Managing and modeling web service evolution in SOA architecture

Wei Zuo

N°d'ordre NNT : 2016LYSEI068

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

(L'institut National des Sciences Appliquées de Lyon)

Ecole Doctorale N° 512
(Informatique et Mathematiques)

Spécialité de doctorat : Informatique
Discipline : **Informatique**

Soutenue publiquement le 05/07/2017, par :
(Wei ZUO)

Managing and Modeling Web Service Evolution in SOA Architecture

---

Devant le jury composé de :

NURCAN, Selmin	HDR Université Panthéon Sorbonne, Paris 1	Rapporteur
VERDIER, Christine	Professeur Université de Grenoble Alpes	Rapporteur
MARET, Pierre	Professeur Université Jean Monet	Examinateur
AMGHAR, Youssef	Professeur INSA-Lyon	Directeur de thèse
BENHARKAT, Nabila	Maître de Conférences	Co-directrice de thèse

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

INSA LYON
Service Scolarité
Bât Camille Claudel

Année Universitaire 2015-2016

ECOLES DOCTORALES – SPECIALITES
A REMPLIR LORS DE VOTRE INSCRIPTION

A établir obligatoirement avec votre directeur de thèse

Nom : _____ZUO_____ Prénom : _____Wei_____ **Signature :**

ECOLES DOCTORALES n° code national	SPECIALITES	Cocher la case correspondante
ED CHIMIE DE LYON **(Chimie, Procédés, Environnement) EDA206**	Chimie	☐
	Procédés	☐
	Environnement	☐
HISTOIRE, GEOGRAPHIE, AMENAGEMENT, URBANISME, ARCHEOLOGIE, SCIENCE POLITIQUE, SOCIOLOGIE, ANTHROPOLOGIE (ScSo) EDA483	Géographie – Aménagement - Urbanisme	☐
ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE (E.E.A.) EDA160	Automatique	☐
	Génie Electrique	☐
	Electronique, micro et nanoélectronique, optique et laser	☐
	Ingénierie pour le vivant	☐
	Traitement du Signal et de l'Image	☐
EVOLUTION, ECOSYSTEMES, MICROBIOLOGIE , MODELISATION (E2M2) EDA 341	Paléoenvironnements et évolution	☐
	Micro-organismes, interactions, infections	☐
	Biologie Evolutive, Biologie des Populations, écophysiologie	☐
	Biomath-Bioinfo-Génomique évolutive	☐
	Ecologie des communautés, fonctionnement des écosystèmes, écotoxicologie	☐
INFORMATIQUE ET MATHEMATIQUES DE LYON (InfoMaths) EDA 512	Informatique	☐
	Informatique et applications	☐
	Mathématiques et applications	☐
INTERDISCIPLINAIRE SCIENCES-SANTE (EDISS) EDA205	Biochimie	☐
	Physiologie	☐
	Ingénierie biomédicale	☐
ED MATERIAUX DE LYON EDA 034	Matériaux	☐
MEGA DE LYON (MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE) (MEGA) EDA162	Mécanique des Fluides	☐
	Génie Mécanique	☐
	Biomécanique	☐
	Thermique Energétique	☐
	Génie Civil	☐
	Acoustique	☐

Pour le Directeur de l'INSA Lyon
Et par délégation
Marie-Christine BAIETTO
Directrice du Département FEDORA
Formation par la Recherche Et des Etudes Doctorale

Cette liste est mise à jour annuellement par le Département FEDORA en collaboration avec les Ecoles Doctorales.

Managing and Modeling Web Service Evolution in SOA Architecture

Abstract

The context of this thesis concerns the evolution of web services in SOA architectures. We mean by evolution all changes of one or more elements of the service contract resulting each time a new version of the service. In addition, we are in the event where versions of services are preserved and maintained as they cannot be all used simultaneously. We are also interested in this thesis, in the effects of these developments on the entire information system and the actors who interact with services. This work is therefore in the field of service versions management with significant ramifications in the areas of business processes and software development. To ensure a smooth and consistent transition between the different versions of a Web service, we advocate for a change-centric model in which necessary changes are identified, planned, implemented, tested, and then notified to all necessary stakeholders. A major consequence of changes in Web services is to review the mechanisms that bind organization applications to these Web services. This review is usually time-consuming and error-prone and sometimes requires the suspension of ongoing operations prior to shifting to new applications. To mitigate this review's consequences on applications, organizations tend to be passive by either ignoring the changes or delaying their adoption. In either case there is a high risk that providers of Web services stop supporting old versions (e.g., too costly to maintain), forcing organizations to take immediate actions, which could turn into chaos. Even if an organization is willing to embrace the changes, there are no guarantees that the transition to a new version will be a success. Organizations end-up using different versions of the same Web service, which is simply "unhealthy".

This thesis aims to build a holistic model for managing the Web Service evolution in service-oriented architecture taking into account services versions. The main work of this thesis is a set of theoretical models and approaches that facilitate the Web Service consumer and provider to handle the issues of Web Service evolution. Additionally, we also provide an implementation methodology which presents and validates the feasibility of the proposed model. Along with the theory and practice contribution of the work, we build a complete scenario to evaluate the whole work. The main contributions are i) the development of holistic theoretical change-centric model for managing Web Service evolution, ii) the change specification for representing Web Service evolution in the context of versions management, iii) the change impact analysis approach for Web Service evolution, and iv) a semi-automatic client adaptation for Web Service evolution.

Key words: Web Service, evolution, changes, dynamic, adaptation, WSDL, impact analysis, programming.

Gestion et Modélisation pour l'Evolution des Services Web dans L'Architecture SOA

Résume

Cette thèse traite de l'évolution des services web dans les architectures SOA. L'évolution s'entend ici comme tout changement impactant les contrats de service à chaque nouvelle version de service. Nous nous inscrivons volontairement dans le cadre de la préservation des versions et de leurs utilisations par des compositions de services ou par des applications quelconques. Nous nous intéressons également aux effets et impacts de ces changements sur l'ensemble du système d'information en particulier sur les acteurs et les processus d'affaires. Ce travail se situe principalement dans le domaine de la gestion des versions de services avec des ramifications dans le domaine des processus d'affaires et du génie logiciel. Afin de pouvoir utiliser des versions différentes en fonction de règles imposées par les consommateurs de service ou les fournisseurs, nous proposons un modèle capable de prendre en compte les changements en termes d'identification, de planification, d'implémentation, de tests et de notifications aux acteurs du système d'information. Dans ce contexte, un des problèmes majeurs est celui de relier les applications et les consommateurs aux nouveaux services. La résolution de problème est a priori difficile si on considère que les solutions qui pourraient y être apportées sont consommatrices en temps d'exécution, génératrice d'erreurs voire entrainant des arrêts de services. Ce coût du changement conduit souvent à ne pas entreprendre des évolutions ce qui en fin de compte est dommageable pour les organisations en général. Quoi qu'il en soit, les migrations de services d'une version à une autre peuvent conduire les consommateurs à éviter les nouvelles versions en dépit de la plus-value que ces dernières peuvent apporter car trop couteuses à maintenir).

Pour répondre à cette problématique, nous proposons un modèle holistique capable de décrire l'évolution des services dans les architectures SOA en prenant en compte les différentes versions de services durant leur cycle de vie. Ce modèle fait l'objet d'une méthodologie spécifique qui conduit à son implantation avec pour but de montrer sa faisabilité et sa validité. Cette méthodologie s'appuie sur un scenario qui permet de confronter toutes les notions du modèle. Plus précisément, nos contributions portent sur i) l'élaboration d'un modèle orienté-changement pour modéliser l'évolution des services, ii) une spécification semi-formelle pour la représentation interne de l'évolution en prenant en compte les versions de services, iii) une approche analytique pour interpréter l'évolution des services sur le système d'information, et iv) une adaptation semi-automatique de la partie client lors de l'évolution de services.

Mots-Clés: service Web, evolution, changements, dynamique, adaptation, WSDL, analyse d' impact, programmation.

Table of contents

List of figures

List of tables

1. Introduction

1.1 Research context

We live in a dynamic world. New business opportunities (e.g., fair trade) rise daily, new IT gadgets (e.g., tablets) enhance productivity, and new applications (e.g., Facebook) bridge the gap between people. Like any traditional software, Web services are subject to continuously changing and need to be dynamic so they can respond to users' constant changing needs and requirements. To cope with changes Web services are fine-tuned in terms of the functionalities they offer and the non-functional performance they achieve. We refer to the fine-tuning as Web service evolution. It is defined as a continuous development process of a Web service through a series of consistent and unambiguous change. Web service evolution is always expressed through the creation, provisioning, and decommissioning of different variants of the service called versions – during its life time.

To ensure a smooth and consistent transition between the different versions of a Web service, we advocate for a change-centric model in which necessary changes are identified, planned, implemented, tested, and then notified to all necessary stakeholders. This model also establishes who did what, when, and where. A major consequence of changes in Web services is to review the mechanisms that bind organization applications to these Web services. This review is usually time-consuming and error-prone and sometimes requires the suspension of ongoing operations prior to shifting to new applications. To mitigate this review's consequences on applications, organizations tend to be passive by either ignoring the changes or delaying their adoption. In either case there is a high risk that providers of Web services stop supporting old versions (e.g., too costly to maintain), forcing organizations to take immediate actions, which could turn into chaos. Even if an organization is willing to embrace the changes, there are no guarantees that the transition to a new version will be a success. Organizations end-up using different versions of the same Web service, which is simply "unhealthy".

The context of this thesis concerns the evolution of web services in SOA architectures. We mean by evolution all changes of one or more elements of the service contract resulting each time a new version of the service. In addition, we are in the eventwhere versions of services are preserved and maintained as they cannot all be used simultaneously. We are also interested in this thesis, in the effects of these developments on the entire information system and the actors who interact with services. This work is therefore in the field of service versions management with significant ramifications in the areas of business processes and software development.

By studying the history of software engineering, it is commonly admitted that the most important features which the software engineers are seeking for are modularization and dynamization. The modularization requires that the different software modules should be separated with explicit boundaries to improve the efficiency of development and maintenance. The dynamization requires that the

software should be able to be adjusted at runtime to lower the cost of updating it by shutting it down. In addition, the need for managing versions of Web Services leads to some difficulties in the maintenance of applications. A software architecture which presents higher modularization and dynamization is usually proved more successful. During the past dozen years, most of the efforts are made for the dynamization feature. One of the best is so called Service Oriented Architecture (SOA). Within SOA, service consumer uses a set of strategies and constraints to discover and select the useful Web Services. Usually, once a Web Service is discovered and selected to be integrated in the business process, service consumer and service provider are in a long term relationship. Except particular situations, there is no need to repeat the search process to discover another service. Therefore, it is legitimate to ask what to do if one of the software functional modules is changed when the business is still working. The software evolution which safe dynamic feature brings more complex challenges particularly within a large scale distributed environment which is the case of SOA.

1.1.1 Web Service evolution in SOA

Web Service is a type of software and thus it is also subject of evolution. Compared to traditional software, Web service is a bit special because it brings both convenience and challenges to deal with the question of the evolution of SOA. These can be summarized in:

Conveniences

- SOA is designed with a set of standard specifications. The existing Web Service tools perfectly support modern SOA specifications like Service Component Architecture (SCA), Enterprise Service Bus (ESB), and BPEL etc. They provide a good platform to take further research on the specification of “changes”.
- Web Service separates the concerns of software. The participants do not need to know the implementation details of the Web Services. The problem of the evolution of the Web service will then decline in issues related to the interface, quality of service (QoS) protocols, the business semantics, etc.).

Challenges

- Web Service is deployed and works in distributed environments. Heterogeneous devices, languages, platforms and limited resources make the context of evolution quite hard. Modeling information system taking into account the Web Service Evolution is a complex task.
- Web Service often belongs to service composition. A change of one Web Service will propagate the impact to the whole business

process. The impact analysis will be more challenging when the Web service evolution happens in SOA.

– The participants[1] of Web Service have weak control of each other. SOA is designed with respect to the loose-coupling principle. The changes induced by Web Service evolution are difficult to overcome the participants.

W3C defines other notions attached to Web Service. One can highlight for example Business Process Execution Language (BPEL). Web Ontology Language for Web Service (OWL-S) which consists in a set of markup language constructs for describing the properties and capabilities of Web services. The Web Service evolution has to deal with the new features (semantics, business protocols etc.) that are different from the traditional software.

1.2 Motivation

The research in the domain of Web Service evolution aims at improving the enterprise agility and flexibility. When this is applied to the SOA context, modeling evolution of Web Service is even more urgent to be carried out. Normally, the problem of software evolution is handled through well designed application architecture to reach forward-compatibility and backward-compatibility. Usually this is carried out by people with a high experience in the field of software engineering. However, when the problem is applied in the SOA environment, the huge number of Web Services and the participants that are out of control result in great challenges for the traditional approaches.

For the Web Service providers, current tools such as Apache CXF, .Net, Eclipse, WebLogic and so on can easily help them to develop and deploy Web Services. However, such tools are limited to establish a mechanism to facilitate evolution-oriented Web Service development. Firstly, the Web Service changes are not formally modeled and propagated. The changes only exist in the heads of the designers and engineers. Secondly, current tools do not support dynamic Web Service evolution. Most of them require the developers to modify the static source code of the Web Services rather than the abstract service models, which decrease the agility of enterprise business. There is lack of the tools which can help the providers to change and deploy the Web Services with high level APIs at runtime.

For the Web Service consumer, when a Web Service has been changed by publishing a new version, the consumer may be interested in the new version. The consumer may also want to be consistent with the new version to profit from the advantages introduced by the new version such as bug fix, function enhancement, and improvement of Quality of Service (QoS). Moreover, some of the Web Service providers in some cases do not maintain a versioning strategy

[1] A participant can be any consumer, provider, developer, and broker of web service

for their Web Services. The substitution strategy is always applied for these cases. Therefore, the consumers are forced to deal with the changes of the Web Service evolution. No matter the consumers are passively or initiatively involved in the issue of Web Service compatibility, it is necessary to build the models and tools for them to deal with Web Service changes.

For the Web Service integrators who make use of a set of the existing Web Services to orchestrate or choreograph enterprise business processes, they will be also interested in the evolution of Web Services with publishing new versions. The Web Service evolution will firstly affect its client application. Then the impact of the Web Service evolution can be higher in terms of costs on the information system.

In summary, to perform dynamic software evolution for Web Services, it is necessary to develop a set of models and tools to support the Web Service stakeholders with the related issues. However, the research in the domain on Web Service evolution is still facing great challenges. In practical, most of the contributions are based on domain experiences and best practices to manage Web Service evolution. They always try to solve the problems using techniques of classic software engineering. In the community, researchers are still struggling with many issues in this field. These challenges will be presented in Part I.

1.3 Research questions

From the previous discussion, it is clear that there is necessary to deal with the issues of Web Service evolution in a context including services versions. A set of models and tools are required to support the stakeholders to produce, analyze and react to the changes that are involved with the evolution process. To properly define the boundaries of the research problem in this thesis, we adopt the following preconditions and assumptions:

1. We assume that either the Web Service or the Web Service client is designed under the principle of concerns separation. This can ensure that the change of the Web Service only refers to the change of Web Service interfaces. Any changes to the implementation of the Web Service do not affect the stakeholders.
2. We do not specify the pattern of Web Service composition in this thesis (choreography, orchestration or SCA). We assume that the model and the process of the Web Service composition are known in advance.

By defining the boundaries of the research, we decompose the problem of Web Service evolution into the following research questions:

1. How to model the Web Service evolution? How to model the Web Service changes and the behavior of the stakeholders during the Web Service evolution.
2. How to extract exactly and completely the changes for the stakeholders? There are many solutions in the community. However, a question remains: how to ensure the completion and accuracy.
3. How to evolve the Web Service at runtime in a graceful manner?
4. How to analyze impact of the Web Service changes? How to analyze the impact for the Web Service client and how to analyze the impact for the composed Web Services?
5. How to determine the compatibility of the Web Service client applications and the evolved Web Service?
6. How to adapt the Web Service client applications to the evolved Web Services in an automatically and dynamically way?

The term "Web Service" indicates the general concept for the popular software services based on Web applications. The theory model in this paper can be used to treat with any services in a uniform manner. However, in this thesis, we specify this term as the SOAP-based Web Service rather than RESTful Web Service. In Section I.5, we discuss why the SOAP-based Web Service is chosen.

1.4 Research methodology

This research aims to provide a set of models and tools to facilitate the Web Service stakeholders (provider, consumer and broker) to deal with Web Service evolution in the context of versions management. We are trying to build a systematic theory and related programming techniques to solve the problems. Thus, we focus on both the theory models and the engineering approaches. Particularly, we decompose the research process into 5 steps:

STEP 1: Problem Definition

First of all, we determine the problem that we will put effort to solve. The problem of Web Service evolution comes from the general software versioning issues. The goal in this step is to obtain the evolution related problems that are concerned by Web Service stakeholders and construct the boundaries of the research. However, when the research progresses the research questions and boundaries may evolve with the state of art of the research.

STEP 2: Dive into the State of Art

When the problem has been defined, the second step is to explore the literature to find out the explicit approaches in the community, which problems have been solved, and which are the current challenges. The literature can also provide a concrete catalog for the defined problem. This does much help for us to make clear the structure of the knowledge in this field and provide a holistic view of the research situation. In this step, we divided the major problem into several sub problems. Especially, we have studied both academic publications and industry attempts to establish the state of art.

STEP 3: Define the model

To solve a problem theoretically, the important step before developing the approach is to build a model for this solution. The theory model defines the roles and behaviors of the approach, indicates the path to solve the problems, and describes the system architecture of the solution. The model is an abstract of the solution to the problem that is defined in step 1.

STEP 4: Implementation

The research target is to build an approach to deal with the Web Service evolution problems. It must be partly or fully implemented to validate the availability and reliability of the approach since the research object is the type of software. Moreover, the implemented prototype can be also used to make the tests for evaluating the model.

STEP 5: Evaluation

When the prototype of the approach has been finished, it is time to work out a scenario to evaluate the model. We design a set of tests to obtain the experiment result of the proposed approach as well as the approaches listed in the literatures. In the end of this step, a conclusion is made to comment the experiment results. It must state what are the advantages and disadvantages of this model and if the model has solved the initial problem.

1.5 Contributions

This thesis aims to build a holistic model for managing the Web Service evolution in service-oriented architecture taking into account services versions. The main work of this thesis is a set of theoretical models and approaches that facilitate the Web Service consumer and provider to handle the issues of Web Service evolution. Additionally, we also provide an implementation

methodology which presents and validates the feasibility of the proposed model. Along with the theory and practice contribution of the work, we build a complete scenario to evaluate the whole work. We present a list here to summarize the main contributions.

- **Holistic theoretical change-centric model for managing Web Service evolution**. Unlike the contributions listed in chapter 2 which provide some approaches to address the problem, this thesis presents a full-stack theory and solution to deal with Web Service evolution. This work firstly defines all the types of roles and behaviors during Web Service evolution. Secondly, this work covers the whole lifecycle of Web Service changes during the evolution process. By this way we consider a family of Web Service composed by all its versions.
- **Change specification for representing Web Service evolution in the context of versions management.** This thesis uses a theoretical and formal model to define and represent the Web Service changes during the evolution. Along with the formal description of the Web Service changes, a set of traces is also built for describing Web Service changes based on XML.
- **Change impact analysis approach for Web Service evolution.** Analyzing the impact of Web Service changes is an important task for all the approaches that are related to Web Service evolution. To deal with the evolution for the Web Service consumers, the first step is to obtain the impact that may be caused by the evolution. In this thesis we study the current analysis approaches and then propose an impact analysis model for both of the client applications and the Web Service compositions.
- **Semi-automatic client adaptation for Web Service evolution**. To facilitate the Web Service stakeholders to react to the Web Service evolution constitutes the last contribution of our work. Through analyzing the impact of the Web Service evolution, we propose a model and a recommended architecture for the client applications of the Web Service. This model defines and determines the interface compatibility between the Web Service and its consumers. Moreover, a prototype is presented to evaluate the model on JavaEE platform.

The rest of the thesis will be organized into the following parts. Part I presents the related works related to the software evolution and Web Service evolution. In Part II, Section II.1 introduces the motivation scenario that we use to describe our problems. Then in Section II.2, a theory model for Web Service evolution is presented. In Section II.3, we describe how to implement the proposed model in Section II.1. Section II.4 will take an evaluation on the whole

model. Finally in Part III, we will make a conclusion on this thesis and have perspectives in future research.

Part I: State of art

I.1 State of art overview

The issue of Web Service evolution comes from the domain of software maintenance and their configuration. Given that companies take a service orientation and choose to migrate their software on service-oriented architecture, it is urgent to improve the theories around the management of the evolution of services in this context of SOA. So the Web Service evolution becomes a widely discussed topic in the community. In this part of the thesis, we are going to analyze literature which covers both of the concepts of the Web Service versioning and the evolution of the Web Services in the SOA architecture. We have also covered the state of the art in the domain of the software development in order to review the tools and models which are used to deal with the software development.

In Part I, we study and present the state of the art in the field of the research on Web Service evolution. We investigate the evolution issues in software engineering, service-oriented architecture, change management and business process. We present both of the contributions and the issues brought by the literatures. For each literature listed in this chapter, we propose to outline the problems which are not addressed. Then we conclude this chapter with a summary of the challenges in the current state of art.

I.2 Software evolution

The earliest pioneering research on software evolution was conducted by M. M. Lehman et al in [6, 7, 8, and 9] since 1969. As Lehman writes in [6], the evolution is an intrinsic, feedback driven, property of software. At this stage, researchers established the point that the real world software systems require continuous changes and enhancements to satisfy new and changed user requirements and expectations, to adapt to new and emerging business models and organizations, to adhere to changing legislation, to cope with technology innovation, and to preserve the system structure from deterioration [7]. The software evolution is more like a general software maintenance and configuration.

According to Lehman's summary in [9] with his "35 years' study on software evolution", the nature of evolution can be concluded with its causes, properties, characteristics, consequences, impact, management, control and exploitation. Lehman and his colleagues consider that the changing and adapting requirements from the real world software systems drives the application evolve with an inevitable and continual feedback showing in Figure 1.

By studying the natures of software evolution and the IBM programming processes, Lehman contributed the 8 laws of software evolution in [9] that govern software evolution. According to Lehman, the 8 laws are still applicable to all the systems that are compatible with E-type specifications.

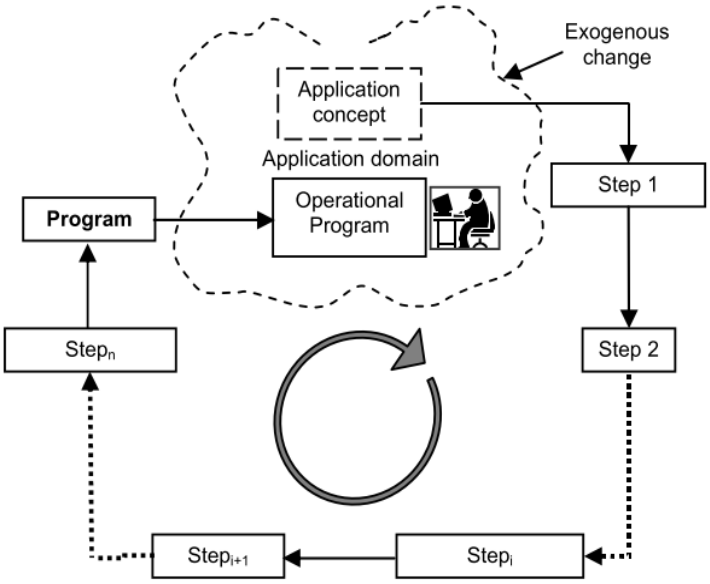

Figure 1 Lehman's feedback for software evolution [9]

The first three laws were proposed in 1974.

1. **Continuing Change:** The software system must be continually adapted, otherwise it will be less useful.

2. **Increasing Complexity:** The complexity of a software program will increase unless the people make efforts to maintain or reduce the complexity.

3. **Self-Regulation:** The software program evolution process is self-regulated.

Then in 1980, Lehman introduced 2 additional laws to the theory.

4. **Conservation of Organizational Stability:** The average effective global activity rate in a system is invariant during a product's life time.

5. **Conservation of Familiarity:** The next release of the evolved system should be statistically invariant to the previous release.

One more law was introduced in a footnote in 1991.

6. **Continuing Growth:** Functional content of a program must be continually increased to maintain user satisfaction over its lifetime.

The 2 remaining laws were introduced in [9] with a conclusion.

7. **Declining Quality:** The software system will be of quality if it is rigorously maintained and adapted to the changing environment.

8. **Feedback System:** The software programming process must be treated to be successfully modified or improved with multi-loop and multi-feedback systems.

Lehman's theory on the software system (or E-type system) establishes a basis of guidelines, methodologies and thoughts for further research in this field.

In the early stage, people discussed more about principles, laws and policies than concrete approaches or models that could be widely applicable.

I.2.1 Software evolution issues

Software evolution always implies two questions:

1. How to evolve the software?
2. How should one (re)act with software that has evolved?

Evolving the software in a better way and reacting to the software evolution in an automatic way are the two basic issues in the field of software evolution. For the first question, traditional solution is to improve the approaches of software development and deployment. To the second question, the users of services have to accommodate with incompatible interfaces of modified modules and must adapt to these changes. Normally the solutions to software evolution are categorized as static evolution and dynamic evolution.

- Static evolution: refactoring the software at development stage and redeploy the new version by shutting down the running application. Static evolution does not need to deal with the state transfer of software and can ensure the quality before it is online. However, static evolution makes the software temporarily unavailable and thus it may cause loses for enterprise business.
- Dynamic evolution: adjusting the behavior of the software at runtime without breaking down the business. Dynamic evolution improves the software adaptability and can be more competitive in the modern complex and distributed environments.

Dynamic software evolution is more compliant with the rapid development computing environment. This is why it attracted the attention of the research community. However, the problems that it brings are also challenging:

1. The dynamic software evolution removes the test stage from the development process. This requires that there must be a set of more effective mechanism to ensure the quality of the dynamic generated software.
2. The dynamic software evolution needs to deal with the impact that the evolution induces on the customers.
3. Dynamic software evolution brings the issue of how to deal with the stateful Web Service. The topic of the dynamic software evolution is proposed to meet the fast changing environment of the software. However, the evolved version of the software cannot be compatible with business processes without further intervention of the humans. For example, if the customer has finished the order online for booking a flight ticket and is going to pay when the website is

updating its software services, the information of the order may be lost or mismatched with the new version of the Web site.

4. Dynamic software evolution brings more complex challenges when people try to apply it to the large scale distributed environment, especially the Web Service dynamic evolution in the Service Oriented Architecture.

I.2.2 Dynamic software evolution and adaptation

The studies on software evolution conducted by Lehman have presented the basic rules for evolving or maintaining the software. They present the rules and models for software evolution allowing the software to change in the best way. However, most of the rules and models are just for solving the problems in general static maintenance or evolution of software [10-15]. With the further development of software engineering, the execution environment of software became more dynamic and complex. Traditional methods of maintenance and software evolution are facing major challenges related to various customers quickly changing needs. Therefore, some researchers begin to propose new models and tools for evolving the software at runtime to meet the more complex requirements.

In the past, the most common approach to perform the versioning to the software is to shut down the system and then install the new versions [16] according to Gupta D et al. However, this type of modification sometimes induces unacceptable delay to the business. Dynamic software evolution is different from the general traditional software maintenance and configuration. According to Gupta D, the latter one controls changes to preserve the integrity of the software when dealing with dynamic software system, but it still installs the changes in the traditional static manner. Dynamic evolution requires the software to be able to adjust its behaviors at runtime. Most of the adjustments may be taken to react to the changes of user requirements and execution environments. In this context, we can cite the following examples.

I.2.2.1 DYMOS

In the beginning of this research, researchers and engineers developed several systems that allowed runtime modification to the programs. Insup Lee presented the DYMOS (a DYnamic MOdification System) in 1983 in his PhD dissertation [17]. Before Lee's work, the modifications to the running program had to be done by patching machine code. DYMOS realized a mechanism to modify and recompile the source code of procedures and modules that need to be replaced. Then the modified code can be inserted to the program architecture to run parallel with the other processes.

As shown in Figure 2, in DYMOS the programmer passes the modification commands to the system. Four modules are prepared to handle the 3 types of commands.

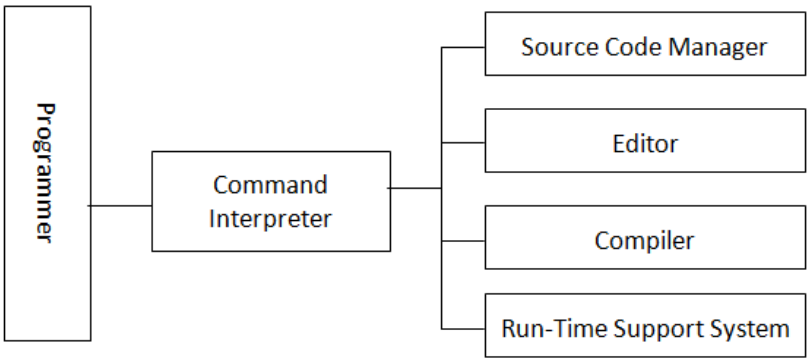

Figure 2 Overview of DYMOS architecture

- Edit command which is passed to the Editor module. The Editor Module locates and modifies the source code from the Source Code Manager module. The Editor module will generate a set of source code of a certain procedure or module.

  – Compile command which is passed to the Compiler module. The Compiler module compiles the modified source code from the Editor module and generates the new object code.

  – Update command which is accepted by the Run-Time Support System for the inserting of the new object code into the execution memory.

DYMOS can accept only the changes that will be correctly compiled with system. Actually, DYMOS only accepts commands with large granularity. The modification command encapsulates the logic for generating a procedure or module. The programmers are unable to get the details of the procedure or module. Generally speaking, the modified code is prepared in advance. The system is designed with an architecture that can support dynamically inserting program code into the system. In summary, DYMOS presents general principles for modifying the running software system. It can modify the running program with adding / updating the modules with the large granularity changes for the architecture. DYMOS is the foundation of the successive solutions to deal with the problems about runtime evolution. However, the modification is limited and this approach only supports the runtime evolution on the system designed with the specific architecture. In other words, users can modify the system dynamically with change actions such as “add a module” but cannot perform the change action such as “modify a parameter”.

I.2.2.2 K-Component

To achieve adaptable software evolution, it is widely accepted 2 types of approaches:

1. Separating adaptation logic from computational code.
2. Building adaptable architecture for dynamic evolution.

Regarding to the guidelines above, Dowling J. et al in [18, 19] choose to build dynamic software architectures to enable adaptable features of software systems. To achieve such dynamic architectures, the architecture meta-model so called K-component is introduced. K-Components are components with architecture meta-model and adaptation contracts to support their dynamic reconfiguration. According to Dowling, K-component builds configuration graph to describe the software architecture for the part of adaptation logic. In K-component, the reconfiguration of the software architecture is presented by pursuing graph transformation. To ensure the meaningful transformations, K-component framework introduces the adaptation contracts. An adaptation contract contains a series of conditional rules for the transformation of the meta-level configuration graph.

For the part of computational logic, K-component uses K-IDL (presented by OMG-CORBA) to define the components of the software. All the components are dynamically generated and compiled. The components can be either primitive or composite. K-component generates and compiles primitive components at runtime. The composite components are combined under the abstract configuration contracts.

According to the K-component and its previous contributions, an effective adaptive system must follow the 3 principles:

1. Dynamic code generation is the only way to realize software runtime evolution.
2. A system should be built with uncoupled components. The high dependency of the components on each other may lead to unexpected compiling errors.
3. To enable dynamic evolution, it’s unavoidable to specify the adaptable software architecture.

K-component framework defines a meta-model for software architecture to provide possibilities for dynamic adaptation. K-component does not model the changes when the software evolves. This makes it difficult the cooperation with the other participants in the system. Moreover, K-component does not explicitly explain how to realize the adaptation events, which is the challenge of software adaptation. K-component aims at achieving an evolvable architecture by defining the meta-model. But it does come with any implementations or approaches on components substitution, which is also important in the field of software evolution.

I.2.2.3 OSGi

To improve the efficiency use of the limited resources in the embedded devices, applications are often required to start only the necessary modules and stop the unnecessary modules at runtime. Then IBM and SUN developed an architecture specification called OSGi (Open Service Gateway initiative) [20]. It describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model. OSGi defines a rigorous component boundary by making use of the bytecode isolation mechanism of Java class loader. The static external dependencies of the component must be explicitly specified in the meta data file of each component. Such a modularization definition makes it easy to perform dynamic substitution for the components. OSGi can start, stop and replace any of its components at runtime. It's designed with a SOA similar service system. The service objects of the OSGi components provide software services by publishing the service to the registry of OSGi. OSGi maintains the service provider, broker and consumer. When the OSGi components are evolved with a new version, OSGi automatically forward the consumers' requests to the new service objects that come from the new version. OSGi solves the problems of software evolution from an engineering perspective and provides a complete implementation. However, OSGi does not concern the software adaptation. It will definitely produce errors when the updated OSGi services have been changed in interface. Moreover, just like any previous contributions, OSGi does not treat with objects state migration when updating the components. The substituted components will lose all the saved sessions or states.

I.3 Web Service evolution

I.3.1 Service oriented architecture

Service-oriented architecture is a model that is used to transform the components of an information system into services which can be integrated to build cross-business processes. Services are provided to other components via a communication protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology.

This model often uses the industry standards such as Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI). SOA has been quickly and widely accepted with the following principles that bring great advantages for enterprises business:

- **Standardization**. SOA requires using a set of common languages for the distributed heterogeneous applications to publish, discover and communicate with each other.

- **Modularization**. A functional module is defined as a service which is suggested to be highly encapsulated with larger granularity.
- **Separation of concerns**. A service uses standard interfaces to expose its internal logics. The details of the implementation are obviously hidden. This allows implementing the service with different technologies and platforms.
- **Autonomy**. A service should have full control over its internal logical.
- **Loose-coupling**. SOA advocates weakening the control of the software services on other services. Furthermore, a service should avoid of directly requesting another service.

Compared to the other principles, the most important one is the standardization. It defines the specifications for all the types of application to discover and communicate with each other. In one hand, this also makes it possible to make use of legacy systems to cooperate with modern novel technologies. In another hand, the standard interface description makes it possible to implement the service with different languages, platforms, object model, or messaging systems. To establish the standardization of SOA, W3C published the specification that uses a standard interface definition model called WSDL (Web Service Description Language).

WSDL defines the details of the abstract service information model. According to the specification of WSDL, a service is defined as a network endpoint, or ports. A WSDL document uses the following elements in the definition of a Web Service:

- **Types** – a container for data type definitions (such as XSD).
- **Message** – an abstract, typed definition of the data being communicated.
- **Operation** – an abstract description of an action supported by the service.
- **Port Type** – an abstract set of operations supported by one or more endpoints.
- **Binding** – a concrete protocol and data format specification for a particular port type.
- **Port** – a single endpoint defined as a combination of a binding and a network address.
- **Service** – a collection of related endpoints. [1]

Besides the standard interface description, SOA also defines a general communication model which is shown in Figure 3. Basically, there are 3 roles in this model.

- **Web Service provider (SP)**. SP designs, develops, publishes and maintains the Web Service.
- **Web Service broker (SB)**. SB maintains a Web Service Registry which stores the information of a list of Web Services that are published by different SPs.
- **Web Service consumer (SC)**. SC designs and develops the client applications which post invocation requests for the Web Services.

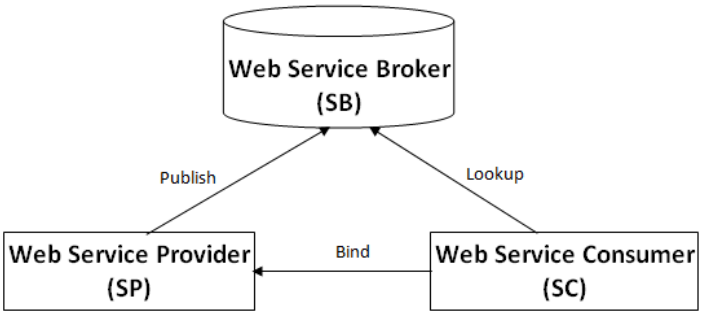

Figure 3 General SOA communication model

When the system is ready, SP publishes WSDL documents and the other information (Semantics, QoS, Yellow page etc.) to the Web Service Registry of SB. SC obtains the WSDL documents of its target service through the discovery action. Then the client application of SC creates a stub for local execution environment (for example a Java object in JVM) according to the obtained WSDL document. The stub handles the logic of the client application and encapsulates the invocation request into SOAP message for transferring on HTTP to the address indicated by the WSDL document. The Web Service parses and handles the SOAP message. Then it encapsulates the result into SOAP message and returns it to the client application of SC.

I.3.2 Web services evolution issues

The questions are similar as traditional software evolution:

1. How to model Web Service evolution?
2. How to analyze the impact of Web Service evolution on applications and information system?
3. How to perform adaptation when dealing with Web Service evolution?

To face these challenges, researchers have contributed with lots of approaches, models and tools.

The key problem of service evolution is that the compatibility between the service and its consumers may change when the service evolves. One of the major objectives of the research on Web Service evolution is to reduce the

unexpected effect caused by incompatibilities. As concluded by Andrikopoulos in [21], we also categorize the approaches of service evolution as corrective and preventive evolution.

Corrective – adaptation-based approaches that actively enforce the non-breaking of existing consumers by modifying the service, and

Preventive – that attempt to confine and forbid changes that would disrupt the consumers (instead of fixing them).

I.3.3 Corrective approaches

Some researchers consider that the evolution of Web Service is not controllable for all the actors. Because the Web Service provider and consumer have weak control on each other, none of them could predict or expect the changes of the Web Service in future. Then the distributed environment is supposed to produce service mismatches all the time. The corrective approaches aim at eliminate these mismatches between Web Service provider and consumer. In these contributions, service adaptation is always adopted to minimize the impact of the mismatches.

Adaptation has been introduced in the component-based software area where adapting a component-based system means modifying one or more of its components. In practice, most components cannot be integrated directly into a system because they are incompatible. The adaptation component aims to generate the most automatic as possible, adapters to compensate the gap between the interfaces and / or components behavior. Several adaptation approaches have been proposed for example in [22-27].

In [22] the authors propose a model-based adaptation approach focusing on software interface mismatch appearing at the behavioral level. The approach takes as input the behavioral interfaces of components to be adapted, and an adaptation contract - an abstract description of the constraints which must be respected to make the involved components work together. Given these two elements, an adapter protocol is generated in an automatic way. A synchronous vector method is provided for the adaptation contract language to make explicit the interactions. The work in [23] focuses on the signature level component adaptation such as names and parameters, and proposes a checking mechanism to find the signature level mismatch. In [24], the authors build an approach that uses a classification of component mismatches and identifies some patterns to be used for eliminating them. In [25], the authors address the problem of whether incompatible component interfaces can be made based on game theory by inserting a converter between them which satisfies specified requirements.

Compared to traditional component adaptation, the research on Web Service adaptation focuses mainly on the evolution of the service interfaces and

compositions. The adaptation may happen at both of the service side and the consumer side to correct the mismatches.

I.3.3.1 Chain of adapters

Kaminski presented an adaptive design for Web Service evolution in [28] to realize backwards compatibility for the evolving Web Services. Kaminski considers that the traditional versioning approaches for Web Service failed in treating with the following issues:

- Keeping backward compatibility for the Web Service clients.
- One Data source for all the versions.
- Avoiding of bugs propagation caused by code duplication.
- Unconstraint evolution.

If the system does not cover these issues, the design may break some rules and principles of software engineering. So Kaminski developed a design called chain of adapters as shown in Figure 4 to deal with the versioning of Web Services.

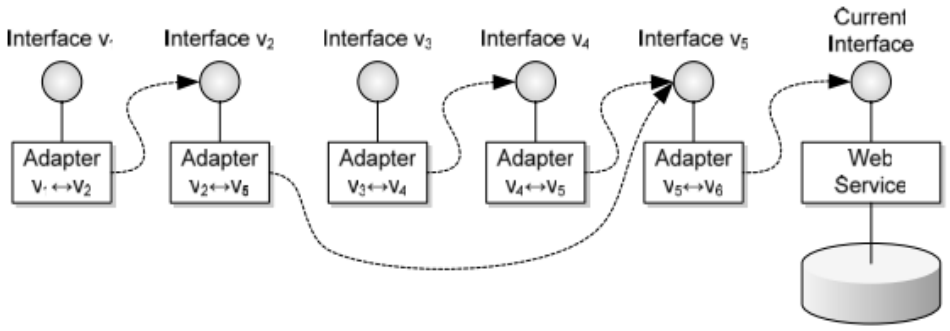

Figure 4 Kaminski's chain of adapters [28]

Kaminski develops one or several adapters for each version of the Web Service. Each of the adapters translates the user's request into a compatible one to its target version. An adapter could adapt one version to any target version that exists in the system. The adapters do not contain any business logic. In the whole system, there only exists one instance of the Web Service and one data source. The design of the adapter chain chooses to adapt the Web Service evolution at the service side since Kaminski believes that only the Web Service provider understands correctly what has been changed and how to adapt to the new versions. However, the disadvantages are also obvious:

1. Kaminski does not point out how to realize these adapters. There are no more steps to describe how to generate these adapters for dynamic and automatic adaptation.

2. When the Web Service is under great load pressure, the performance decreases because the chain of the translation actions of the adapters.
3. Not all the types of changes can be adapted to the current version. For example, if the consumer requests a deleted operation, the adapter has to request an old version of the Web Service, which breaks the initiative of this design.

Despite these drawbacks, the approach based on adapter chain solves the version problems based on the principles of software engineering. It explains how to manage Web Service evolutions more efficiently. Especially, it provides a possibility to enable backward compatibility for the evolving Web Service at the service provider side. However, it does not dive into all the problems that are related to the topic of Web Service evolution and adaptation such as the impact analysis.

I.3.3.2 WSDarwin

Marios Fokaefs presented WSDarwin and related contributions in several papers [29-32]. WSDarwin is a toolkit which could support the clients to co-evolve with the Web Services. Differently from Kaminski's adapter chain, WSDarwin attempts to solve the mismatches from the client side. First of all in [29], Fokaefs develops an approach called VTracker to compare the differences between XML documents. Using VTracker, the Web Service consumer could automatically determines the changes between two versions of the WSDL documents. Then in [30], Fokaefs describes the Overview of WSDarwin shown in Figure 5.

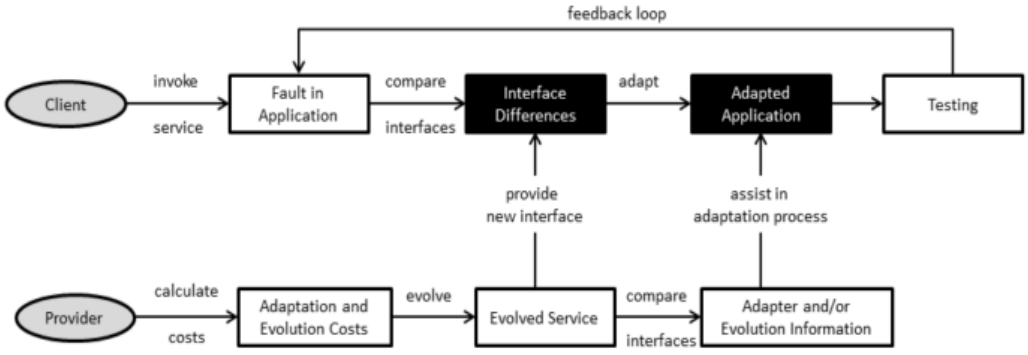

Figure 5 Overview of WSDarwin [30]

When the client proxy meets an invocation fault for one of the Web Service since the service has been evolved, the client proxy will uses the VTracker tool to compare the differences of the WSDL document from its previous version. Under the assistance of the provider proxy, the client proxy could automatically translate the requests to make it adaptable with new version of the WSDL document. The adaptation process shown in Figure 6 is similar as the generation of the adapters for the component adaptation.

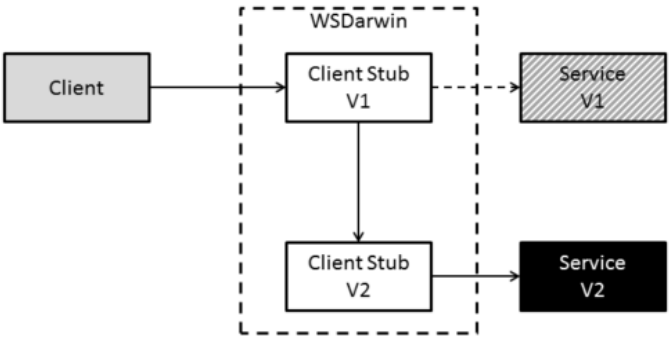

Figure 6 Adaptation process of WSDarwin [33]

The client proxy of WSDarwin will generate a client stub for the new version of the Web Service according to the changes that detected by VTracker. The generation processes is taken on the fly. So the invocation and adaptation is totally transparent to the client applications. WSDarwin provides a solution to help the Web Service consumer adapts to and survives in the "nature". This approach refers to Charles Darwin's *"It is not the strongest of the species that survive, but the one most responsive to change"*. Kaminski treats the Web Service clients as the nature and adapts the Web Service to the nature. Fokaefs treats the Web Services as the nature and adapts the Web Service clients to the nature. WSDarwin is a complete solution to deal with Web Service evolution including the changes discovery, adapter generation and impact analysis. However, some imperfections are still found in WSDarwin.

1. The principal drawback of WSDarwin and other similar approaches such as those presented in [33-37] is that they treat the syntactic differences of the interface documents as the changes when they are dealing with Web Service evolution. Most of them use the differential methods to obtain the differences of two Web Service versions by comparing the WSDL documents.
2. WSDarwin does not indicate how the Web Service provider could perform the adaptation assistance.
3. WSDarwin does not explain how to resolve the potential problems in the adaptation processes such as "how to deal with the dependencies and name spaces when generating and compiling the client stub". In fact, WSDarwin seems not implementable by most of the popular platforms such as Java.

I.3.3.3 Gensis

Treiber and his colleagues from Vienna University contributed the Gensis and related technologies in [38-42] for Web Service evolution. To cover this topic, Treiber et al. firstly analyze the Web Service changes in [38] and develops

Gensis to facilitate the programming evolvable Web Services. The Web Service changes that are triggered by different sources will have impacts on different parties. As shown in Table 1, they concluded the qualitative result of the impact of Web Service changes and the related entities that involved in the Web Service Evolution.

The entities implied in the Web Service evolution include the provider, developer, integrator, and user. The changing primitives include the interface, implementation, Service Level Agreement (SLA), QoS, usage, pre-pos conditions and feedback. This table encourages the researchers to make further contributions on the type, trigger and impact of the Web Service changes. Treiber et al did not cover all the types of changes in their successive work in [41, 42]. But they developed Gensis and a programming model as shown in Figure 7 for adaptive computing on the interface and implementation changes during Web Service evolution.

Gensis is a framework which provides high level programming APIs for modifying Web Service including the interface and implementation. Especially, Gensis reorganizes the Web services at the provider side and build a set of behavior modules for migrating to different hosts. A behavior module is an atomic entity which is composed of one or more Web Services. It is strictly self-contained so that it can be independently deployed. Gensis gives the system the ability of modifying Web Service at runtime. However, the programming model does not explain which kinds of changes need to be adapted. In fact, Gensis does not model Web Service evolution. It is just a programming framework for developing Web Service with high level APIs. Many of the issues involved in Web Service evolution such as the change management, versioning, and adaptation are not discussed.

Table 1 The change impact and Entities [41]

Observed Change	Trigger	Impact on	Modification of	Effect on
Interface	Provider, User, Service, Integrator	Integrator, Developer	Implementation	QoS, SLA, Usage
Implementation	Developer	Integrator, User	Implementation	QoS, Interface
QoS	Usage	Provider	Implementation, Interface	Interface, QoS, SLA, Usage
Usage	User	Provider	Contact user	SLA, QoS
Requirement	User, Integrator	Provider, Developer	Interface, Implementation, SLA	Usage
SLA	Provider	User, Developer, Integrator	Usage	Requirement, QoS, Implementation
Pre-Post Conditions	Provider, Developer	User, Integrator	SLA	Implementation
Feedback	User, Integrator	Provider, Developer	SLA	Usage

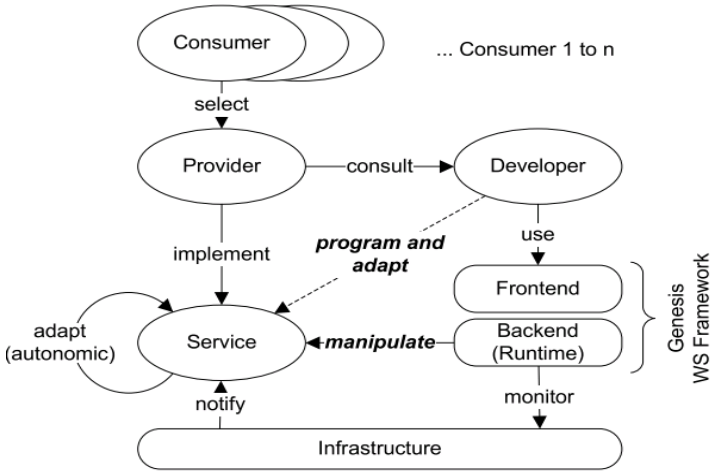

Figure 7 Treiber's Adaptive Programming Framework for Web Service Evolution [42]

I.3.4 Preventive approaches

Some other researchers, especially the ones from the industry field, advocate the preventive approaches when dealing with Web Service evolution for the following reasons:

1. The adaptation behaviors for both of the providers and consumers will lead the system to an irreversible chaos states since the corrective approaches assume that the evolution of the Web Service cannot be confined.
2. Not all the changes need to be adapted and not all the changes can be adapted without manual intervention.
3. The corrective approach does not lower the cost of the evolution of the Web service as they promise with an effort on automation. The cost of the effort is just moved to maintain the adapters.

Preventive approaches try to confine the changes that may be produced during the evolution process. They believe that the best practice is to evolve the Web Service in a more graceful way. They always build explicit or implicit agreements between the Web Service provider and consumer constraining the evolution of the services to make it compatible for the consumers. Contributions presented in [21, 43-58] fall in this type of approach. Most of them try to develop the compatible service versioning strategies to fit the requirements of the consumers. Andrikopoulos in his Ph.D thesis [21] has presented a typical and complete preventive solution for Web Service evolution. Before any further actions, preventive approaches firstly need to determine which changes are compatible. Andrikopoulos defines the Web Service full compatibility as the one which satisfies both horizontal and vertical compatibility as shown in Figure 8.

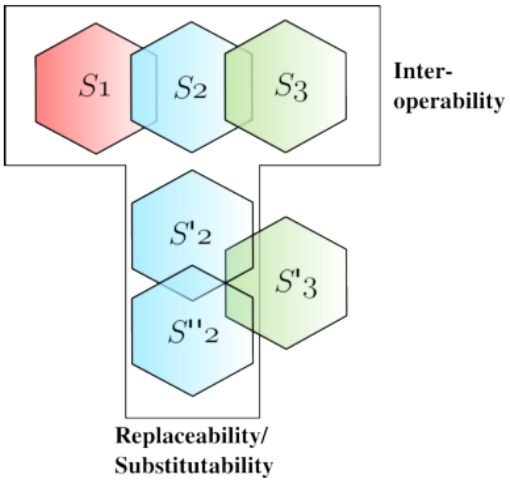

Figure 8 Andrikopoulos's Web Service compatibility [21]

Horizontal compatibility or interoperability of two services expresses the fact that the services can participate successfully in an interaction as service provider and service consumer. Vertical compatibility or substitutability (from the provider's perspective) or replaceability (from the consumer's perspective) of service versions expresses the requirements that allow the replacement of one version by another in a given context. The changes that meet both horizontal and vertical compatibility of the Web Service are called "T-Shaped changes" (as shown in the above figure). Secondly, preventive approaches should decide which changes are compatible for the evolution. Andrikopoulos concluded the backward compatible changes during his dissertation in Table 2.

Table 2 The compatible types of Web Service Changes

Change	Backwards Compatible
Add (Optional) Message Data Types to Input	Yes
Add (New) Operation (and respective Message Data Types)	Yes
Remove Operation	No
Modify Operation (Includes renaming and changing parameters, parameter order and message exchange pattern.)	No
Modify Message Data Types	No
Modify Service Implementation (As long as it has no effect on the service interfaces.)	Yes

According to Andrikopoulos, the implementation changes do not affect the invocation of the Web Service, so only the interface changes are considered. Finally, the preventive approaches introduce the service contracts to help the providers and the consumers to make an agreement on the use and the evolution of the Web Service. This service contract describes the changing patterns of the Web Service. It ensures that all the changes can fall in the T-Shaped changes set.

The adaptive and preventive approaches have been both widely discussed and applied. From the industry's perspective, preventive approaches seem more practical since it is easy to reach the goal. Conversely, the adaptive approaches

are facing great challenges when dealing with the adaptation dynamically and intelligently.

In practice, most of the Web Service changes are caused by changes of business, environment, policies and context. The adaptation for the Web Service interfaces [59-66] does not actually solve the problems. They could just provide a supporting platform and some possibilities to deal with these problems at the software engineering level. At the business level or the semantic level, these problems become more complicated. Researchers have to seek for more models and even more domain knowledge to improve the correctness and completion for the adaptation. Let's note that in this thesis, the proposed solution falls in the adaptive approaches from the following perspectives:

1. One of the principles of SOA is to uncouple the software components or services. More contracts between the Web Service providers and consumers will limit the development of SOA.
2. SOA systems should be designed with an open environment. Regarding the increasing of Web Service providers and consumers, the control of the stakeholders of each other will be weaker and weaker. However, the preventive approaches imply intervention on business processes to respect service contracts.
3. It is difficult to have an extensive and complete adaptive approach. Methods, models, and tools to standardize the Web Service evolution, are still necessary to reach the objective of a well adaptive approach.

I.4 Evolution impact analysis

Before performing an adaptive evolution of Web service, most approaches should carry out an impact analysis of changes. The approaches of the impact assessment are distinguished into three categories:

- Impact analysis at the service side.
- Impact analysis at the consumer side.
- Impact analysis on the Web Service compositions.

The first one is used to estimate the impact of the Web Service evolution on its consumers to decide if the Web Service should be evolved and which changes should be performed on the Web Service. The second one is considered to estimate the impact of the Web Service evolution on a single consumer for determine further strategies reacting to the evolution. The third one allows the estimation of the impact of the Web Service evolution on the whole business process. To complete Treiber’s qualitative impact analysis presented in Section I.3.3.3, in this thesis we will focus on both of the qualitative and the quantitative feature of impact analysis.

I.4.1 Impact analysis at the service side

Marcelo Yamashita in [67] presents the change impact analysis based on usage profile. This approach permits to empower providers with an understanding of the overall impact of changes in the whole set of client applications, enabling sound decisions on evolution strategies. The key point of Marcelo's approach is to introduce an interceptor within the Change Management Framework at the service side as shown in Figure 9.

The interceptor captures the interaction information between the Web Service and the consumer. Then the profile manager translates the raw information into "Usage Profile" which records how the Web Service consumers depend on the Web Service by several metrics. Marcelo's original metrics include: i) the number of applications and, ii) the number of requests of an operation or number of times a message was exchanged. Finally, the Version Manager will decide the compatibility between the new version and the client. The set of compatible changes and the determination algorithm are similar as Andrikopoulos's. All the analysis for the clients will be rolled up to the final result to reveal how much incompatibility will be induced by evolving the Web Service. After the analysis, the Web Service provider can estimate the risk of losing customers if he performs the evolution. This approach supports their decision making for evolving or not.

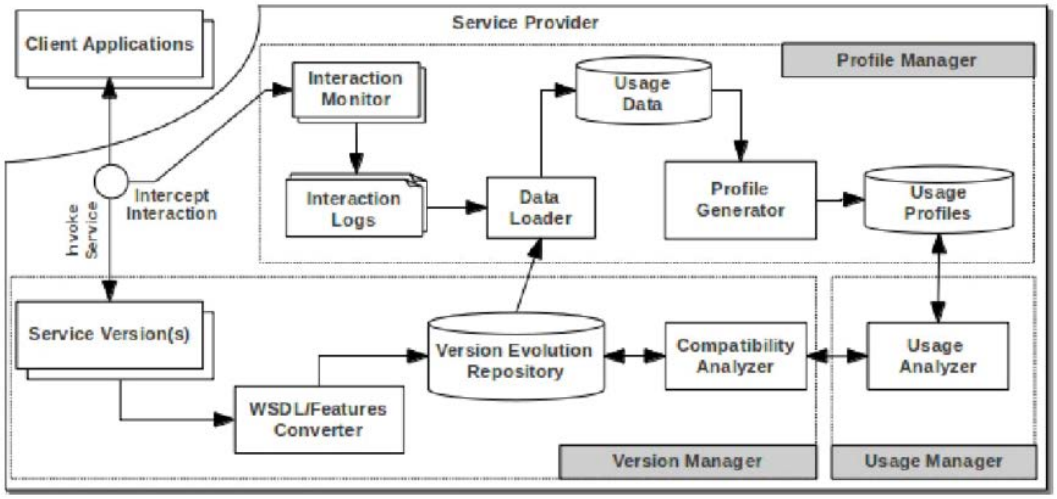

Figure 9 Overview of Marcelo's Change Management Framework [67]

Marcelo showed that the impact analysis of the Web Service evolution should be connected with the usage of the consumers. Unfortunately, Marcelo's contribution seems far from the final target of the research on Web Service evolution.

1. Marcelo does not specify a detailed model for the metrics of the Usage Profile. This question of which metric will be used is left for the users of the framework.

2. This approach can only answer the question of how much incompatibility will be induced by the evolution. It could not provide more precise prediction on the impact. All changes do not affect web services in the same manner.
3. This approach only supports the decision to make evolve the Web Service or not. It could not provide solutions to adapt to the evolution. It does not contribute anything on reducing the cost of evolution. It does not contribute anything on reducing the cost of evolution.

I.4.2 Impact analysis at the consumer side

Shuying WANG in [68-71] proposed a dependency impact analysis model for Web Service evolution. Dependency model is an approach for analyzing the dependency links among Web Services which work in collaboration. It extracts the degree of dependency for each link between the elements in one Web Service (so called intra relation) or between services (so called inter relation) as show in Figure 10.

Intra-service relation matrix of Customer service

	e1	e2	e3	e4	e5	e6
e1	1	0	0	0	0	0
e2	1	1	0	0	0	0
e3	1	0	1	0	0	0
e4	0	0	0	1	0	0
e5	0	0	0	1	1	0
e6	1	1	1	1	0	1

Inter-service relation matrix for Customer service to Order service

S2 \ S1	e1	e2	e3	e4	e5	e6
e1	0	0	0	0	0	0
e2	0	0	0	0	0	0
e3	0	0	0	0	0	0
e4	0	0	0	0	0	0
e5	0	0	0	1	0	0
e6	0	0	0	0	1	0
e7	0	0	0	0	0	1

Intra-service relation matrix of Order service

	e1	e2	e3	e4	e5	e6	e7
e1	1	0	0	0	0	0	0
e2	0	1	0	0	0	0	0
e3	0	1	1	0	0	0	0
e4	1	1	0	1	0	0	0
e5	0	0	0	0	1	0	0
e6	0	0	0	0	1	1	0
e7	0	1	0	0	1	0	1

Figure 10 Dependency Model for Impact Analysis [68]

Shuying counts all the elements which exist in the WSDL file and note them as changeable primitives. The model aggregates all the dependency relations within a dependency matrix. Then, Shuying obtains the final impact of the Web Service evolution by multiplying the dependency matrix by the changes matrix. It is a foundation for most of the successive works in this field such as [70-78].

Let's consider the figure from Shuying's model. For the customer service, e2->e1, e3->e1, e5->e4, e6->e1, e6->e2, e6->e3, and e6->e4 are defined as intra-service dependencies with a value 1. For the order service, e3->e2, e4->e1, e4->e2, e6->e5, e7->e2, e7->e2 are also defined as intra-service dependencies. For the relation between the order service and customer service, e5 of S2->e4 of S1,

e6 of S2->e5 of S1, e7 of S2->e6 of S1 are defined as inter-service dependencies. Any of the changes on an element will induce a change vector. For example, according to the intra service relation matrix, the change of element e1 for Customer service s1 will cause the change of the elements e2, e3, and e6. The change vector is then represented as <1, 1, 1, 0, 0, 1>. Then the impact of this change is expressed as the product of the dependency value by the change value (if changed it is 1 or not 0). The advantages of dependency model include: i) the ease of understanding and extending; ii) the quickness of implementing the model in real world systems. However, the disadvantages are also obvious:

- Too simple dependency between elements. In the dependency model, all the types of dependencies between elements are set to the same value. It does not consider the relationships betwin the elements in business and the practical cases.
- Manually retrieving dependencies. The model assumes that the dependencies are known at design time. When the system scale becomes larger and the Web Service evolves more quickly, the model will cost too many manual operations.
- Change types confusion. The model does not distinguish the change types add, remove and modify. It considers that each type of change result the same impact. Different types of changes will definitely induce different impact on the client.

Shuying WANG uses his dependency matrix to analyze the impact of the Web Service evolution on the business processes. It counts all the dependencies among all the member services in a composition and uses these dependency relations for calculating Web Service impact. Actually, what this approach calculates is the entropy of dependencies among all the member services. It provides an overview of the impacts but does not support any further actions such as Web Service adaptation.

I.5 Discussion

I.5.1 Why not RESTful Web Services?

When one talks about Web Services in the industry field, one always refers to the two types of Web Services:

- RPC style Web Service, especially the SOAP-based Web Service. It uses XML based standard document description (WSDL) and message transferring format (SOAP). SOAP-based Web Service follows the style of remote procedure call (RPC).
- REST (Representational State Transfer) Web Services or RESTful Web Services. Presented by Dr. Roy Fielding in 2000, it proposes to

treat the Web Service as a type resource rather than a remote procedure. It uses standard HTTP or HTTPS protocols and standard HTTP method such as GET, POST, DELETE and so on.

Both of the two types of Web Services are widely discussed and adopted in the enterprise applications. For a long time, the researchers and developers have taken much effort on the SOAP-based Web Service and its related techniques. SOAP-based Web Service has been extended with lots of industry specifications such as Enterprise Service Bus (ESB), BPEL-S, OWL-S, and Service Level Agreement. However, RESTful Web Service has captured attention from the practitioners of Internet since several leading enterprises (amazon, yahoo and google) have decided to migrate from SOAP-based Web Service to RESTful Web Service. Anyone who talks about the techniques of Web Service should not ignore the RESTful ones.

RESTful Web Service advocates to simplifying the service-oriented architecture. It abandons the XML format to exchange messages. Instead, JSON is the recommended format because of the fast parsing speed. RESTful proposes to use directly the standard HTTP methods to operate the Web Services that are considered as Web resources. RESTful Web Service does not constraint the approach to build it, so the Web Service providers can use any Web Server container to implement RESTful Web Services. Compared to SOAP-based approaches, RESTful Web Service does not constraint the format for exchanging messages, so it does not need to create a local agent for sending and receiving messages. Therefore, RESTful Web Service is considered more lightweight than the traditional RPC approaches.

Unfortunately, RESTful Web Service only defines the style to construct service-oriented architectures. Currently, it lacks of many industry standards which is important to realize software automation. RESTful does not define the approach and specification to describe the concrete Web Service information model. Now many restful Web Services cannot be discovered and invoked by software applications since they do not have standard description specifications such as WSDL. Though we have found that there are some attempts to deal with this problem such as Google Discovery service format, Web Application Description Language (WADL), and IBM's WSDL-based REST description, they do not seem to become or will become the industry standards. Most of the RESTful Web Service APIs are read and used by humans rather than software applications. This is useful when the enterprise is providing the relatively stable Web APIs to the public. However, when it is applied to the complex enterprise business processes, the lack of standard definitions for RESTful Web Services will greatly lower the software automation and intelligence. Relatively, SOAP-based approaches are supported by lots of languages and standard specifications, it is more suitable for solving the problems that come from the dynamic and complex distributed systems.

I.6 Summary

In this section, we have explored the related contributions in the field on Web Service evolution. We firstly introduced the early contributions on software evolution. Researchers focused more on the architectures to facilitate the software reconfiguration and the model of software evolution. Secondly we introduced the dynamic evolution and adaptation of software. When the software met the more complex environment in the distributed computing world, the software evolution was facing great challenges in reacting to the environment at runtime. In this stage, people were trying hard to develop the architectures and tools to enable the software being reconfigured and modified dynamically. Thirdly, we explored the issues about Web Service evolution. When more and more enterprises choose SOA as the principle to develop and deploy software services, the approaches for managing software services need to be renewed for reducing the cost of developing and maintaining applications. Especially, we compared the preventive approaches and adaptive approaches for Web Service evolution. Finally, we presented the impact analysis approaches when people deal with the Web Service evolution.

The selected contributions that we mentioned above have been proved successful for dealing with the problems that are related to Web Service evolution. However, challenges still exist to make the software be more close to the ideal shape in the field of Web Service evolution.

Firstly, the change management does not provide enough supports for enabling Web Service evolution. Most of the contributions, such as IBM Research's typical change management framework in [79-83] try to obtain the Web Service changes through comparing two versions of WSDL documents. However, as we have mentioned above, these differential approaches can only obtain differences, not changes. Different understandings for the results of the differential methods can lead to different results of the changes.

Secondly, there is still lack of models for Web Service evolution. Current models only focus on one or a few aspects of Web Service evolution. For example, Foaeks [29-31] focuses on the adaptation when the Web Service has been evolved. His approach is not interested to develop Web services. Andrikopoulos in [13] only focuses on the preventive evolution model. He does not pay attention to the impact and adaptation. Actually, each of the aspects of Web Service evolution should be modeled in a holistic way.

Thirdly, the impact analysis of Web Service evolution only stays in the stage of quantitative analysis. It provides the reference solution to the further strategies of the Web Service stakeholders. However, it is not the solution.

Table 3 shows the aspects that the typical approaches related to the field of Web Service evolution cover and compare them to Wei's approach [100-103] in this thesis.

Finally, we have argued in the last Section I.5, why we did not deal with RESTful Web Services.

What we are seeking for are a set of models, tools, and frameworks to facilitate stakeholders to deal with Web Service evolution more efficiently. The proposed approach in this thesis aims at reducing the manual cost when evolving, analyzing, and reacting to the Web Service evolution.

Table 3 Comparison of the Web Service evolution related approaches

Aspects / Approache	Evolution Model	Impact Analysis	Client Adaptation	Service Adaptation	Programming Support
Andrikopoulos [13]	Preventive	None	None	None	None
Kaminski [18]	Corrective	None	None	Adapter-Chain	None
WSDarwin [20]	Corrective	None	Semi-automatic	None	None
Gensis [27]	Corrective	None	None	None	Strong
Marcelo[44]	None	Usage-Profile	None	Future Plan	None
Wang Shuying [45]	None	Dependency Model	None	None	None
Wei[100-103]	Change-centric Corrective	Precise	Semi-automatic	None	Strong

Part II: Contributions

II.1 Motivation scenario

For explaining the proposed approach in a practical way and validate it with simulation, in Section I.1, we will present a scenario which presents the problems of Web Service evolution from the perspective of software engineering. As we mentioned for several times, the proposed approach aims at reducing the Web Service evolution related cost. As we know most of the cost comes from the development and maintenance of software. So in this section, we will present most of the technology details of the scenario such as WSDL descriptions and class graphs.

This scenario is based on the Travel Planning Service (TPS). TPS integrates thousands of Hotel Services (HS) and Train Ticket Services (TS) to provide the service of booking transport tickets and hotels for its clients. In normal situations, the clients can book any tickets and hotels that the TPS selects for them. However, when we deal with evolution of Web services, and we are faced with thousands of TS and HS, it is hard to make them evolve to improve the quality and functionality of services. Since there are thousands of competitors, the Web Service providers are facing great pressure for publishing new versions. For their client TPS, as the Web Services HS and TS are continuously evolving, it has to adjust the client application and usage strategies to ensure the business process not to be broken down. Firstly, it should obtain the changing information of the Web Services. Secondly, it should find out what kind of changes can affect and how they affect the client applications. Finally, TPS needs to determine how to adapt the client applications to the evolution of HS and TS.

We present the scenario in this section to describe the motivation of the proposed approach. In the rest sections of Part II and Part III, we will review this scenario to present examples for the theoretical model. We will also present how the approach solves the problems of this scenario in the evaluation section in Part II.

Travel Planning Service (TPS) is a scenario for describing how to realize the service of travel planning by Web Service based applications. A customer uses some basic information as the input to obtain the reservation for the transportation and accommodation. TPS calculates the best itinerary and accommodation places for the customer and generates the bills for paying. Then the customer pays for the bills at the Bank Service. Finally, TPS obtains the reservation of the transportation and accommodation and return to the customer. Figure 11 shows the process of the operation “plan travel”. The thin solid arrows show the mapping relationship between the process and the Web Service.

Each process for this operation has invoked a Web Service in this system. Generally, TPS depends on HS, TS, and BS (Bank Service) for this operation as shown in Figure 11 with dotted arrows. The composition of this operation is hard coded in TPS. TPS is a Web Service integrator and provides composite

Web Service to the customer. Our concern isn't the evolution of TPS because it does not affect the others stakeholders of this operation.

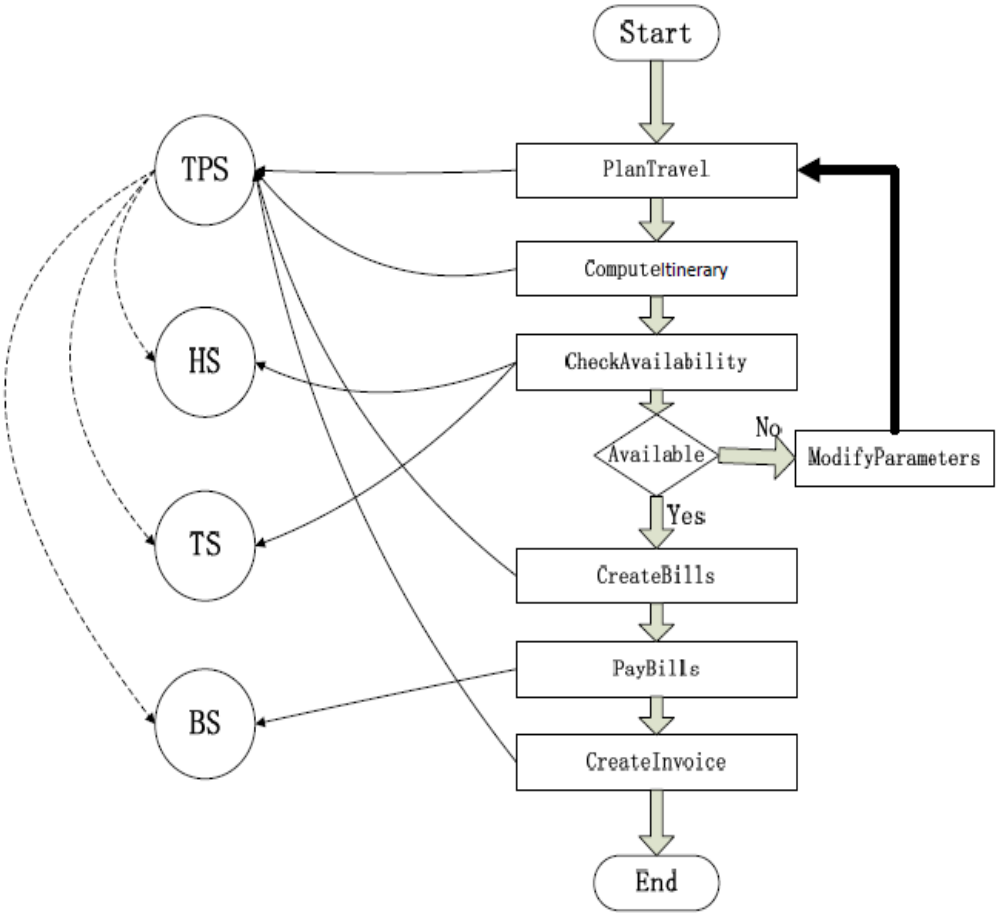

Figure 11 The working process of operation "plan travel"

II.1.1 Information model of the scenario

To describe the information models and the evolution issues of the involved Web Services, we need to present the Web Service description documents for each service. To make it more convenient for the reading, we only use the simplified descriptions. The detailed WSDL files will be presented in the appendix.

Figure 12 shows the portType of one of the TS "TrainTicketService". It contains two operations of "checkAvailable" and "bookTrainTickets". The operation "checkAvailable" is used by TPS to verify if the customer's request can be satisfied. The operation "bookTrainTickets" is used for perform the booking command.

The input and output messages of the operation "checkAvailable" are shown in Figure 13. The element "arg0" indicates the required departure time of the train. The element "arg1" indicates the number of the required tickets. The

element “arg2” indicates the identifier of the train. The output message of the operation returns a Boolean value to decide if the requirement is satisfied.

```
<wsdl:portType name="TrainTicketServiceInt">
  <wsdl:operation name="checkAvailable">
    <wsdl:input message="tns:checkAvailable"
    name="checkAvailable">
  </wsdl:input>
    <wsdl:output message="tns:checkAvailableResponse"
    name="checkAvailableResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="bookTrainTickets">
    <wsdl:input message="tns:bookTrainTickets"
    name="bookTrainTickets">
  </wsdl:input>
    <wsdl:output message="tns:bookTrainTicketsResponse"
    name="bookTrainTicketsResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

Figure 12 TrainTicketService portType

```
<xs:complexType name="checkAvailable">
  <xs:sequence>
    <xs:element minOccurs="0" name="arg0"
    type="tns:date"/>
    <xs:element name="arg1" type="xs:int"/>
    <xs:element minOccurs="0" name="arg2"
    type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="checkAvailableResponse">
  <xs:sequence>
    <xs:element name="return"
    type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

Figure 13 Input and Output Message of “checkAvailable”

The WSDL description of the input and output messages of the operation “bookTrainTickets” are similar as the ones of “checkAvailable”. The difference is that the operation “bookTrainTickets” returns a ComplexType called “PayModel”, which indicates the payment model for the travel plan. The structure of the “PayModel” is shown in Figure 14. The element “account” indicates the identifier of the target account that the customer wants to transfer money to. The element “bankService” indicates the reference of the bank service in a string value. The customer needs to transfer money to the specified account of the specified bank service.

Figure 15 shows the portType of one of the HS “YouthHotelService”. It contains 2 operations of “bookHotelRooms” and “checkAvailableRooms”. The operation “checkAvailableRooms” is used for obtaining the number of the available rooms with specified conditions. The operation “bookHotelRooms” is used for booking the rooms with the specified input message.

```
<xs:complexType name="payModel">
  <xs:sequence>
    <xs:element minOccurs="0"
    name="account" type="xs:string"/>
    <xs:element minOccurs="0"
    name="bankService" type="xs:string"/>
    <xs:element name="amount"
    type="xs:int"/>
    <xs:element name="transactionID"
    type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

Figure 14 The structure of the complexType "PayModel"

```
<wsdl:portType name="YouthHotelServiceInt">
  <wsdl:operation name="bookHotelRooms">
    <wsdl:input message="tns:bookHotelRooms"
    name="bookHotelRooms">
  </wsdl:input>
    <wsdl:output message="tns:bookHotelRoomsResponse"
    name="bookHotelRoomsResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="checkAvailableRoomNum">
    <wsdl:input message="tns:checkAvailableRoomNum"
    name="checkAvailableRoomNum">
  </wsdl:input>
    <wsdl:output message="tns:checkAvailableRoomNumResponse"
    name="checkAvailableRoomNumResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

Figure 15 The portType of "YouthHotelService"

Figure 16 shows the XML schema of the operation "checkAvailableRoomNum". The input message contains three elements. The element "arg0" indicates the room type with a string parameter. The element "arg1" indicates the start date of the reservation. The element "arg2" indicates the end date of the reservation. The output of this operation is an integer value which indicates the available number of the rooms that satisfy the requirements.

```
<xs:complexType name="checkAvailableRoomNum">
  <xs:sequence>
    <xs:element minOccurs="0" name="arg0" type="xs:string"/>
    <xs:element minOccurs="0" name="arg1" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="arg2" type="xs:dateTime"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="checkAvailableRoomNumResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

Figure 16 Input and Output Message of "checkAvailableRoomNum"

Similarly, the operation "bookHotelRooms" has the same input message. The output message of this operation is also a "PayModel" that is shown in Figure 14.

Figure 17 shows the portType of one of the BS “BankService”. It contains 3 operations of “pay”, “generateBillModel” and “checkBillPayed”. The operation “pay” is used for the customer to perform the payment action. This is done by TPS in this scenario. The operation “generateBillModel” is used by HS and TS in this scenario to create bills for paying. The operation “checkBillPayed” is used by TPS for verifying if the customer has paid the bill.

```
<wsdl:portType name="ChinaBankServiceInt">
  <wsdl:operation name="pay">
    <wsdl:input message="tns:pay" name="pay">
  </wsdl:input>
    <wsdl:output message="tns:payResponse"
    name="payResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="generateBillModel">
    <wsdl:input message="tns:generateBillModel"
    name="generateBillModel">
  </wsdl:input>
    <wsdl:output message="tns:generateBillModelResponse"
    name="generateBillModelResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="checkBillPayed">
    <wsdl:input message="tns:checkBillPayed"
    name="checkBillPayed">
  </wsdl:input>
    <wsdl:output message="tns:checkBillPayedResponse"
    name="checkBillPayedResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

Figure 17 The portType of the “BankService”

```
<xs:complexType name="pay">
  <xs:sequence>
    <xs:element minOccurs="0"
    name="arg0" type="xs:string"/>
    <xs:element minOccurs="0"
    name="arg1" type="xs:string"/>
    <xs:element minOccurs="0"
    name="arg2" type="tns:billModel"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="payResponse">
  <xs:sequence>
    <xs:element minOccurs="0"
    name="return" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Figure 18 Input and Output Message of “pay”

Figure 18 shows the XML schema for the operation “pay”. The element “arg0” indicates the identifier of the target account in this bank service. The element “arg1” indicates the source account which will transfer money. The element “arg2” indicates the object of the bill with a complexType of “BillModel”.

"BillModel" is requested by HS or TS and generated by BS. It is used for identifying the payment transaction which contains the information of transaction identifier and the amount of payment. The structure of the Bill is shown in Figure 19.

```
<xs:complexType name="billModel">
  <xs:sequence>
    <xs:element minOccurs="0"
    name="exchangeID" type="xs:string"/>
    <xs:element name="amount"
    type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

Figure 19 The structure of "BillModel"

The operation "generateBillModel" is requested by the enterprises which create accounts in this bank service. In this scenario the Web Service "YouthHotelService" and "TrainTicketService" do this. When there is payment request for HS and TS, they will invoke the operation "generateBillModel" to obtain the bill and send it back the customer. Then the customer pays for the bill and completes the process of the reservation. The operation "checkBillPayed" is used by HS and TS to verify if the bill is paid. This is not important for this scenario, so we ignore the descriptions.

II.1.2 Web Service implementation of the scenario

We have introduced the interface description of the Web Service as the information model. Generally, the Web Service consumers are only interested in the description documents of the Web Service such as WSDL, OWL-S, SLA, and so on. When the Web Service has been evolved, the change of the Web Service implementation sometimes implies the change of the Web Service semantics. It may not induce invocation errors for the Web Service consumers, but it cannot ensure the result of the invocation for the new versions is completely correct. However, at the Web Service provider's side, the evolution can usually happen on both of the descriptions and the implementations of the Web Service during the Web Service evolution. It is unavoidable to concern the implementation when talking about Web Service evolution. Web Service evolution involves the process of software development. One of the goals of the proposed approach is to reduce the cost of the Web Service evolution. A set of high level APIs must be provided to deal with the changes on both of the interface and implementation.

In this scenario, we do not provide the details of the implementation of each of the presented Web Services since it does not affect the evaluation in Part II. However, we will provide the method to deal with the changes of Web Service implementation.

II.1.3 Web Service evolution of the scenario

Normally, all the Web Services used by TPS will cooperate in long-time with it. However, the Web Service providers of HS, BS, and TS are continuously driving their developers and business designers to evolve the Web Services to meet the more and more complex environment and the competition pressure. Web Service evolves through publishing new versions with lots of changes generated. The reasons that drive the Web Service evolve can be fairly various. To fix the bugs, enhance the functionalities, and comply with the upper layer policies can all induce Web Service changes.

In this scenario, we assume that there are thousands of HS and TS but only one BS used by TPS. We assume that the thousands of HS and TS quickly evolve their Web Services. As a result, the development and deployment of the Web Service struggle hardly with the time. The client applications of TPS also encounter lots of troubles. It cannot predict and control the changes of the services. They have to develop methods and tools to obtain and deal with the changes during the Web Service evolution.

Table 4 The changes of TS

Change Primitive	Initiative	Change Action	Description
Operation	Enhance Functionality	add	Add a new operation named “bookFlight” to provide additional air transportation.
Element	Policy Change	add	The input message of the operation “bookTrainTickets” must be updated since the government requires this service to provide the customer personal information. But according to the policy, this is not necessary and is adaptable.
Element	Enhance Functionality	modify	The type of the element “arg0” of the input message “checkAvailable” will be replaced by the standard date type “dateTime”.
Implementation	Fix Bugs	modify	The implementation of the operation “bookTrainTickets” will be rewritten

In this scenario, each provider of the HS, TS, and BS evolves his Web Services in different ways. To simplify our presentation, we assume that we use the same model for HS and TS. Then we also assume that there are 4 types of changes for all the TS as shown in Table 4. Each HS performs all the changes by loop with rollback evolution. We assume that the whole system is deployed in a very large distributed environment. Each of the HS and TS will evolve in a very fast speed. In the evaluation section in Part II, we will set the evolution

frequency as that the change will happen every 3 seconds with this scenario. There is no possibility to employ someone to obtain, analyze, and adapt the client applications to the evolution in such a short time.

Similarly, we also design 4 types of changes for HS in Table 5.

Table 5 The changes of HS

Change Primitive	Initiative	Change Action	Description
Implementation	Enhance Functionality	modify	The provider ameliorates the quality of the service to provide better experience.
Element	Policy Change	add	The input message of the operation “bookHotelRooms” must be updated by adding an element “customer identifier” since the government requires this information when the customer wants to book a hotel.
Operation	Policy Change	modify	The name of the operation “bookHotelRooms” will be renamed to “bookRooms” for simplification.
Output Message	Policy Change	delete	The operation “checkAvailableRoomNum” does not need to return a message since it will be obtained by other means.

For BS, the change of BS does not affect very much the TPS since it is relatively stable. So we do not make it evolve in this scenario. All the Web Services are evolving with high speeds. The Web Service providers usually do not maintain all the versions of the Web Services since it will waste too much resources.

II.1.4 Summary

In this section, we have described the business process of an operation “plan travel” for TPS. We also presented the interface descriptions of each involved Web Service. Now we have got the full scope of the scenario. Thousands of HS and TS are constantly publishing new versions which contain the changes on the interface or implementation of their Web Services. We have also described the changes that will occur during the evolution. When TPS meets the evolution of HS and TS, the old version of the invocation can usually generate errors or incorrect results. At this time, he has always 3 options:

- Discard the evolved service and choose another similar one that satisfies the client application.

- Do nothing with the evolution and return an exception to the users.
- Modify the client application to adapt it to the evolving Web Service.

No matter which option will be chosen by TPS, it must obtain the information of the evolution, estimate the impact of the evolution and then make decision.

The first option seems feasible because it does not need to change the client applications and does not need to deal the Web Service evolution. If the first option cannot be realized, TPS has to choose the second option. In this scenario, we assume that the Web Services are similar on interface definition. However, they are actually different with each other since they have different business logics and semantics. The first option is nearly impossible in this scenario. As we have mentioned above, the Web Service provider does not maintain too many versions for one Web Service because it is a waste of resources. So TPS does not have too many chances to use the old version when the Web Services of TS or HS are evolving in such a high speed. Then if the Web Services that they are using have been evolved, there is no possibility to keep using the previous compatible versions of the Web Service. At last, TPS has to choose the second option to throw exceptions for the customers and interrupt the business.

Generally speaking, the third option is the best way to help TPS deal with the Web Service evolution of HS and TS. To adapt the client applications to the evolving Web Service, TPS needs to firstly discover exactly what the changes are, secondly find out if and which of the changes can be adapted, and thirdly decide how to take adaptation.

However, current approaches presented in Part I have less discussed the solutions which can cover all the issues mentioned above. We will start to present our contributions for the issues of Web Service evolution in Part II.

II.2 Change-Centric model for web service evolution

Everything evolves along with the time. To control the evolution, one must to catch the things which do not change. When the Web Service evolves along with the time, the only things that never change are the changes. Web Service delta is a set of changes from one version to its next version of the Web Service. Delta comes from the mind of the Web Service provider at design time and is executed by the Web Service developer. Once determined, delta will never change again and it keeps globally the system consistent during the whole lifecycle of the Web Service. Once designed in a standard and formalized way, it becomes the consensus for all the stakeholders to understand Web Service evolution. Most of the current approaches consider Web Service delta as the result of evolution so they propose different approaches to find it out. In this thesis, we take into account of another aspect of delta. We make that delta can be considered as both the reasons and results of Web Service evolution. Once delta comes out of the mind of Web Service provider, it involves all the stakeholders to take corresponding actions that are related to evolution. That means, all the actions that are performed under guiding of delta when evolution occurs in SOA include: 1) the actions of designing, executing, and publishing the Web Services versions belongs to the provider, 2) the actions of storing and distributing Web Service versions belongs to the broker, and 3) the actions of monitoring, analyzing, and adapting to versions belongs to the consumer. The Web Service provider develops delta rather than the Web Service versions. The Web Service broker manages delta rather than Web Service versions. The Web Service consumer reacts to the delta rather than the Web Service versions.

Figure 20 indicates that delta actually can be involved throughout the behaviors in SOA, which is so called “change-centric”.

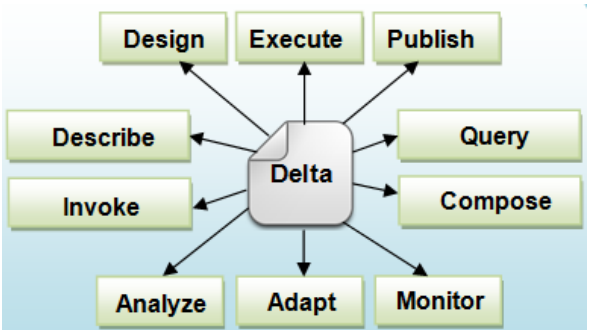

Figure 20 The “Change-Centric” SOA

II.2.1 Web Service changes

Current solutions to manage Web Service evolution have required too much work on the discovery of the Web Service changes. Typical approaches are analyzing the Web Service description documents (WSDL, OWL-S) to find out the differences between two versions of the Web Service. The advantage of such approaches is that it does not need to build any other mechanism to discover the changes. Current SOA system and related frameworks can well support the obtaining and parsing the description documents. However, the differences do not refer to the changes. The approaches which obtain the changes by analyzing the differences can usually lead to ambiguity. For example, imagine that we have a piece of WSDL description as shown in Figure 21.

```
<wsdl:portType name="TrainTicketServiceInt">
  <wsdl:operation name="checkAvailable">
    <wsdl:input message="tns:checkAvailable"
    name="checkAvailable">
  </wsdl:input>
    <wsdl:output message="tns:checkAvailableResponse"
    name="checkAvailableResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="bookTrainTickets">
    <wsdl:input message="tns:bookTrainTickets"
    name="bookTrainTickets">
  </wsdl:input>
    <wsdl:output message="tns:bookTrainTicketsResponse"
    name="bookTrainTicketsResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

Figure 21 The original version of the Web Service

And after the Web Service has been evolved, we got another piece of WSDL description as shown in Figure 22.

```
<wsdl:portType name="TrainTicketServiceInt">
  <wsdl:operation name="checkAvailable">
    <wsdl:input message="tns:checkAvailable"
    name="checkAvailable">
  </wsdl:input>
    <wsdl:output message="tns:checkAvailableResponse"
    name="checkAvailableResponse">
  </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="bookTickets">
    <wsdl:input message="tns:bookTickets"
    name="bookTickets">
  </wsdl:input>
    <wsdl:output message="tns:bookTicketsResponse"
    name="bookTicketsResponse">
  </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

Figure 22 The evolved version of the Web Service

Now there are two possibilities to obtain the changes by current differential approaches.

1. The semantics doesn't change: Updating the operation "bookTrainTickets" to rename it to "bookTickets".

2. The semantics changes: Delete the operation “bookTrainTickets” and add a new operation named “bookTickets”.

These two cases are both right if we use VTracker and similar tools to analyze the WSDL documents of the two versions. However, it is obvious that the two possibilities are totally different and will have different impact on the consumers. For the first possibility, the Web Service only changes the name of the operation “bookTrainTickets” and does not change the semantics, the implementation, and the other related primitives. The client applications only need to modify the interface to invoke the Web Service correctly. For the second possibility, the operation “bookTrainTickets” will be never available since it is deleted. The new operation “bookTickets” will have new semantics and cannot be directly used by the old version of the client applications without any analysis. Whatever the choice adopted by different tools, one can not really understand what were the changes.The Web Service changes come from the mind of the Web Service provider who is responsible of the maintenance and development of the Web Service. The Web Service provider is the first one and the only one who understands what the changes really are when the Web Service has evolved. According to this assumption, we consider to model the Web Service changes at the provider side and propagate the changes over the system. The underlying framework that we will present in this thesis will be responsible for hiding the details of the creation and propagation of the Web Service changes. It helps the system to control the whole lifecycle of the Web Service changes.

II.2.1.1 Roles involved in Web Service evolution

Before demonstrating how to control the changes in the system, we firstly define the participants that are related to the Web Service evolution.

Table 6 The Roles and Behaviors in Evolution SOA

Roles	Behaviors
Provider	• Designs / Creates / Modifies / Delete Web Services • Designs / Creates / Implements Web Service changes
Broker	• Maintains Web Service registries • Maintains Web Service changes registries • Propagates Web Service changes
Consumer	• Develops client applications • Subscribe evolution events • Reacts to the Web Service evolution

In typical SOA model introduced in the introduction section, the roles are provider, broker and consumer that are related to the Web Service interaction. The Web Service broker works like the Web Service registry which accepts the Web Service providers to publish their Web Services. The Web Service provider develops the Web Services and publishes them to the Web Service broker. The

Web Service consumer looks up in the registry of the Web Service broker to select the Web Service desired and bind to it.

When the issue of Web Service evolution is posed in the typical SOA model, the behaviors of each role will be a little different. We use Table 6 to describe the roles and the evolution related behaviors of each role.

We also use Figure 23 to describe the interactions among the roles to deal with Web Service evolution on the basis of the SOA model.

The provider plans, designs, and implements the changes and provides the infrastructure for deploying Web service instances. When a provider publishes a new version of a Web service on the registry of the broker, the Web Service changes from the previous version are also published along with the WSDL documents. Actually, the Web Service changes are extensions for the WSDL documents. The broker then adjusts its registry by keeping track of all the changes per version and notifies consumers of the changes. Consumer includes client application developers and final users of the Web services. When the consumer obtains the WSDL documents, he also obtains the Web Service changes. Finally the consumer might decide to adapt their client applications to respond to these changes.

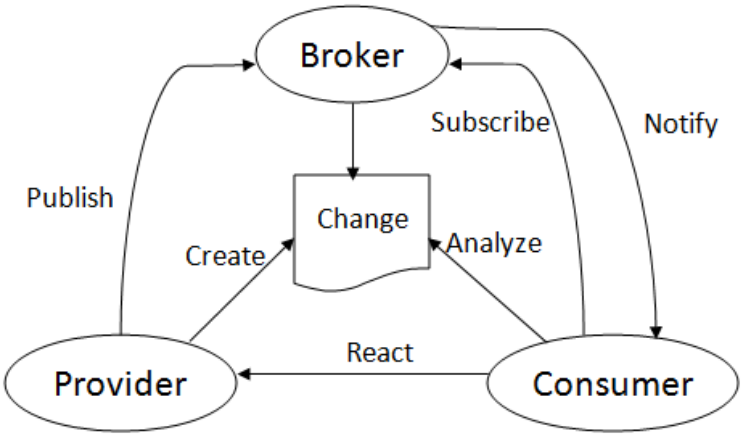

Figure 23 Web Service Evolution in SOA

II.2.1.2 Change specification of Web Service

II.2.1.2.1 Classic models for Web Service changes

The Web Service changes are always related to the behaviors of the Web Service during the evolution. Thus, we will need a standard Changes Specification which ensures the consensus on the formalization of the Web Service changes for all the stakeholders.

Current Web Service information model such as WSDL and OWL-S do not report on the version related information. Classical contributions on the classification and description of the Web Service changes include Leitner [62] and Juric [83] which have presented their own changes models.

Leitner explicitly defines the primitives and types of changes as shown in Figure 24. This classification scheme distinguishes three top-level types of changes: (1) Non- Functional Changes are all changes in the non-functional interface of the Web service as defined above, (2) Interface Changes represent all changes in the functional interface, and (3) Semantic Changes cover all changes that are not contained in the WSDL description of the service, such as a changed understanding (but not changed structure) of operations, parameters or return values.

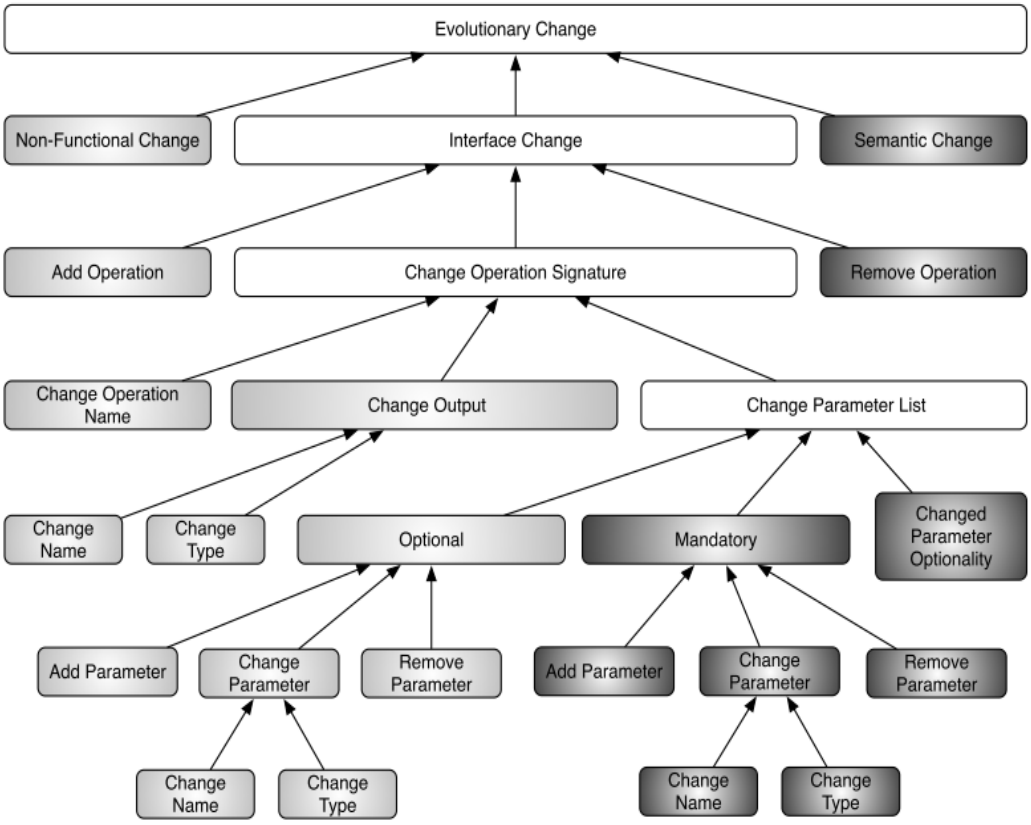

Figure 24 Leitner's Changes Model

Leitner's model covers most of the possible changes for the real world Web Service evolution. He does not think that the non-functional changes including QoS and semantic changes should be considered when dealing with Web Service evolution for the following reasons:

1. The QoS changes are monitored at runtime. One cannot define the exact values of QoS when evolving the Web Service.
2. The semantic description such as SAWSDL is not widely applied in the real world system.

In other words, Leitner only focuses on the interface changes to manage Web Service evolution. Unfortunately, not all the changeable primitives in WSDL interfaces are covered and there is no further discussion on how to model the Web Service changes formally. For example, the changes of the messages are not included in Leitner's model.

Unlike Leitner's model, Juric builds a complete set of XML extensions for current UDDI and WSDL specification to identify the Web Service changes. The main contribution of this approach is that it creates new tags for WSDL and UDDI documents for Web Service changes related description. For example, it introduces the tag <wsdlx:version> for WSDL documents as shown in Figure 25.

This type of description does not describe the changes. Instead, it describes the "differences".

```
<description xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://uni-mb.si/example">

 <!-- Version declarations -->
 <wsdlx:versions mode="serviceLevel">
  <wsdlx:version vid="1.0">
    <wsdlx:initialVersion/>
    <wsdlx:deprecated deprecationDate="2009-01-01"/>
    <wsdlx:versionInfo>Short description of version 1.0.</wsdlx:versionInfo>
  </wsdlx:version>
  <wsdlx:version vid="2.0" type="backward-incompatible" intent="revision">
    <wsdlx:previousVersion vid="1.0"/>
    <wsdlx:versionInfo>Modified XML payload, an obligatory element added.</wsdlx:versionInfo>
  </wsdlx:version>
  <wsdlx:version vid="2.1" type="backward-compatible" intent="variant">
    <wsdlx:previousVersion vid="2.0"/>
    <wsdlx:defaultVersion/>
    <wsdlx:versionInfo>Added new operation.</wsdlx:versionInfo>
  </wsdlx:version>
  <wsdlx:useDefaultInterfaceMapping
         mode="endpoint|parameter"
         default="true"/>     <!-- Optional for version unaware service consumers -->
 </wsdlx:versions>

 <wsdlx:version vid="1.0">  <!-- Definition of version 1.0 -->
   …
 </wsdlx:version>
 <wsdlx:version vid="2.0">  <!-- Definition of version 2.0 -->
   …
 </wsdlx:version>
 <wsdlx:version vid="2.1">  <!-- Definition of version 2.1 -->
   …
 </wsdlx:version>
</description>
```

Figure 25 Juric's WSDL extension

Juric uses the <wsdlx:version> tag to identify the different parts for each version of the Web Service. One WSDL document will contain all the descriptions of all the versions. Although it has some special tags for decreasing the unnecessary descriptions for versioning information, the size of the WSDL document will be also unacceptably huge if it is applied in a system that evolve quickly. When there are too many versions for one Web Service, the system will encounter performance decreasing on the following aspects:

- Huge size WSDL documents will block too much of the network bandwidth.
- It will cost too much time the parse the huge WSDL documents.
- The WSDL document will become unreadable.

II.2.1.2.2 Definition of Web Service changes

After having studied the existing models for Web Service changes, we worked out our own Changes Specification to formally describe the changes.

We define the Web Service information model to point out the primitives that are sensitive of Web Service evolution.

Definition II.1.1: a Web service is characterized by 3 dimensions WS^D where:

$$D = \begin{cases} \text{I: interface, WSDL annotations and XML Schemas} \\ S: \text{ Semantics, OWL based descriptions} \\ \text{Q: QoS properties} \end{cases}$$

Definition II.1.2: when a Web service WS_i is subject to changes, we consider the delta noted as Δ between two versions:

$$\Delta\,(v_j\,, v_{j-1}) = v_j(WS_i) - v_{j-1}(WS_i),$$

Similarly, we have

$$v_j(WS_i) = \Delta\,(v_j\,, v_{j-1}) + v_{j-1}(WS_i),$$

and

$$v_{j-1}(WS_i) = v_j(WS_i) - \Delta\,(v_j\,, v_{j-1})$$

This can ensure that each version of the Web Service is able to rollback and able to rollforward.

$v_j(WS_i)$ indicates version j of Web service i. One also notes the change description with three dimensions:

$$\Delta^D(v_j, v_{j-1}) = \{\ \Delta^I(v_j, v_{j-1}), \Delta^S(v_j, v_{j-1}), \Delta^Q(v_j, v_{j-1})\ \}.$$

Definition II.1.3: According to Definition II.1.2, Web service evolution is defined by:

$$\Omega^D = \{\Delta_j\ /\ \Delta_{j-1} \ll \Delta_j <=> v_{j-1}(S_i) \ll v_j(S_i)\},$$

The expression $a \ll b$ means that a precedes b.

Web Service interface changes can directly affect the invocation. So we must firstly define the Web Service interface.

Definition II.1.4: A Web Service interface $I = \{T, OP\}$, where T is a set of Web Service schema complex types (defined in WSDL) and $T = \{t_1, t_2, t_3 ... t_n\}$. t is a sequence of e which represents the schema element (tag $< xs : element >$ defined in WSDL) which belongs to the sequence of complex type t. $OP = \{It, Ot, N\}$ where It represents input message of the operation and $It = \{it_1, it_2, it_3 ... it_n\}$. Ot represents the output message of the operation and $Ot = \{ot_1, ot_2, ot_3 ... ot_n\}$. N represents the signature of the operation. And for the Web Service interface, we have:

$$\forall it \in It, ot \in Ot, t \in T, \exists e \to it, e \to ot, e \to t, e \in I\ .$$

Then we can define the Web Service changes on interface.

Definition II.1.5. $\Delta^I(v_j, v_{j-1})$ is denoted as a set of interface changes that $\Delta^I(v_j, v_{j-1}) = I_j - I_{j-1}$, where $\Delta^I = \{c_1, c_2, c_3 ... c_n\}$ and $c_n =< a_n, e_n >$. e represents a changed primitive in the Web Service. So we denote $\Delta^I =< A, I >$. A is a set of change actions among $\{delete, add, modify\}$.

We have explicitly defined the Web Service evolution and related interface changes in this section. As we have mentioned above, the Web Service changes comes from the Web Service provider's mind and are propagated by Web Service broker. In next section, we will introduce how to produce the formal described changes with our programming framework.

II.2.1.2.3 XML Annotation of Web Service changes

Unlike the previous description of Web Service changes, the proposed approach in this thesis extends current WSDL annotations to independently and directly describe the changes by <change> tags. Each WSDL document will have a version and change descriptions by extended tags. Table 10 contains a part of the tags, attributes and contents of the extensions. The XML annotation presented is the implementation of the abstract Change Specification.

Table 7 XML Annotations for Change Specification

Tags	Attributes	Values	Description
<changeoperation>	type *	{add, modify, delete}	The change types for the operation. The "modify" type only modifies the signature of the operation.
	name *	[xsstring]	The name of the operation that needs to be changed.
	adapt	[xsboolean]	Specify if the change is adaptable. "add" type is set to "true" by default, unless is "false" by default.
<changeinput>	type *	{add, modify, delete}	The change types for the input. The type "modify" only modifies the signature of the input.
	name *	[xsstring]	The name of the input that needs to be changed.
	operation *	[xsstring]	The name of operation to which the input belongs to.
	adapt	[xsboolean]	Specifies if the change is adaptable or not.

Tags	Attributes	Values	Description
<changeoutput>	type *	{add, modify, delete}	The change types for the output. The “modify” type only modifies the signature of the operation.
	name *	[xsstring]	The name of the output that needs to be changed.
	operation *	[xsstring]	The name of operation to which the output belongs to.
	adapt	[xsboolean]	Specifies if the change is adaptable or not.
<changecomplexType>	type *	{add, modify, delete}	The change types for the “complexType”. The “modify” type only modifies the signature of the “complexType”.
	name *	[xsstring]	The name of the “complexType” that needs to be changed.
	adapt	[xsboolean]	Specifies if the change is adaptable or not. “add” type is set to “true” by default, unless it is “false” by default.
<changemessage>	type *	{add, modify, delete}	The change types for the “message”. The “modify” type only modifies the signature of the “message”.
	name *	[xsstring]	The name of the “message” that needs to be changed.
	adapt	[xsboolean]	Specify if the change is adaptable or not. “add” type is set to “true” by default, unless it is “false” by default.
<changeelement>	type *	{add, modify, delete}	The change types for the WSDL tag <xselement>. The “modify” type only modifies the signature of the <xselement>.
	name *	[xsstring]	The name of the “element” that needs to be changed.
	parent *	{[xsstring]}	The parent of the element. Usually it is <xsschema> or <xscomplexType>
	adapt	[xsboolean]	Specify if the change is adaptable or not. “add” type is set to “true” by default, unless it is “false” by default.

Actually, the XML tags support more changes to describe by further extensions. We only present the ones that involved with Web Service interfaces in WSDL document at current stage.

II.2.2 Programming framework for Web Service evolution

The previous work on the programming such as Gensis for Web Service evolution, described above in the state of the art, aims at enabling the Web Service developers to program the Web Service with high level APIs. The programming framework can reduce the cost of the Web Service development as well as operation and maintenance.

Compared with Gensis, in this thesis, the programming framework does not only support high level APIs, but also it records also each change action to the Web Service and automatically generates the formalized Web Service changes that meet the XML extensions presented above. When the Web Service provider is going to make evolve a Web Service, he should follow the steps of: 1) design changes, 2) apply or execute changes, and 3) deploy the new versions. The whole process is shown in Figure 26.

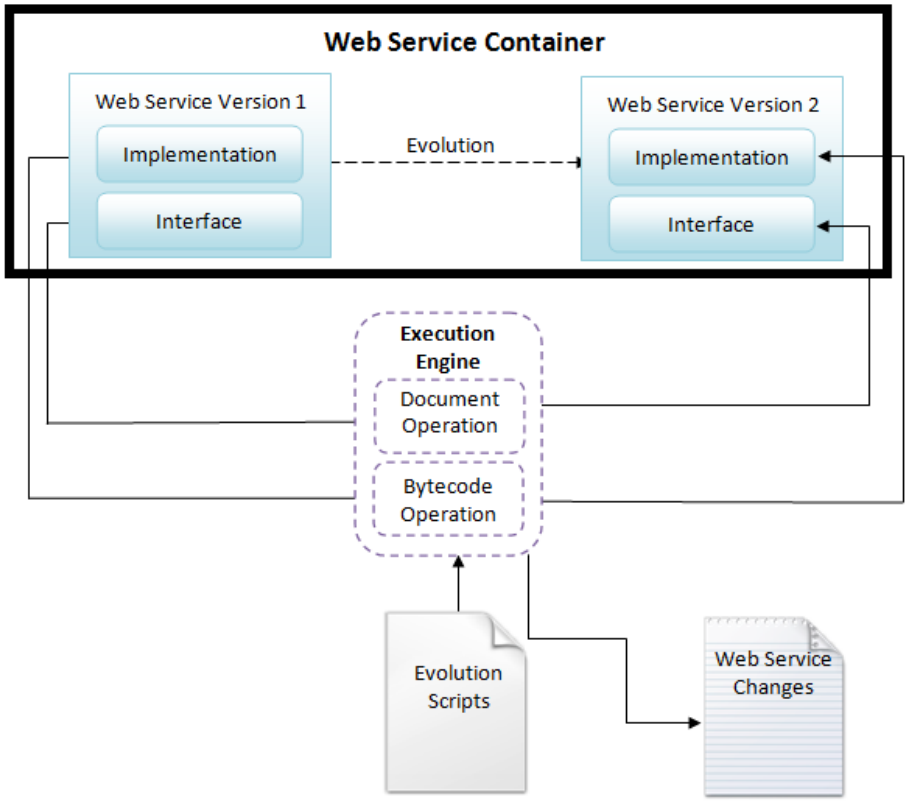

Figure 26 The process for evolving a Web Service

The first step is to develop a plan to evolve the Web service. The supplier must clearly define what part of the Web Service will be changed and what is

the result of the expected change.The provider needs to build up a set of evolution scripts as input of the Web Service evolution execution engine.

The second step is to apply the changes and generate the new version of Web service. We provide an execution engine to analyze and execute the scripts described in the first step. The execution is performed on both the information and implementation aspects of Web services. The Web service instance is generated by the bytecode operation module of the execution engine. The input of the execution engine includes a complete Web service instance deployed in the Web Service container and the designed scripts from the first step. The output of the execution is a complete Web service instance identified with a new version including the documents, the source code and the byte code.

When a new version is generated by the execution engine, a change description will be also published on the broker following the third step. The broker maintains the registry of the Web services published by different providers and stores the subscription information with the Web service consumers and the Web service evolution event that they are interested in. The data structure of the Web service registry in the Web service broker can be considered as a list of Web service descriptions.

II.2.2.1 Web Service evolution APIs

To obtain a new version of the Web Service, the proposed model generates the change description for each version with the execution engine. The generation process is totally transparent for the Web Service developers. What they only need to learn is the scripting APIs to let the execution know how to execute the changes.

The scripting framework is a set of APIs which help the Web Service developers to program the Web Service evolution at high level. It hides the details of the generation of the changes and directly produces a new version of the Web Service including the WSDL descriptions and implementations.

Take a look at the “Document Operation” in Figure 26, we pick up a piece of the script API including the “Interface” class and the “Operation” class for example as shown in Figure 27 to explain how the API produces the Web Service changes.

When the programming script is executed by the execution engine, the change description will be also generated in XML format. If the Web Service provider makes evolve the Web Service with the programming framework, all the changes will be captured.

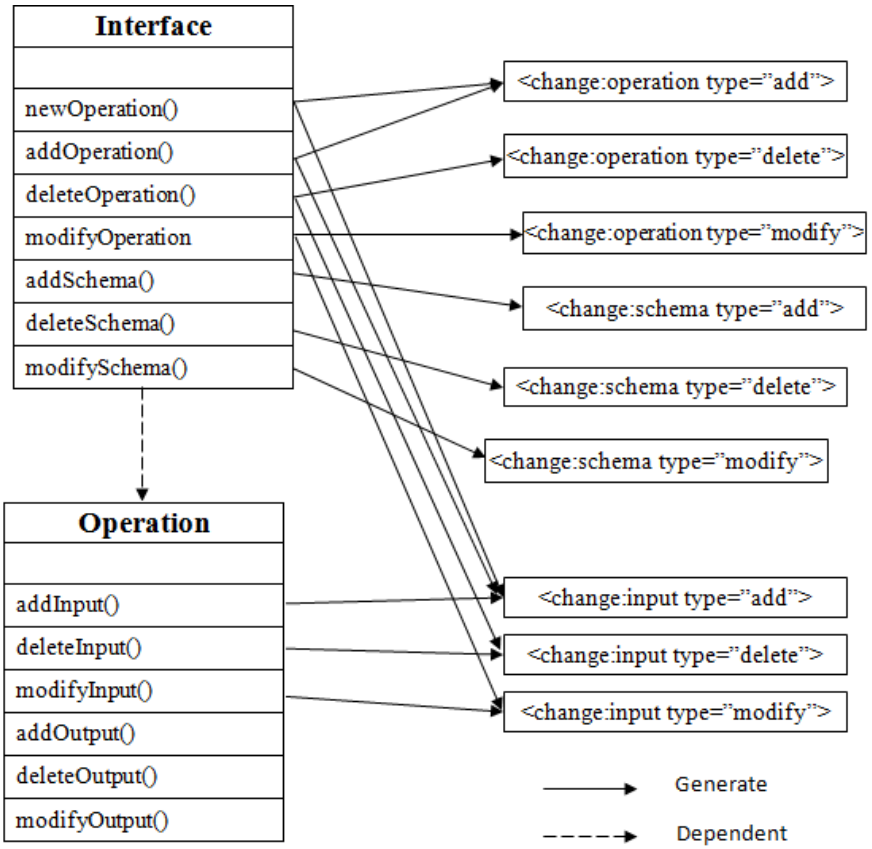

Figure 27 Part of the scripting API

II.2.3 Resource management for runtime versioning

We have introduced the programming framework for Web Service evolution. When we mentioned that the Web Service execution engine picks up the Web Service from the Web Service container, we implied that the Web Service evolution happens at run time. The execution engine accepts the evolution scripts and performs the Web Service evolution on the fly. However, when there exist multiple versions in the Web Service container, the latter can waste much of the memory resources. This can directly induce performance decrease and event stack overflow. For static evolution, there is no such problem since the resources are saved in the persistent drivers such as hard disk.

We consider that the Web Service is actually a package of code resources. When the execution engine has to create a Web Service, it will manage all the resources that are included in the Web Service.

To isolate all the versions of one Web Service, the Web Service container usually allocate independent resource spaces to each version. This type of resource management will waste lots of system resources. It will cause great performance bottleneck when there are lots versions and lots of Web Services are deployed.

The change-centric model uses the inherited resource loaders to manage runtime Web Service versions as shown in Figure 28. The resources of each version of the Web Service are managed by the resource loader. The modified resource for one version, for example the interface and implementation bytecode, will be automatically specified with “changed” tags as the one notated by “1” in Figure 28. The circle notated by “4” is specified with “manual” tags. When the resource loader is prepared to load the resources, it firstly checks if the resource is specified with the two tags. If yes, it will load the resource to current loader. If no (for example circle “2” and “3”), the loading job will be delegated to its parent loader. The parent loader corresponds to the previous version from which the current version has evolved.

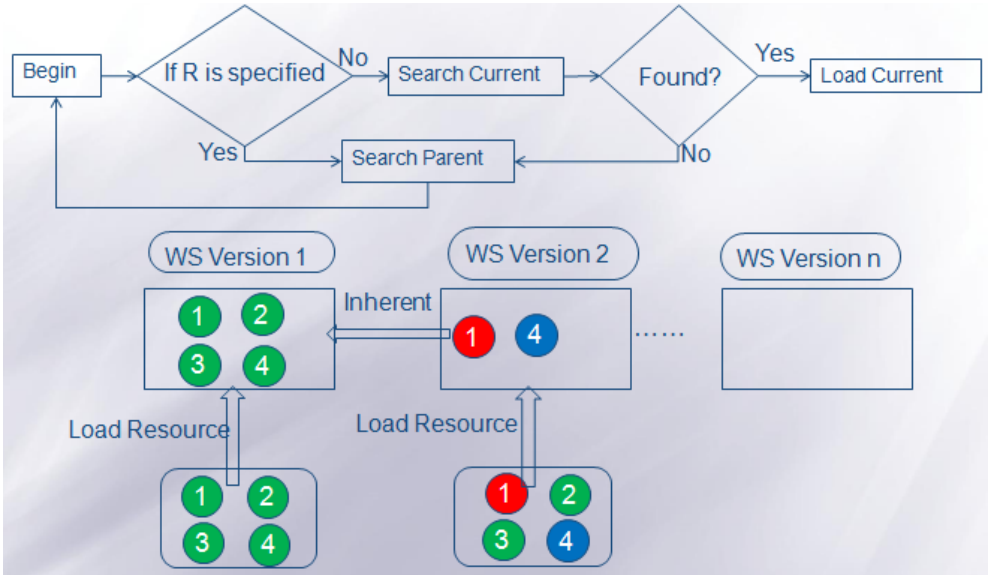

Figure 28 Version Management for Change-centric Model

① Changed resource.
②③Unchanged resource
④Manual defined changed resource

II.2.4 Impact analysis for Web Service evolution

The Web Service evolution can directly affect the invocation of the client applications and even the whole business. To help the stakeholders who are involved with the impact of the Web Service evolution to deal with it, the first step is to estimate the impact. Current approaches can estimate part of the impact such as Wang in [45] and Yamashita in [44]. As we have discussed the limitations of them, we will present the impact analysis model for Web Service evolution which can eliminate these limitations. The impact analysis is categorized as the impact analysis on the client applications and the impact analysis on the Web Service composition.

The impact analysis will answer the two questions:

1. How much is the cost for the stakeholders to adapt to the changes?
2. If the evolved Web Service is adaptable for the consumers?

II.2.4.1 Web Service evolution impact analysis on Web Service client applications

Definition II.1.5 has defined $\Delta^I =< A, I >$. The interface changes of the Web Service may result in some side-effects on the clients such as:

- The unexpected results of the invocation from the Web Service.
- Fatal errors of the client application.
- Incorrect understanding of the consumer for the Web Service.

To estimate the cost of using the new version after the evolution, it is necessary to introduce a metric which could provide a quantitative view of the side-effects that are caused by the Web Service changes during the Web Service evolution. We denote I_Δ as this type of metric which indicates a float numeric value varying from 0 and 1. It measures the change impact of the Web Service evolution and the cost that must be taken for the consumers to adapt to the evolution. I_Δ also indicates the compatibility degree of the new version of Web Service to its clients. $I_\Delta = 0$ always means that the new version of the Web Service is completely compatible with the applications of its current consumers without any updates to them. A greater value of I_Δ indicates a greater impact on the applications of the current clients.

On concluding the side-effects that may be caused by the Web Service changes, we consider that the value of I_Δ depends on 3 aspects: 1) the changes; 2) the usage of the consumer; 3) the Web Service information model.

Firstly, the change impact depends on the types and the numbers of the changes. Unfortunately, only adding new operations, adding output messages, adding new schemas and adding new complex types are compatible with the current clients of the service. The rest types of changes can directly or potentially affect the consumers. The more changes may induce a greater impact degree on the clients.

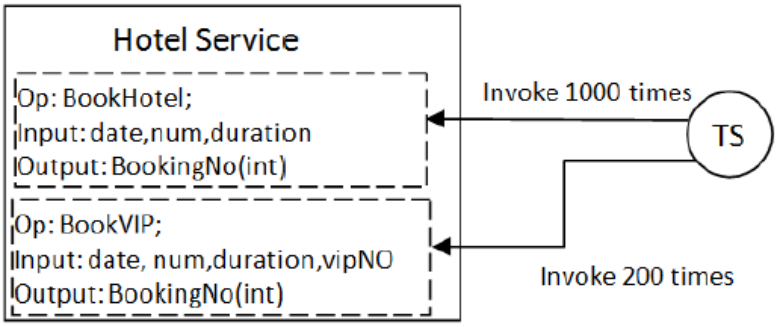

Figure 29 Part of the scripting API

Secondly, the change impact depends on the client usages of the service. Different consumers may have different dependencies on one Web Service. These dependencies can be retrieved through analyzing the historical behaviors records of the consumers. Differently from Wang's dependency model [68], we consider that the same change actions (add, remove or modify) to the different elements of the Web Service may cause different impacts on the same consumer. For example, there are two operations in the HS shown in Figure 29. BookHotel has been invoked for 1000 times. TransferMoney has been called for 200 times. We consider the impact degree of changing BookHotel should have a greater value than the changing TransferMoney on its consumer TravelService.

Thirdly, the change impact also depends on the changing of the dependency graph for the Web Service information model. When a change action is taken to the Web Service, the availability of the Web Service is also changed due to the change of the Web Service information model. Some of the change may not affect the current invocation of the clients, but it may potentially affect the future invocation since it has changed the understanding of the consumers for the Web Service. The contribution of the changing of the Web Service information model is determined by the changing type and the number of the changed primitives. For example in Figure 30- Figure 32 which shows the changing of the Web Service information model.

The change action of adding an element on the Web Service information model can directly change the global understanding of the consumers on the Web Service even if this change action is completely adaptable for the Web Service client applications. This type of changing scenario always refers to the change action of adding a message the Web Service interface. This contribution of changing the Web Service information model is roughly set as 1/10=0.1.

Figure 31 shows that a modification on e5 also induces another modification on e9. This type of changing scenario always refers to the modification on the name of the Web Service so that the input message, output message and the complex type have all been influenced. In this case, the contribution of changing the Web Service information model is roughly set as 2/9=0.2222.

Figure 32 shows that the change action of deleting e2 induces the delete action for all of its sub elements e4, e5, e6, e8, e9. All of the mentioned elements will be also deleted. In this case, the contribution of changing the Web Service information model is roughly set as 6/9=0.6667. Considering all the possible factors listed above, we can give the following definition of I_{Δ}.

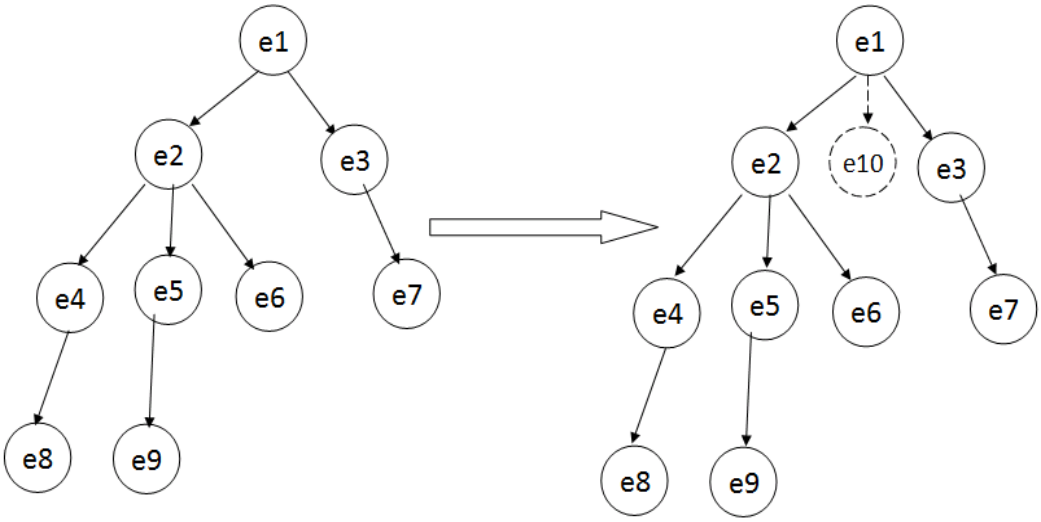

Figure 30 Adding an element e10 to the Web Service information model

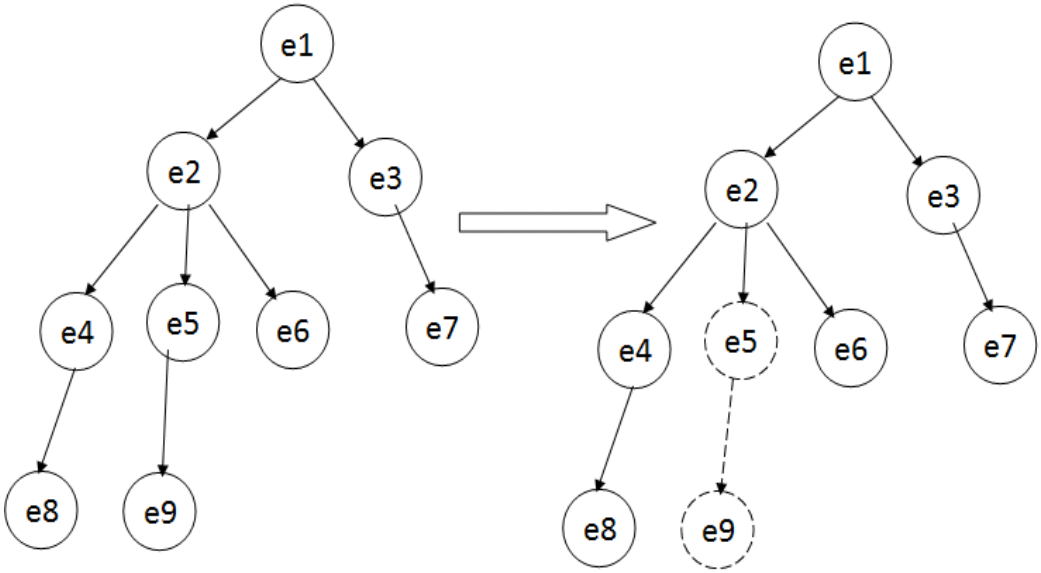

Figure 31 Modifying the e5 to the Web Service information model

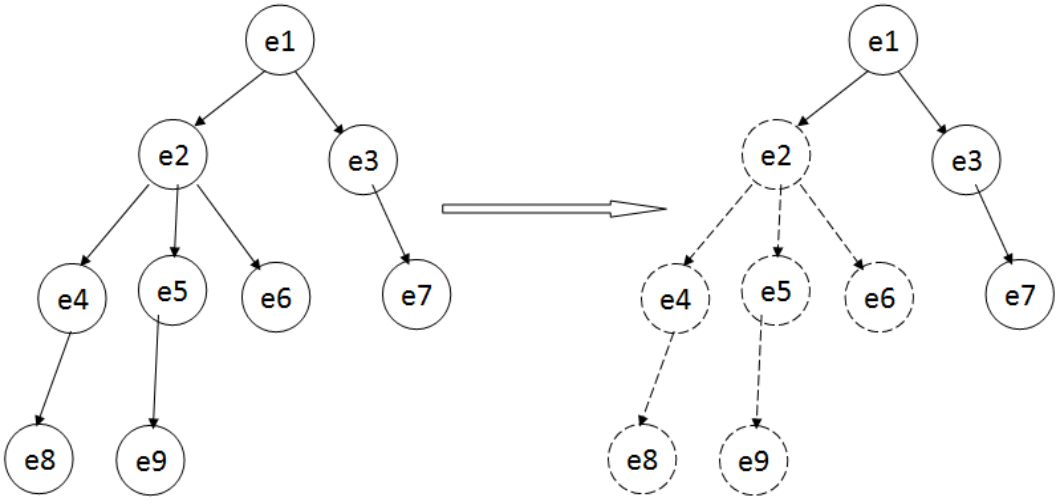

Figure 32 Deleting the e2 from the Web Service information model

Definition II.1.6. The change impact of Web Service evolution on the clients represents the metric of the side-effect on the clients that may be caused by the changes during the Web Service evolution. We assume that a service is changed from S_{v1} to S_{v2} . Then the change impact of this evolution is denoted as $impact_{\Delta}$ and:

$$I_{\Delta}((S_{v1}, S_{v2}), m) = \sum_{i=1}^{n} IC(c_i) \times d(e_i, m)$$

$d(e_i, m)$ represents the dependency between the consumer m and e_i . n represents the total number of changes happen to the Web Service. $e_i \in S_{v1}$ and will be introduced in Definition 7. $IC(c_i)$ indicates the impact of change c_i and will be introduced in Definition II.1.7.

Definition II.1.7. The impact of change action c_i depends on the Web Service information model and the change type. For a given Web Service S , we denote the impact of one change action c as $IC(c)$ and:

$$IC(c) = IF(c) + IR(c)$$

Definition II.1.8. $IF(c)$ indicates the impact of change c to the Web Service information model and:

$$IF(c) = \begin{cases} \dfrac{1}{|S|+1}, if \ \ a(c) = \text{add} \\ 0, if \ \ a(c) = \text{modify} \\ \dfrac{1}{|S|-1}, if \ \ a(c) = \text{delete} \end{cases}$$

where a(c) indicates the action type of change c .

Definition II.1.9. $IR(c)$ represents the impact caused by the type of change action c . It depends on whether the change action is compatible or not. We have noted in Section I.3.4 that the limited types of changes in Web Service evolution are compatible. Now we use Θ to denote the set of compatible changes in Web Service evolution.

$$IR(c) = \begin{cases} 1, if \ \ c \notin \Theta \\ 0, if \ \ c \in \Theta \end{cases}$$

Definition II.1.10. $d(e_x, m)$ represents the dependency degree between the consumer m on the element e_x in the Web Service. It is determined by the historical invocation records of consumer m to Web Service element e_x.

$$d(e_x, m) = \frac{k_x}{\sum_{i=1}^{n}(k_i) + 1}$$

where k_i represents the invocation times of the consumer m to the element i of the current version of the Web Service.

II.2.4.2 Adaptable Web Service changes

Section II.2.1.2.3 has presented the XML annotations for describing the Web Service changes. We have noticed that there is an attribute named “adapt” with a type of <xsboolean> for each type of “change” tags. Normally, this attribute is automatically assigned by the execution engine according to the compatible changes that we mentioned above. Actually, there are two possibilities to make the Web Service changes adaptable to the old version of the client applications.

1. The changes do not affect any of the elements that are not used by the Web Service consumer. For example, adding an operation or changing an input message that a consumer does not use both fall in this type.
2. The changes do not affect the types of the invocation of the client applications and the semantics of the Web Service. For example, changing the signature of an operation and the “adapt” attribute of this change tag is set to “true” falls in this type.

Table 8 The Adaption Behavior

Change c	Adaptation behavior b	Description
<changeoperation type=”modify”, adapt=”true”>	<adaptoperation type=”default”>	default adaptation action is to directly invoke the evolved Web Service by changing the name of the operation
<changeelement type=”add” adapt=”true” parent=”[xscomplexType]” >	<adaptelement type=”value”> [value] </adaptelement>	Assign a default value (null) to the additional element on the already known complex type.
<changeelement type=”delete” adapt=”true” parent=”[xscomplexType]” >	<adaptelement type=”default”>	Ignore the value for the deleted element of one complex type
<changeinput type=”delete” adapt=”true” parent=”[xsschema]”>	<adaptinput type=”default”>	The same as the adaptation for deleting element.
<changeinput type=”add” adapt=”true” parent=”[xsschema]”>	<adaptinput type=”value”> <sequence><adaptelement type=”value”>[value]</adapt element> </sequence></adaptinput>	The same as the adaptation for adding an element for a complex type.

With consideration of the two possibilities, we have the formal descriptions for the adaptable Web Service evolution. A Web Service change c is an adaptable change if it satisfies at least one the following conditions.

Adaptation Condition 1: Let's assume that $c =< a, e >$, if $d(e, m) = 0$, then c is an adaptable change for consumer m .

Adaptation Condition 2: if $c(adapt) = true$ and there exists an adaptation behavior specified by the provider for consumer m noted as $b(m)$, then c is an adaptable change for consumer m.

For the adaptation behavior, Table 11 lists part of the possible types.

To assign the adaptable changes can greatly reduce the cost of evolution for the client applications. To evaluate how the Web Service can support this feature, we define another attribute for the quality of the service so called "Adaptability".

Definition II.1.11: The attribute adaptability indicates how the version of the Web Service supports the backward compatibility for their consumers and:

$$QoS_{adaptability} = \frac{N_{adaptablechanges}}{N_{changess}}$$

The denominator represents the number of generated changes when one version is published comparing with its previous version. The numerator represents the number of adaptable changes among all the changes.

When the Web Service integrator is considering selecting its desired Web Service, he may also want to know if and how the Web Service is compatible with its client applications during its life cycle.

II.2.4.3 Web Service evolution impact analysis on Web Service compositions

The Web Services always work within compositions to complete the business process. The impact of the Web Service evolution on the compositions is determined by how the member services of the composition depend on each other. If more than one member services are changed during the evolution, the impact on the composition is the sum of each impact of the member service on the composition. The impact analysis on the Web Service composition is taken for the Web Service integrators or the users to have a quantified view of the effect of the composition that may be caused by the evolution of the Web Services. It also measures the effort that should be taken to adapt the composition to the evolution.

	TS	TPS	HS	BS
TS		0	0	$I_\Delta((BS_{v1}, BS_{v2}), TS)$
TPS	$I_\Delta((TS_{v1}, TS_{v2}), TPS)$		$I_\Delta((HS_{v1}, HS_{v2}), TPS)$	$I_\Delta((BS_{v1}, BS_{v2}), TPS)$
HS	0	0		$I_\Delta((BS_{v1}, BS_{v2}), HS)$
BS	0	0	0	

Figure 33 Impact Matrix for the motivation scenario

We have got the result of the evolution impact on the client applications. Then we can get the evolution impact on the whole compositions through an impact matrix evolved from the dependency matrix illustrated by Wang [44]. Taking the scenario described in Section II.1 as the example, we have the impact matrix as shown in Figure 33.

Definition II.1.12. Given a Web Service evolution composition, the impact of Web Service evolution on the composition is denoted as IP and:

$$IP = \sum_{i=i}^{n} \sum_{j=1}^{m} I_\Delta((S_i), S_j)$$

Taking Figure 33 as an example, the evolution of TS, BS and HS will result in an impact on the whole business process with the value of:

$$IP = I_\Delta((BS_{v1}, BS_{v2}), TS) + I_\Delta((TS_{v1}, TS_{v2}), TPS) + I_\Delta((HS_{v1}, HS_{v2}), TPS) + I_\Delta((BS_{v1}, BS_{v2}), TPS) + I_\Delta((BS_{v1}, BS_{v2}), HS)$$

II.2.4.4 Discussion

In Section II.2.4, we have explained how to analyze the impact of the Web Service evolution for the client application and the Web Service composition. For the client application, the evolving Web Services may cause impact on them. These impacts can cause incorrect results or even fatal errors to the business processes. To lower the side effect of the Web Service evolution, we proposed a method to analyze the impact of the Web Service evolution on the client applications. This method provides the foundation for any stakeholders to deal with Web Service evolution. We analyze the impact of the Web Service evolution in both of the quantitative and qualitative aspects. Qualitative analysis helps the client application to determine if the Web Service evolution is adaptable. Quantitative analysis helps the Web Service consumers to estimate

the costs of adaptation for the Web Service evolution. Besides, we also provide a method to analyze the impact of Web Service evolution on the Web Service composition based on Wang's dependency model.

With the proposed approach, we could precisely locate the problems of the Web Service evolution. To be closer to the intelligent software, we need the methods to solve these problems. So in next section we will present the approach of client adaptation for the Web Service evolution.

II.2.5 Client adaptation for Web Service evolution

When the Web Service consumers have obtained and analyzed the changes, it is possible to make further strategies to adapt the client applications to the evolution to keep the business processes unbroken. However, the client applications do not adapt to the evolution automatically and dynamically. Thus, people need a mechanism to facilitate the Web Service consumers to react to the Web Service evolution more agilely.

Fokaefs's approach for WSDarwin shows that the generation of the client stub is the only way to adapt the application to the changes. However, WSDarwin only support the adaptation to the limited originally compatible changes as listed in Table 2. To adapt more types of Web Service changes to the client application dynamically, we need a corresponding new method.

II.2.5.1 Overview

Firstly we introduce the overview of the Web Service we propose to perform client adaptation as shown in Figure 34.

The Invocation Interceptor is used for intercepting the dependency relation information that we mentioned in Definition II.1.10. It obtains the dependency information and transfers it for the Impact Analyzer.

The Evolution Monitor is used for discovering and obtaining the change descriptions that are published by the Web Service provider. Then it also transfers the change descriptions for the Impact Analyzer.

The Impact Analyzer accepts the dependency information and change description from the upper layer modules and generates the information about the impact of the evolution on this consumer as described in Section II.2.4.

The Adaptation Designer analyzes the result of the Impact Analyzer to determine if and which of the Web Service changes could be adapted. Once it determines that the Web Service evolution can be adapted, it will generate the adaptation strategies that are corresponding to the adaptation behaviors described in Table 11 for the Web Service Proxy Factory.

The Web Service Proxy Factory generates the Web Service references for the Client Business Module. Normally, the Client Business Module invokes the Web Services through the framework that we provide for the Web Service client application by the Web Service interfaces. The generated Web Service references always implement the interface. Thus, the Web Service references can be always compatible with the Web Service Client Business Module. The process of adaptation is entirely transparent to the Client Business Module. The detailed method and algorithm will be introduced in Section II.3.3.2 with the implementation of the client adaptation.

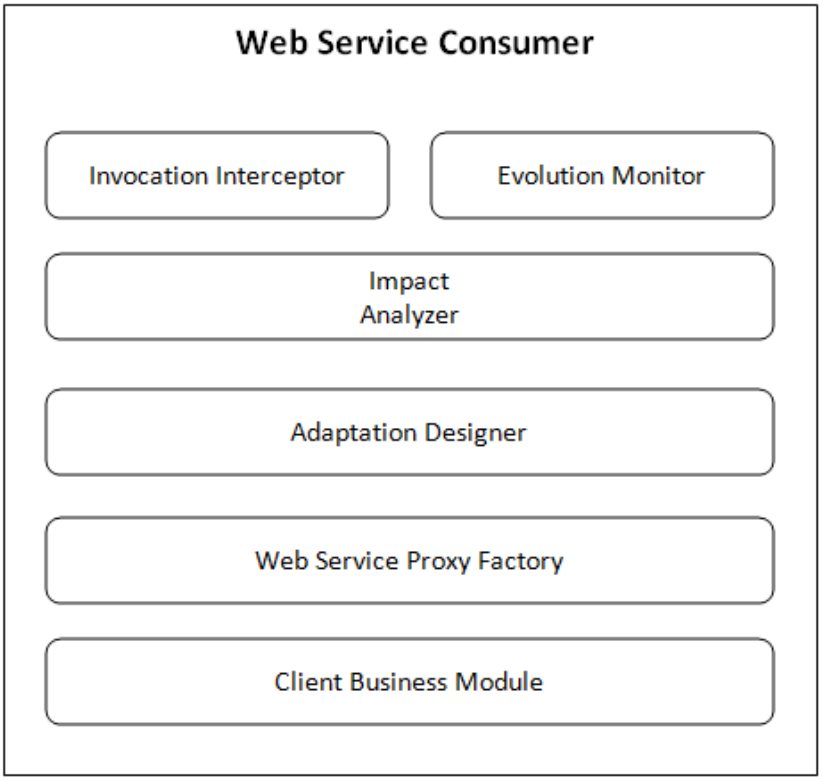

Figure 34 Web Service consumer for adaptation to the evolution

II.2.6 Summary

In Section II.2, we have described several aspects of the change-centric Web Service evolution model.

The Web Service provider is responsible for maintaining and evolving the Web Service. The provider maintains the Web Service container and Web Service execution engine. When the user is planning a Web Service evolution, he firstly prepares an evolution script following the API in Section II.2.2.1. This script is actually a formalized implementation of the user's evolution idea. When the script is imported in the execution engine, the execution engine starts to work with the Web Service evolution. The execution engine grabs the deployed Web Service instance which is specified by the script as the previous version. Then it executes the script to generate both of the interface and implementation

parts of the new version of the Web Service as well as the changes descriptions. Notice that the changes descriptions are automatically generated by the programming framework of the execution engine. The changes descriptions obey the change specification that presented in Section II.2.1.2. When the new version is generated, the execution engine starts to deploy the new version. When the execution engine deploys the new version to local Web Service container, it uses the resource management that introduced in Section II.2.3 to manage multiple versions of the service. When the execution engine deploys the new version to the Web Service broker's Web Service Registry, it also publishes the changes descriptions to the Web Service broker's change interests registry.

The Web Service consumer is responsible for maintaining the client applications and reacting to the Web Service evolution. To keep the client business unbroken, the Web Service consumer should be facilitated to adapt the client applications to the Web Service evolution. However, before the reactions, the consumer should firstly be equipped with the impact analysis as introduced in Section II.2.4 for the Web Service evolution. The impact analysis aims at estimate the compatibility between the client application and the evolved Web Service and the adaptability of the evolved Web Service. The compatibility evaluation is quantitative analysis to estimate the cost of adapting the client application to the evolution. With the compatibility evaluation, the decision maker could have a quantitative view of the impact of the Web Service evolution to support his further decision on whether to manually modify the client applications to adapt to the Web Service evolution. The adaptability evaluation is qualitative analysis to estimate how many of the Web Service changes during the evolution could be adaptable. The adaptation behaviors that are defined in Table 8 determine the adaptable changes and the related adaptation methods. Finally in Section II.2.5, we briefly introduced the method for client adaptation for Web Service evolution.

Generally speaking, we have presented a theory model to make a set of specifications for multiple aspects of the Web Service evolution including versioning, change management, discovery, adaptation and execution. However, some of the details are lack and we did not describe how these theory models can be applied with the software systems and the proposed scenario. In next section, we will introduce the implementation of the change-centric model on Java platform to cover these shortages in this section.

II.3 Execution model

In last section, we have presented the theoretical models and methods to deal with Web Service evolution in several aspects. In this section, we will describe how to implement each part of the proposed model with software engineering technologies. We will show how to build a system that uses the popular real world technologies to facilitate the stakeholders to deal with Web Service evolution. The implementation part will be presented based on the Java platform and related tools and frameworks from the perspective of software engineering. Most of the implementation descriptions will be accompanied with a set of algorithms and critical code. Especially, some examples that are related with the motivation scenario are also presented along with the implementation. To make it clear to specify the implementation of the proposed approach, we denote the implementation of the change-centric model as the Web Service Evolution Platform.

In this section, the presentation is separated in two major parts. The first part introduces the implementation of the programming framework and execution engine of Web Service provider. The second part introduces the implementation of the Web Service impact analysis and the client application adaptation for the Web Service consumer. Finally, there is a conclusion to analyze the advantage and limitations of the implementation.

Table 9 shows the tools that are involved within the execution model.

Table 9 Tools and Frameworks Involved

Items	Version	Descriptions
Apache CXF	Version 3.04	Web Service Toolkit
JBoss Javassist	Version 3.20GA	Generating Web Service code
Groovy Library	Version 2.6	Scripting Engine for Web Service evolution
dom4j	Version 1.6.1	Generating Web Service change descriptions
Java Development Toolkit	Version 1.7	Basic runtime
Eclipse Development Environment	Version Mars	Development Environment
Play Framework	Version 2.4	Web Server
HttpClient	Version 4.4.1	Connecting Web server and stakeholders

II.3.1 System architecture

Figure 35 shows the architecture of the proposed execution model. All the parts in this figure have been presented and discussed in previous sections. In Section II.3, we will introduce how to implement them using our proposed execution model.

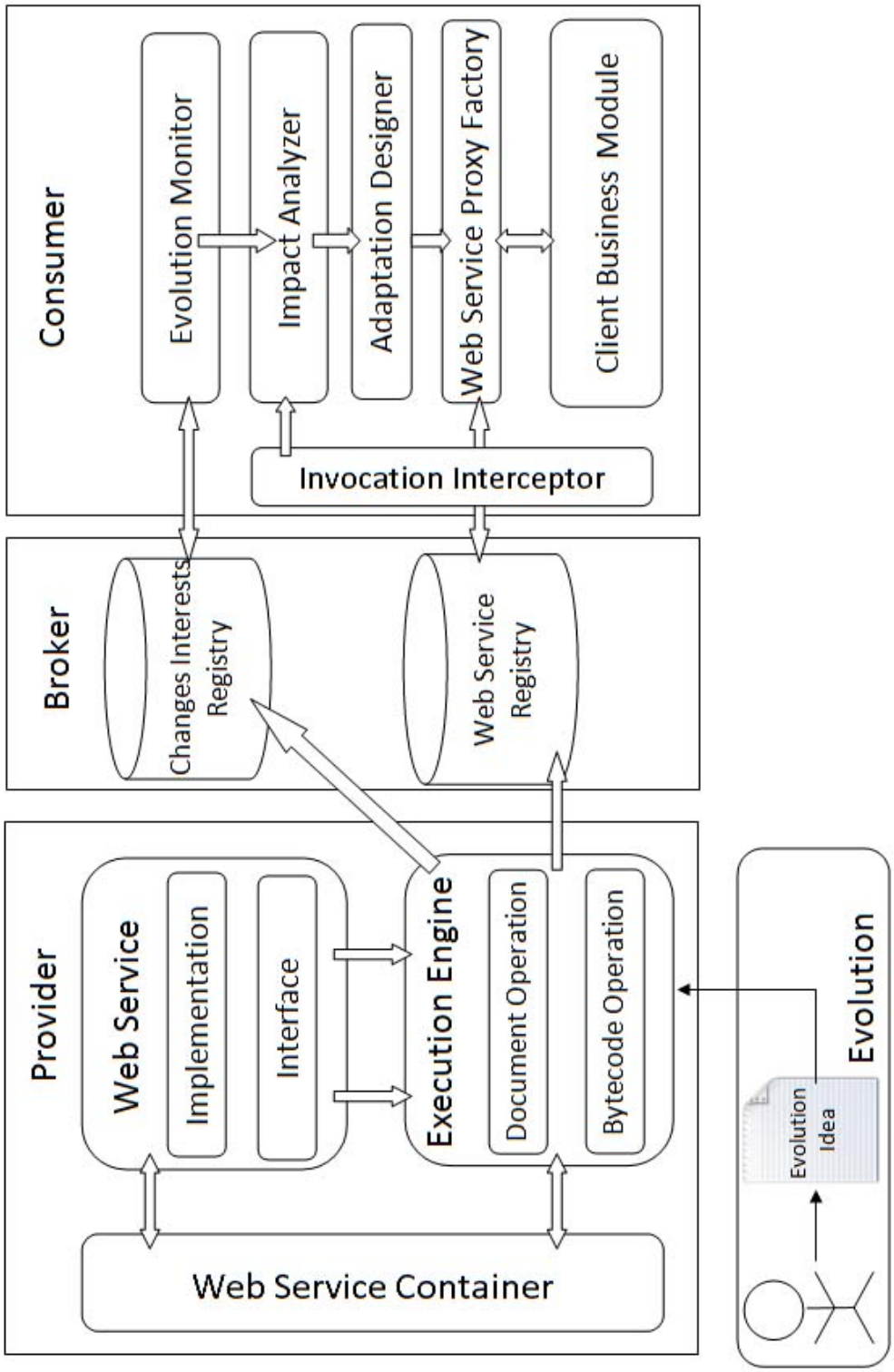

Figure 35 System Overview of Change-centric Model

II.3.2 Web Service provider

The Web Service provider is responsible to develop, evolve and maintain the Web Services. According to the motivation scenario, the Web Service provider quickly updates his Web Services by publishing new versions. In the change-centric model, the Web Service provider uses the programming framework to provide evolution script to describe the requirements for publishing the new versions of the Web Services. The execution engine at the Web Service provider side executes the input scripts and creates the instance of the new version of the Web Services.

II.3.2.1 Programming framework

The programming framework for the change-centric model organizes a set of APIs to help the Web Service developers to quickly publish new versions. The system overview of the implementation of the programming framework is shown in Figure 36.

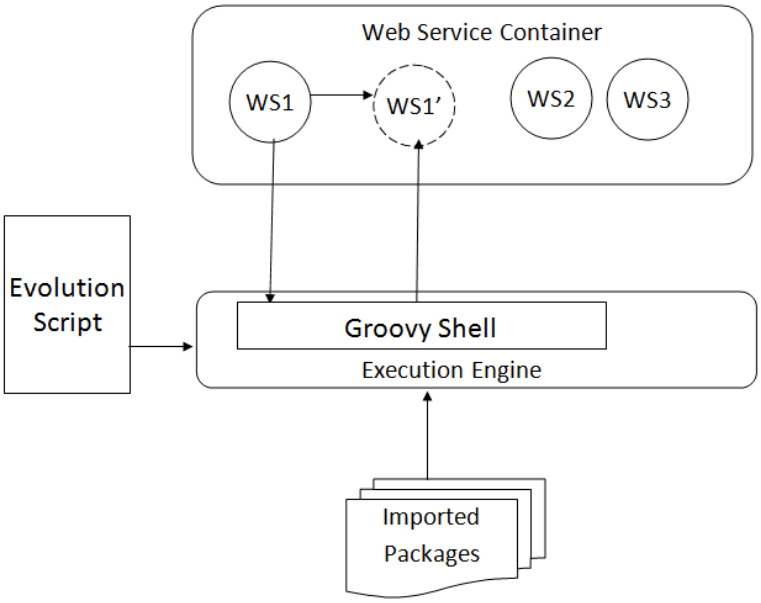

Figure 36 Overview of programming framework

The execution engine is actually a groovy shell which accepts the script, the imported packages and the Web Service WS1 will be injected into the Groovy Shell. The evolution script is written by Web Service developers to specify the description of the evolution process. The imported packages are the Java packages that are specified by the users for the Groovy Shell to resolve the dependencies. The execution engine extracts the model of the Web Service instance of WS1 from the Web Service container in the memory or from the persistence devices under the guiding of the evolution script. When the execution is finished, a new instance WS1' is generated and deployed in the Web Service container.

When the Web Service is evolving, the groovy shell of the execution engine is firstly injected with a Java object named “service”. This object is a copy of the existing Web Service WS1. It includes the service object and is created in a new namespace and within a new resource manager. The users operate this object to realize Web Service evolution execution by calling its public operations.

Especially, the internal process of the execution engine is shown in Figure 37

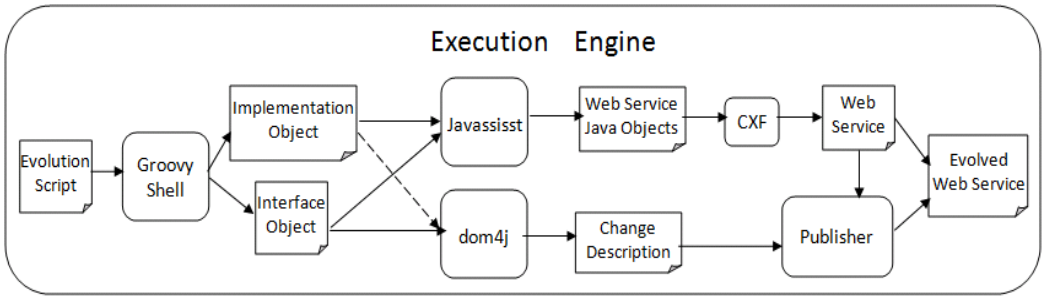

Figure 37 Internal process of execution engine

The evolution script is injected into the Groovy Shell. The target of the Groovy Shell is to produce an implementation object and an interface object. The implementation object and the interface object are both java objects that include the changed information specified by the Evolution Script. Both of the two objects will be handled by Javassist component which produce the java classes and objects for CXF. The interface object is also used by dom4j component to produce the change description in XML format. The implementation object also contains changes. However, usually the implementation changes do not affect the consumers. The user will determine if the implementation change will be handled by dom4j component. The CXF component takes the Web Service objects as input. The Web Service objects include the necessary parameters for CXF to publish Web Service:

- Service Class (Java interface).
- Service Bean (Java class).
- Publish Address (java.lang.String).

Once the CXF component generated the Web Service and dom4j generated the change descriptions, the publisher component will publish both of the two to the Web Service broker as an evolved Web Service.

II.3.2.2 Generating Web Service changes

The Groovy Shell handles user’s evolution script and generates the Web Service changes. This process is completely automatic and transparent for the Web Service developers. Before introducing the generation process for the Web

Service changes, we should understand the model of the Web Service changes. In Section II.2.1.2 we described the model for Web Service changes. For the implementation in Java platform, the class diagram for the changes package is shown in Figure 38 with the part of interface changes. The implementation changes are not included.

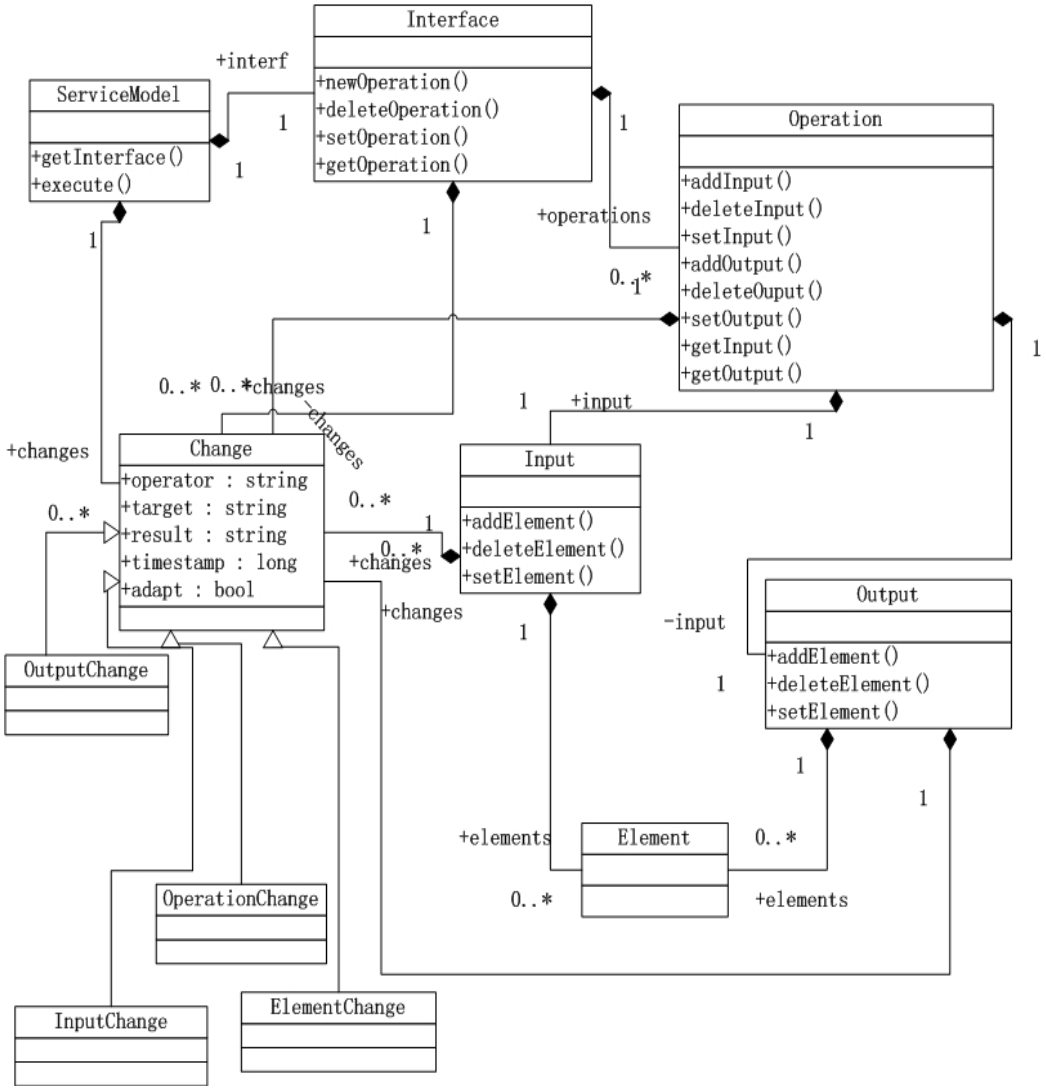

Figure 38 Class Diagram for Web Service Changes

We can see that each changeable primitive has a field named "*changes*" with the type of "*java.util.ArrayList*" which is used to record all the changes during the evolution on this object. For each change operation, there will be one or more objects of the abstract class "*Change*" being added to the list as we have introduced in Section II.2.2.1. In this phase, the programming framework records all the Web Service changes for each change action. Now we will give the pseudo code shown in Figure 39 which is used for recording the Web Service changes for change action of adding an operation to an existing Web Service.

Once a version of the Web Service is executed by the execution engine, the execution engine will firstly create a timestamp in long type to ensure that all the changes are generated at the same time and to avoid of multi change being performed on the same target. For example, when a new operation is added to the interface, a new input message is also added automatically. If the Web Service developer modifies the input message by adding the necessary xml elements to the complex type of the message later, there are two changes on the same object. Finally for the stakeholders who are interested in this change, they may obtain different results by different XML parsing tools since the procedure of parsing the XML document is disordered. This can usually lead to ambiguity for different consumers.

```
1.  package zw.provider.execution.api.serivce.interf

2.  long timeStamp=getTimeStamp();
3.  ArrayList<Change> changes=getChanges();
4.  Interface interf=getInterface();
5.  ArrayList<Operation> operations=getOperations();

6.  function newOperation(String operationName)
7.  {
8.      Operation operation=new Operation(operationName);
9.      operations.add(operation);
10.     Change change=new OperationChange();
11.     change.operator=”add”;
12.     change.target=operationName;
13.     change.timestamp=timeStamp;
14.     setTimeStamp(timestamp++);
15.     change.result=””;
16.     changes.add(change);

17.     //Add Input Message
18.     Input input=operation.addInput();
19.     //Add OutputMessage
20.     Output input=operation.addOutput();
21.     change.children.add(input.changes);
22.     change.children.add(output.changes);
23.     return operation;
24. }
```

Figure 39 Pseudo code for adding an operation

When the evolution script is executed, an instance of the java class *ServiceModel* is created which includes the service changes, service class (Interface), bean class (Implementation), class pool for dependencies. The next step is to generate the XML descriptions for the service changes. As we have mentioned that in Figure 38 that there are 4 types of sub class that extends the abstract *Change* class. So now we need to explore the instance of *changes* of

ServiceModel to find out all the changes. For simplifying, we only introduce how to generate interface changes as shown in Figure 40.

```
createAnnotation("change") as annotation_change
For change_I in changes of Interface
    If change_I instanceof OperationChange
        CreateAnnotation("changeoperation") as annotation_op
            annotation_op.name=change_I.target.
            annotation_op.type=change_I.operator
            If annotation_op.type=="add" annotation_op.adapt="true"
            else annotation_op.adapt="false"
            If change_I.adapt == "true" and annotation_op.type=="modify" or "delete"
                annotation_op.createAnnotation("adaptoperation")
            annotation_op.timestamp=change_I.timestamp
            annotation_change.addChild(annotation_op);
    If change_I instanceof InputChange
        CreateAnnotation("changeinput") as annotation_input
            annotation_input.name=change_I.target
            annotation_input.type=change_I.operator
            annotation_input.adapt="false"
            If change_I .adapt== "true"
                annotation_input.createAnnotation("adaptinput")
            annotation_input.timestamp=change_I.timestamp
            annotation_change.addChild(annotation_input);
    If change_I instanceof OutputChange
        CreateAnnotation("changeoutput") as annotation_output
            annotation_output.name=change_I.target
            annotation_ output.type=change_I.operator
            annotation_ output.adapt="false"
            If change_I .adapt== "true"
                annotation_ output.createAnnotation("adapt output")
            annotation_ output.timestamp=change_I.timestamp
            annotation_change.addChild(annotation_output);
    If change_I instanceof ElementChange
        CreateAnnotation("changeelement") as annotation_element
            annotation_ element.parent=change_I.parent
            annotation_ element.name=change_I.target
            annotation_ element.type=change_I.operator
            annotation_ element.adapt="false"
            If change_I .operator== "add" or "modify"
                annotation_ element.createAnnotation("xselement", change_I.source)
            If change_I .adapt== "true"
                annotation_ element.createAnnotation("adapt element")
            annotation_ element.timestamp=change_I.timestamp
            annotation_change.addChild(annotation_element);

    change_I.createAnnotation("children") as children_I
    For change_child in change_I.children
        Repeat 3 with change_child
   Return  annotation_change
```

Figure 40 Pseudo code for generating changes

II.3.2.3 Examples of the evolution script for the motivation scenario

To demonstrate more explicitly how the programming framework works, now we give the examples of the programming script for the implementation of the motivation scenario. According to the Table 5, there are 4 types of changes for TS.

II.3.2.3.1 Example of the implementation for the change action 1 of TS

We have described the details for the change action 1 of TS in the scenario section as show in Table 10.

Table 10 The changes action 1 of TS

Change Primitive	Initiative	Change Action	Description
Operation	Enhance Functionality	add	Add a new operation named “bookFlight” to provide additional air transportation.

For the changing action 1 of TS, the programming script is shown in Figure 41.

```
1.  Service service=createService(“http://localhosts:9001/TransportService”,”1.0”);
2.  Interface interf=service.getInterface();
3.  Operation op_addBookFlight_interface=Interf.newOperation(“bookFlight”);
4.  Input input=op_addBookFlight_interface.getInput();
5.  input.addElement(“tns:date”);
6.  input.addElement(“xsint”,”amount”);
7.  input.addElement(“xsstring”,”customerID”);
8.  Output output= op_addBookFlight_interface.getOutput();
9.  output.addElement(“tns:PayModel”,”return”);

10. Implementation impl=service.getImplementation();
11. Service.impl.Operation
    op_addBookFlight_impl=impl.newOperation(“bookFlight”);
12. Service.impl.Input input_impl= op_addBookFlight_impl.getInput();
13. input_impl.addElement(“zw.util.Date”);
14. input_impl.addElement(“int”);
15. input_impl.addElement(“java.lang.String”);
16. Service.impl.Output output_impl= op_addBookFlight_impl.getOutput();
17. output_impl.setElement(“zw.util.PayModel”);
18. Service.impl.Body body= op_addBookFlight_impl.getBody();
19. body.setText(“return new
    PayModel(\“TS\”,\”BS\”,$2*5,\”TSTrans_\”+this.counter++);”);

20. service.execute();
21. service.publish(“http://localhosts:9001”,”1.1”);
```

Figure 41 Evolution Script for the change action 1 of TS

When this piece of script is executed by the groovy shell, an evolved version of TS is generated and published to the address "http://localhost:9001/TransportService/V1.1". At the same time, a change description is also generated. The key lines of changes generations are line 3, 5, 6, 7, 9.

Line 3 adds a new empty operation named "bookFlight" to TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

- Input Message named "bookFlight".
- Complex Type named "bookFlight"
- Output Message name "bookFlightResponse".
- Complex Type named "bookFlightResponse"
- Operation named "bookFlight".

And also the change description of the adding action is generated as shown in Figure 42.

```
<changeoperation type="add"  name="bookFlight"
adapt="true" timestamp="0">
</changeoperation>
```

Figure 42 Generated change descriptions of the adding "bookFlight" action

Notice that adding an operation to a Web Service interface is an adaptable change in default.

Line 5 adds a new xml element named "arg0" to the input message of the operation "bookFlight" of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

- The xml element named "arg0" with the type of "tns:date" to the Complex Type "bookFlight" of the message "bookFlight".

And also the change description of this adding xml element action is generated as shown in Figure 43.

```
<changeelement type="add" name="arg0"
parent=" bookFlight" adapt="false">
        <xselement name="arg0" type="tns:date"/>
</changeelement>
```

Figure 43 Generated change descriptions of the adding "arg0" action

Notice that adding the xml element "arg0" to a message is not an adaptable change by default.

Line 6 adds a new xml element named "amount" to the input message of the operation "bookFlight" of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

– The xml element named “amount” with the type of “xsint” to the Complex Type “bookFlight” of the message “bookFlight”.

And also the change description of this adding xml element action is generated as shown in Figure 44.

```
<changeelement type=”add” name=”amount”
 parent=” bookFlight” adapt=”false”>
        <xselement name=”amount” type=”xsint”/>
</changeelement>
```

Figure 44 Generated change descriptions of the adding “amount” action

Notice that adding the xml element “amount” to a message is not an adaptable change by default.

Line 7 adds a new xml element named “customerID” to the input message of the operation “bookFlight” of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

– The xml element named “customerID” with the type of “xsstring” to the Complex Type “bookFlight” of the message “bookFlight”.

And also the change description of this adding xml element action is generated as shown in Figure 45.

```
<changeelement type=”add” name=”customerID”
 parent=” bookFlight” adapt=”false”>
        <xselement name=”amount” type=”xsstring”/>
</changeelement>
```

Figure 45 Generated change descriptions of the adding “customerID” action

Notice that adding the xml element “customerID” to a message is not an adaptable change by default.

Line 9 adds a new xml element named “return” to the output message of the operation “bookFlight” of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

– The xml element named “return” with the type of “tns:PayModel” to the Complex Type “bookFlightResponse” of the message “bookFlightResponse”.

And also the change description of this adding xml element action is generated as shown in Figure 46.

```
<changeelement type=”add” name=”return” parent=” bookFlightResponse”
adapt=”false”>
        <xselement name=”amount” type=”tns:PayModel”/>
</changeelement>
```

Figure 46 Generated change descriptions of the adding “return” action

Notice that adding the xml element “return” to a message is not an adaptable change by default.

Finally, the execution engine will generate a completely Web Service changes description as shown in Figure 47.

```
<evolution target=http://localhost:9001/TransportService
 previousVersion=”1.0” version=”1.1”/>
<changeoperation type=”add” name=”bookFlight” adapt=”true”>
</changeoperation>

<changeelement type=”add” name=”arg0”
parent=”bookFlight” adapt=”false”>
        <xselement name=”arg0” type=”tns:date”/>
</changeelement>

<changeelement type=”add” name=”amount”
parent=” bookFlight” adapt=”false”>
        <xselement name=”amount” type=”xsint”/>
</changeelement>

<changeelement type=”add” name=”customerID”
parent=” bookFlight” adapt=”false”>
        <xselement name=”customerID” type=”xsstring”/>
</changeelement>

<changeelement type=”add” name=”return”
parent=” bookFlightResponse” adapt=”false”>
        <xselement name=”return” type=”tns:PayModel”/>
</changeelement>
```

Figure 47 Generated change descriptions for the change action 1 of TS

II.3.2.3.2 Example of the implementation for the change action 2 of TS

We have described the details for the change action 2 of TS in the scenario section as show in Table 11.

Table 11 The changes action 2 of TS

Change Primitive	Initiative	Change Action	Description
element	Policy Change	add	The input message of the operation “bookTrainTickets” must be updated by adding an input parameter “customerID” since the government requires this service to provide the customer personal information. But according to the policy, this is not necessary and is adaptable.

For the changing action 2 of TS, the programming script is shown in Figure 48.

```
1. Service service
=createService("http://localhosts:9001/TransportService","1.1");
//change the interface
2. Interface interf=service.getInterface();
3. Operation op_ bookTrainTickets
   _interface=Interf.getOperation("bookTrainTickets");
4. Input input=op_ bookTrainTickets _interface.getInput();
5. input.addElement("xsstring","customerID");

//change the implementation

6. Implementation impl=service.getImplementation();
7. Service.impl.Operation op_ bookTrainTickets
   _impl=impl.getOperation("bookTrainTickets");
8. Service.impl.Input input_impl= op_ bookTrainTickets _impl.getInput();
9. input_impl.addElement("java.lang.String");
10. service.execute();
11. service.publish("http://localhosts:9001","1.2");
```

Figure 48 Evolution Script for the change action 2 of TS

When this piece of script is executed by the groovy shell, an evolved version of TS is generated and published to the address "http://localhost:9001/TransportService/V1.1". At the same time, a change description is also generated. The key lines of changes generations are line 5 and 9.

Line 5 adds a new xml element named "customerID" to the input message of the operation "bookTrainTickets" of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

- The xml element named "customerID" with the type of "xsstring" to the Complex Type "bookTrainTickets" of the message "bookTrainTickets".

And also the change description of this adding xml element action is generated as shown in Figure 49.

```
<changeelement type="add" name="customerID" parent="operation:
bookTrainTickets:input" adapt="true">
        <xselement name="arg0" type="xsstring"/>
</changeelement>
```

Figure 49 Generated change descriptions of the adding "customerID" action

Notice that adding the xml element "customerID" to a message is not an adaptable change by default. However, here we set it as "true" because the client

application has to take the adaptation for it. About the adaptation process, we will present it later in Section II.3.3.

Finally, the execution engine will generate completely Web Service changes description as shown in Figure 50.

```
<evolution target=http://localhost:9001/TransportService
 previousVersion=”1.0” version=”1.1”>
        <changeelement type=”add” name=”customerID” parent=”operation:
bookTrainTickets:input” adapt=”true”>
                <xselement name=”arg0” type=”xsstring”/>
        </changeelement>
```

Figure 50 Generated change descriptions for the change action 2 of TS

II.3.2.3.3 Example of the implementation for the change action 3 of TS

We have described the details for the change action 3 of TS in the scenario section as show in Table 12.

Table 12 The changes action 3 of TS

Change Primitive	Initiative	Change Action	Description
element	Enhance Functionality	modify	The type of the element “arg0” of the input message “checkAvailable” will be replaced by the standard date type “dateTime”.

For the changing action 3 of TS, the programming script is shown in Figure 51.

When this piece of script is executed by the groovy shell, an evolved version of TS is generated and published to the address “http://localhost:9001/TransportService/V1.2”. At the same time, a change description is also generated. The key lines of changes generations are line 5 and 9.

Line 5 modifies the xml element named “arg0” of the input message of the operation “checkAvailable” of TS. Once this line is executed, the execution engine generates the following contents to the Web Service interface.

- The xml element named “arg0” with the type of “xsdataTime” to the Complex Type “checkAvailable” of the message “checkAvailable”.

Notice that line 9 sets the type of the parameter with “java.util.Date”. This type will be automatically translated into “xsdateTime” for the XML document by the Web Service engine (eg. CXF).

```
1. Service service
=createService(“http://localhosts:9001/TransportService”,”1.2”);
//change the interface
2. Interface interf=service.getInterface();
3. Operation op_ checkAvailable _interface=
Interf.getOperation(“checkAvailable”);
4. Input input=op_ checkAvailable _interface.getInput();
5. input.setElement(“arg0”,”arg0”,”xsdataTime”,false);
//change the inmplementation

6. Implementation impl=service.getImplementation();
7. Service.impl.Operation op_ checkAvailable
   _impl=impl.getOperation(“checkAvailable”);
8. Service.impl.Input input_impl= op_ checkAvailable _impl.getInput();
9. input_impl.modifyParameter(0, java.util.Date);
10. service.execute();
11. service.publish(“http://localhosts:9001”,”1.3”);
```

Figure 51 Evolution Script for the change action 3 of TS

And also the change description of this modifying xml element action is generated as shown in Figure 52.

```
<changeelement type=”modify” name=”arg0” parent=”operation:
checkAvailable:input” adapt=”false”>
        <xselement name=”arg0” type=”xsdataTime”/>
</changeelement>
```

Figure 52 Generated change descriptions of the modifying “arg0” action

Notice that modifying the xml element “arg0” of the message is not an adaptable change by default. However, here we specify it as “false” because we are not going to take the adaptation for it. About the adaptation, we will present it later in Section III.3.3.

Another thing should be noticed is that the modification on the input message is always accompanied by the modification on the body of the method. Here we just ignore it. We only concern on what will be presented to the Web Service consumer.

Finally, the execution engine will generate a completely Web Service changes description as shown in Figure 53.

```
<evolution target=”http://localhost:9001/TransportService”
previousVersion=”1.2” version=”1.3”>
<changeelement type=”modify” name=”arg0” parent=”operation:
checkAvailable:input” adapt=”false”>
        <xselement name=”arg0” type=”xsdataTime”/>
</changeelement>
```

Figure 53 Generated change descriptions for the change action 3 of TS

II.3.2.3.4 Example of the implementation for the change action 4 of TS

We have described the details for the change action 4 of TS in the scenario section as show in Table 13.

Table 13 The changes action 4 of TS

Change Primitive	Initiative	Change Action	Description
Implementation	Fix Bugs	modify	The implementation of the operation "bookTrainTickets" will be rewritten

For the changing action 4 of TS, the programming script is shown in Figure 54.

```
1.  Service service
=createService("http://localhosts:9001/TransportService","1.3");

//change the implementation

2.  Implementation impl=service.getImplementation();
3.  Service.impl.Operation op_ bookTrainTickets
    _impl=impl.getOperation("bookTrainTickets");
4.  op_ bookTrainTickets _impl.setBody("
5.
6.  //Fixed body
7.
8.  ",adapt=true);
9.  service.execute();
10. service.publish("http://localhosts:9001","1.4");
```

Figure 54 Evolution Script for the change action 4 of TS

When this piece of script is executed by the groovy shell, an evolved version of TS is generated and published to the address "http://localhost:9001/TransportService/V1.3". At the same time, a change description is also generated. The key lines of changes generations are line 4.

Line 4 modifies the body of the operation "bookTrainTickets" which is fixed with eliminate the bugs. Because the change of fixing bugs does not affect the client, the execution engine does not need to add the change description for this change action.

II.3.2.3.5 Example of the implementation for the change action 1 of HS

This type of change has the similar description as the example in II.2.3.2.4. So we omitted this description.

II.3.2.3.6 Example of the implementation for the change action 2 of HS

This type of change has the similar description as the example in II.2.3.2.2. So we also omitted this description.

II.3.2.3.7 Example of the implementation for the change action 3 of HS

We have described the details for the change action 3 of HS in the scenario section as show in Table 14.

Table 14 The changes action 3 of HS

Change Primitive	Initiative	Change Action	Description
Operation	Policy Change	modify	The name of the operation "bookHotelRooms" will be renamed to "bookRooms" for simplifying.

For the changing action 3 of HS, the programming script is shown in Figure 55.

When this piece of script is executed by the groovy shell, an evolved version of HS is generated and published to the address "http://localhost:9001/HotelService/V1.3". At the same time, a change description is also generated. The key lines of changes generations are line 4.

Line 4 modified the name of the operation "bookHotelRooms" to "bookRooms" for simplifying the invocation from the consumers. Once this line is executed, the execution engine modifies the following contents to the Web Service interface.

```
1. Service service=createService("http://localhosts:9001/HotelService","1.2");

//change interface
2. Interface interf=service.getInterface();
3. Operation op_ bookHotelRooms
   _interface=Interf.getOperation("bookHotelRooms");
4. op_ bookHotelRooms _interface.setName("bookRooms",adapt=true);
//change  implementation

5. Implementation impl=service.getImplementation();
6. Service.impl.Operation op_ bookHotelRooms
   _impl=impl.getOperation("bookHotelRooms");
7. op_ bookHotelRooms _impl.setName("bookRooms");
8. service.execute();
9. service.publish("http://localhosts:9001","1.3");
```

Figure 55 Evolution Script for the change action 3 of HS

– The operation named "bookHotelRooms" is changed into "bookRooms".

– The message named “bookHotelRooms” is changed into “bookRooms”.

– The message named “bookHotelRoomsResponse” is changed into “bookRoomsResponse”.

– And the related complex type names.

And also the change description of this adding xml element action is generated as shown in Figure 56.

```
<changeoperation type=”modify” name=” bookHotelRooms adapt=”true”>
        < value>bookRooms</ value>
</changeoperation>
```

Figure 56 Generated change descriptions for the modifying operation action

II.3.2.3.8 Example of the implementation for the change action 4 of HS

We have described the details for the change action 4 of HS in the scenario section as show in Table 15.

Table 15 The changes action 4 of HS

Change Primitive	Initiative	Change Action	Description
Output Message	Policy Change	delete	The operation “checkAvailableRoomNum” does not need to return a message since it will be obtained by other means.

For the changing action 4 of HS, the programming script is shown in Figure 57.

```
1. Service service=
createService(“http://localhosts:9001/HotelService”,”1.3”);

//change interface
2. Interface interf=service.getInterface();
3. Operation op_ checkAvailableRoomNum
   _interface=Interf.getOperation(“checkAvailableRoomNum”);
4. op_ checkAvailableRoomNum _interface.deleteOutput();


//change implementation
5. Implementation impl=service.getImplementation();
6. Service.impl.Operation op_ checkAvailableRoomNum
   _impl=impl.getOperation(“checkAvailableRoomNum”);
7. op_ checkAvailableRoomNum _impl.setReturnType(null);
8. service.execute();
9. service.publish(“http://localhosts:9001”,”1.4”);
```

Figure 57 Evolution Script for the change action 4 of HS

When this piece of script is executed by the groovy shell, an evolved version of HS is generated and published to the address "http://localhost:9001/HotelService/V1.4". At the same time, a change description is also generated. The key lines of changes generations are line 4.

Line 4 deletes the output message of the operation "checkAvailableRoomNum". Once this line is executed, the execution engine also deletes the following contents to the Web Service interface.

- The message named "checkAvailableRoomNumResponse".
- And the related complex type names.

And also the change description of this action is generated as shown in Figure 58.

```
<changeoutput type="delete" name=" checkAvailableRoomNum
adapt="true">
```

Figure 58 Generated change descriptions for the modifying operation action

II.3.2.4 Versions isolation and resource management

Usually, the Java objects of the different versions of the Web Services always run in the same Java Virtual Machine (JVM) instance. All the versions of one Web Service share the same Web Service name (ex. TransportService) but different version identifiers (ex. V1.0, V1.1, V1.2…). Each version of the Java object should be separated by different class loaders. For example, in the change action 1 of TS in the scenario section, the Resource Manager is implemented by javassist.ClassPool which includes a list of bytecode resources (java classes). Each object of the javassiss.ClassPool is managed by an object of javassist.Loader. The class of javassist.Loader implements the abstract class of java.lang.ClassLoader. A little difference of the javassist.Loader from java.lang.ClassLoader is the class loading algorithm. The class loading strategy for javassist.Loader is shown as Figure 59.

We find that the loader will firstly search the local class pool to get the resource if there is no specified delegation. If the resource is found locally, the loader will load the resource and return it. If the resource is not found, the loading will be delegated to the parent loader. All the resource loaders of the versions of the Web Services share the same parent class loader which is the context class loader of the main thread (CCL). CCL is a standard app class loader implemented by Apache CXF which is responsible for loading the system resources such as "java.lang.*".

As shown in Figure 60, the instance of the javassist.ClassPool of the TS version 1.1 (CP2) inherits from the instance of the javassist.ClassPool of the TS version 1.0 (CP1). When the version 1.1 is created, the CP2 will refer the list of

resources of CP1. When the execution engine has modified the resources (red ones in Figure 60), the resources are created locally in CP2.

```
protected Class loadClass(String name, boolean resolve)
throws ClassFormatError, ClassNotFoundException
{
    name = name.intern();
    synchronized (name) {
     Class c = findLoadedClass(name);
     if (c == null) {
       c = loadClassByDelegation(name);
     }
     if (c == null) {
       c = findClass(name); //search the local class pool
     }
     if (c == null) {
       c = delegateToParent(name);
     }
     if (resolve) {
       resolveClass(c);
     }
     return c;
    }
}
```

Figure 59 javassist.Loader.loadClass()

As a result, the resources TransportService and TransportServiceImpl are loaded locally in CP2 by the loader. The resources PayModel and the other Web Service related dependencies are loaded by delegating to the parent loader for version 1.0 and loaded in CP1.

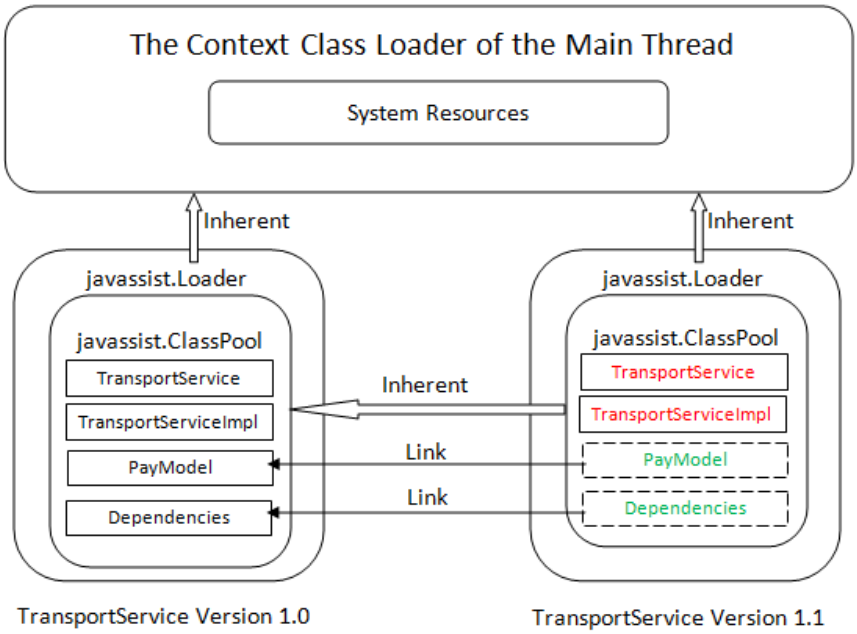

Figure 60 Resource Loading mechanism for the Web Service versioning

Finally, all the instances of the Web Service resources of different versions are isolated through the class loading mechanism. And also the system avoids of a waste of space though the class pool inheritance mechanism.

II.3.2.5 Web Service performance monitor

This section introduces the implementation of an extra functionality of the Web Service execution engine which will be used in the evaluation section. This is not included in the main contributions of this thesis, but it is important to build the basis for testing and evaluating the proposed change-centric model.

Before the execution engine creates the instance for each version of the Web Service, it also weaves the block of monitoring code (shown in Figure 47) for the implementation of each operation of the Web Services to capture the information of the invocation latencies. The object of the performance monitoring is to evaluate the impact of the proposed evolution approaches on the runtime performance of the Web Services. As shown in Figure 61, the weaving process follows the principle of Aspect-Oriented Programming (AOP) [84, 85]. The weaving process is entirely transparent for the Web Service programmers and is executed at runtime in memory through the monitoring framework. Two blocks are weaved in each of the Web Service methods which contain the Java signature “public”. The weaved Java classes are recompiled at runtime with an independent class loader and an independent namespace. The recompiling process is also implemented by JBoss-Javassist as well as the execution engine. The two weaving blocks compute the execution time of the current method and report it to the performance repository of the monitor center. The Monitor Center is a global single instance which collects the performance information of all the versions of the Web Service that deployed in one Web Service provider.

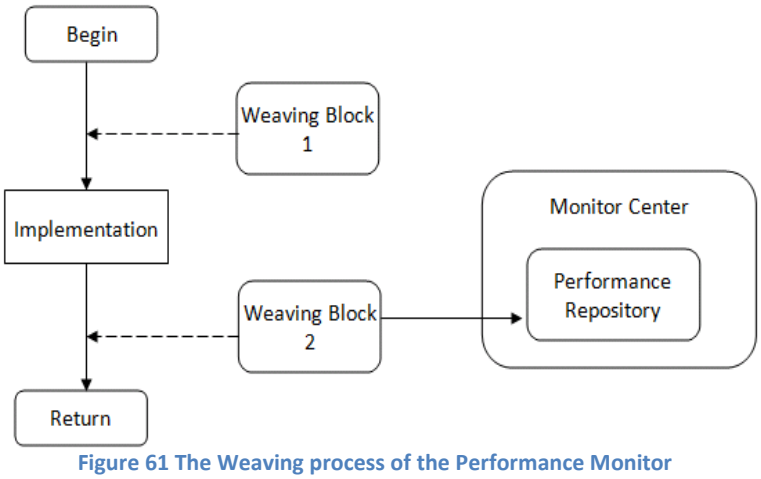

Figure 61 The Weaving process of the Performance Monitor

The Web Service performance does not affect the business logic of the Web Service implementation. The Web Service is loaded from static storages and changed at runtime. The weaving process has a little influence on the runtime

performance of the Web Service. In this thesis, we assume that this little impact can be ignored.

Actually in practice, the performance monitor is integrated in the execution engine as shown in Figure 62. Before each version of the Web Service is published, it will be firstly passed to the performance monitor of the execution engine to weave the monitor blocks. This AOP process compiles and generates the desired Web Service instances with the function of performance monitor transparently. So it can report the more precise latency of the methods of the Web Service than any other external monitor tools and services.

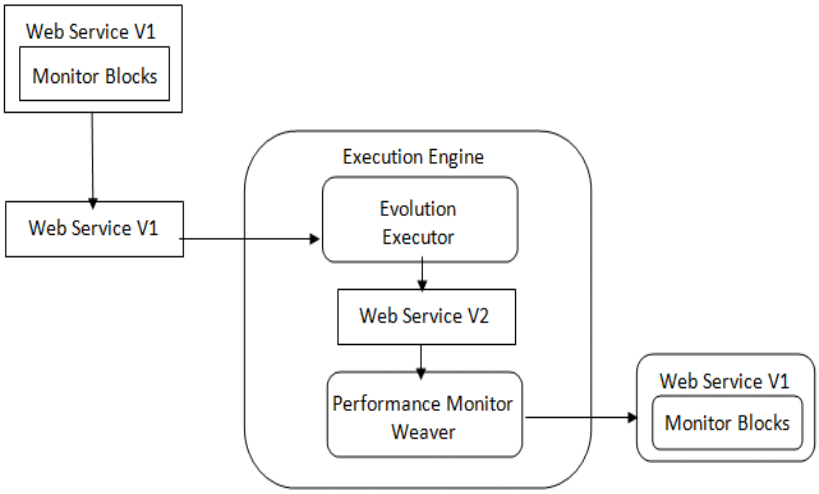

Figure 62 Execution Engine integrates Performance Monitor Weaver

Figure 63 present the implementation code for the weaving process in Java. This method takes the type of “java.lang.Class” and the current binding address of the Web Service as input and generates an object of the “java.lang.Class” which has been weaved with the monitor blocks. The weaver firstly injects a local static field named “myAddress” to store the publishing address of the Web Service so that the weaving blocks can report the correct Web Service addresses to the Monitor Center. Then the weaver injects the monitor blocks for each method of the input instance which is the type of “java.lang.Class”. Finally it calls the method “addNewLatency” of the global instance “zw.Provider.Monitor”. For the sake of simplication of this prototype, the performance monitor exists in the same JVM instance with all the Web Service, so the calling is the direct Java style. In future, the calling process can be implemented in the other ways such as remote method invoke.

```
1.  public static Class weave(Class clazz,String address)
2.  {
3.  Class c=null;
4.  ClassPool cp = ClassPool.getDefault();
5.  try {
6.  CtClass ctClazz=cp.get(clazz.getName());
7.  Loader loader=new
    Loader(Thread.currentThread().getContextClassLoader(),cp);
8.  loader.delegateLoadingOf("zw.provider.Monitor");
9.  CtField f=CtField.make("public String myAddress=\""+address+"\";",
    ctClazz);
10. ctClazz.addField(f);
11. for(CtMethod m:ctClazz.getDeclaredMethods())
12. {
13. m.addLocalVariable("monitor$startTime", CtClass.longType);
14. m.addLocalVariable("monitor$endTime", CtClass.longType);
15. m.insertBefore("zw.provider.Monitor.getInstance()
.addNewCall(myAddress,\""+m.getName()+"\");");
16. m.insertBefore("monitor$startTime=
System.currentTimeMillis();");
17. m.insertAfter("monitor$endTime=System.currentTimeMillis();");
18. m.insertAfter("zw.provider.Monitor.getInstance()
.addNewLatency(myAddress,\""+m.getName()+"\",monitor$endTime-
monitor$startTime);");
19. }
20. c=ctClazz.toClass(loader);
21. } catch (Exception e) {
22. e.printStackTrace();
23. }
24. return c;
25. }
```

Figure 63 Weaving code for Web Service Performance Monitoring

II.3.3 Web Service consumer

The Web Service consumer is responsible for reacting to the Web Service evolution. As described in the scenario section, the Web Service of the TS and HS evolve in a very fast speed so that it is a great pressure for the Web Service consumer to deal with the Web Service evolution. The changes that occur during the Web Service evolution have different types in different levels. To keep the client business unstopped, the Web Service consumer has to catch up with the evolution of the Web Services in using. Otherwise, the consumer has to stop the client application and choose another Web Service. The last option may drive the consumer to change the business logics of the client applications if the consumer cannot find a new Web Service which provides the same functionality for the consumer. This process may induce great manual effort which may result in loss of cost and time. For the Web Service consumer, the first option will be

better to deal with Web Service evolution. In this scenario, we assume that the consumer could not use a Web Service with an old version which is originally fully compatible with the current client applications of the consumer for the following reasons.

1. The Web Service evolves in a very fast speed so that the Web Service provider only maintains a limited number of the Web Service versions. Thus, the Web Service consumer is not always able to obtain the compatible old version.
2. The Web Service always evolves with the enhancement of the functionality or the quality of the service. The Web Service consumer is encouraged to adapt the client applications to the evolution to benefit the better Web Service.

Thus, the tasks of the Web Service consumer to deal with the Web Service evolution are:

1. To estimate the impact of the Web Service evolution on the client applications to determine if and how to adapt to the Web Service evolution.
2. To adapt the client applications to the Web Service at runtime.

There is one convention of the usage of the evolution platform for the Web Service consumer. The Web Service consumer communicates with Web Service by interface. The reference object of the Web Service is generated by the programming framework.

II.3.3.1 Implementation of the impact analysis

In section I.2.4, we have introduced the model of impact analysis on the change-centric Web Service evolution model. According to the presentation in section I.2.4, this analysis requires that the client application should build a mechanism to obtain, parse and deal with the Web Service changes as well as to retrieve the dependencies from the client application to the Web Service. Thus, the content for this section is divided into 2 parts. Section II.3.3.1.1 introduces how to build a mechanism to obtain and analyze the Web Service changes. Section II.3.3.1.2 introduces how to extract the dependency relations between the Web Service consumer and the Web Services.

II.3.3.1.1 Web Service changes analysis

Web Service is used to build a communication mechanism for heterogeneous platform and application components though a set of industry standards. Besides, SOA defines the cooperation patterns for different roles involved. It indicates the method for the Web Service consumer to look up and bind to the expected Web Service.

To communicate with heterogeneous applications, there should be a Web Service Description Language to describe the functions and messages for a Web Service. To communicate with the Web Service changes, there should be also a Web Service changes description as we have presented in Section II.2. For the topic of managing Web Service evolution, people also needs to be equipped with communication mechanism to help the Web Service consumers and providers to synchronize the Web Service changes.

As described in Section II.3.1, the generated Web Service will be published in the Web Service broker as a document in XML format. To make the client be aware of the updating information from the remote machine, there are basically 3 options in the industry world. The first one is to synchronize the data of the client with the server only when the client meets the unexpected result. The second one is to make the requester to post heart beating message for the remote server like comet in [86] and Sencha EXTJS direct in [87]. The third one is to build an independent communication mechanism with a set of new protocols such as Web Socket in [88].

In Section I.3.3.2 with Figure 5, WSDarwin use a feedback loop in the adaptation process. The Web Service client starts the adaptation process which includes obtaining the versioning information only when the invocation meets an error. This approach falls into the first option. In Section I.4.1 with Figure 9, Marcelo's change management framework advocates synchronizing the client application every time when the client posts a request to the Web Service to ensure that the invocation will not result in the failures. WSDarwin's feedback approach can avoid of ambiguity understanding for the updating of the Web Service versions when it encounters the invocation faults. However, WSDarwin does not explain how to obtain and analyze the Web Service changes. No one ensures that the client application can correctly and precisely locate the root cause of the fault.

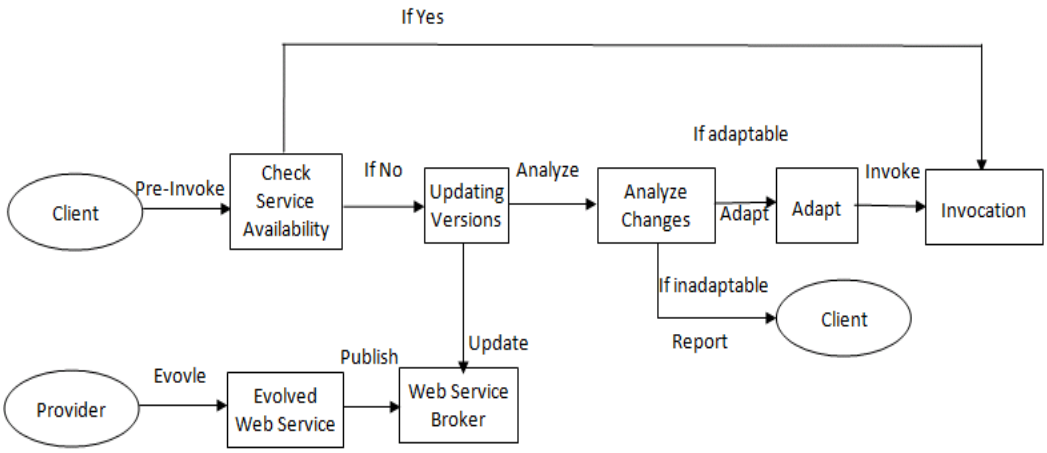

Figure 64 Process for Web Service client to analyze the Web Service changes

In summary, the design for the implementation for analyzing the Web Service changes in the change-centric model should follow the principles:

– Feedback invocation.

– Version information updating.

Based on the loop presented by WSDarwin, we improve the process for the client application to obtain and analyze the Web Service as shown in Figure 64.

Normally, the Web Service consumer uses the evolution platform that we propose to control the invocation of the Web Services. The evolution platform consistently provides the Web Service reference for the Web Service client applications to complete the business. The client applications will check the availability of the Web Service with the specified version ID each time when the request for invocation is coming. The method for checking the availability is simply to obtain the HTTP response by sending a "GET" HTTP request for the location of the WSDL document of the version of the Web Service. The detailed code for Java platform is shown in Figure 65.

```
public boolean checkAvailability(wsdlLocation) {
        try {
            HttpURLConnection.setFollowRedirects(false);
            HttpURLConnection connection =
(HttpURLConnection) new URL (wsdlLocation).openConnection();
            connection.setRequestMethod("GET");
            return connection.getResponseCode() ==
HttpURLConnection.HTTP_OK;
          } catch (Exception e) {
            return false;
    }
}
```

Figure 65 Check the availability of the Web Service

If the version of the Web Service is available, the client applications will be granted with the available Web Service reference. If the Web Service is not available, the consumer will look up the version information on the Web Service to check if there is new version published for this Web Service. When the consumer obtains the Web Service evolution information which contains the Web Service change description, the consumer analyzes the Web Service changes to find out the impact of the Web Service evolution on the client and the adaptability of new version for the client applications. If the new version of the Web Service is adaptable, the Web Service evolution platform will take the client adaptation and return a new Web Service reference to the Web Service client application.

Notice that no matter the new version of the Web Service is adaptable or not, the evolution platform will always report the Web Service impact to the consumer for them to make further decision to react to the Web Service evolution.

II.3.3.1.2 Implementation of the impact analysis of the Web Service changes on the client

Section II.3.3.1.2 introduces the approach for obtain and retrieve the Web Service changes. To implement the Web Service impact analysis presented in Section II.2.4, the Web Service consumer needs also to retrieve the dependency information from the consumer to the Web Service. Wang in [68] has introduced the dependency model which contains "intra" and "inter" relations between two Web Services. In this step, we do not consider the intra relations specified by Wang between two Web Services. But similarly, we also introduce the inter and intra relations for the Web Service and its client.

Assuming that each Web Service primitive that exists in the WSDL description is denoted as e . And the dependency graph which describes the intra-service relation is shown in Figure 66 as an example.

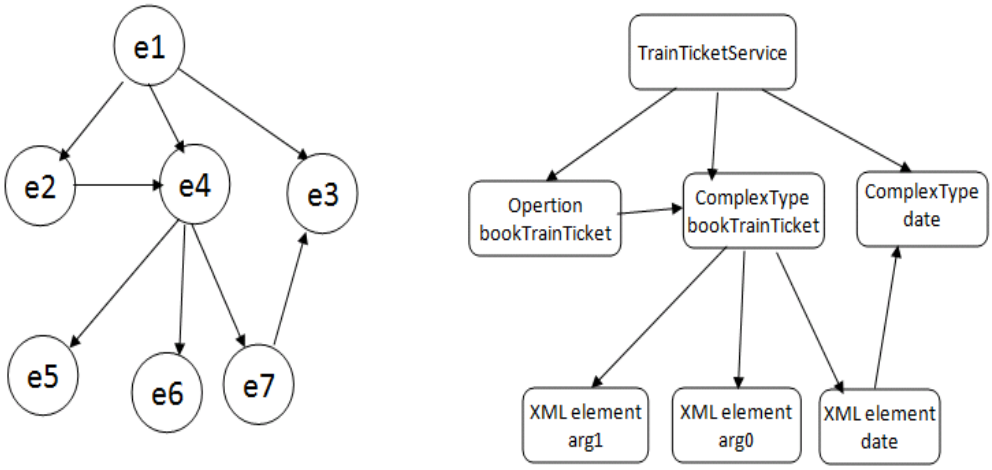

Figure 66 Web Service Dependencies

The primitives of e1-e7 represent the corresponding relations of the primitives for "TrainTicketService". For any primitive that is determined as the dependency by the Web Service client at each time when the invocation happens, we consider that all of its children will also be added to the dependency set. They are called the intra-service relations. The dependency from the client to the set is so called inter-service relations. And when the evolution happens, all of its children will be also influenced. There is one problem for retrieving the dependencies. The intra-service relations are determined by the explicitly presented WSDL descriptions. There may still be some implicit intra-service dependency relations that are difficult to discover. For example, an internal process of an operation calls another operation of the Web Service. For this issue, it should introduce the other methodologies such as the approaches in [89-91]. However this is beyond the scope of this thesis. We only use the explicit descriptions in current stage.

For example for e2 in Figure 66, when e2 is invoked by a client application, e4-e7 and e3 will be all added to the dependency set and be considered to contribute to the value of k of Definition II.1.10 in Section II.2.4.1.

II.3.3.2 Implementation of the client adaptation

The part of the impact analysis for the Web Service evolution will report two types of information. The first one is the quantitative impact of the Web Service evolution on the client applications which reveals the compatibility of the new version for the old version of the client applications. The second is the adaptability of the Web Service which reveals that fact that if the new version of the Web Service is adaptable for the client applications. Once it is determined that the new versions is adaptable, the client application will automatically adjust its behavior to make itself fully compatible with the new version. Notice that this approach does not attempt to explain how to develop strategies for client application. It just provides a mechanism to facilitate client adaptation. The further decisions involve with business process have to be made by human brains.

The common idea for the adaptation is to generate a new temporary proxy similar as an adapter for the upper layer requesters as shown in Figure 67.

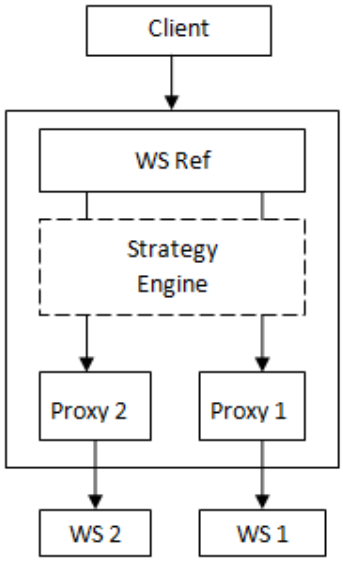

Figure 67 Generating Web Service Proxy

There are two Web Service versions WS2 and WS1. The client application does not communicate directly with the Web Service. The proxy translates the requests for each version of the Web Service. The following steps show how to generate the proxy in Java platform.

1. Generate a new class which implements the current client proxy for the Web service.
2. Generate a new interface to the new version of the Web service according to the changes.

3. Create a new member with the type of the new interface in step 2 for the new class in step 1 which refers to the new versions of Web service with correct format.
4. Set the body of the modified operation as calling the new operation.
5. Return a Web Service reference (WS Ref) of a new instance of the generated class to consumer.

We take the adaptation for the second change of TS described in Section II.1 as an example. By default, the change action of adding an XML element to the input message of an operation is inadaptable. However, the Web Service developer or the provider sets the property “adapt” to “true” for this adding xml element. When the Web Service client meets this case, the Web Service client has to take adaptation for this change action. Assuming that currently the Web Service client is using this Web Service through an interface as shown in Figure 68:

```
package zuowei.provider;

@javax.jws.WebService
public interface ITrainTicketService$v10 {

    public Ticket bookTrainTicket(Date date, int num, String trainID);

}
```

Figure 68 Interface of the Web Service TrainTicketService

```
package zuowei.provider;
import org.apache.cxf.frontend.ClientProxyFactoryBean;
public class ITrainTicketService$v11$classProxy implements
ITrainTicketService$v10 {
    public ITrainTicketService$v11 ref;
    public PayModel bookTrainTickets(Date date, int num, String trainID)
    {
        return ref.bookAirTicket(date,num,trainID, “”);
    }
    public ITrainTicketService$v11$classProxy(String serviceAddress)
    {
        ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
        factory.setServiceClass(ITrainTicketService$v11.class);
        factory.setAddress(serviceAddress);
        ref = (ITrainTicketService$v11) factory.create();
    }
}
```

Figure 69 Generating Web Service Proxy

When executing the step 1, the generated new class is shown in FFigure 69. The constructor of the new proxy creates a new Web Service reference to the

new version of the Web Service. When the client application calls any of the operations of the Web Service, the proxy translates them into the request to the new version. In default, this adding element with the type of string passes a string value “” to the operation. The new proxy is entirely generated and compiled at runtime by the module of JBOSS-javassist.

Therefore, the Web Service client application can still obtain the Web Service reference by using the old version of the Web Service interface. It cannot be aware of any changes on the usage of the Web Service.

II.3.4 Summary

In this section, we have introduced the implementation of the change-centric model to facilitate Web Service evolution for the Web Service provider and consumer from the perspective of software engineering. We use a series of code, pseudo code, algorithms, UML graphs and examples to describe how to realize the project which implements the change-centric model presented in Section II.2.

For the Web Service providers, the evolution platform enables them to modify and publish the Web Service versions at runtime with high level APIs. Under the support of the evolution platform, the Web Service provider can also publish the formalized description for the Web Service changes during the evolution processes. The Web Service changes are automatically generated by the programming framework that the evolution platform provides according the modification that the providers or developers take on the Web Service. Thus, the system can ensure that all the Web Service stakeholders have the same correct and standard understanding on the Web Service changes.

For the Web Service consumers, the evolution platform enables them to retrieve and analyze the Web Service changes that are published by the Web Service providers. The evolution platform can also help the consumers to analyze the impact of the Web Service changes through calculating the Web Service dependencies. Finally, we presented the implementation for the client adaptation. According to the analysis of the change impact, the evolution platform generates and compiles automatically and dynamically the proxy adapters to facilitate the Web Service consumer to adapt their client applications to the new versions of the Web Service. The evolution platform also reports the result of the impact analysis to the Web Service consumers to help them make further decision to react to the Web Service evolution.

II.4 Evaluation

In last section, we have described the execution model for the proposed change-centric Web Service evolution model. By the implementation and related monitor system, we can take experiments to evaluate the proposed approaches in this section. The objectives of the evaluation include:

1. Validate the feasibility of the proposed change-centric model.
2. Test the performance of the implementation.
3. Estimate the impact of the implementation on the original system.

We have built a working scenario in Section II.1. In this section, the evaluation will be taken based on this scenario to show how the proposed model can work with the scenario and how the model performs. We use lots of testing data, tables and charts to present the results.

To simplify the evaluation progress, we also list 3 items that we are going to evaluate:

1. Evaluation for the performance of Web Service generation with the scenario.
2. Evaluation for the impact analysis of the Web Service evolution with the scenario.
3. Evaluation for the client adaptation with the scenario.

We organize this section as follows. Section II.4.1 describes the general process of the evaluation. Section II.4.2 evaluates the performance of the Web Service generation. Section II.4.3 evaluates the impact analysis. Section II.4.4 evaluates the client adaptation. Section II.4.5 concludes this section.

II.4.1 General description

In this section, we try to build several testing cases to take experiments for evaluating the proposed model. In Figure 70, we present the architecture of the testing system. There are 3 monitors which collect the testing data. The first one is located in the process of client adaptation which is responsible to capture the adaptation actions that the client application takes. The second is located in the process of collecting and analyzing the Web Service changes which is responsible to capture the impact analysis result that taken by the evolution platform. The third is located in the process of generating Web Service versions which is responsible to capture the latency of this process.

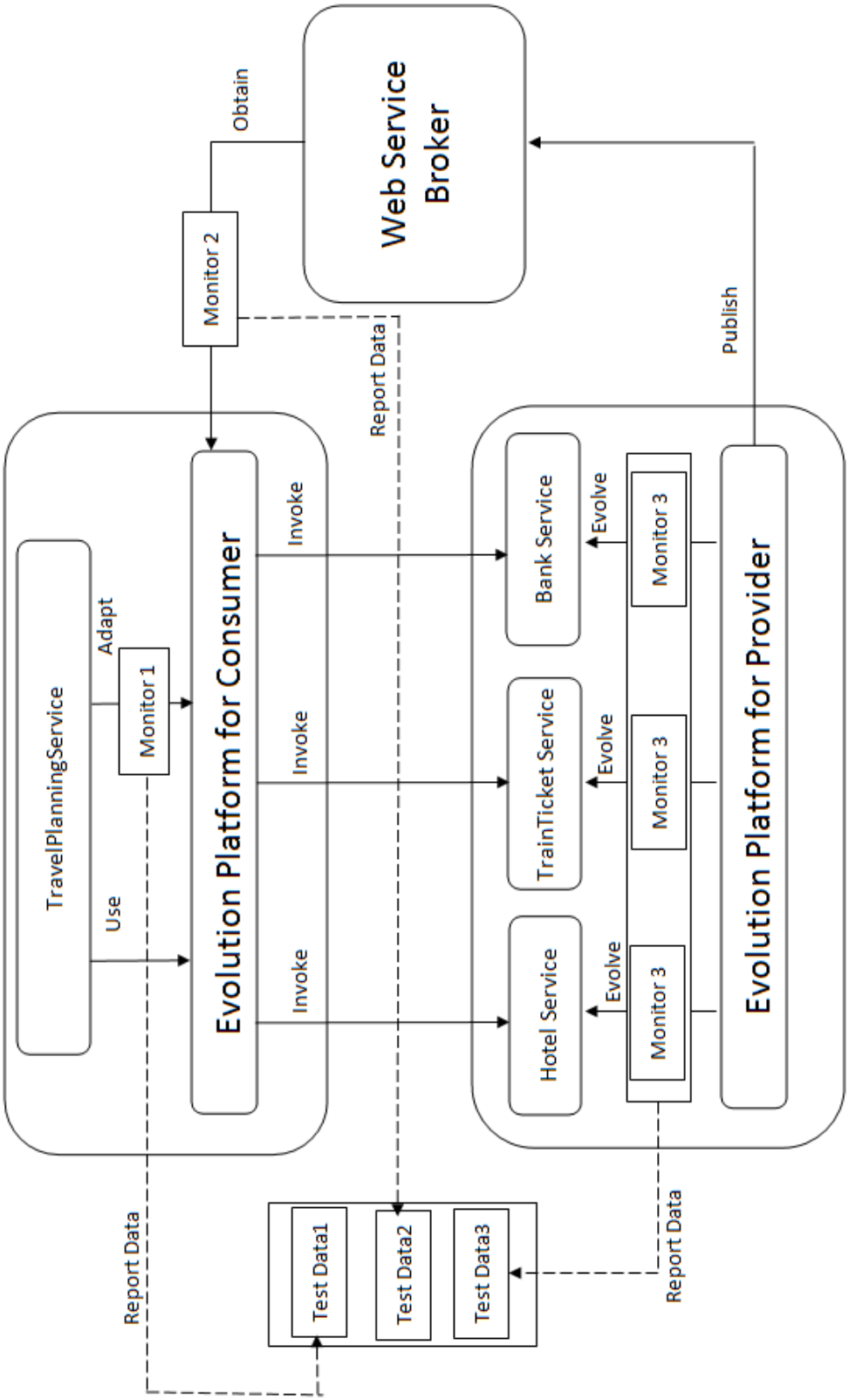

Figure 70 Overview of testing system

Table 16 presents the test environment that we use to generate the result data.

As described in the scenario section, we have defined the settings and pre-conditions for the evaluation as following list.

- The two Web Service providers evolve their Web Services (TS and HS) in a very fast speed (higher than 1 minute per version).
- TPS uses lots of instances (more than 50) of TS and HS and has to deal with the evolution for each of them.
- When each of the Web Service has been changed according to the description presented in Table 4, it also takes the corresponding roll back change and loop as so on. For example, each time when the TS has taken the change action 1 (adding an operation "bookFlight"), it will also taken a reverse change action (deleting an operation "bookFlight").

Table 16 Test Environment

Items	Descriptions
Machine	Hasee K580s i7 D1
Hard Drive	Crucial 120G SSD + Toshiba 1T HD
Memory	Kingston 16G
CPU	Intel i7-3610Q 2.3GHz
Operation System	Windows 7 Home Premium SP1
Graphics	Intel HD Graphics 4000+NVDIA GeForce GT 650

II.4.2 Evaluation for the Web Service generation

The evolution platform facilitates the Web Service provider with the high level evolution APIs for Web Service and the ability of dynamic Web Service evolution. In general, this process shortens the interval of publishing new versions and lowers the cost of updating the system by shutting down the machines. However, dynamic generation of the Web Service still results in performance decrease in practical systems comparing to the normal publishing actions. In Section II.4.2, the evaluation is to estimate how the performance will be decreased with the Web Service generation. The testing process is to compare the latency of the publishing action with and without Web Service generation for each Web Service with a very high frequency of 1 version per 10 mile seconds. The evolution script for each Web Service is listed in Section II.3.2.3. The evaluation base line is the normal publishing action typically using "org.apache.cxf.frontend.ServiceFactoryBean.create ()" method. Then for each change action of the two Web Services TS and HS, the testing results are shown in Figure 71– Figure 76.

II.4.2.1 Change action 1 for TS

The change action 1 for TS is to add an operation to the existing Web Services. We stimulate the process by continuously adding the operation to its last version to examine if the generation process cost too much time when the Web Service adds more things to its model.

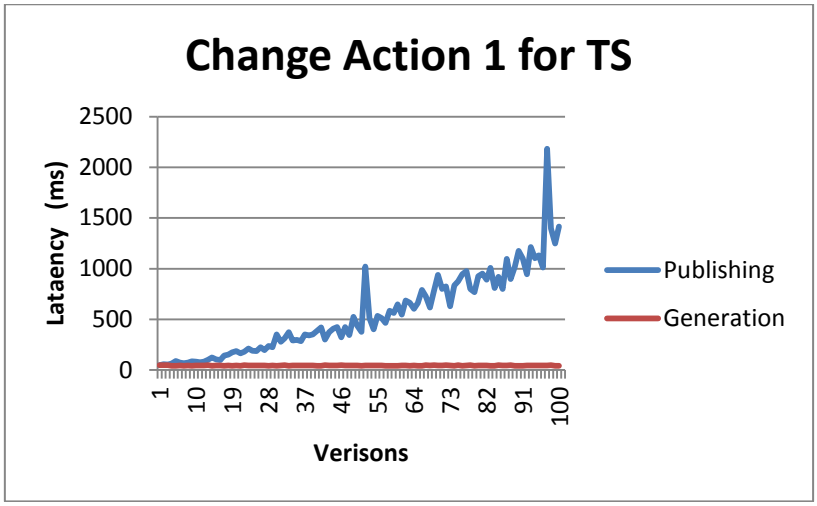

Figure 71 Result for Change Action 1 for TS

We can see that in Figure 71, when there are more and more operations added to the Web Service, the latency for publishing increases because it needs more time for Apache CXF to create the instance of the Web Service especially the WSDL documents. However, the latency for Web Service generation remains at a stable level even when the Web Service becomes bigger. Compared to publishing the Web Service, the latency of the Web Service generation is almost ignorable.

II.4.2.2 Change action 2 & 3 & 4 for TS

The change actions 2, 3, 4 for TS are similar to a Web Service since they are both change actions to modify the messages of the Web Service. The results of the three simulations also show that they behave similar in the patterns of the chart. To simulate the three change actions, we do the change actions and the opposite rollback change action and loop so on.

No matter how the Web Service execution engine modifies the Web Service messages and how long it will take to publish the Web Services, the latency of the Web Service generation almost remains at a relatively low and stable value. Peaks occasionally occur. Notice that the average value of the latency for change action 4 is a little lower than the change action 2 and 3 because the change action 4 only changes the implementation of the Web Service. It does not change the interface. The latency growths for publishing Web Service of change

actions 2, 3, and 4 are less than change action 1. They are caused by CXF Web Service management.

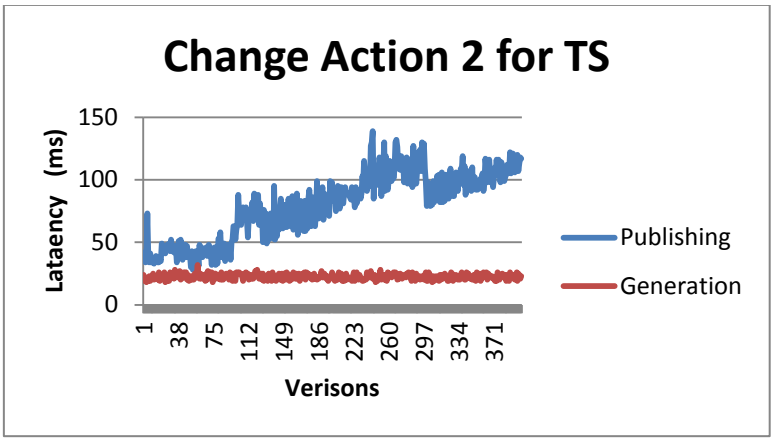

Figure 72 Result for Change Action 2 for TS

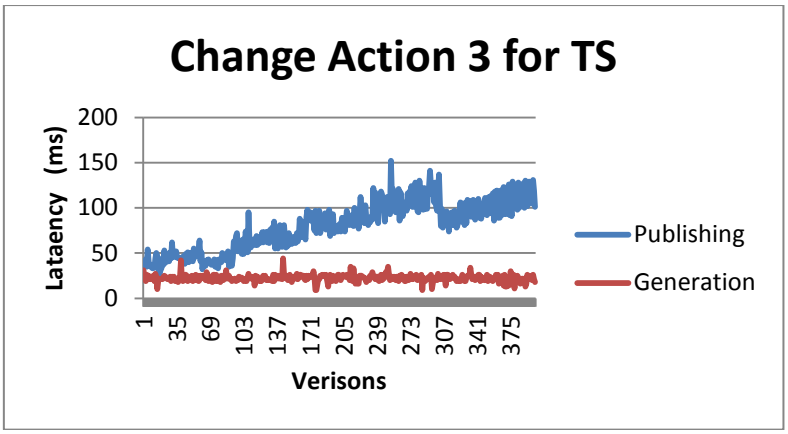

Figure 73 Result for Change Action 3 for TS

In conclusion, the Web Service execution engine makes little impact on the runtime performance of publishing the Web Services when it generates the Web Service versions dynamically.

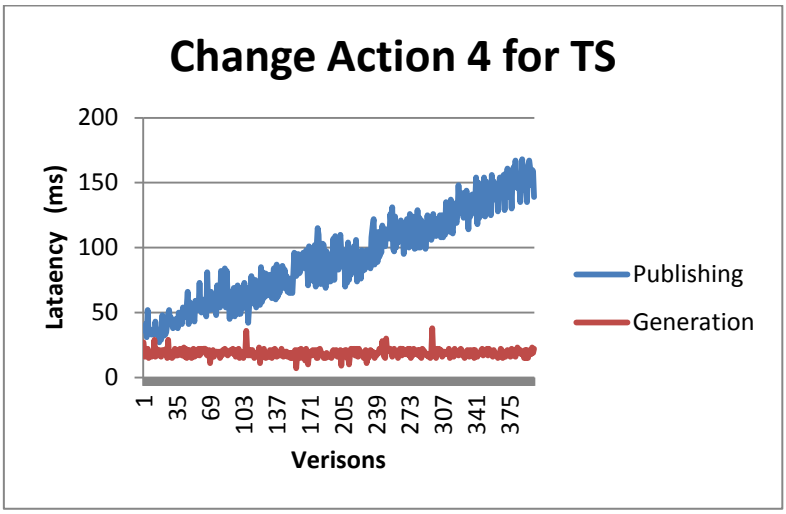

Figure 74 Result for Change Action 4 for TS

II.4.3 Evaluation for the impact analysis

Table 17 Result of Impact Analysis for Web Service Evolution

Change Action	Change	Action	IR Def II.1.9	IF Def II.1.8	IC Def II.1.7	d Def II.1.10	$IC \times d$
C1TS	Operation: bookFlight	Add	0	1/23	0.0435	0	0
	XML Element: arg0	Add	0	1/23	0.0435	0	0
	XML Element: amount	Add	0	1/23	0.0435	0	0
	XML Element: customerID	Add	0	1/23	0.0435	0	0
	Message: bookFlight	Add	0	1/23	0.0435	0	0
	Message: bookFlight Response	Add	0	1/23	0.0435	0	0
	XML Element: bookFlight return	Add	0	1/23	0.0435	0	0
C2TS	XML Element: bookTrainTickets customerID	Add	1	1/23	1.0435	0.0268	0.0280
C3TS	XML Element: checkAvailable arg0	Modify	1	0	1	0.0357	0.0357
C4TS	Implementation: bookTrainTickets	Modify	0	0	0	0.0268	0
C1HS	Implementation: bookHotelRooms	Modify	0	0	0	0.0390	0
C2HS	XML Element: bookHotelRooms customerID	Add	1	1/17	1.0588	0.0390	0.0413
C3HS	Operation: bookHotelRooms	Modify	1	0	0	0.0390	0
C4HS	XML Element: checkAvailable RoomNumReponse return	delete	1	1/16	1.0625	0.0519	0.0551

When the Web Service provider TS and HS have generated lots of new versions during the Web Service evolution, TPS has to estimate the impact of the changes caused by the evolution for each Web Services. For the change actions that we listed in Table 4 and Table 5, we got the result of the impact values showing in Table 17 according to the analysis model presented in Section II.2.4. Notice that "C1TS" means the change action 1 of TS.

According to the Table 17, there are something meaningful with the data presented in the Table.

- The change action of adding an operation is totally adaptable for a Web Service client. (C1TS)
- The change action of modifying the implementation is totally adaptable for a Web Service client. (C4TS, C1HS)
- The change action of renaming a Web Service is totally adaptable for a Web Service client since the system can automatically adapt the client to this change action. (C3HS)
- The change action of modifying an element that is more dependent for a Web Service client can causes higher impact. (C2TS compared with C3TS)
- The change action on an element in a smaller Web Service results in a higher impact. (C2HS compared to C2TS)
- The change action of removing an element can result in high impact. (C4HS)

Table 18 Dependency Information

Consumer	Dependent Operation	Dependent Service	Calls (k)
TPS	checkAvailable	TS	2000
TPS	bookTrainTicket	TS	1500
TPS	checkAvailableRoomNum	HS	2000
TPS	bookHotelRooms	HS	1500

Table 17 shows the impact analysis result without the adaptation mechanism that the evolution platform provides for the system. With the support of the client adaptation, the Web Service consumer can lower the impact of the Web Service evolution as shown in Table 19. We have indicated in Section II.3.2.3 that which of the change actions are set with an attribute “adapt” to “true”.

The cell where is marked with “m” means that the change action is manually defined as adaptable change. The last change action “C4HS” has not been specified with “adapt=true” by the Web Service provider. That means it may cost more to adapt the client application to the Web Service evolution by manual operations.

That is to say, the adaptation of the proposed model can reduce the value of the change impact ($IC \times d$). According to Table 19, the change impact of C2TS, C2TS, and C2HS have been reduced. The change impact of C4HS cannot be decreased because the change action is not adaptable.

Table 19 Result of Lowed Impact Analysis for Web Service Evolution

Change Action	Change	Action	Lowered IR Def II.1.9	IF Def II.1.8	IC Def II.1.7	d Def II.1.10	Lowered $IC \times d$
C1TS	Operation: bookFlight	Add	0	1/23	0.0435	0	0
	XML Element: arg0	Add	0	1/23	0.0435	0	0
	XML Element: amount	Add	0	1/23	0.0435	0	0
	XML Element: customerID	Add	0	1/23	0.0435	0	0
	Message: bookFlight	Add	0	1/23	0.0435	0	0
	Message: bookFlightRespons e	Add	0	1/23	0.0435	0	0
	XML Element: bookFlight return	Add	0	1/23	0.0435	0	0
C2TS	XML Element: bookTrainTickets customerID	Add	0 (m)	1/23	0.0435	0.0268	0.0011
C3TS	XML Element: checkAvailable arg0	Modify	1	0	0	0.0357	0.0357
C4TS	Implementation: bookTrainTickets	Modify	0	0	0	0.0268	0
C1HS	Implementation: bookHotelRooms	Modify	0	0	0	0.0390	0
C2HS	XML Element: bookHotelRooms customerID	Add	0 (m)	1/17	0.0588	0.0390	0.0023
C3HS	Operation: bookHotelRooms	Modify	0 (m)	0	0	0.0390	0
C4HS	XML Element: checkAvailableRoo mNumReponse return	delete	1	1/16	1.0625	0.0519	0.0551

II.4.4 Evaluation for the client adaptation

In the last section, we have obtained the impact of each change action. In Table 19, the impact value for a change action lowered $IC \times d$ where it is 0 indicates that this change action can be automatically adapted by the evolution

platform for the Web Service consumer. Normally, the attribute “adapt” is defined by Web Service provider with a corresponding adaption behavior indicated in Table 8. In our experiments, we have defined an adaption behavior for each adaptable change presented in Table 19 (C2TS, C2HS and C3HS) action as shown in Table 20.

According to Table 20, the evolution platform will automatically and dynamically generate new Web Service references for the Web Service consumers to enable them to catch up with the new version of the Web Service by using the old version of the client.

For C2TS, the automatically generated proxy is shown in Figure 67 presented in Section II.3.3.2.

Table 20 Adaption Behaviors for the Adaptable Change Actions

Change Action	Adapt Behavior
C2TS	<adaptelement type=”value”> </adaptelement>
C2HS	<adaptelement type=”value”> </adaptelement>
C3HS	<adaptoperation type=”default”>

For C2HS, the automatically generated proxy is shown in Figure 75.

```
package zuowei.provider;
import org.apache.cxf.frontend.ClientProxyFactoryBean;
public class YouthHotelService$v11$classProxy implements
IYouthHotelService$v10 {
   public IYouthHotelService$v11 ref;
   public PayModel bookHotelRooms(String type, Date start, Date end)
   {
      return ref.bookHotelRooms(type, start, end, “”);
   }
   public YouthHotel$v11$classProxy(String serviceAddress)
   {
      ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
      factory.setServiceClass(IYouthHotelService$v11.class);
      factory.setAddress(serviceAddress);
      ref = (IYouthHotelService$v11) factory.create();
   }
}
```

Figure 75 Generating Web Service Proxy for C2HS

For C3HS, the automatically generated proxy is shown in Figure 76.

The generation process for adapting the client applications to the Web Service evolution is not free. There will also be performance decrease comparing with the normal usage, especially when the Web Service evolves in a

high speed. To understand the performance decrease for the Web Service client adaptation, we need to observe the latency for the Web Service invocation from the perspective of the business module of the client application. To evaluate the performance, we set the Web Service invocation without client application as the base line. We let the Web Service client application of TPS continuously invoke the planning trip business and capture the latency for each of the invocation 10 times per second and 2000 times in total. At the provider side, the provider periodically modifies the Web Service TS and HS with C2TS, C2HS, and C3HS with the frequency of 1 time per second. When a new version of the Web Service is published, the old one is deprecated so that the client application of TPS can only visit the newest version. If occasionally the client application cannot obtain the new published version, it will keeps waiting and searching for the newest version on the broker.

```
package zuowei.provider;
import org.apache.cxf.frontend.ClientProxyFactoryBean;
public class YouthHotelService$v11$classProxy implements
IYouthHotelService$v10 {
   public IYouthHotelService$v11 ref;
   public PayModel bookHotelRooms(String type, Date start, Date end)
   {
      return ref.bookRooms(type, start, end);
   }
   public YouthHotel$v11$classProxy(String serviceAddress)
   {
      ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
      factory.setServiceClass(IYouthHotelService$v11.class);
      factory.setAddress(serviceAddress);
      ref = (IYouthHotelService$v11) factory.create();
   }
}
```

Figure 76 Generating Web Service Proxy for C3HS

By pre-estimation, the cost of the performance for the client adaptation includes 3 parts:

- The test of checking the availability. This is mainly determined by the network status. In this scenario, we run all the tests in one single machine, so this part contributes little to the result.
- The re-look up for the evolved Web Services.
- The adaptation including retrieving the changes through XML parser, analyzing the impact, and the creation of the proxies.

As a result, the generated data is shown in Figure 77. We can see that the cost of the client adaptation varies from 20 to 80 ms for the latency of the Web Service invocation. Most of the performance lost is contributed by the creation of the Web Service client by Apache CXF. Peaks only appear when HS and TS

take evolution by publishing new versions. The average cost of the client adaptation is only around 45 mille seconds. This result proves that the proposed approach for client adaptation is worth of further investigation in the aspect of engineering.

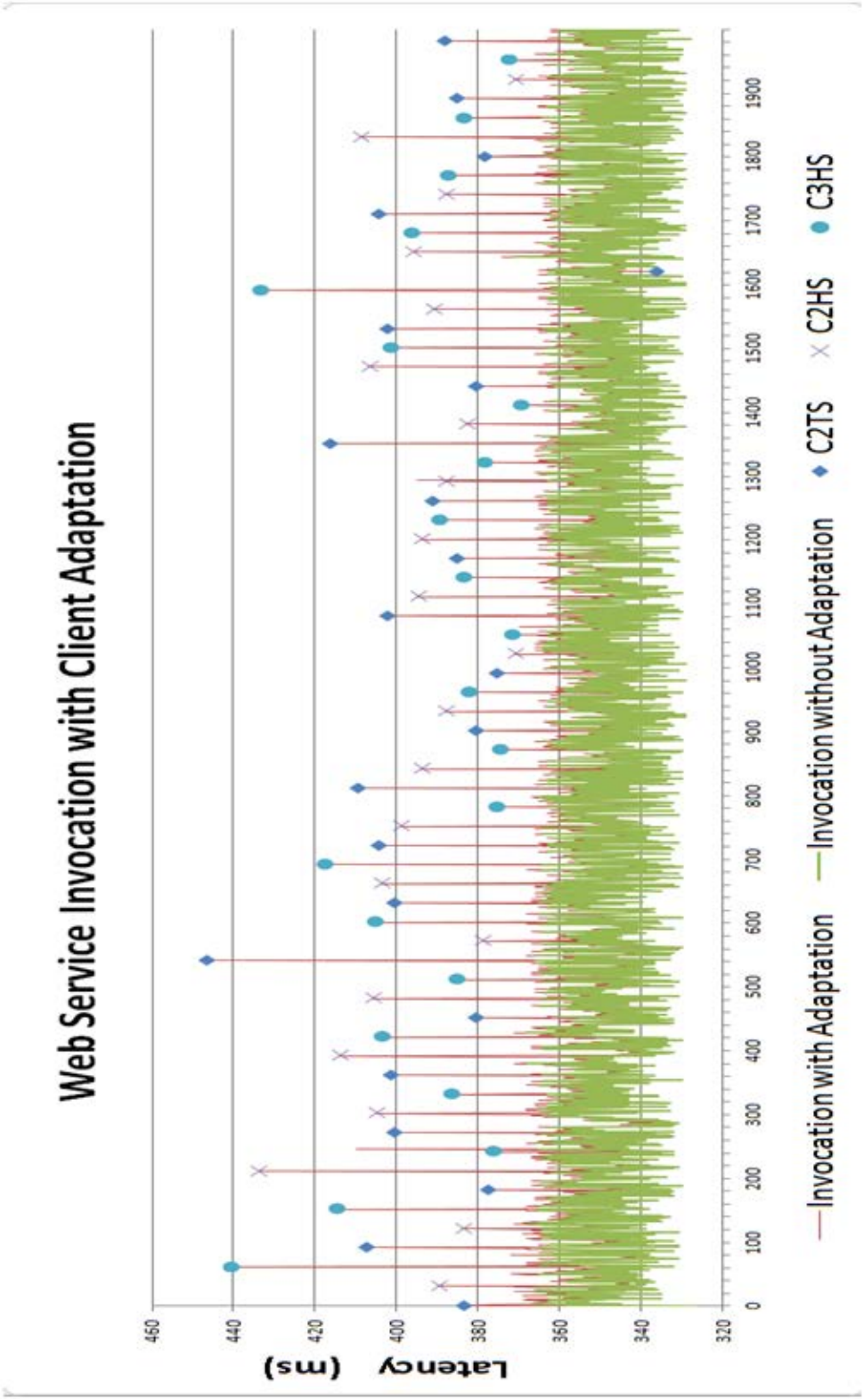

Figure 77 Cost of Client Adaptation

II.4.5 Limitations

The proposed model performs quite well during the experiments except some blind sites. We present the defects and limitations from two perspectives. The first is the theoretical part which reveals the limitations of the proposed model. The second is the engineering part in which there seems to be some unhealthy problems during the experiments.

From the theoretical perspective, there are two main problems exist in the model:

1. The types of change actions during Web Service evolution are limited. Until now we only introduce the methods to deal with interface changes and part of the methods for dealing with the implementation changes. However, there are more extra types of changes in practical systems and they are also very interesting for the stakeholders. There should be more ideas for generating, propagating, analyzing and taking adaptation to more types of changes.
2. The Web Service client adaptation is fairly constraint with the adaptation types. Only several adaptation behaviors are defined in Table 1. This is quite complex since the change of the Web Service systems always implies the change of the business logics. To automate the issues of Web Service evolution in the upper layer, people must prepare more complex domain knowledge and business rules. However, modeling the domain knowledge is a little far away from the software systems and is not included in the context of this thesis.

From the engineering perspective, some metrics perform quite problematic during the evaluation progress.

1. In Section II.4.2, the latency of generating the Web Service increases significantly along with the increase of the number of the published Web Service versions. For Figure 71, this is normal because we constantly add more operations to the Web Service which makes it more and more bloated so the cost for the creation of the WSDL documents increases. However, it is quite confusing that Figure 72, Figure 73, and Figure 74 also perform like this since we did not add too many things to the Web Service. Through a set of tests for each of the components, we have found that the Apache CXF was the bottleneck. But after all, the generation of the Web Service is very safe if the number of the maintained Web Service in one single JVM is under 100. In practice, this is a common case.
2. The impact analysis lacks of more complex scenarios to assess its reliability. In this scenario, we only define 4 types of changes for each of HS and TS. To compare the results for each type of change

action, we properly explained why the value is high or low. But it is quite complex to speak out which value is the best and which value can be considered as safe and so on.

Part III: Conclusions and perspectives

III.1 Conclusion

In this thesis, we have presented a model for Web Service evolution which facilitates the Web Service stakeholders to deal with the issues involved in Web Service evolution. Let's go back to the questions that we proposed in the introduction section and see how the proposed model answers them.

1. How to model Web Service Evolution?

   In this thesis, we define the roles and their corresponding responsibilities during the Web Service evolution. Firstly, we explained what the Web Service evolution is through defining the Web Service changes, and secondly when to take actions to deal with the Web Service evolution. Thirdly we provided the methods and tools to deal with the Web Service for each of the stakeholders.

2. How to extract exactly and completely the changes for the stakeholders?

   In our change-centric model, the Web Service stakeholder does not need to retrieve the Web Service changes through different approaches like in the past since they can result in omissions and even errors. Each of the published versions of a Web Service is accompanied with a Web Service change description which is generated automatically by using the underlying programming framework at the provider side. Thus, all of the other participants like the brokers and the consumers can keep the same understanding for the changes as the provider.

3. How to evolve the Web Service at runtime in a graceful manner?

   The programming framework introduces the script system with a set of high level APIs which makes it easy for the Web Service developer to evolve the Web Service at runtime. The script system takes the user's commands as input and automatically applies them on both of the interfaces and implementations of the deployed Web Services. The Web Service developer does not need to maintain the versions of the Web Service and does not need to involve in the management of the changes.

4. How to analyze impact of the Web Service changes?

   Thanks to the definition of the Web Service evolution and the formalization of the Web Service change description. The Web Service consumer can obtain the correct and complete set of the Web Service changes. Then we proposed a method for the impact analysis of the Web Service changes according to 3 facts. The first is that the impact of the Web Service evolution depends on the change itself. The second is that the impact of the Web Service evolution depends on the usage of each consumer. The second is

that the impact of Web Service evolution depends on how the change action alters the Web Service information model. Based on the 3 facts, the change-centric model proposes an impact analysis method to help the Web Service consumer to determine if the Web Service is adaptable and estimate the cost of adapting to the Web Service evolution.

5. How to adapt the Web Service client applications to the evolved Web Services in an automatic and dynamic way.

   After analyzing the impact of the Web Service evolution, the Web Service consumer can determine which changes are compatible, adaptable, and inadaptable. Once the change is considered as an adaptable change, the client application can take automatically and dynamically adaptation by a specified architecture style and a progress of proxy generation. The client adaptation is semi-automatic because it needs the support at the provider side to specify the “adapt” attribute.

At last, a set of experiments have been performed to evaluate the proposed model. The results in the evaluation section show that the proposed model and implementation present well in dealing with the issues presented in the proposed motivation scenario in Section II.1. It could support both of the Web Service provider and the consumer to deal with the Web Service evolution in a graceful way and little performance lost to the system.

In general, this thesis contributes both of the theoretical study and engineering work to the issues of the Web Service evolution. Although the experiments show that all the promises have been realized except few limitations, this is only the beginning of our perspectives.

III.2 Perspectives

III.2.1 Future work on the Change-centric model for Web Service evolution

Firstly, we present the future work that may be done for this thesis which could cover the existing limitations, disadvantages, and defects.

As we have mentioned in Section II.4.5, there should be more work to be done on extending the types of changes for the change-centric model. At least the following types of changes should be considered.

Semantic changes. Before, people are only interested in the Web Service interfaces and do not pay attention to how the Web Service is implemented when they are using the Web Service. Now, as the Web Service interface descriptions

are more and more constraint in the discovery and ranking for the Web Services and the number of the Web Service become larger and larger, more and more researchers in the community are involved in the issues of Web Service semantics. The Web Service semantics describe the meanings of multiple aspects of the Web Service with the ontology of the domain knowledge. It is quite useful in discovering and selecting the Web Service in a very large pool. In current stage, the Web Service semantics are relatively more stable than the Web Service interfaces so less people focus on the evolution of the Web Service semantics. However, with the growth of the Web Services, there will be more and more scenarios in which the stakeholders have to take into account the evolution of the Web Service semantics.

QoS changes. Strictly speaking, the change of the quality of service is not or not only decided by the Web Service provider so it is not suitable to adopt the proposed model in this thesis to deal with QoS changes. For a certain consumer, the meaningful QoS is the set of parameter values that he observes from his point of views. There is no need for the generation of the Web Service changes and no need for the propagation of the Web Service changes. Furthermore, the adaptation is also complicated and needs more extra studies on this QoS changes.

Change-centric model is also limited when dealing with the changes which are involved in lots of dependencies. In current stage in Section II.2.3, we present the resource management by manually marking the changed or unchanged resources to determine if they will be loaded or be linked with the old ones. If too many resources that the Web Service is dependent on need to be marked as changed, it will be also costing time when evolving the Web Services.

Another limitation is the way to deal with the status of the Web Service. Unlike the restful Web Service which recommends using stateless Web Services, the SOAP based Web Service usually holds the transaction states for the business process. No matter the online system is shut down or not when evolving the Web Service, the retired version of the Web Service will lose all the transaction states at runtime. To migrate the Web Service status from one version to another is quite complex and needs more discussions.

III.2.2 Future work on the Web Service evolution

After describing the future work of this thesis, we secondly discuss the future of Web Service evolution. As we mentioned in the introduction section, the major target of our research on the software engineering is to realize dynamization and modularization. Furthermore, the major target of the research on the Web Service evolution is to realize the autonomic computing.

The research on the Web Service evolution pays more attention to the dynamization aspect of the Web Service. We emphasize the dynamic features of the system in many places in this thesis such as dynamically generating the Web

Service versions and dynamically taking client adaptation. In fact, the dynamic Web Service evolution is only the first step. The final target is to build a type of software component which could adapt itself to the environment through dynamically adjusting its runtime behavior.

The Figure 78 is showing that the circle becomes bigger when the further achievement is made on the Web Service. Initially people are developing Web Services functionalities. When people intend to make use of these functionalities across the platforms and environments, they have to build a series of standard descriptions for the Web Service to encapsulate the Web Services such as interface, semantics and QoS. A further step is to grant the Web Service systems the abilities of Web Service evolution such as descriptions, impact analysis, generations, monitors and strategies through a set of models and tools. This step is covered in this thesis. And then a final step is to enable the Web Service with self-healing, self-managing, self-adaptation and auto-composition.

Figure 78 Class Graph for Web Service Evolution

Autonomic computing advocates to enable the Web Service to automatically and intelligently adjust its internal behaviors to fix the problems, optimize the resources, adapt to the environments, and protect the Web Service itself once it is deployed at runtime. Let's take a preliminary simple scenario that shown in Figure 79.

Figure 79 Web Service Self-healing

There is a Web Service named "WeatherReportService" which provides several operations to provide all the data of metrics of the weather. In one moment, the operational manager finds that all the operations of this Web Service present the unhealthy patterns on the monitor view and many of their customers are blocked by the great latencies. After a series similarity analysis, random walking, and root cause rankings [92], he finds that the operation "QueryRain" results in a great cost in the garbage collection process. To rapidly recover the whole Web Service temporarily, the most effective method is to shutdown the operation "QueryRain()" so that the other operations can work normally. The behavior of the detecting and shutdown is actually a type of the self-healing process. And one of the key features of the systems like this is the ability of the Web Service dynamic evolution.

III.2.3 Web Service evolution and the Big Data

The example we proposed in Figure 79 is part of the application performance management (APM). As widely known, the four steps of APM is to 1) monitor, 2) analyze, 3) optimize, and 4) predict. Figure 79 can actually cover the first 3 steps. Nowadays, the hottest topic for prediction is the technology of big data. The big data technology reveals the intrinsic features of the things without common logical reasoning. Instead, it takes effect by modeling and analyzing all of its related data using the computer technologies such as machine learning, data mining, and cloud computing. In normal situation, most of the

people consider that the Web Service belongs to or partly belongs to the cloud computing. In future, the next target of the research on the Web Service evolution is also the prediction using the big data technologies.

The Web Service developers and the managers of system operation handle the mistakes and the problems of the Web Services. However, none of them can discover the natures of the Web Service when it evolves like any other creatures so no one can predict the upcoming behaviors of the Web Services. Now if we consider the big data technologies, what can be done next?

We have built a change specification which provides a standard and formal definition for the Web Service changes in this thesis. Each published version is accompanied with a change description in XML format. The change description is propagated through the whole system and analyzed by anyone who is interested in the evolution of the Web Service. However, it is not the ending. If we collects all the change descriptions from all the Web Service brokers, we actually obtain a Web Service Evolution Repository as shown in Figure 80 which can be used for the big data technologies to analyze and predict the future of software services.

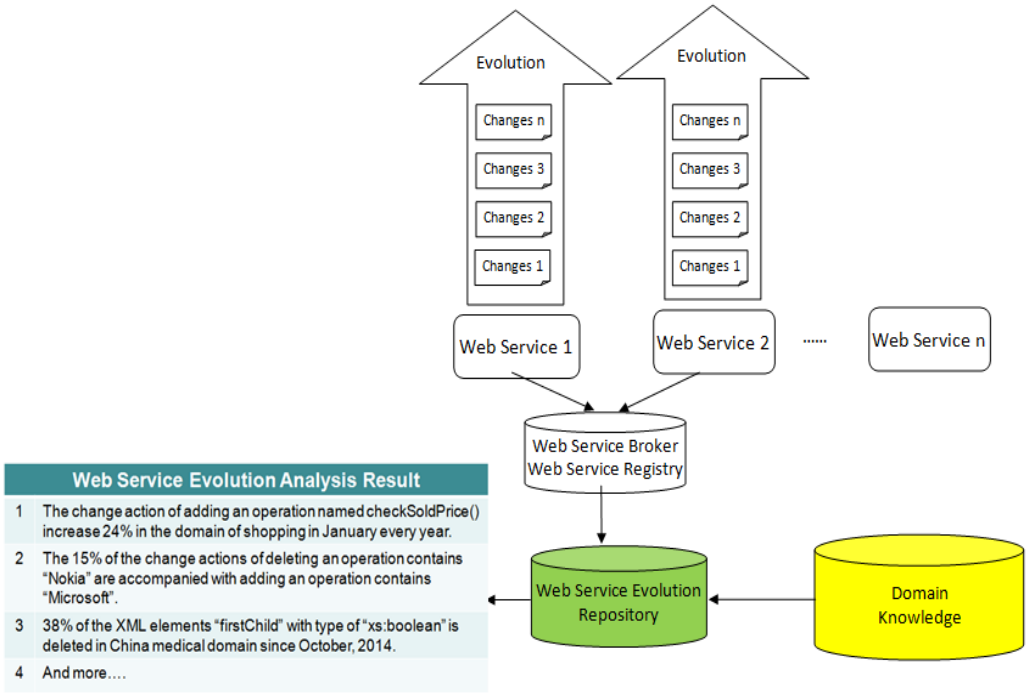

Figure 80 Big Data analysis on the Web Service evolution

In this Figure we give some examples of the result of the Web Service evolution analysis using the big data. Taking the first one as an example, it is quite possible that the shopping service may add an operation named "checkSoldPrice" because it is close to the sales season in Europe every year. Then for the Web Service integrators and consumers who are using the shopping service, they are recommended to add new features to their applications to integrate this feature of the shopping services.

Bibliography

[1] Web Service Architecture. [Online] available at: https://www.w3.org/TR/ws-arch/

[2] Pan W, Chen S, Feng Z. Service-Oriented ontology and its evolution[M]//Advances in Grid and Pervasive Computing. Springer Berlin Heidelberg, 2012: 109-121.

[3] Sathyavathy P, Sneha C, Uma B, et al. Semantic and QoS based Web Service Selection using a Multi Agent System[C]//IICAI. 2005: 2521-2533.

[4] Ryu S H, Casati F, Skogsrud H, et al. Supporting the dynamic evolution of web service protocols in service-oriented architectures[J]. ACM Transactions on the Web (TWEB), 2008, 2(2): 13.

[5] Xie Q, Wu K, Xu J. QoS driven web services evolution[C]//Complex, Intelligent and Software Intensive Systems (CISIS), 2011 International Conference on. IEEE, 2011: 329-334.

[6] Lehman M M. Programs, life cycles, and laws of software evolution[J]. Proceedings of the IEEE, 1980, 68(9): 1060-1076.

[7] Lehman M M, Belady L A. Program evolution: processes of software change[M]. Academic Press Professional, Inc., 1985.

[8] Lehman M M. Laws of software evolution revisited[M]//Software process technology. Springer Berlin Heidelberg, 1996: 108-124.

[9] Lehman M M, Ramil J F. Software evolution—Background, theory, practice[J]. Information Processing Letters, 2003, 88(1): 33-44.

[10] Agusa K. Software Engineering Evolution[C]//Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of. IEEE, 2004: 3-8.

[11] Mens T, Buckley J, Zenger M, et al. Towards a taxonomy of software evolution[C]//Proceedings of the International Workshop on Unanticipated Software Evolution. 2003 (LAMP-CONF-2003-005).

[12] Oreizy P, Medvidovic N, Taylor R N. Architecture-based runtime software evolution[C]//Proceedings of the 20th international conference on Software engineering. IEEE Computer Society, 1998: 177-186.

[13] Gurguis S A, Zeid A. Towards autonomic web services: Achieving self-healing using web services[C]//ACM SIGSOFT Software Engineering Notes. ACM, 2005, 30(4): 1-5.

[14] Becker S, Brogi A, Gorton I, et al. Towards an engineering approach to component adaptation[M]. Springer Berlin Heidelberg, 2006.

[15] Canfora G. Software evolution in the era of software services[C]//Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of. IEEE, 2004: 9-18.

[16] Gupta D, Jalote P, Barua G. A formal framework for on-line software version change[J]. Software Engineering, IEEE Transactions on, 1996, 22(2): 120-131.

[17] Lee I. Dymos: a dynamic modification system[D]. University of Wisconsin, Madison, 1983.

[18] Dowling J, Cahill V, Clarke S. Dynamic software evolution and the k-component model[C]//Workshop on Software Evolution, OOPSLA. 2001, 2001.

[19] Dowling J, Cahill V. The k-component architecture meta-model for self-adaptive software[M]//Metalevel Architectures and Separation of Crosscutting Concerns. Springer Berlin Heidelberg, 2001: 81-88.
[20] OSGi Alliance. [Online] available at: http://www.osgi.org.
[21] Andrikopoulos V. A theory and model for the evolution of software services[R]. School of Economics and Management, 2010.
[22] Canal C, Poizat P, Salaun G. Model-based adaptation of behavioral mismatching components[J]. Software Engineering, IEEE Transactions on, 2008, 34(4): 546-563.
[23] Xie X, Zhang W. A checking mechanism of software component adaptation[C]//Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference. IEEE, 2006: 347-354.
[24] Chainbi W, Mezni H, Ghedira K. An autonomic computing architecture for self-* Web services[M]//Autonomic Computing and Communications Systems. Springer Berlin Heidelberg, 2009: 252-267.
[25] Passerone R, De Alfaro L, Henzinger T A, et al. Convertibility verification and converter synthesis: Two faces of the same coin[C]//Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. ACM, 2002: 132-139.
[26] Hiel M, Weigand H, Van Den Heuvel W J. An adaptive service-oriented architecture [M]//Enterprise Interoperability III. Springer London, 2008: 197-208.
[27] Di Nitto E, Ghezzi C, Metzger A, et al. A journey to highly dynamic, self-adaptive service-based applications[J]. Automated Software Engineering, 2008, 15(3-4): 313-341.
[28] Kaminski P, Müller H, Litoiu M. A design for adaptive web service evolution [C] // Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems. ACM, 2006: 86-92.
[29] Fokaefs M, Mikhaiel R, Tsantalis N, et al. An empirical study on web service evolution[C]//Web Services (ICWS), 2011 IEEE International Conference on. IEEE, 2011: 49-56.
[30] Fokaefs M, Stroulia E. WSDarwin: Automatic web service client adaptation[C]//Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research. IBM Corp., 2012: 176-191.
[31] Fokaefs M, Stroulia E. The WSDarwin Toolkit for Service-Client Evolution[C]//Web Services (ICWS), 2014 IEEE International Conference on. IEEE, 2014: 716-719.
[32] Fokaefs M, Stroulia E. WSDARWIN: A Decision-Support Tool for Web-Service Evolution[C]//Software Maintenance (ICSM), 2013 29th IEEE International Conference on. IEEE, 2013: 444-447.
[33] Marwaha P, Banati H, Bedi P. WSDL-TC: Temporal Customization of Web Services[J]. Journal of Network and Innovative Computing, 1(2013): 234-247.
[34] Banati H, Bedi P, Marwaha P. WSDL-temporal: An approach for change management in web services[C]//Uncertainty Reasoning and Knowledge Engineering (URKE), 2012 2nd International Conference on. IEEE, 2012: 44-49.
[35] Chaturvedi A. Automated Web Service Change Management AWSCM-A Tool[C]//Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014: 715-718.

[36] Wang Y, Wang Y. A survey of change management in service-based environments[J]. Service Oriented Computing and Applications, 2013, 7(4): 259-273.
[37] Romano D, Pinzger M. Analyzing the evolution of web services using fine-grained changes[C]//Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, 2012: 392-399.
[38] Treiber M, Juszczyk L, Schall D, et al. Programming evolvable web services[C]//Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems. ACM, 2010: 43-49.
[39] Treiber M, Andrikopoulos V, Dustdar S. Calculating service fitness in service networks[C]//Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. Springer Berlin Heidelberg, 2010: 283-292.
[40] Treiber M, Truong H L, Dustdar S. Semf-service evolution management framework[C]//Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference. IEEE, 2008: 329-336.
[41] Juszczyk L, Truong H L, Dustdar S. Genesis-a framework for automatic generation and steering of testbeds of complexweb services[C]//Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on. IEEE, 2008: 131-140.
[42] Treiber M, Truong H L, Dustdar S. On analyzing evolutionary changes of web services[C]//Service-Oriented Computing–ICSOC 2008 Workshops. Springer Berlin Heidelberg, 2009: 284-297.
[43] Andrikopoulos V, Benbernou S, Papazoglou M P. Managing the evolution of service specifications[C]//Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2008: 359-374.
[44] Vara J M, Verde J, Andrikopoulos V, et al. An EMF-based toolkit for reasoning on web services evolution[C]//Proceedings of the workshop on ACadeMics Tooling with Eclipse. ACM, 2013: 4.
[45] Papazoglou M P. The challenges of service evolution[C]//Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2008: 1-15.
[46] Chaturvedi A. Automated Web Service Change Management AWSCM-A Tool[C]//Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014: 715-718.
[47] Zou Z L, Fang R, Liu L, et al. On synchronizing with web service evolution[C]//Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008: 329-336.
[48] Kajko-Mattsson M, Lewis G A, Smith D B. Evolution and maintenance of soa-based systems at sas[C]//Hawaii International Conference on System Sciences, Proceedings of the 41st Annual. IEEE, 2008: 119-119.
[49] J. Kenyon, “Web service versioning and deprecation,” Jan. 2003. [Online]. Available: http://soa.sys-con.com/node/39678
[50] J. Evdemon, “Principles of service design: Service versioning,” Aug. 2005. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms954726.aspx
[51] M. Russell, “Manage message contract changes with versioning,” Aug. 2005. [Online]. Available: http://www.ibm.com/developerworks/web/library/wa-msgvers/index.html

[52] M. Endrei, M. Gaon, J. Graham, K. Hogg, and N. Mulholland, “Moving forward with web services backward compatibility,” May 2006. [Online]. Available: http:// www.ibm.com/developerworks/java/library/ws-soa-backcomp/index.html?ca=drs-
[53] G. Bechara, “Web services versioning,” Apr. 2007. [Online]. Available: http: //www.oracle.com/technology/pub/articles/web services versioning.html
[54] K. Jerijærvi and J. Dubray, “Contract versioning, compatibility and com-posability,” Dec. 2008. [Online]. Available: http://www.infoq.com/articles/ contract-versioning-comp2
[55] D. Parachuri and S. Mallick, “Service versioning in SOA,” Dec. 2008. [Online]. Available: http://www.infosys.com/offerings/IT-services/soa-services/ white-papers/pages/index.aspx
[56] G. Flurry, “Service versioning in SOA,” Oct. 2008. [Online]. Avail- able: http://www.ibm.com/developerworks/websphere/techjournal/0810 col flurry/ 0810 col flurry.htm
[57] Narayan A, Singh I. Designing and versioning compatible Web services[J]. IBM DeveloperWorks, 2007, 28: 30.
[58] K. Brown and M. Ellis, “Best practices for web services versioning,” Jan. 2004. [Online]. Available at: http://www.ibm.com/developerworks/webservices/library/ws-version/
[59] Kajko-Mattsson M, Lewis G A, Smith D B. A framework for roles for development, evolution and maintenance of SOA-based systems[C]//Proceedings of the international workshop on systems development in SOA environments. IEEE Computer Society, 2007: 7.
[60] Liu R, Chen F, Yang H, et al. Agent-based web services evolution for pervasive computing[C]//Software Engineering Conference, 2004. 11th Asia-Pacific. IEEE, 2004: 726-731.
[61] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du, “A version- aware approach for web service directory,” in ICWS 2007, Jul. 2007, pp. 406–413.
[62] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, “End-to-End versioning support for web services,” in IEEE International Conference on Services Computing, 2008., vol. 1, Jul. 2008, pp. 59–66.
[63] Kajko-Mattsson M, Tepczynski M. A framework for the evolution and maintenance of web services[C]//null. IEEE, 2005: 665-668.
[64] R. Weinreich, T. Ziebermayr, and D. Draheim, “A versioning model for enterprise services,” in Advanced Information Networking and Applications Workshops, 2007, AINAW ’07. 21st International Conference on, vol. 2, 2007, pp. 570–575.
[65] K. Becker, A. Lopes, D. S. Milojicic, J. Pruyne, and S. Singhal, “Automatically determining compatibility of evolving services,” in ICWS 2008, 2008, pp. 161–168.
[66] Khater M, Malki M. An approach for adapting web services [C]//Multimedia Computing and Systems, 2009. ICMCS'09. International Conference on. IEEE, 2009: 56-61.
[67] Yamashita M, Vollino B, Becker K, et al. Measuring change impact based on usage profiles[C]//Web Services (ICWS), 2012 IEEE 19th International Conference on. IEEE, 2012: 226-233.

[68] Wang S, Capretz M A M. A dependency impact analysis model for web services evolution[C]//Web Services, 2009. ICWS 2009. IEEE International Conference on. IEEE, 2009: 359-365.
[69] Wang S, Capretz M A M. Dependency and entropy based impact analysis for service-oriented system evolution[C]//Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01. IEEE Computer Society, 2011: 412-417.
[70] Yau S S, Ye N, Sarjoughian H S, et al. Toward development of adaptive service-based software systems[J]. Services Computing, IEEE Transactions on, 2009, 2(3): 247-260.
[71] Na J, Gao Y, Zhang B, et al. Improved adaptation of Web service composition based on change impact probability[C]//Dependability (DEPEND), 2010 Third International Conference on. IEEE, 2010: 146-153.
[72] Feng Z, Chiu D K W, He K. A Service Evolution Registry with Alert-Based Management[C]//Service Science and Innovation (ICSSI), 2013 Fifth International Conference on. IEEE, 2013: 123-130.
[73] Feng Z, He K, Peng R, et al. Taxonomy for evolution of service-based system[C]//Services (SERVICES), 2011 IEEE World Congress on. IEEE, 2011: 331-338.
[74] Mateos C, Crasso M, Rodriguez J M, et al. Measuring the impact of the approach to migration in the quality of web service interfaces[J]. Enterprise Information Systems, 2015, 9(1): 58-85.
[75] Oliva G, Gerosa M, Milojicic D, et al. A change impact analysis approach for workflow repository management[C]//Web Services (ICWS), 2013 IEEE 20th International Conference on. IEEE, 2013: 308-315.
[76] Qi S, Li B, Liu C, et al. A Trust Impact Analysis Model for Composite Service Evolution[C]//Software Engineering Conference (APSEC), 2012 19th Asia-Pacific. IEEE, 2012, 1: 73-78.
[77] Wang M, Cui L Z. An impact analysis model for distributed web service proces[C]//Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on. IEEE, 2010: 351-355.
[78] Wang Y, Yang J, Zhao W. Service change analyzer: An enabling tool for change management in service-based business processes[C]//e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on. IEEE, 2011: 237-244.
[79] Motahari Nezhad H R, Benatallah B, Martens A, et al. Semi-automated adaptation of service interactions[C]//Proceedings of the 16th international conference on World Wide Web. ACM, 2007: 993-1002.
[80] Fang R, Chen Y, Fong L, et al. A version-aware approach for web service client application[C]//Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on. IEEE, 2007: 401-409.
[81] Frank D, Lam L, Fong L, et al. Using an interface proxy to host versioned web services[C]//Services Computing, 2008. SCC'08. IEEE International Conference on. IEEE, 2008, 2: 325-332.

[82] Fang R, Lam L, Fong L, et al. A version-aware approach for web service directory[C]//Web Services, 2007. ICWS 2007. IEEE International Conference on. IEEE, 2007: 406-413.
[83] Juric M B, Sasa A, Brumen B, et al. WSDL and UDDI extensions for version support in web services[J]. Journal of Systems and Software, 2009, 82(8): 1326-1343.
[84] Kongdenfha W, Saint-Paul R, Benatallah B, et al. An aspect-oriented framework for service adaptation[M]//Service-Oriented Computing–ICSOC 2006. Springer Berlin Heidelberg, 2006: 15-26.
[85] Aspect Oriented Programming. [Online] available at http://docs.jboss.org/aop/1.0/aspect-framework/userguide/en/html/what.html
[86] Comet Technology. [Onlline] available at: https://en.wikipedia.org/wiki/Comet_(programming)
[87] Sencha EXTJS [Online] available at : https://docs.sencha.com/extjs/6.0/backend_connectors/direct/specification.html
[88] Web Socket. [Online] available at http://www.websocket.org/.
[89] Na J, Gao Y, Zhang B, et al. Improved adaptation of Web service composition based on change impact probability[C]//Dependability (DEPEND), 2010 Third International Conference on. IEEE, 2010: 146-153.
[90] Kongdenfha W, Motahari-Nezhad H R, Benatallah B, et al. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters[J]. Services Computing, IEEE Transactions on, 2009, 2(2): 94-107.
[91] Feng Z, He K, Ma Y, et al. A Requirements-Driven and Aspect-Oriented Approach for Evolution of Web Services Composition[C]//Web Mining and Web-based Application, 2009. WMWA'09. Second Pacific-Asia Conference on. IEEE, 2009: 201-204.
[92] Kim M, Sumbaly R, Shah S. Root cause detection in a service-oriented architecture[J]. ACM SIGMETRICS Performance Evaluation Review, 2013, 41(1): 93-104.
[93] Buckley J, Mens T, Zenger M, et al. Towards a taxonomy of software change[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2005, 17(5): 309-332.
[94] Bennett K H, Rajlich V T. Software maintenance and evolution: a roadmap[C]//Proceedings of the Conference on the Future of Software Engineering. ACM, 2000: 73-87.
[95] Becker K, Lopes A, Milojicic D, et al. Automatically determining compatibility of evolving services[C]//Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008: 161-168.
[96] Aguilera M K, Mogul J C, Wiener J L, et al. Performance debugging for distributed systems of black boxes[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 74-89.
[97] Zhao X, Zhang Y, Lion D, et al. lprof: A non-intrusive request flow profiler for distributed systems[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 629-644.

[98] Mei H, Huang G, Xie T. Internetware: A software paradigm for internet computing[J]. Computer, 2012 (6): 26-31.
[99] Li J, Xiong Y, Liu X, et al. How does web service API evolution affect clients?[C]//Web Services (ICWS), 2013 IEEE 20th International Conference on. IEEE, 2013: 300-307.
[100] Zuo Wei, Youssef Amghar & Benharkat Aïcha-Nabila (2015). « The Impact Analysis Model for Web Service Evolution. ». IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 9 décembre 2015, Singapore (Singapour), pp 457-460. doi : 10.1109/WI-IAT.2015.199. HAL : hal-01278226
[101] W. Zuo, Y. Amghar, A. Benharkat. Programming Framework based on change-centric web service evolution model. In The 4th International Symposium on Web Services (WSS'2014), Sfax, Tunisia. 2014.
[102] W. Zuo, A. Benharkat, Y. Amghar. Holistic and Change-centric Model for Web Service Evolution. In 2014 SERVICES Workshops-IEEE Fourth International Workshop on the Future of Software Engineering for/in the Cloud (FoSEC- 2014), IEEE ed. Anchorage, Alaska. 2014
[103] W. Zuo, A. Benharkat, Y. Amghar. Change-centric Model for Web Service Evolution . In IEEE ICWS 2014,ALASKA. 2014