + ");\n}";
    code = code
        + "catch(Exception e) \n {
            e.printStackTrace();\n}\n";
}
code = code + "}"
return code;
}
}

```

===== **End of ProxyCodeCreator class** =====

17- UpdateServiceCodeChecker

*Fields

- oldService: the service which we want to update.
- newService: the updated service.
- map: save how to map between the two services.

* Getters and Setters.

* Public Methods

- checkCode: take a parameter of String of updatedCode and call other methods to check and extract

* Private Methods

- createXMLFile: take a string and save it as XML file then check it using the schema.
- checkServicePart: check service element of the XML file.
- checkOperationPart: check an operation element.
- checkAddParamsPart: check addedparams element.
- checkRemoveParamsPart: check removedParams element.
- checkUpdateParamsPart: check updatedParams element.

```
package com.yahoo.alwasouf.soa.tools.xmlToJava.outputUtilities;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.Vector;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.JAXPChecker;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.OperationInterface;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.Param;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.Service;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.UpdateOperationMap;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.WsdlFileReader;
public class UpdateServiceCodeChecker {
    private static Service oldService;
    private static Service newService;
    private static Vector<UpdateOperationMap> map;
    static {
        map = new Vector<UpdateOperationMap>();
        index = 0;
    }
    public static String checkCode(String code, WsdlFileReader reader) {
```

```

String message = creatXMLFile("xmlUpdateServiceFile", code);
if (message != null)
    return message;
JAXPChecker checker = new JAXPChecker("xmlUpdateServiceFile");
checker.check();
Node rootNode;
try {
    rootNode = checker.getParser().parse("xmlUpdateServiceFile");
} catch (SAXException e) {
    return e.getMessage();
} catch (IOException e) {
    return e.getMessage();
}
oldService = null;
newService = null;
map = new Vector<UpdateOperationMap>();
message = checkServicePart(rootNode,
                            oldService, newService, reader);
if (message != null)
    return message;

return null;
}

private static String creatXMLFile(String fileName,
                                    String FileContents) {
    File xmlFile = new File(fileName);
    Writer output;
    try {
        xmlFile.createNewFile();
        output = new BufferedWriter(new FileWriter(xmlFile));
        output.write(FileContents);
        output.close();
    } catch (IOException e) {
        e.getMessage();
    }
    return null;
}

```

```

private static String checkServicePart(Node rootNode,
                                       Service oldService,
                                       Service newService,
                                       WsdLFileReader reader){
    Node serviceNode = rootNode.getChildNodes().item(0);
    Element serviceElement = (Element) serviceNode;
    String oldServiceName = serviceElement
        .getAttributeNode("oldService")
        .getValue();
    String newServiceName = serviceElement
        .getAttributeNode("newService")
        .getValue();
    oldService = reader.getServiceByName(oldServiceName);
    newService = reader.getServiceByName(newServiceName);
    if (oldService == null)
        return "Error : Message (the old service name are
            not found in the services list, check it)";
    if (newService == null)
        return "Error : Message (the new service name are
            not found in the services list, check it)";
    NodeList operationNodes =
        rootNode.getChildNodes().item(0)
            .getChildNodes();
    for (int i = 0; i < operationNodes.getLength(); i++) {
        String message = checkOperationPart(operationNodes
            .item(i), oldService, newService);
        if (message != null)
            return message;
    }
    return null;
}

private static String checkOperationPart(Node operationNode,
                                       Service oldService, Service newService) {
    Element operationElement = (Element) operationNode;
    String oldOperName = operationElement.getAttributeNode("oldName")
        .getValue();
}

```

```

String oldReturnName = operationElement.getAttributeNode(
    "oldReturnType").getValue();
String newOperName = operationElement.getAttributeNode("newName")
    .getValue();
String newReturnName = operationElement.getAttributeNode(
    "newReturnType").getValue();
OperationInterface oldOpr = oldService
    .getOperationByName(oldOperName);
OperationInterface newOpr = oldService
    .getOperationByName(newOperName);
UpdateOperationMap tempMap = new UpdateOperationMap(
    oldOperName, newOperName);
if (oldOpr == null)
    return "Error : operation name("
        + oldOperName
        + ") is not found at the same service,
        you maybe mistyped the name, check it";
if (newOpr == null)
    return "Error : operation name("
        + newOperName
        + ") is not found at the same
        service, you maybe mistyped the name, check it";
if (oldReturnName != oldOpr.getReturnParam()
    .getType().getName())
    return "Error : you mistyped the return type of the
        operation"+ oldOperName;
if (newReturnName != newOpr.getReturnParam().getType().getName())
    return "Error : you mistyped the return type of the
        operation" + newOperName;
NodeList operationChildNodes = operationNode.getChildNodes();
String message = "";
for (int i = 0; i < operationChildNodes.getLength();i++) {
    if (operationChildNodes.item(i).getNodeName() ==
        "addParams") {
        message = checkAddParamsPart(
            operationChildNodes.item(i),
            oldService, newService, newOpr);
    } else if (operationChildNodes .item(i).getNodeName() ==

```



```

        "removeParams") {
            message = checkRemoveParamsPart(
                operationChildNodes.item(i),
                oldService, newService, oldOpr);
        } else if (operationChildNodes.item(i).getNodeName() ==
            "updateParams") {
            message = checkUpdateParamsPart(
                operationChildNodes.item(i),
                oldService, newService,
                oldOpr, newOpr);
        }
        if (message != null)
            return message;
    }
    map.add(index, tempMap);
    index++;
    return null;
}

private static String checkAddParamsPart(Node addNode,
                                         Service oldService,
                                         Service newService,
                                         OperationInterface
                                             newOpr){
    NodeList nodes = addNode.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++) {
        Element addElement = (Element) nodes.item(i);
        String name = addElement
            .getAttributeNode("name").getValue();
        String type = addElement
            .getAttributeNode("type").getValue();
        String defValue = addElement.getAttributeNode("defaultValue")
            .getValue();

        Param param = newOpr.getParamByName(name);
        if (param == null)
            return "Error :Added param " + name
                + " doesn't belong to operation"
                + newOpr.getName()
                + " parameters, check it.";
    }
}

```

```

        if (param.getType().getName() != type)
            return "Error :Added param type" + type
                + "doesn't match with type of "
                + name
                + "Param of operation"
                + newOpr.getName() + "check it.";
        map.elementAt(index - 1).addAddedParam(param, defValue);
    }
    return null;
}

private static String checkRemoveParamsPart(Node removeNode,
                                             Service oldService,
                                             Service newService,
                                             OperationInterface oldOpr) {
    NodeList nodes = removeNode.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++) {
        Element removeElement = (Element) nodes.item(i);
        String name = removeElement
            .getAttributeNode("name").getValue();
        String type = removeElement
            .getAttributeNode("type").getValue();
        Param param = oldOpr.getParamByName(name);
        if (param == null)
            return "Error :Added param " + name
                + "doesn't belong to operation"
                + oldOpr.getName()
                + " parameters, check it.";
        if (param.getType().getName() != type)
            return "Error :Added param type" + type
                + "doesn't match with type of "
                + name
                + "Param of operation"
                + oldOpr.getName()
                + "check it.";
        map.elementAt(index - 1).addRemovedParam(param);
    }
    return null;
}
}

```

```

private static String checkUpdateParamsPart(Node updateNode,
    Service oldService, Service newService,
    OperationInterface oldOpr, OperationInterface newOpr) {
    NodeList nodes = updateNode.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++) {
        Element updateElement = (Element) nodes.item(i);
        String oldName = updateElement.getAttributeNode("oldName")
            .getValue();
        String oldType = updateElement.getAttributeNode("oldType")
            .getValue();
        String newName = updateElement.getAttributeNode("newName")
            .getValue();
        String newType = updateElement.getAttributeNode("newType")
            .getValue();
        String mappingMethod = updateElement
            .getAttribute("mappingMethod");
        Param oldParam = oldOpr.getParamByName(oldName);
        if (oldParam == null)
            return "Error :Added param " + oldName
                + "doesn't belong to operation"
                + oldOpr.getName()
                + " parameters, check it.";
        if (oldParam.getType().getName() != oldType)
            return "Error :Added param type" + oldType
                + "doesn't match with type of "
                + oldName
                + "Param of operation"
                + oldOpr.getName() + "check it.";
        Param newParam = newOpr.getParamByName(oldName);
        if (newParam == null)
            return "Error :Added param " + newName
                + "doesn't belong to operation"
                + newOpr.getName()
                + " parameters, check it.";
        if (newParam.getType().getName() != oldType)
            return "Error :Added param type" + newType
                + "doesn't match with type of "
                + newName
    }
}

```

```

        + "Param of operation"
        + newOpr.getName() + "check it.";
    map.elementAt(index - 1).addPair(oldParam,
                                     newParam, mappingMethod);
    }
    return null;
}
public static Service getOldService() {
    return oldService;
}
public static Service getNewService() {
    return newService;
}
public static Vector<UpdateOperationMap> getMap() {
    return map;
}
}

===== End of UpdateServiceCodeChecker class =====

```

18 - UpdateServiceCodeCreator Class

* Public Methods

- getPrixyJavaFile: get old and new services and their mapping to produce java source code.

* Private Methods

- isItUpated: check if a specific method has been updated or not.

```

package com.yahoo.alwasouf.soa.tools.xmlToJava.outputUtilities;
import java.util.Vector;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.OperationInterface;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.Param;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.Service;
import com.yahoo.alwasouf.soa.tools.xmlToJava.module.UpdateOperationMap;
public class UpdateServiceCodeCreator {
    public static String getProxyJavafile(Service oldService,
        Service newService, Vector<UpdateOperationMap> map)

```

{

```

String code = "";
// packages section
for (int i = 0; i < oldService.getPackages().size();i++){
    code = code + "import " +
                oldService.getPackages().get(i) + ";\n";
}
for (int i = 0; i < newService.getPackages().size(); i++) {
    code += "import " + newService.getPackages().get(i) + ";\n";
}
code += "import javax.xml.ws.WebServiceRef;\n";
// class section
code += "public class" + oldService.getPortType().getName()
                + "Proxy {\n";
code += "@WebServiceRef(wsdlLocation =\"" +
                newService.getWsdLFileReader()
                .getWsdLFileUrl() + "\")\n";
code += "private static" + newService.getServiceName()
                + " service;\n";
for (int i = 0; i < oldService.getOperations().size(); i++) {
    String returnType;
    if (oldService.getOperations().get(i).getReturnParam()
                .getType() == null)
        returnType = "void";
    else
        returnType = oldService.getOperations()
                .get(i).getReturnParam()
                .getType().getName();
code += "public " + returnType + oldService.getOperations()
                .get(i).getName() + "(";
    for (int j = 0; j < oldService.getOperations()
                .get(i).getParams()
                .size(); i++) {
        if (j == oldService.getOperations().get(i).getParams()
                .size() - 1)
            code = code+ oldService.getOperations()
                .get(i).getParams().get(j)
                .getType()
                + " "

```

```

        + oldService.getOpetaiions()
        .get(i).getParams().get(j)
        .getName();
    else
        code = code
        + oldService.getOpetaiions()
        .get(i).getParams().get(j).getType()
        + " "
        + oldService.getOpetaiions()
        .get(i).getParams().get(j).getName()
        + ",";
}
code += ") { \n";
code += "try {\n";
code += newService.getPortType().getName()
    + " port = service."
    + newService.getPortType().getName()
    + "Port();\n";
// map into the new operation
int found = isItUpdated(oldService
    .getOpetaiions().get(i), map);
String returnPart;
if (returnType == "void")
    returnPart = "";
else
    returnPart = "return ";
if (found != -1) {
    String newOprName = map.get(found).getNewOprName();
    OperationInterface newOpr = newService
        .getOperationByName(newOprName);
    Param param = null;
    String dfValue = null;
    String paramOprType = "";
    String mappingMethod = "";
    String newOprCall = newOprName + "(";
    for (int k = 0; k < newOpr.getParams().size(); k++) {
        map.get(found).getParamOperationType(
            newOpr.getParams().get(k).getName(), param,

```

```

        paramOprType, dfValue, mappingMethod);
    if (paramOprType == "pair") {
        code += newOpr.getParams().get(k).getType()
                .getName()
                + " " + param.getName()
                + " = new"
                + newOpr.getParams()
                .get(k).getType().getName()
                + "();";
        code += mappingMethod;
    } else if (paramOprType == "added") {
        newOprCall += dfValue;
    } else if (paramOprType == "removed") {
        // nothing to do just ignore it ...
    }
}
for (int k = 0; k < newOpr.getParams().size(); i++) {
    map.get(found).getParamOperationType(
        newOpr.getParams()
            .get(k).getName(), param,
            paramOprType, dfValue,
            mappingMethod);
    if (paramOprType == "pair") {
        newOprCall += "("
            + newOpr.getParams().get(k)
                .getType().getName()
            + ")" + param.getName();
    } else if (paramOprType == "added") {
        newOprCall += dfValue;
    } else if (paramOprType == "removed") {
        // nothing to do just ignore it ...
    }
    if (k != newOpr.getParams().size() - 1)
        newOprCall += ", ";
}
code += returnPart + (oldService.getOpetaions()
        .get(i).getReturnParam()

```

```

        .getType().getName()) + "port."
        + newOprCall;
    } else {
        code += returnPart + "port."
            + oldService.getOpetations()
                .get(i).getName() + "(";
        for (int k = 0; k < oldService.getOpetations().get(i)
            .getParams().size(); i++) {
            if (k == oldService.getOpetations()
                .get(i).getParams()
                    .size() - 1)
                code += oldService.getOpetations()
                    .get(i).getParams()
                        .get(k).getName();
            else
                code += oldService.getOpetations()
                    .get(i).getParams()
                        .get(k).getName()
                            + ", ";
        }
    }
    code += ");\n}";
    code += "catch(Exception e) \n {
        e.printStackTrace();\n}\n}\n";
}
code += "};";
return code;
}

private static int isItUpdated(OperationInterface opr,
    Vector<UpdateOperationMap> map) {
    for (int i = 0; i < map.size(); i++) {
        if (opr.getName() == map.get(0).getOldOprName())
            return i;
    }
    return -1;
}

}

===== End of UpdateServiceCodeCreator class =====

```