Building Reliable Web Services: Methodology, Composition, Modeling and Experiment

CHAN Pik Wah

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

Supervised by

Michael R. Lyu

March 2008

Abstract of thesis entitled:

Building Reliable Web Services: Methodology, Composition, Modeling and Experiment

Submitted by CHAN Pik Wah
for the degree of Doctor of Philosophy
at The Chinese University of Hong Kong in March 2008

One of the latest achievements of the Internet usage is the availability of Web services technology and its dependability is becoming one of the most critical goals in Web related research. In this thesis, we propose a design paradigm for reliable Web services and a Web service composition algorithm. We describe the methods of dependability enhancement by redundancy in space and redundancy in time, using Round-robin scheduling technique, N-version programming and recovery block. The Web services are coordinated by a replication manager. It provides a Round-robin algorithm for scheduling the workload of the Web services and keeps updating the availability of each Web service. The replication algorithm and the detailed system configuration are described.

In the paradigm, N-version programming technique is applied to increase the diversity of the system. As different versions of Web services or even different versions of their components are abundantly available in the Internet, the combination of different versions of the Web service or their components is thus becoming critical

for enabling different versions in a server application using the N-version approach. We propose a dynamic Web service composition algorithm and evaluate with Petri-Net for verification purposes.

Moreover, we model the Web services with Markov chains and Petri-Nets to demonstrate the performance and reliability of the constructed Web services. Also, we develop the mathematical models to analyze the reliability of the Web services.

Finally, we perform a series of experiments employing several replication schemes and compare them with a non-redundant single service. Through the experiments, we evaluate both the reliability of the Web service paradigm and the correctness of the Web service composition algorithm.

摘要

論文題目：修造可靠的網服務：方法，構成，塑造和實驗

作者：陳碧華
修讀學位： 哲學博士
香港中文大學計算機科學及工程學部
日期： 二零零七年十月

其中一個互聯網用法的最新的成就是網服務技術的可及性，並且它的可靠性是成爲的一個最重要的目標在網相關的研究。
在這份論文，我們提出一個設計範例爲可靠的網服務和網服務構成演算法。 使用聯名聲明預定的技術， N 版本編程和補救塊，我們由多餘在空間和多餘描述可靠性改進方法及時。 網服務由複製經理協調。 它爲預定網服務的工作量提供一種聯名聲明演算法並且繼續更新每項網服務的可及性。 複製演算法和詳細的系統佈局被描述。

在範例，申請 N 版本編程技術增加系統的變化。 因爲網服務的不同的版本甚至他們的組分的不同的版本是非常可利用的在互聯網，使用 N 版本方法，網服務或他們的組分的不同的版本的組合在伺服器應用因而變得 重要爲使能不同的版本。 我們提出一種動態網服務構成演算法並且評估與陪替氏網驗證目的。 並且，執行一系列的實驗使用幾份複製計畫的我們並且他們與一非冗余單路供電比較。 終於，我們塑造網服務與馬爾可夫鏈和陪替氏網展示被修建的網服務的表現和可靠性。

Acknowledgement

I would like to take this opportunity to express my gratitude to my supervisor Prof. Michael R. Lyu, for his generous guidance and patience given to me in the past five years. His numerous support and encouragement, as well as his inspiring advice are extremely essential and valuable in my research papers (conference papers published in ICDAT'2005, Aerospace'2005, ISAS'2006 and ICWS'2007) and my thesis.

I am also grateful for the time and valuable suggestion that Prof. Irwin King, Prof. Malek and Prof. Sun have given in marking my term paper. Without their effort, I will not be able to strengthen and improve my research project and papers.

I would also like to show my gratitude to the Department of Computer Science and Engineering, CUHK, for the provision of the best equipment and pleasant office environment required for high quality research.

Special thanks should be given to Mr. Edward Yau who has given me valuable suggestions, encouragement and supports. And I would like to give my thanks to my fellow colleagues, Alex Fok, Jill Law, Alan Chu, Brian Tuse, Food Lam, KK Lo, Steven Hoi and Wyman Wong. They have given me support, and a joyful and wonderful university life.

Finally, I am deeply indebted to my family for their unconditional love and support over the years.

This work is dedicated to my family for the support and patience

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A Web service is based on Service-oriented Architectures (SOA) [26]. This approach simplifies interoperability as the only standard communication protocols and simple broker-request architectures are needed to facilitate exchanges of services. Web services are becoming more popular and are beginning to pervade all aspects of life. However, due to our increasing dependency on these services, the problems of service dependability, security and timeliness are becoming critical.

Not surprisingly, the use of services, especially Web services, has become a common practice. The expectations are that services will dominate the software industry within the next five years.

One important element in the delivery of reliable Web services is that the software itself should be reliable. To achieve this, software needs to be fault-tolerant. Several fault tolerance approaches have been proposed for Web services in the literature [10, 42, 38, 51, 23, 69], but the field still requires theoretical foundations, appropriate models, effective design paradigms, practical implementations, and in-depth experimentation. We attack these issues in a unified

approach in our research, which is aimed at building reliable Web services with credible modeling and critical analysis.

1.1 Background

Web service is a major trend in the industry for loosely coupled service-oriented architecture and interoperable solutions across heterogeneous platforms and systems. It receives great attention and adoption by the industry and standard bodies [13, 19, 22, 25, 41, 52, 58]. It is well suited for integrating disparate systems, particularly those systems evolve over time. By enabling existing enterprise resources through Web services, these enterprises can be expanded to provide services to a wider variety of clients [40].

1.2 Research Objective

There are many fault-tolerant techniques that can be applied to Web services including replication and diversity. Replication is one of the efficient ways for creating reliable systems by time or space redundancy. Redundancy has long been used as a means of increasing the availability of distributed systems, with key components being re-executed (replication in time) or replicated (replication in space) to protect against hardware malfunctions or transient system faults. Another efficient technique is design diversity. By independently designing software systems or services with different programming teams, diversity provides an ultimate resort in defending against permanent software design faults.

In this thesis, we focus on the systematic analysis of the replication techniques when applied to Web services. We analyze the performance and the reliability of the Web services using spatial and temporal redundancy and study the tradeoffs between them. A generic Web service system with spatial as well as temporal replication is proposed and constructed for experiment.

Furthermore, nowadays there are abundant of Web services available in the Internet. To provide more efficient and suitable services for different clients, combining different Web services would be an efficient approach. In this thesis, we propose an dynamic Web services composition algorithm and evaluate it with a series of experiments.

1.3 Contribution

Our research work has the following contributions:

- Surveyed on reliability methodologies
- Surveyed on Web services reliability and Web service composition
- Proposed an architecture for dependable Web services
- Proposed an algorithm for Web services composition
- Developed reliability models for the proposed scheme
- Performed experiments for evaluating the reliability of the system and the correctness of the algorithm

Surveying on fault tolerance, Web services reliability and Web service composition

We perform a complete survey on the current fault tolerance technologies and the Web services reliability techniques. Currently, there are few experimental investigations to evaluate the reliability and availability of Web services systems. We propose a reliable Web service paradigm and evaluate it with experiments. We also perform a survey on the Web service composition which enable us to enhance our system's reliability.

Proposing an architecture for dependable Web services

In this thesis, we first identify the parameters which impact the Web services dependability. Then, we describe the methods of dependability enhancement by redundancy in space and redundancy in time. Furthermore, we perform a series of experiments to evaluate the reliability of Web services. To increase the reliability of the Web service, we use several replication schemes and compare them with a single service. The Web services are coordinated by a replication manager.

Applying Round-robin, N-version programming and recovery block approach to the paradigm

To increase the reliability of the system, we propose three approaches, Round-robin [64], N-version programming [45] and recovery block [64], to be intergraded with our system. Each of them has different characteristics which improve the reliability of the system in different ways.

Proposing an algorithm for Web services composition

We propose an algorithm for composing Web services. Together with an N-version programming Web service, it improves the reliability of the overall system. In the algorithm, WSCI [4] and BPEL [3] are employed to enable the Web services composition. The composition algorithm is verified to be correct and deadlock-free through the Petri-Nets.

Modeling on proposed scheme

We develop reliability models for our proposed Web service paradigm by using Markov chains model [24] and Petri-Nets [53]. Through the models, the reliability, performance and throughput of the proposed paradigm are evaluated and the characteristics of the Web services are shown. Also, we develop mathematical models for each approached of the proposed paradigm. Thus, we can compare and evaluate the reliability of different approaches.

Experiments

We perform a series of experiments, which are designed for evaluating the reliability of the Web services. We apply retry, reboot and spatial replication with Round-robin, N-version programming Web services or recovery block to our system. We perform the experiments with different combinations. Also, according to the Web service composition algorithm proposed, different versions of the experiment system are composed and the program metrics are measured. Furthermore, we perform experiments to evaluate the correctness and performance of the composition algorithm.

1.4 Structure of Thesis

The thesis is organized as follows. The next chapter introduces the issues related to Web services, fault-tolerance techniques, and describes a survey on the current reliable Web service technologies and Web service composition. We then propose the methodologies for reliable Web services, present our approaches with a list of key parameters, describe the architecture and configuration of the system, and propose a roadmap for further development and experimentation in Chapter 3. In Chapter 4, we present our dynamic Web service composition algorithm and analysis with examples. Reliability models of the proposed system are developed and examined in Chapter 5. Then we document execution of Web service experiments and present the results in Chapter 6. Finally, in Chapter 7 we draw some conclusions and sketch the future work.

□ **End of chapter.**

Chapter 2

Literature Review

2.1 Introduction

Web service is a self-contained, modular application built on deployed network infrastructure including XML and HTTP. It uses open standards for description (Web Service Definition Language, WSDL), discovery (Universal Description, Discovery, and Integration, UDDI) and invocation (Simple Object Access Protocol, SOAP). Web service becomes more popular and fault tolerance becomes essential property for a Web service. There are different proposed approaches for improving the reliability of the Web services, including N-version programming, replication, reliable messaging, message ordering and duplicate elimination.

Let us first have a brief introduction of Web service, and an overview of state-of-the-arts technologies in reliability and Web service composition techniques. Then, a literature review of current reliable Web service systems and Web service composition is presented.

2.2 Web Services

The W3C defines a Web service as a software system designed to support interoperable machine to machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server, a description in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client-side code generation in the mainstream Java and .NET SOAP frameworks. Some industry organizations, such as the WS-I (Web services Interoperability Organization), mandate both SOAP and WSDL in their definition of a Web service.

Web services are self-contained business functions that operate over the Internet. They are written to strict open specifications to work together and with other similar kinds of components.

Web services are useful to business as they enable systems in different companies to interact with each other, more importantly, in a far easier way than before. With business needing closer operations between suppliers and customers, engaging in more joint ventures, and facing the prospect of more mergers and acquisitions, companies need the capability to link up their established systems

quickly and efficiently with other companies. Thus Web services give companies the capability to do more e-business, with more potential business partners, in more and different ways than before, and at reasonable cost.

The recent growth in use of the World Wide Web on the Internet has caused a significant increase in the demand on Web services. Web services have gained high popularity in the development of distributed application systems. Some critical applications also consider using Web services paradigm due to the benefit of interoperability, reusability, and adaptability. To support critical applications, existing Web service models need to be extended to assure survivability.

2.2.1 Technologies in Web Services

A Web service is programmable application logic accessible using standard Internet protocols. Web services combine the best aspects of component-based development and the Web. Like components, Web services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web services are not accessed via object-model-specific protocols, such as DCOM, RMI, or IIOP. Instead, Web services are accessed via ubiquitous Web protocols (such as HTTP) and data formats (such as XML).

A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service performs a specific task or a set of tasks. A Web service is described using a standard, formal XML notation,

called its service description, that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. Web service descriptions are expressed in WSDL.

The specifications that define Web services are intentionally modular, and as a result there is no one document that contains them all. Additionally, there is neither a single, nor a stable set of specifications. There are a few ”core” specifications that are supplemented by others as the circumstances and choice of technology dictate, including:

SOAP An XML-based, extensible message envelope format, with ”bindings” to underlying protocols. The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written.

WSDL An XML format that allows service interfaces to be described, along with the details of their bindings to specific protocols. Typically used to generate server and client code, and for configuration.

UDDI A protocol for publishing and discovering metadata about Web services, to enable applications to find Web services, either at design time or runtime.

2.2.2 Work-flow of Web Services

A service provider creates a Web service and its service definition and then publishes the service with a service registry based on a standard called the Universal Description, Discovery, and Integration (UDDI) specification.

Once a Web service is published, a service requester may find the service via the UDDI interface. The UDDI registry provides the service requester with a WSDL service description and a URL (uniform resource locator) pointing to the service itself. The service requester may then use this information to directly bind to the service and invoke it.

The architecture of Web Service is shown in Figure 2.1

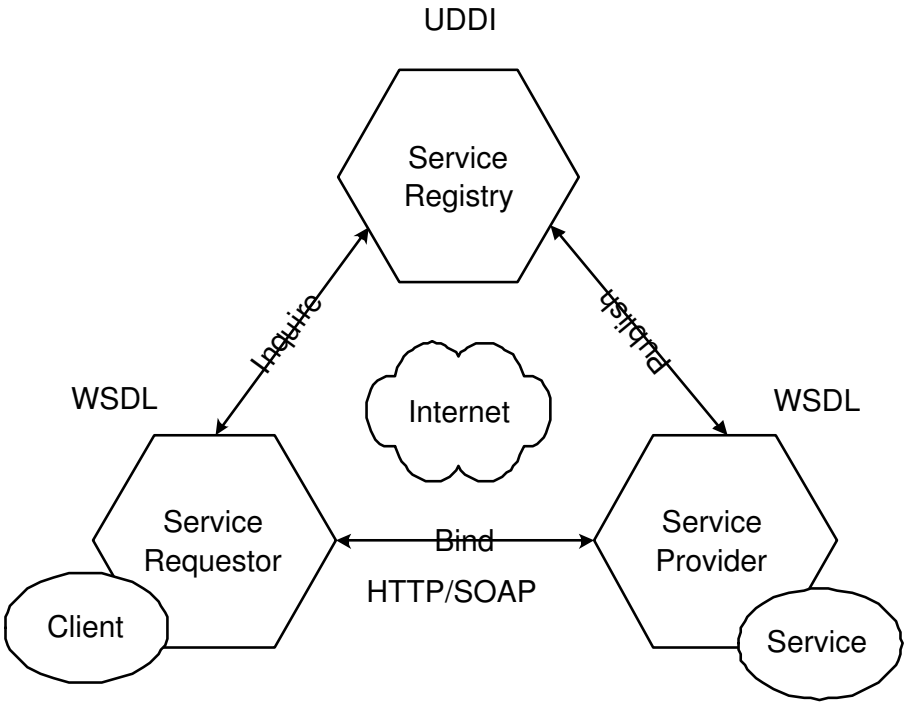

Figure 2.1: Architecture of Web service

Properties of Web services:

- Perform encapsulated business functions using request/reply as well as business process interactions
- Looser coupling via less reliance on pre-defined interfaces
- Can be mixed and matched to create complete process
- Enable dynamic integration through embedded capability of service discovery and binding

2.2.3 Problems of Current Web Services

Transaction

Atomicity is not provided. The key point is that HTTP is stateless, however, business processes or transactions are useful.

Security

Add-on measures such as Encryption are needed to deal with the insecure Internet transportation.

Interoperability

IBM, Microsoft, intel, BEA and other companies formed the Web services Interoperability Organization (WS-I), a non-profit organization for promoting Web services standard. The idea behind WS-I was not create new standards, but rather to assemble ”profiles” of standards from the W3C, OASIS and others. The profiles are sets of related standard against which conformance tests and certifications can be established.

Reliability

The Internet is inherently unreliable. Currently there is no singel underlying transport protocols (HTTP, FTP, SMTP) addresses all reliability issues, namely guaranteed delivery, ordered delivery, and duplicate elimination. For collaborative e-business and e-transaction scenarios, message reliability becomes a critical issue.

Composition

It is often assumed that a business process or application is associated with some explicit business goal definition that can guide a planning-based composition tool to select the right service [50]. Unfortunately, we found that explicit goals are usually not available from an industrial perspective. A business process model describes the processing of persistent data objects in discrete process steps. The real goal of a business often remains implicit in these models and is rather expressed at a higher level using often using balanced score cards, while the implicit goal of a business process is the correct handling or the creation of data objects manifested in persistent documents [66].

In this thesis, we focus on solving the reliability and composition problems. We propose reliable paradigm and composition algorithm for these two issues respectively.

2.2.4 Failure Response Stages of Web Services

Web services will go through different stages when failure occurred and the failure response of Web services can be classified into different stages [46]. When failure occurred, the Web service is confined.

Fault detection techniques are applied to find out the failure causes and the failed components are repaired or recovered. Then, reconfiguration, restart and reintegration will be performed. The flow of the failure response of Web service is shown in Figure 2.2 and the details of each stage are described as follows:

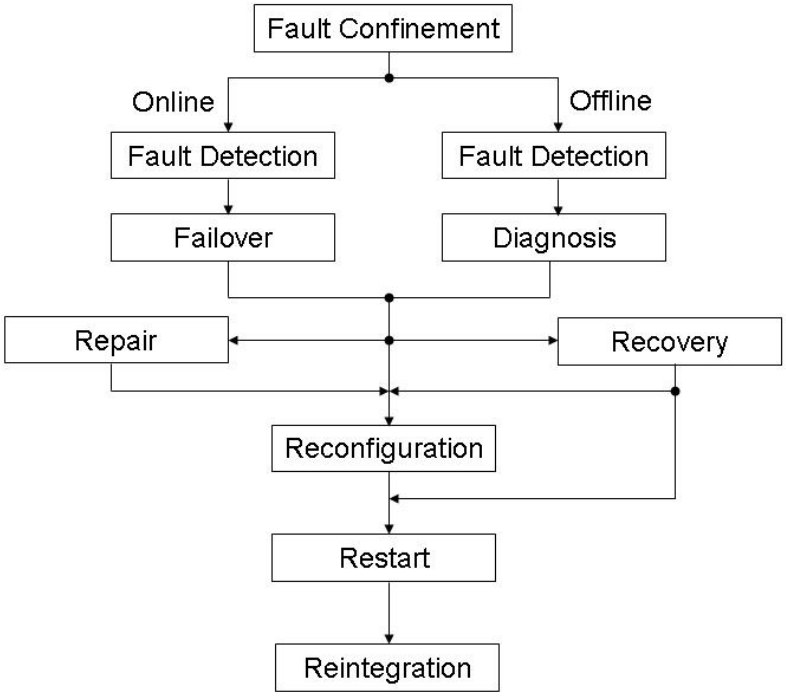

Figure 2.2: Flow of the failure response of Web services.

Fault confinement

This stage limits the spread of fault effects to one area of the Web service, thus preventing contamination of other areas. Fault confinement can be achieved through use of: fault detection within the Web

services, consistency checks and multiple requests or confirmations.

Fault detection

This stage recognizes that something unexpected has occurred in a Web service. Fault latency is the period of time between the occurrence of a fault and its detection. Techniques fall in two classes: off-line and on-line. With off-line techniques, such as diagnostic programs, the service is not able to perform useful work while under test. On-line techniques, such as duplication, provide a real-time detection capability that is performed concurrently with useful work.

Diagnosis

This stage is necessary if the fault detection technique does not provide information about the fault location.

Reconfiguration

This stage occurs when a fault is detected and located. The Web services can be composed of different components. When providing the service, there may be a fault in individual components. The system may reconfigure its components either to replace the failed component or to isolate it from the rest of the system.

Recovery

This stage utilizes techniques to eliminate the effects of faults. Two basic recovery approaches are based on: fault masking, retry and rollback. Fault-masking techniques hide the effects of failures by

allowing redundant information to outweigh the incorrect information. Web services can be replicated or implemented with different versions (NVP). Retry undertakes a second attempt at an operation and is based on the premise that many faults are transient in nature. Web services provide services through network; retry would be a practical approach as requests/reply may be affected by the state of the network. Rollback makes use of the fact that the Web service operation is backed up (checkpointed) at some points in its processing prior to fault detection and operation recommences from that point. Fault latency is important here because the rollback must go back far enough to avoid the effects of undetected errors that occurred before the detected error.

Restart

This stage occurs after the recovery of undamaged information.

- Hot restart: resumption of all operations from the point of fault detection and is possible only if no damage has occurred.
- Warm restart: only some of the processes can be resumed without loss.
- Cold restart: complete reload of the system with no processes surviving. The Web services can be restarted by rebooting the server.

Repair

At this stage, a failed component is replaced. Repair can be off-line or on-line. Web services can be component-based and consist of

other Web services. In off-line repair, either the Web service will continue if the failed component/sub-Web service is not necessary for operation or the Web services must be brought down to perform the repair. In on-line repair, the component/sub-Web service may be replaced immediately with a backup spare or operation may continue without the component. With on-line repair Web service operation is not interrupted.

Reintegration

At this stage the repaired module must be reintegrated into the Web service. For on-line repair, reintegration must be performed without interrupting Web service operation.

2.3 Review on the Methodologies for Reliable Web Services

2.3.1 Introduction

Reliability is a measure of the success with which the system conforms to some authoritative specification. Reliability engineering provides the theoretical and practical tools whereby the probability and capability of parts, components, equipments, products and systems to perform their required functions for desired periods of time without failure, in specified environments and with a desired confidence, can be specified, designed in, predicted, tested and demonstrated [36].

Reliability includes:

- Guaranteed delivery: ensure that all information to be sent actually received by the destination or error reported.
- Duplicate elimination: ensure that all duplicated information can be detected and filtered out.
- Ordering: communication between parties consist of several individual message exchanges. This aspect ensure that Message Exchanges are forwarded to the receiver application in the same order as the sender application issued.
- Crash tolerance: ensures that all information prescribed by the protocol is always available regardless of possible physical machine failure.
- State synchronization: if the minimum error point is cancelled for any reason, then it is desirable for both nodes to set their state as if there were no communication between the parties.

There are numbers of methods can be applied in the reliability issues, including:

- Redundancy
- Diversity

The following sections will discuss these techniques.

2.3.2 Fault Tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components. If its operating quality decreases at all, the decrease is pro-

portional to the severity of the failure, as compared to a naively-designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought-after in high-availability or life-critical systems.

In this section, we first introduce the concepts related to this technique [12, 34].

Failures A failure occurs when the user perceives that a software program is unable to deliver the expected service [32]. The expected service is described by a system specification or a set of user requirements.

Errors An error is part of the system state which is liable to lead to a failure. It is an intermediate stage in between faults and failures. An error may propagate, i.e., produce other errors.

Faults A fault, sometimes called a bug, is the identified or hypothesized cause of a software failure. Software faults can be classified as design faults and operational faults according to the phases of creation. Although the same classification can be used in hardware faults, we only interpret them in the sense of software here.

Design faults A design fault is a fault occurring in software design and development process. Design faults can be recovered with fault removal approaches by revising the design documentation and the source code.

Operational faults An operational fault is a fault occurring in software operation due to timing, race conditions, workload-related stress and other environmental conditions. Such a fault can be removed by recovery, i.e., rollback to a previously saved state and executed again.

Fault avoidance (prevention) To avoid or prevent the introduction of faults by engaging various design methodologies, techniques and technologies, including structured programming, object-oriented programming, software reuse, design patterns and formal methods.

Fault removal To detect and eliminate software faults by techniques such as reviews, inspection, testing, verification and validation.

Fault tolerance To provide a service complying with the specification in spite of faults, typically by means of single version software techniques or multi-version software techniques. Note that, although fault tolerance is a design technique, it handles manifested software faults during software operations. Although software fault tolerance techniques are proposed to tolerant software errors, they can help to tolerate hardware faults as well.

Fault/failure prediction (forecasting) To estimate the existence of faults and the occurrences and consequences of failures by dependability-enhancing techniques consisting of reliability estimation and reliability prediction.

2.3.3 Redundancy

It is a well-known fact that fault tolerance can be achieved via spatial or temporal redundancy, including replication of hardware (with additional components), software (with special programs), and time (with the repetition of operations) [27, 36, 63, 65, 59, 70].

Redundancy can be achieved by replicating hardware modules to provide backup capacity when a failure occurs, or redundancy can be obtained using software solutions to replicate key elements of a business process.

Spatial redundancy can be dynamic or static. Both of them use replication but in static redundancy, all replicas are active at the same time and voting takes place to obtain a correct result. The number of replicas is usually odd and the approach is known as n-modular redundancy. For example, under a single fault assumption, if services are triplicated and one of them fails, the remaining two will still guarantee the correct result. The associated spatial redundancy cost is high (three copies plus a voter). The time overhead of managing redundant modules such as voting and synchronization is also considerably large for static redundancy. Dynamic redundancy, on the other hand, engages one active replica at one time while others are kept in an active or in standby state. If one replica fails, another replica can be employed immediately with little impact on response time. In the second case, if the active replica fails, a previously inactive replica must be initialized and take over the operations. Although this approach may be more flexible and less expensive than static redundancy, its cost may still be high due to the possibility of hastily eliminating modules with transient faults. It

may also increase the recovery time because of its dependence on time-consuming error-handling stages such as fault diagnosis, system reconfiguration, and resumption of execution.

2.3.4 Diversity

In any redundant systems, common-mode failures (CMFs) result from failures that affect more than one module at the same time, generally due to a common cause. These include design mistakes and operational failures that may be caused externally or internally. Design diversity has been proposed in the past to protect redundant systems against common-mode failures [6, 7, 48] and has been used in both hardware and software systems [30, 60]. The basic idea is that, with different designs and implementations, common failure modes will probably cause different error effects. One of the design diversity techniques is N-version programming [6], and another one is Recovery Blocks [55]. The key element of N-version programming or Recovery Block approaches is diversity. By attempting to make the development processes diverse, it is hoped that the independently designed versions will also contain diverse faults that are non-identical or even may be similar. It is assumed that such diverse faults will minimize the likelihood of coincident failures.

These multiple versions are executed either in sequence or in parallel, and can be used as alternatives (with separate means of error detection), in pairs (to implement detection by replication checks) or in larger groups (to enable masking through voting). Three fundamental techniques are known as recovery block, N-version programming and N self-checking programming.

Recovery Block

The recovery block (RB) technique involves multiple software versions implemented differently such that an alternative version is engaged after an error is detected in the primary version [56, 57]. The question of whether there is an error in the software result is determined by an acceptance test (AT). Thus the recovery block uses an acceptance test and backward recovery to achieve fault tolerance. As the primary version will be executed successfully most of the time, the most efficient version is often chosen as the primary alternate and the less efficient versions are placed as secondary alternates. Consequently, the resulting rank of the versions reflects, in a way, their diminishing performance.

The usual syntax of the recovery block is as follows. First of all, the primary alternate is executed; if the output of the primary alternate fails the acceptance test, a backward error recovery is invoked to restore the previous state of the system, then the second alternate will be activated to produce the output; similarly, every time an alternate fails the acceptance test, the previous system state will be restored and a new alternate will be activated. Therefore, the system will report failure only when all the alternates fail the acceptance test, which may happen with a much lower probability than in the single version situation. The recovery block model is shown in Figure 2.3, while the operation of the recovery block is shown in Figure 2.4.

The execution of the multiple versions is usually sequential. If all the alternate versions fail in the acceptance test, the module must raise an exception to inform the rest of the system about its failure.

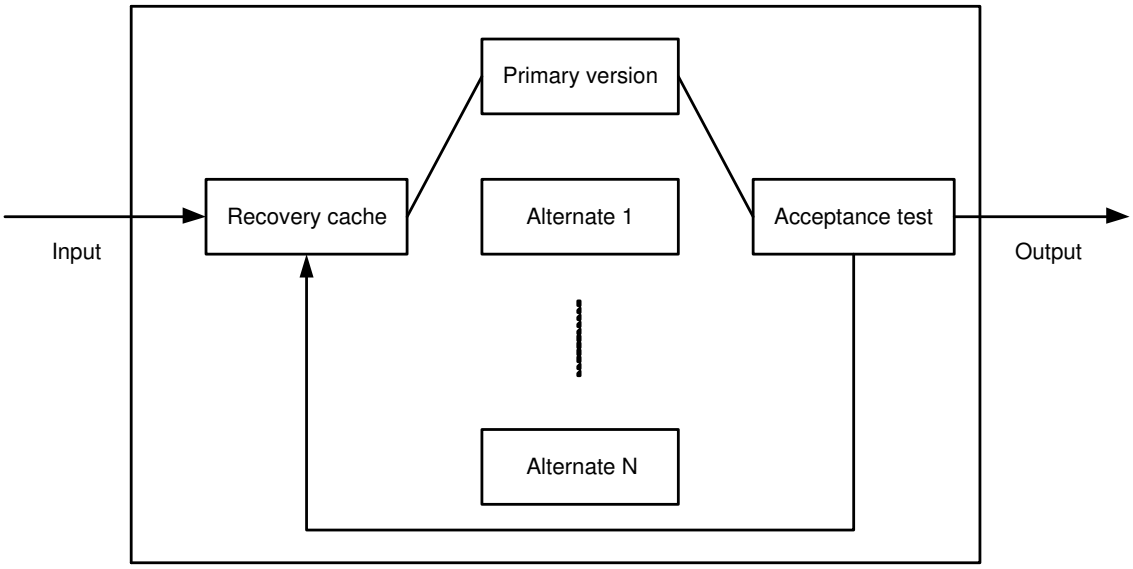

Figure 2.3: The recovery block model.

N-Version Programming

The concept of N-version programming (NVP) was first introduced in 1977 [6]. It is a multi-version technique in which all the versions are typically executed in parallel and the consensus output is based on the comparison of the outputs of all the versions [48]. In the event that the program versions are executed sequentially due to lack of resources, it may require the use of checkpoints to reload the state before a subsequent version is executed. The N-version software model is shown in Figure 2.5.

The NVP technique uses a decision algorithm (DA) and forward recovery to achieve fault tolerance. The use of a generic decision algorithm (usually a voter) is the fundamental difference of NVP from the RB approach, which requires an application-dependent acceptance test. The complexity of the decision algorithm is generally lower than that of the acceptance test. In NVP, since all the versions

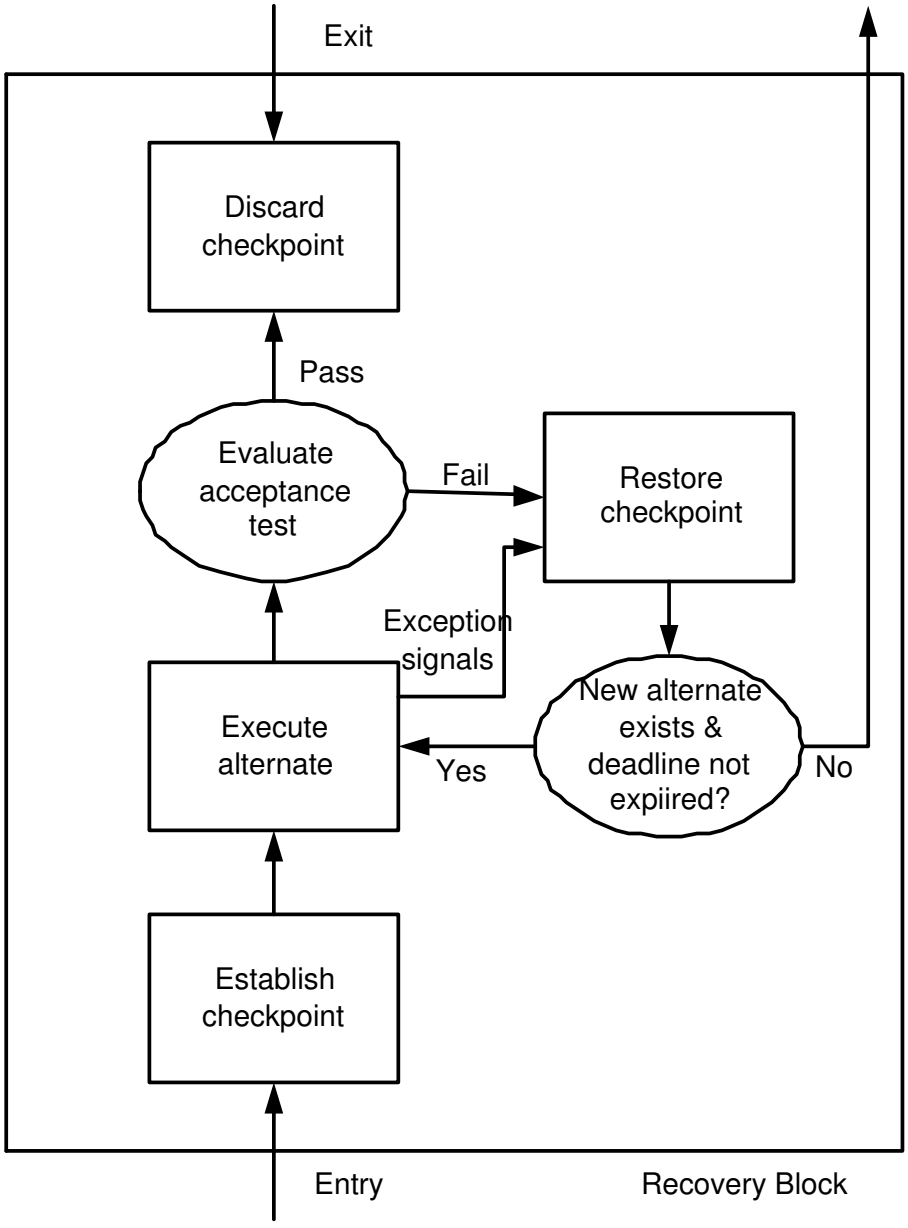

Figure 2.4: Operation of the recovery block.

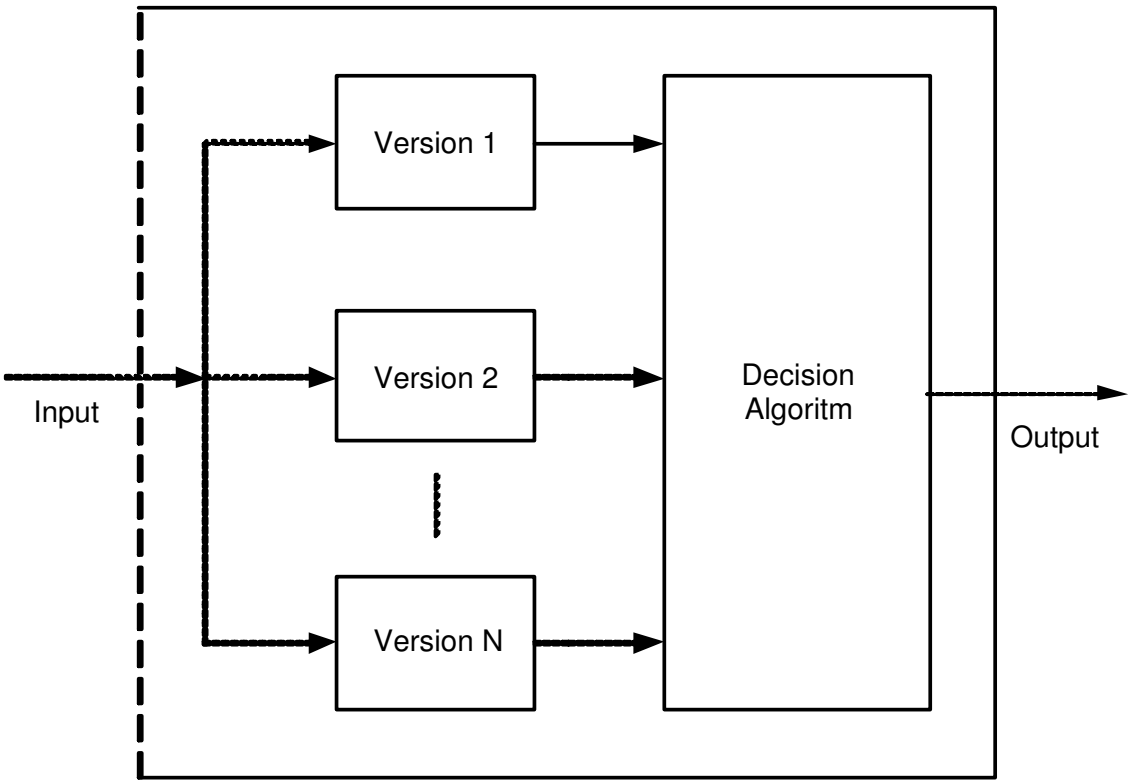

Figure 2.5: The N-version programming model.

are built to satisfy the same specification, it requires considerable development effort but the complexity (i.e., development difficulty) is not necessarily much greater than that of building a single version. Much research has been devoted to the development of methodologies that increase the likelihood of achieving effective diversity in the final product [5, 11, 21, 28].

N-Self Checking Programming

N self-checking programming (NSCP) was developed in 1987 by Laprie et al. [32, 33]. It involves the use of multiple software versions combined with structural variations of the recovery block and N-version programming approaches. Both acceptance tests and decision algorithms can be employed in NSCP to validate the outputs of multiple versions.

The N self-checking programming method employing acceptance tests is shown in Figure 2.6. Same as RB and NVP, the versions and the acceptance tests are developed independently but each designed to fulfill the requirements. The main difference of NSCP from the RB approach is in its use of different acceptance tests for different versions. The execution of the versions and tests can be done sequentially or in parallel but the output is taken from the highest-ranking version that passes its acceptance test. Sequential execution requires a set of checkpoints, and parallel execution requires input and state consistency algorithms.

Each design diversity technique, recovery block, N-version programming, and N self-checking programming, has its own advantages and disadvantages compared with the others.

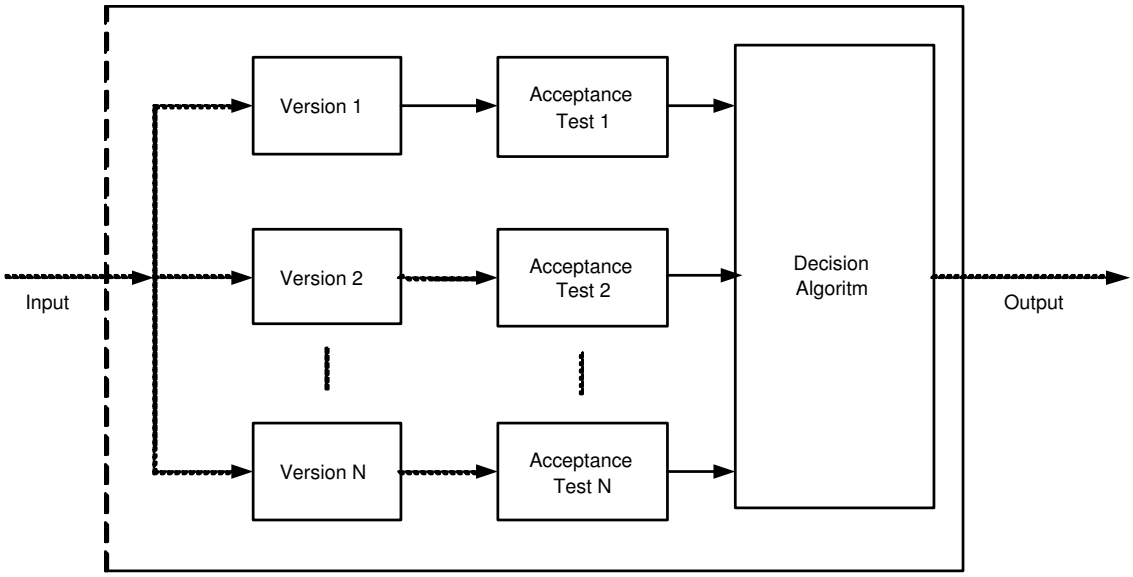

Figure 2.6: N self-checking programming using acceptance test.

2.4 Web Service Composition

Composition of Web services has received much interest to support business-to-business or enterprise application integration [73, 35, 66, 29]. Currently, most of the work is in the description of Web services, the syntax of their flows, and how they could be executed. In the future, it is necessary to view Web services in the context of specifying, validating, and automatically synthesizing complex, and reactive processes. In this section, we review the state-of-the-art composition techniques.

2.4.1 WSCI

The Web Service Choreography Interface (WSCI) [4] is an XML-based interface description language that describes the flow of messages exchanged by a Web service participating in choreographed

interactions with other services.

WSCI describes the dynamic interface of the Web service participating in a given message exchange by means of reusing the operations defined for a static interface. WSCI works in conjunction with the Web Service Description Language (WSDL), the basis for the W3C Web Services Description Working Group; it can, also, work with another service definition language that exhibits the same characteristics as WSDL.

WSCI describes the observable behavior of a Web Service. This is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. WSCI also describes the collective message exchange among interacting Web Services, thus providing a global, message-oriented view of the interactions.

WSCI does not address the definition and the implementation of the internal processes that actually drive the message exchange. Rather, the goal of WSCI is to describe the observable behavior of a Web Service by means of a message-flow oriented interface. This description enables developers, architects and tools to describe and compose a global view of the dynamic of the message exchange by understanding the interactions with the web service.

2.4.2 BPEL

Business Process Execution Language (BPEL) [3] for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment even across multiple organizations, using a combination of Web services. Writ-

ten by developers from BEA Systems, IBM, and Microsoft, BPEL combines and replaces IBM's Web services Flow Language (WSFL) [37] and Microsoft's XLANG specification. [67]. It is serialized in XML and aims to enable programming in the large. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.

There were ten original design goals associated with BPEL [71]:

1. Define business processes that interact with external entities through Web Service operations defined using WSDL, and that manifest themselves as Web services defined using WSDL. The interactions are abstract in the sense that the dependence is on portType definitions, not on port definitions.

2. Define business processes using an XML-based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.

3. Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

4. Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.

5. Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.

6. Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.

7. Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as ”suspend” and ”resume” may be added in future releases for enhanced lifecycle management.

8. Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.

9. Use Web Services as the model for process decomposition and assembly.

10. Build on Web services standards (approved and proposed) as much as possible in a composable and modular manner.

2.4.3 Other Standards

Except from the BPEL and WSCI, there are numbers of other existing standards that is related to Web service composition. In this section, we are going to briefly introduce some of them.

BPML

BPML [1] provides an abstract model and grammar for expressing abstract and executable business processes. Using BPML, enterprise processes, complex web services and multi-party collaborations can be defined. A process in BPML is a composition of activities that perform specific functions. The process directs the execution of these activities. It can also be a part of composition by defining it as a part of its parent process or by invoking from another process. Each activity (both simple and complex) in the process has a context, which defines common behavior for all activities executing in that context. Hence a process can be defined as a type of complex activity that defines its own context for execution. The BPML specification defines 17 activity types, and three process types. The different process types are nested processes which are defined to execute within a specific context and whose definitions are a part of context's definition, exception processes to handle exceptional conditions in executing parent's process and compensation processes to provide compensation logic for their parent processes. Each process definition may specify any of the three ways of instantiating a process: in response to an input message, in response to a raised signal, or invoked from an activity or schedule. BPML specifications support importing and referencing service definitions given in WSDL. It also suggests standardizing BPML documents by using RDF for semantic meta-data, XHTML and Dublin Core metadata to improve human readability and application processability.

BPSS

ebXML [2] is a global electronic business standard envisioned to define a XML based framework that will allow businesses to find each other and conduct business using well-defined messages and standard business processes. ebXML Business Process Specification Schema is a standard for representing models for collaborating e-business public processes. Using XML syntax the parties involved in a collaboration can model and agree on the relevant business process. ebXML BPSS standard can be used to configure the business systems to support the commercial collaboration. Hence this specification determines the actual exchange (identified as patterns) of business documents and business signals between the partners. A library of process templates can be created using BPSS definitions and to support a business process template a user can extract information from the corresponding BPSS and configure his runtime system by agreeing on a pattern and a role that collaborates through a set of choreographed transactions by exchanging Business Documents. However there is no explicit support for describing how data flows between transactions, but there is explicit support for specifying quality-of-service semantics for transactions such as authentication, acknowledgements, non-repudiation, and timeouts.

DAML-S

DAML-S is an initiative to provide an ontology markup language expressive enough to semantically represent capabilities and properties of Web services. DAML-S is based on DAML+OIL and the aim is to discover, invoke, compose, and monitor Web services. It defines an

upper ontology appropriate for declaring and describing services by using a set of basic classes and properties. In DAML-S, each service can be viewed as a process and its Process Model is used to control the interactions with the service. Using the processOntology?s sub-ontologies, ProcessOntology and ProcessControlOntology, it aims to capture the details of the Web service operation. The ProcessOntology describes the inputs, outputs, preconditions, effects, and component subprocesses of the service. ProcessControlOntology is used to monitor the execution of a service. However, current version of DAML-S does not define the ProcessControlOntology. DAML-S also categorizes three types of processes. The first type is atomic processes which do not have any subprocesses and can be executed in a single step. The second type is simple processes which are not invocable as they are used as abstraction for representing atomic or composite processes to use them. Composite processes are of third type which are decomposable into sub-processes. The composite process uses lot of control constructs to specify how inputs are accepted and outputs are returned by subprocesses.

WSCL

Web Services Conversation Language (WSCL) allows defining the external visible behavior of the services by specifying the business level conversations and public processes supported by a Web services. The conversations are defined using XML syntax and the WSCL document also specifies XML documents that are exchanged as a part of conversation and the order in which they are exchanged. WSCL provides a minimal set of concepts necessary for specifying

the conversations. The specification states that typically the conversation is provided from the perspective of the service provider, which can also be used to determine the conversation from the perspective of the user. Though the conversation is defined from the service provider's perspective it separates the conversational logic from the application logic or the implementation aspects of the service.

2.5 Related Work

2.5.1 Reliable Web Services

For reliable Web service, two popular fault tolerance techniques can be applied: redundancy and diversity. Based on the these fault-tolerance techniques, a number of reliable Web services techniques have appeared in the recent literature.

WS-FTM (Web Service-Fault Tolerance Mechanism) is an implementation of the classic N-version model for Web services [42], which can easily be applied to existing systems with minimal change. The Web services are implemented in different versions, and the voting mechanism is conducted in the client program.

FT-SOAP [38], on the other hand, is aimed at improving the reliability of the Simple Object Access Protocol (SOAP) when using Web services. The system includes different approaches to function replication management, fault management, logging/recovery mechanism and client fault tolerance transparency. FT-SOAP is based on the work of FT-CORBA [39], in which a fault-tolerant SOAP-based middleware platform is proposed.

FT-Grid [68] is another design, which is a deployment of design diversity for fault tolerance in Grid. It is not originally specified for Web services, but the techniques are applicable to Web services. FT-Grid allows a user to manually search through any number of public or private Universal Description, Discovery and Integration (UDDI) repositories, to select a number of functionally-equivalent services, to choose the parameters for each service, and to invoke those services. The application can then perform voting on the results returned by the services, with the aim of filtering out any anomalous results.

2.5.2 Web Service Composition

There are a number of techniques to enable the composition of Web services. The Web Service Choreography Interface (WSCI) [4] is an XML-based interface description language that describes the flow of messages exchanged by a Web service participating in choreographed interactions with other services. Business Process Execution Language (BPEL) [3] for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment even across multiple organizations, using a combination of Web services.

Moreover, a number of Web service composition schemes are proposed. SWORD [54] is one of the proposed solution. It is a set of tools for the composition of a class of Web services including “information-providing” services. In SWORD, a service is represented by a rule to express that with given certain inputs, the service is capable of producing particular outputs. A rule-based expert sys-

tem is then employed to automatically determine whether a desired composite service can be realized with the existing services.

When large-scale Web services are available, Chen et al. propose a structure to handle the composition [18]. Then the mutual search operations among Web service operations, inputs and outputs are studied, and a novel data structure called Double Parameter Inverted File (DouParaInvertedFile) is proposed to implement these operations. An algorithm to build DouParaInvertedFile is provided as well.

Apart from the self-contained algorithm, some algorithms are based on the current existing standards. In [20], a BPEL-based Web service composition using high-level Petri-Nets (HPN) approach is proposed. By analyzing the structure of Web service composition based on BPEL, the corresponding HPN is constructed. The dynamism and occurrence are presented in HPN with guard expression with colored token. After translation, the equivalent HPN of the Web service composition based on BPEL can be verified on existing mature tools.

Although a number of approaches have been proposed to aggregate the Web service, there is a need for a dynamic approach to compose the increasing number of Web services to provide new services. In this thesis, we aim at proposing an innovative dynamic Web service composition algorithm.

2.6 Summary

Although a number of approaches have been proposed to increase Web service reliability, there is a need for systematic modeling and

experiments to understand the tradeoffs and to verify the reliability of the proposed methods. We proposed a framework [16] for the deployment of reliable Web services, and enhance the scheme with a Round-robin algorithm, N-version programming and recovery block for Web services [17]. In this thesis, we focus on the systematic analysis of the replication techniques when applied to Web services. A generic Web service system with spatial as well as temporal replication is proposed, and its prototype is implemented as an experimental testbed. Also, there is a lack of dynamic composing approach. As the number of Web services is rapidly increasing, it is necessary to have a dynamic composition algorithm to integrate the new coming Web services with the existing Web services. To make more versions of Web services to be available for the proposed paradigm, a dynamic Web service composition is developed and correctness is verified through different experiments.

□ **End of chapter.**

Chapter 3

Methodologies for Reliable Web Services

3.1 Introduction

In this section, we propose a design paradigm for reliable Web services. Its architecture is shown in Figure 3.1. In our system, a dynamic spatial redundancy approach is adopted.

3.2 Scheme Details

In the proposed system, the Web services are replicated in different servers. We apply three different approaches for managing spatial replication, including a Round-robin (RR) algorithm, N-version programming and recovery block. The replicas are coordinated by the replication manager. More details will be discussed in the following sections. Also, we perform different experiments to evaluate the reliability of the system.

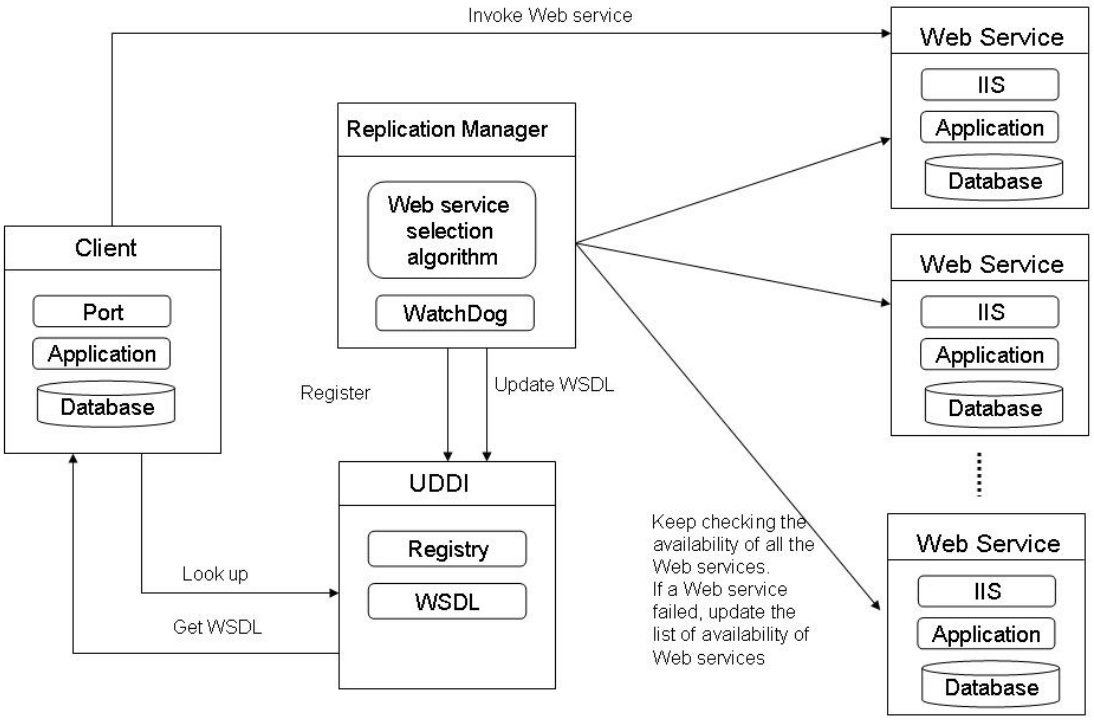

Figure 3.1: Proposed architecture for dependable Web services.

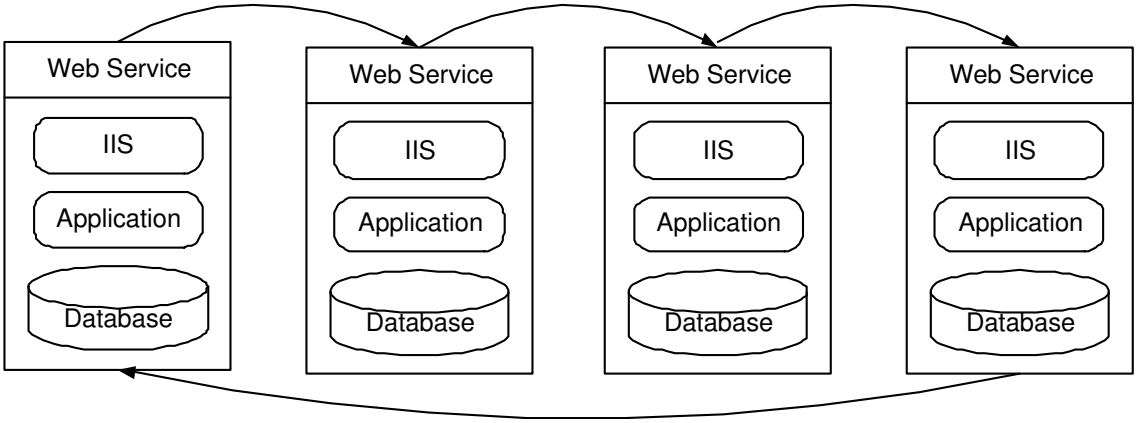

Figure 3.2: Round-robin approach.

3.2.1 Round-robin Approach

In the first approach, the Web servers work concurrently and a Round-robin algorithm [64] is employed for scheduling the work among the Web services. The idea is shown in Figure 3.2. The Web service is replicated on different machines. When there is a Web service failure, other Web servers can immediately provide the required service. This replication mechanism shortens the recovery time and increases the reliability of the system.

The main component of this system is the replication manager (RM), which acts as a coordinator of the Web services. The replication manager is responsible for:

1. Creating a Web service.
2. Choosing (with an anycasting algorithm) the best (fastest, most robust, etc.) Web service [62] to provide the service which is called the primary Web service.
3. Keeping the availability list of the Web services.

4. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).

5. Continuously checking the availability of the Web services by using a watchdog.

6. Applying the Round-robin algorithm for the scheduling the workload of the Web service.

The replication manager schedules the work of the Web service using the Round-robin algorithm; therefore, the resources of the system can be fully utilized. The replication manager distributes the work to different Web servers according to the availability of the servers. The requests are sent to different Web services accordingly. Whenever the server changes, the replication manager maps the new address of the Web service providing the service to the WSDL; thus, the clients can still access the Web service with the same URL. This failover process is transparent to the users.

The workflow of the replication manager is shown in Figure 3.3. The replication manager is running on a server, which keeps checking the availability of the Web services by a polling method: namely it sends messages to the Web services periodically. If it does not get a reply from the primary Web service, it will select another Web service to replace the primary one and map the new address to the WSDL. The system is considered failed if all the Web services have failed.

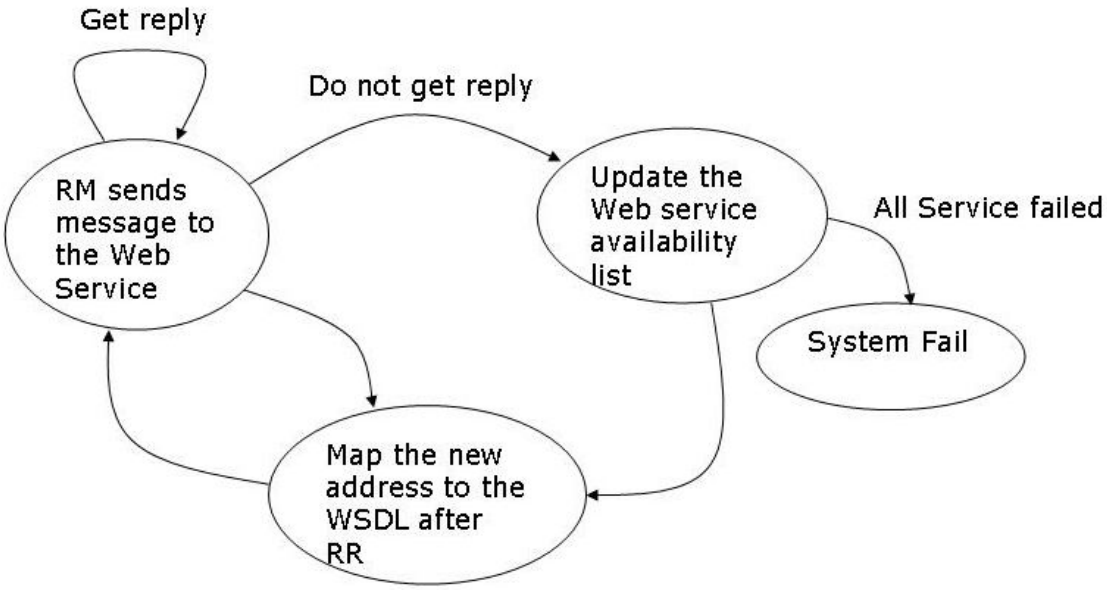

Figure 3.3: Workflow of the Replication Manager

3.2.2 N-version Programming Approach

In the second approach, different versions of the Web service are employed. As shown in Figure 3.4, the requests from the clients are forwarded to all versions of the Web services. When all the results are ready, a voting algorithm is applied to obtain the majority result and return the answer to the corresponding client.

The architecture of the system is similar to the first approach. However, the functionality of the replication manager is different. The replication manager is responsible for:

1. Creating a Web service.

2. Selecting the primary Web service for executing the voting procedure. Once the selected Web service gets the request, it will forward the request to all the Web services.

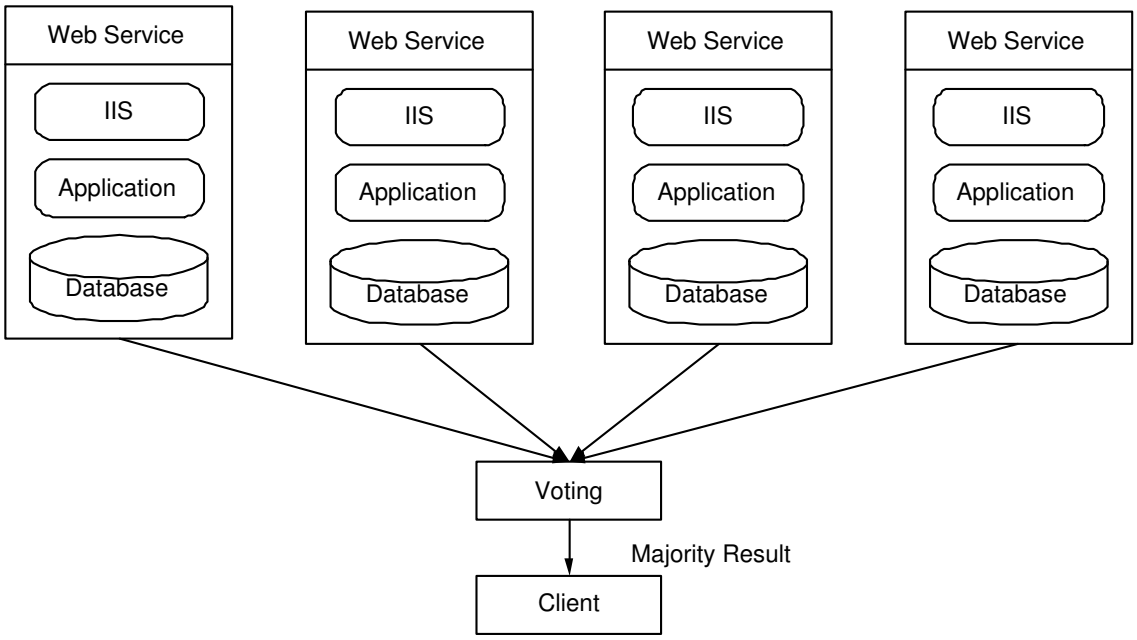

Figure 3.4: N-version programming Web services approach.

3. Keeping the availability list of the Web services.

4. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).

5. Continuously checking the availability of the Web services by using a watchdog.

3.2.3 Recovery Block Approach

In the third approach, different versions of the Web service are employed in the recovery block scheme. The architecture is shown in Figure 3.5. Checkpoints are defined in the Web services and the checkpoints are stored in the recovery cache.

The requests from the clients are sent to the primary Web services. The result will be examined by the acceptance test. If it passes

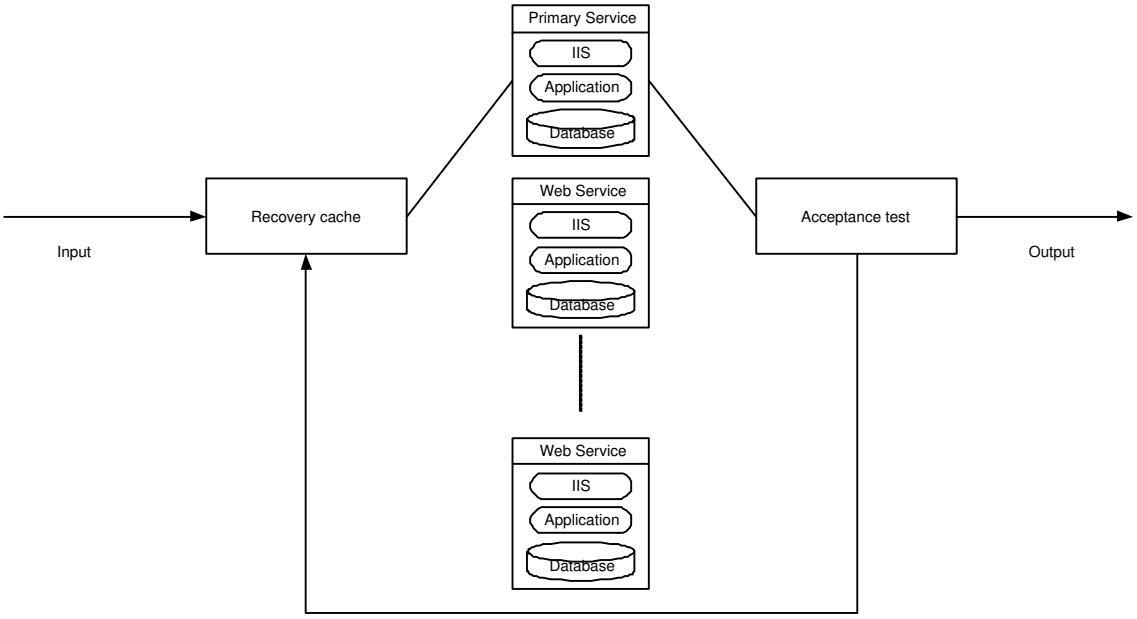

Figure 3.5: Recovery block approach.

the test, the result will be sent to the client. Otherwise, the alternative service will perform the service by rolling back to the checkpoint of the previous service.

The architecture of the system is similar to the pervious approaches. However, the functionality of the replication manager is different. The replication manager is responsible for:

1. Creating a Web service.
2. Selecting the primary Web service for providing the service to the client.
3. Caching the checkpoint of the Web service.
4. Keeping the availability list of the Web services.
5. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).

6. Continuously checking the availability of the Web services by using a watchdog.

7. Running the acceptance test for the Web service, if the test is passed, send the result to the client. Otherwise, rollback to the checkpoint and select an alternate Web service to provide service.

3.3 Roadmap for Experimental Research

We take a pragmatic approach by starting with a single service without any replication. The only approach to fault tolerance in this case is the use of redundancy in time. If a service is considered as an atomic action or a transaction in which the input is clearly defined, no interaction is allowed during its execution, and the outcome has possible states: correct or incorrect. In this case, the only way to make such a service fault tolerant is to retry or reboot it. This approach allows tolerance of temporary faults, but it will not be sufficient for tolerating permanent faults within a server or a service. One issue is how much delay the user can tolerate, and another issue is the optimization of the retry or the reboot time.

If redundancy in time is not appropriate to meet dependability requirements or if the time overhead is unacceptable, the next step is redundancy in space. Redundancy in space for services means replication where multiple copies of a given service may be executed sequentially or in parallel. If the copies of the same services are executed on different servers, different modes of operations are possible:

1. Sequentially, meaning that we await a response from a primary service and in case of timeout or a service delivering incorrect results, we invoke a back-up service (multiple backup copies are possible). This is also known as dynamic redundancy.

2. In parallel, meaning that multiple services are executed simultaneously and if the primary service fails, the next one takes over. Another variant is that the service whose response arrives first is taken.

3. There is also a possibility of majority voting using n-modular redundancy, where results are compared and the final outcome is based on at least $\lfloor n/2 + 1 \rfloor$ services agreeing on the result. This is also known as dynamic redundancy.

If diversified versions of different services are compared, the approach can be seen as either a Recovery Block (RB) system, where backup services are engaged sequentially until the results are accepted (by an Acceptance Test), or an N-version programming (NVP) system where voting takes place and majority results are taken as the final outcome. In case of failure, the failed service can be masked and the processing can continue.

NVP and RB have undergone various challenges and lively discussions. Critics state that the development of multiple versions is too expensive and dependability improvement is questionable in comparison to a single version, provided the development effort equals the development cost of the multiple versions. We argue that, in common with the maturity of service-oriented computing technologies, diversified Web services now predominate and the objec-

tions to NVP or RB can be mitigated. Based on market needs, service providers are competitively and independently developing their services and making them available on the market. With an abundance of services available for specific functional requirements, it is apparent that fault tolerance by design diversity will be a natural choice. Moreover, NVP can be applied to services not only for dependability but also for higher performance purposes, due to locality considerations.

Finally, a hybrid method may be used where both space and time redundancy are applied, and depending on system parameters, a retry might be more effective before switching to the back-up service. This type of approach will require a further investigation.

3.4 Summary

In this chapter, we describe the details of the proposed reliable Web service paradigm. The architecture and flow of work are presented. The system improve the reliability of Web service by applying both spatial and temporal replication. The system is coordinated by a replication manager and different fault-tolerance techniques are employed, including Round-robin scheduling algorithm, N-version programming and recovery block. Also, the roadmap for experimental research is presented.

□ **End of chapter.**

Chapter 4

Web Service Composition

4.1 Introduction

Diversity is one of the key elements in the proposed paradigm. In the emergence of service-oriented computing, different versions of Web services or even different versions of their components are abundantly available in the Internet. The combination of different versions of the Web service or their components is thus becoming critical for enabling different versions in a server application using the N-version programming approach. In this section, we propose an algorithm for composing Web services. With an N-version programming Web service approach, the reliability of the overall system is improved.

4.2 Web Service Description

The description of a Web service is statically provided by WSDL, including Web service functional prototypes. However, its static nature limits the flexibility for composing Web services. Different Web

services provide their services at different time, and so a dynamic composition approach is necessary for composing different versions of Web services and making them to be available in the Internet.

In the Web services, the communication mainly depends on the messages exchange between different Web servers. The Web Service Choreography Interface (WSCI) [4] is an XML-based language for the description of the observable behavior of a Web service in the context of a collaborative business process or work-flow. WSCI describes the dynamic interface of the Web Service participating in a given message exchange by means of reusing the operations defined for a static interface. It defines the flow of messages exchange by a stateful Web service, describing its observable behavior. By specifying the temporal and logical dependencies among the message exchange, WSCI is employed to describe a service in such a way that other Web services can unambiguously interact with the described service in conformity with the intended collaboration. Though WSCI provides a message-oriented view of the process, it does not define the internal behavior of the Web service or the process.

4.3 Proposed Composition Method

Our proposed service composition method is based on two standard Web service languages: WSDL and WSCI. WSDL describes the entry points for each available service, and WSCI describes the interactions among WSDL operations. WSCI complements the static interface details provided by a WSDL file, as WSCI describes the ways operations are choreographed and their properties. This is achieved

with the dynamic interface provided by WSCI through which the inter-relationship between different operations in the context of a particular operational scenario.

4.3.1 Web Service Composition Algorithm

The flow of the composition procedure is as follows: First, get the WSDL of the Web service components from UDDI. Then, through the messages between the Web services, obtain the WSCI of the components. Afterwards, examine the input and output of the components through WSDL and determine the interactions between different components to provide the service through WSCI. Finally, perform the composition of the Web service with the information obtained in the composition procedure. The detailed composition algorithm is shown in Algorithm 1.

In Algorithm 1, we aim to build a tree for the Web service composition. We use a bottom-up approach to perform the composition, that is, we build the composition tree from output to input.

When we get the required output, search the Web services in the WSDL. In the *operation* tag of the WSDL, the output information is stated. When the desired output is found, that Web service component (CP_n) is inserted as the root of the tree. Then, if the input of that operation matches the required input, the searching is finished and the input is inserted as a child of the CP_n. Otherwise, we will search the *action* tag in the WSCI in finding matches to the *operation* in CP_n. After the *action* is completed, we determine the previous *action*. Then, we can find the *operation* prototypes in the WSDL. If the input of this operation matches the required input, then the com-

Algorithm 1 Algorithm for Web service composition

Require: $I[n]$: required input, $O[n]$: required output

1: CP_n: the n^{th} Web services component
2: **for all** $O[i]$ **do**
3: Search the WSDL of the Web services, and find the CP_n 's operation output = $O[i]$. Then, insert CP_n into the tree.
4: **if** the input of the operation = $I[j]$ **then**
5: Insert the input to the tree as the child of CP_n.
6: **else**
7: Search the WSCI of CP_n, WSCI.process.action = operation.
8: Find the previous action needing to be invoked.
9: Search the operation in WSDL equal to the action.
10: **if** input of the operation = $I[i]$ **then**
11: Insert input to the tree as the child of CP_n
12: **else**
13: go to step (8)
14: **end if**
15: **end if**
16: until reaching the root of WSCI and not finding the correct input, search other WSDL with output = $I[j]$, insert CP_m as the child of CP_n and go to step (7) to do the searching in WSCI of CP_m.
17: **end for**

position is finished. Otherwise, we will iterate until the root of the WSCI is reached.

If the desired input is still not found, we will search for the operations in other WSDL whose output is equal to the input of CP_n. If the next Web service component found is CP_m, then CP_m is inserted as the child of CP_n. We perform the searching iteratively and continue to build the tree until all the inputs match the required input.

4.3.2 Case Study

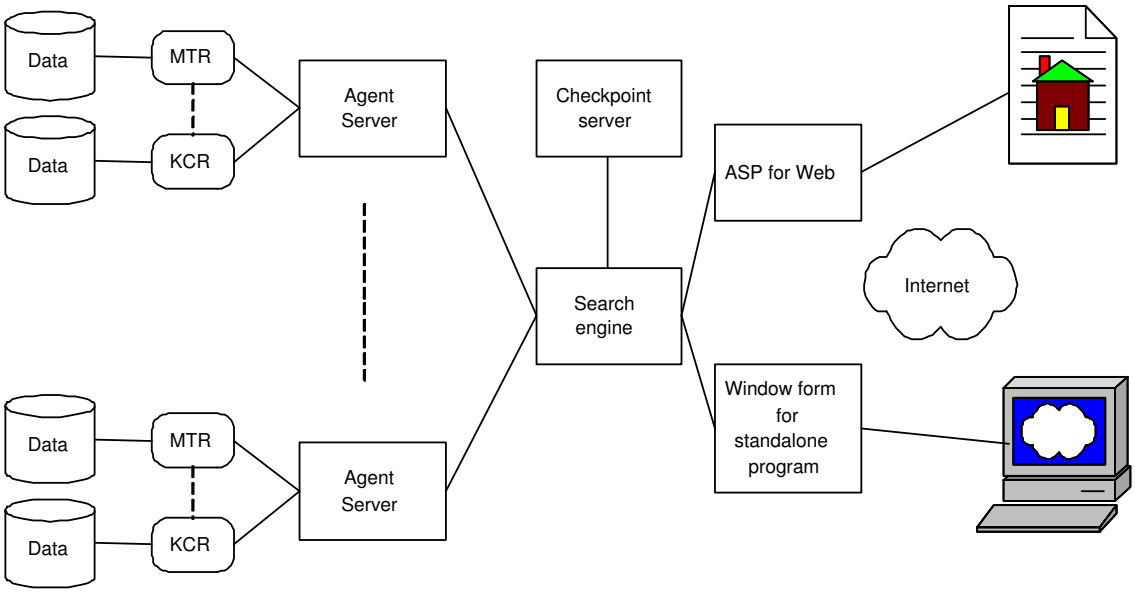

Figure 4.1: Best Route Finding system architecture.

To illustrate the above procedure, we present the Web services composition with the Best Route Finding system (BRF) [15] whose architecture is shown in Figure 4.1. This system suggests the best route for a journey within Hong Kong by public transport, based on input consisting of the starting point and the destination. BRF

consists of different components, including a search engine, agent servers, and the public transport companies. We acquired several versions of BRF, which are implemented by different teams using different components. Also, the Web service components may differ from versions to versions; thus, in this experiment, we try to compose the Web services from different versions with the WSDL and WSCI provided therein.

Also, the following shows part of the WSDL specification of the search engine. The WSDL identifies the input and output parameters of the services provided by the *search engine* of the system.

```
<?xml version="1.0" encoding="UTF-8"?>
 ...
<portType name=BRF">
  <operation name=shortestpath">
    <input message=
           "tns:startpointDestination"/>
    <output message="tns:pathArray"/>
  </operation>

  <operation name=addCheckpoint">
    <input message="tns:pathArray"/>
    <output message=
           "tns:addAcknowledgement"/>
  </operation>
  ...
  </operation>
</portType> </definitions>
```

The following shows part of the WSCI specification of the *search engin*e.

```
<correlation name=pathCorrelation
property=tns:pathID></correlation>

<interface name=busAgent>
  <process instantiation="message">
    <sequence>
       <action name="ReceiveStartpointDest
          role="tns:busAgent
          operation="tns:BRF/shortestpath">
        </action>
        <action name="Receivecheckpoint
          role=" tns:busAgent
          operation="tns:BRF/addCheckpoint">
          <correlate correlation=
             tns: pathCorrelation/>
          <call process=tns:SearchPath/>
        </action>
    </sequence>
  </process>
...
```

Based on Algorithm 1, a composition tree is built, giving the result as shown in Figure 4.2.

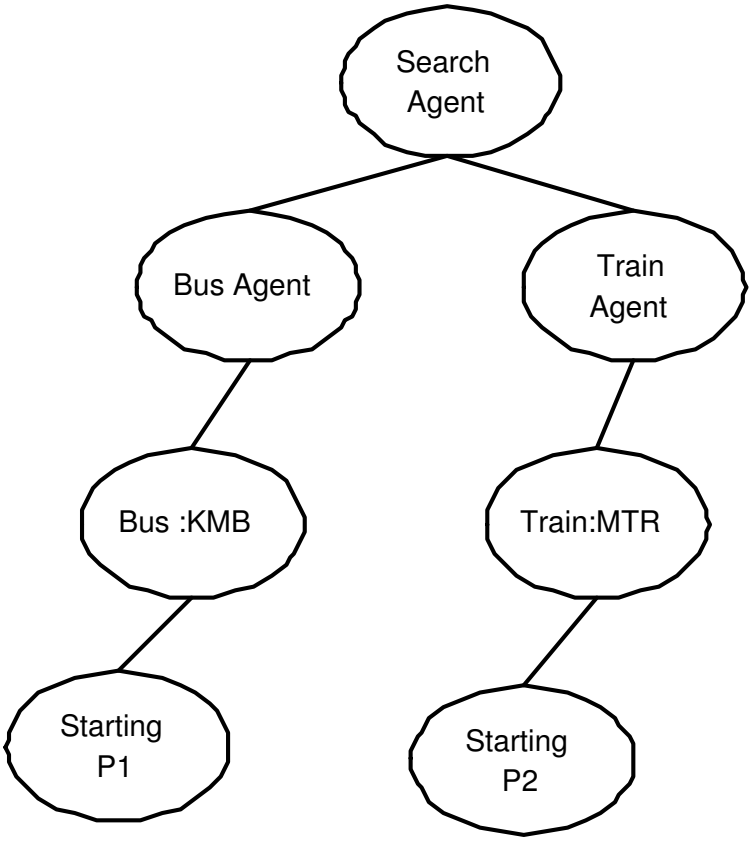

Figure 4.2: Composition tree of BRF.

4.4 Verification with Petri-Net

To verify the correctness of the composed Web service, Petri-Net [53] is employed. We first construct a Petri-Net for the Web service with the information provided in BPEL.

4.4.1 BPEL

By defining and executing business processes involving Web services, BPEL enables the top-down realization of Service-oriented Architecture (SOA) through composition, orchestration, and coordination of Web services. BPEL provides a relatively easy and straightforward way to compose several Web services into new composite services called *business processes*.

After a Web service is composed with the proposed Algorithm 1, a BPEL is constructed. BPEL describes the composition properties of the Web service, such as communication and specific behaviors.

A BPEL process specifies the exact order in which participating Web services should be invoked, either sequentially or in parallel. With BPEL, conditional behaviors can be expressed. For example, an invocation of a Web service can depend on the value of a previous invocation. Also it can construct loops, declare variables, copy and assign values, define fault handlers, and so on. By combining all these constructs, the flow of the Web service can be defined.

4.4.2 Building Block of Petri-Net

In the verification process, we employ Petri-Nets to build the model of the Web service to prevent deadlock and construct dynamic re-

lations. Different building blocks of Petri-Nets are defined according to the activities in BPEL schema, including inner-service, intra-service, inter-activity, and intra-activity. With the defined blocks, we map the operations or activities specified in BPEL to the Petri-Net building blocks. Then, a Petri-Net for a specified Web service is generated. Some major building blocks are defined in Table 4.1 and Table 4.2, respectively.

Table 4.1: Petri-Net building blocks of basic activities

Building Block type	Description
Invoke	The Invoke activity directs a Web service to perform an operation.
Reply	The Reply activity matches a Receive activity. It has the same partner link, port type, and operation as its matching Receive. Use a Reply to send a synchronous response to a Receive.
Empty	The Empty activity is a no operation instruction in the business process.
Assign	The Assign activity updates the content of variables.
Terminate	The Terminate activity stops a business process.
Throw	The Throw activity provides one way to handle errors in a BPEL process.
Wait	The Wait activity tells the business process to wait for a given time period or until a certain time has passed.

A Web service operation is composed of basic activities (includ-

Table 4.2: Petri-Net building blocks of structure activities

Building Block type	Description
While	Repeat the same sequence of activities as long as some condition is satisfied.
Switch	Use ”case-statement” to produce branches.
Sequence	Definition of a series of steps for the orderly sequence.
Link	Link different activities work together.
Flow	A series of steps should be specified in parallel implementation.

ing Receive, Reply, Assign, Invoke, Empty, Terminate, and Wait) and structures activities (including While, Switch, Sequence, Link and Flow. The sample basic activities translation are shown in Figure 4.3 to Figure 4.9 and the structure activities are shown in Figure 4.10 to Figure 4.14. With the activities in BPEL, Web services are described procedurally. In a Petri-Net, a *place* connected to a *transition* intuitively expresses the *states* before and after executing the corresponding *action*. *Firing* a transition means that the corresponding *action* is executed. Moreover, Web service invocation is expressed by entering a *token* in a *place* which denotes the *starting point* of the operation.

Figures 4.3 to 4.14 illustrate some Web service operations composed with Petri-Net building blocks. A *building block* is presented by a *place* with a *token* whose type is specified by the block type.

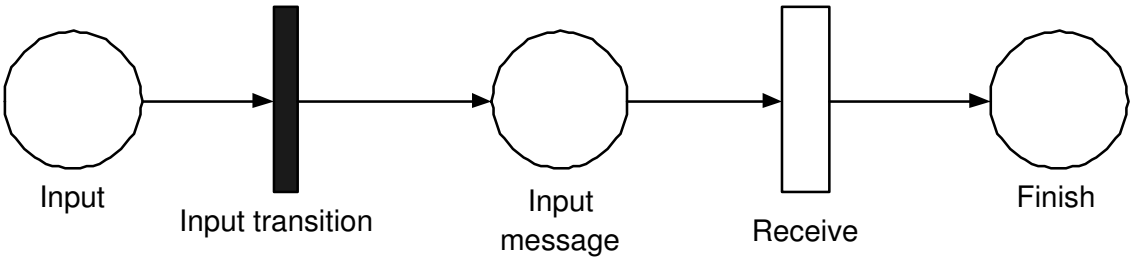

Figure 4.3: Basic Petri-Net building block – Receive.

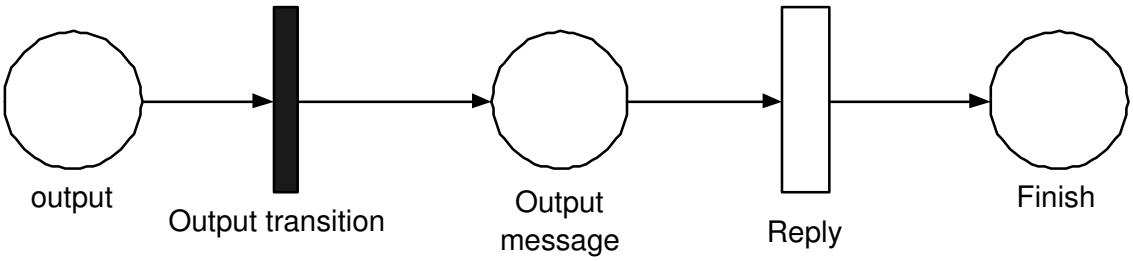

Figure 4.4: Basic Petri-Net building block – Reply.

An *arc* is used to link the transition with another *arc* corresponding to the input or output message consisting of those blocks based on the relationship defined in Table 4.1 and Table 4.2.

With the Algorithm 1 and BPEL Petri-Net building blocks, Petri-Nets of different versions of BRF are generated. One of the composed BRF is shown in Figure 4.16. With the constructed Petri-Net, we perform the operation to check the correctness and verify that the Web service is deadlock-free.

4.5 Summary

In this chapter, we describe the details of the proposed algorithm for Web service composition. The composition algorithm makes use of the WSDL and WSCI. The correctness is checked by the acceptance

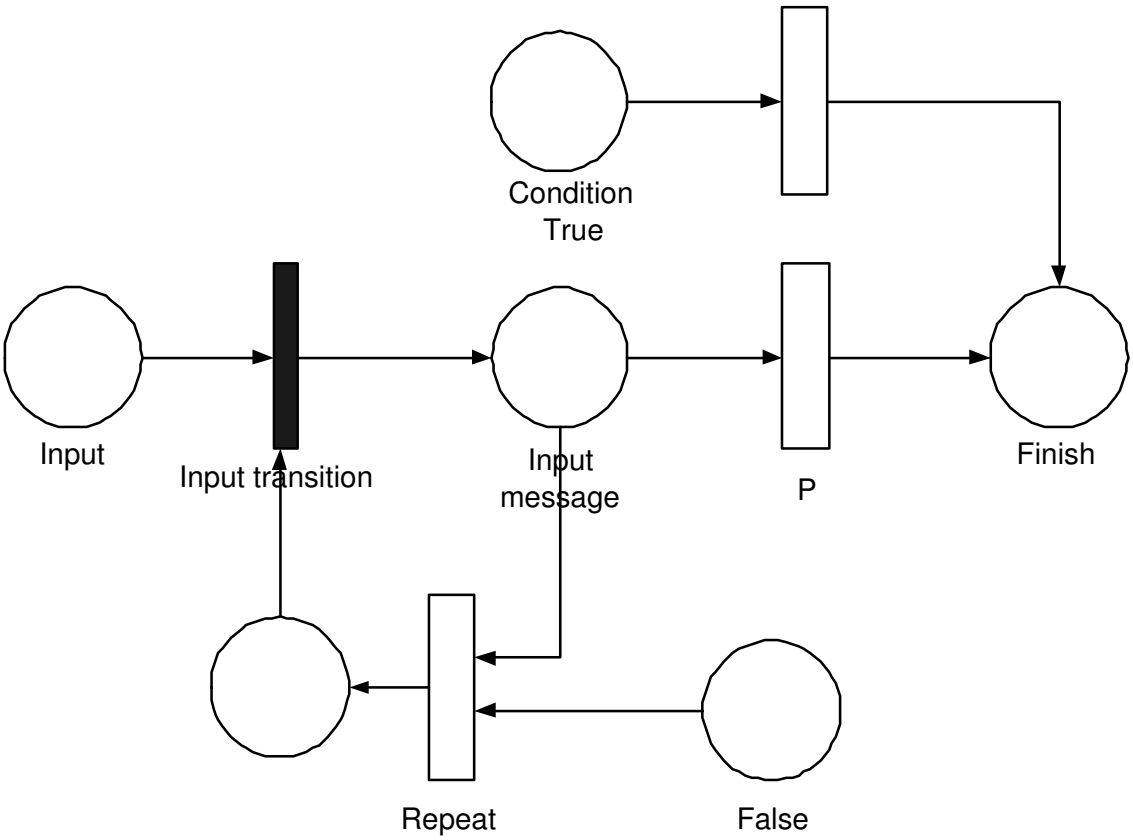

Figure 4.5: Basic Petri-Net building block – Wait.

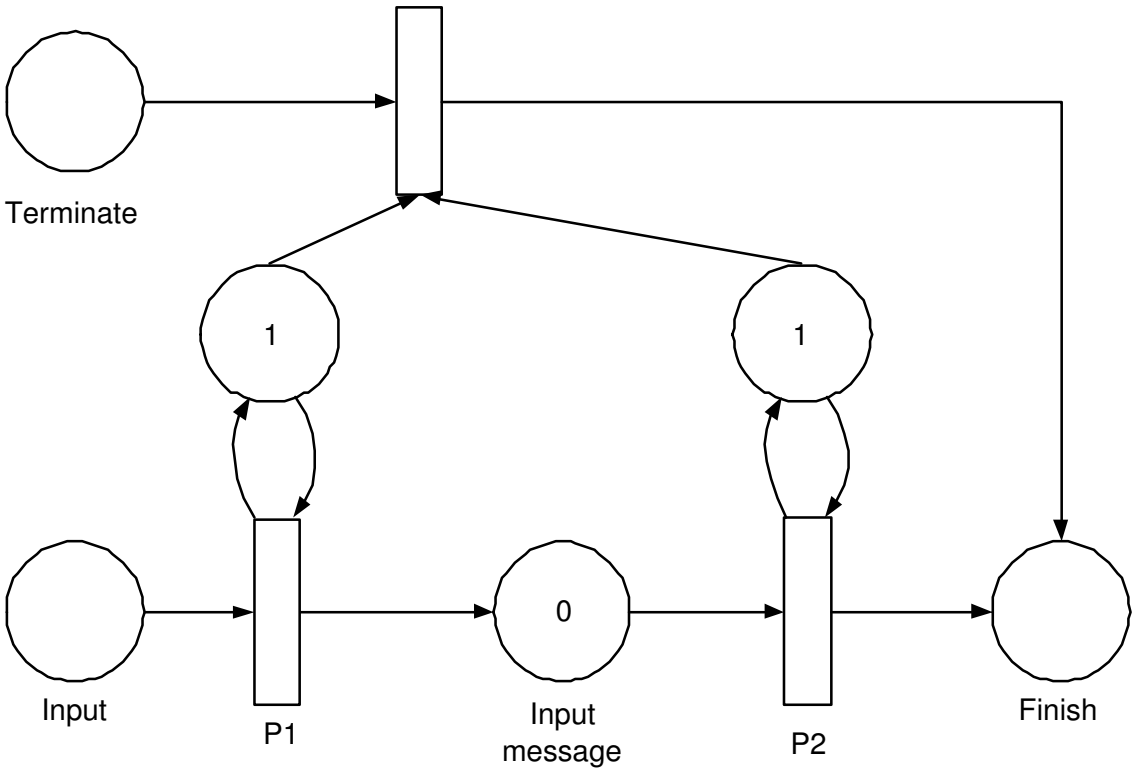

Figure 4.6: Basic Petri-Net building block – Terminate.

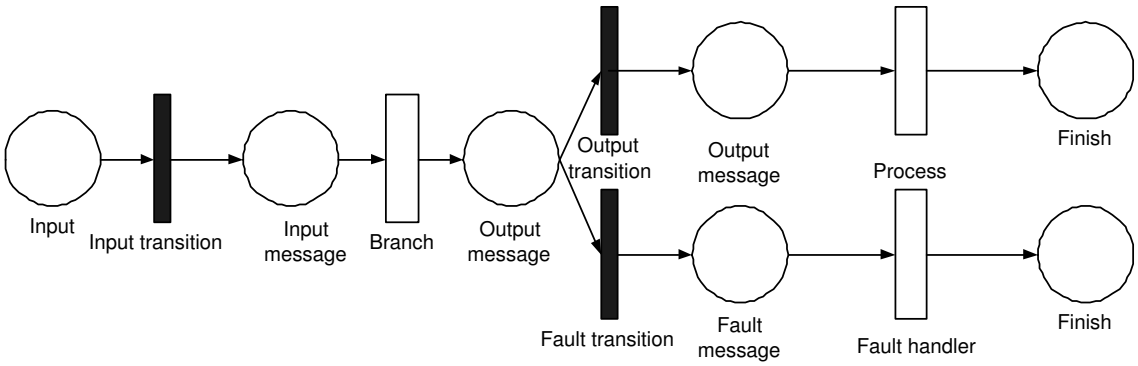

Figure 4.7: Basic Petri-Net building block – Invoke.

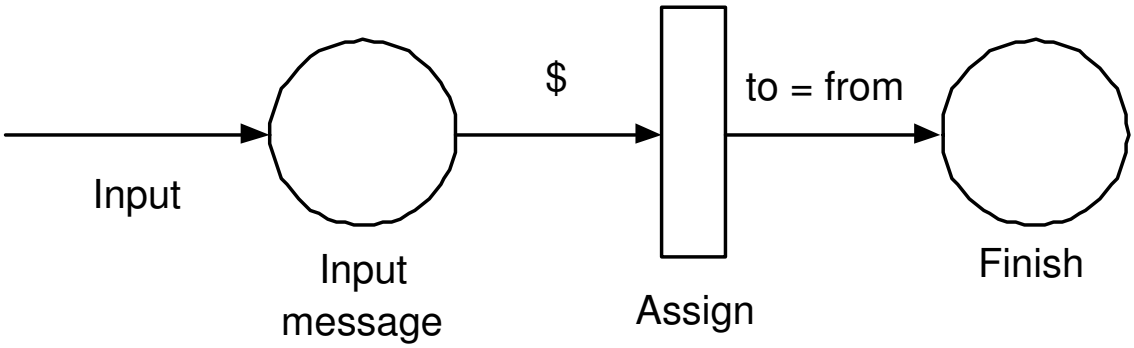

Figure 4.8: Basic Petri-Net building block – Assign.

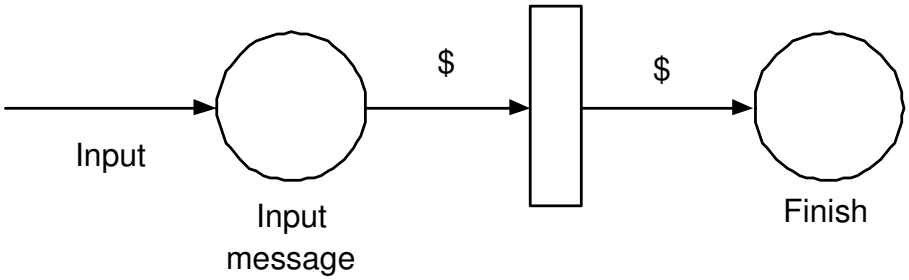

Figure 4.9: Basic Petri-Net building block – Empty.

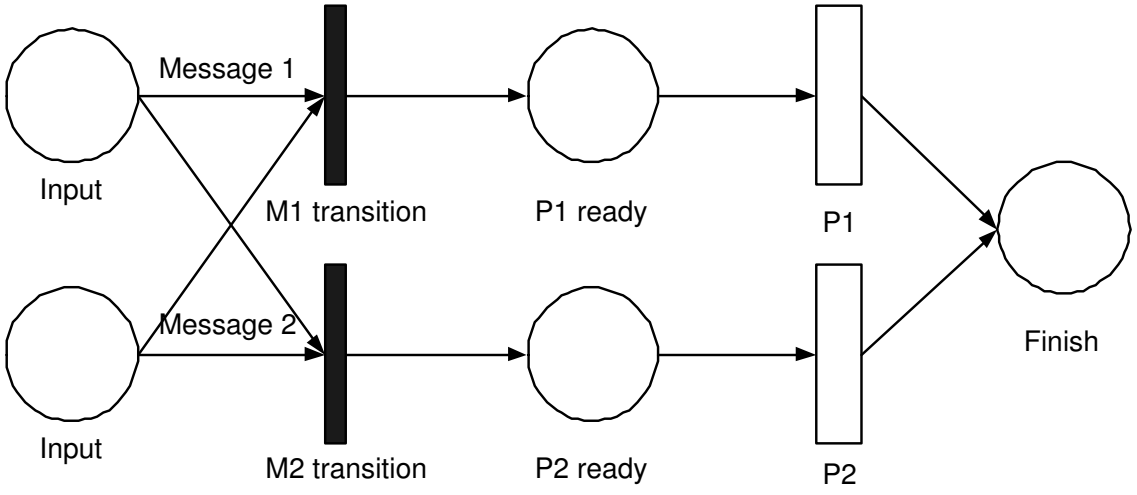

Figure 4.10: Structure Petri-Net building block – Pick.

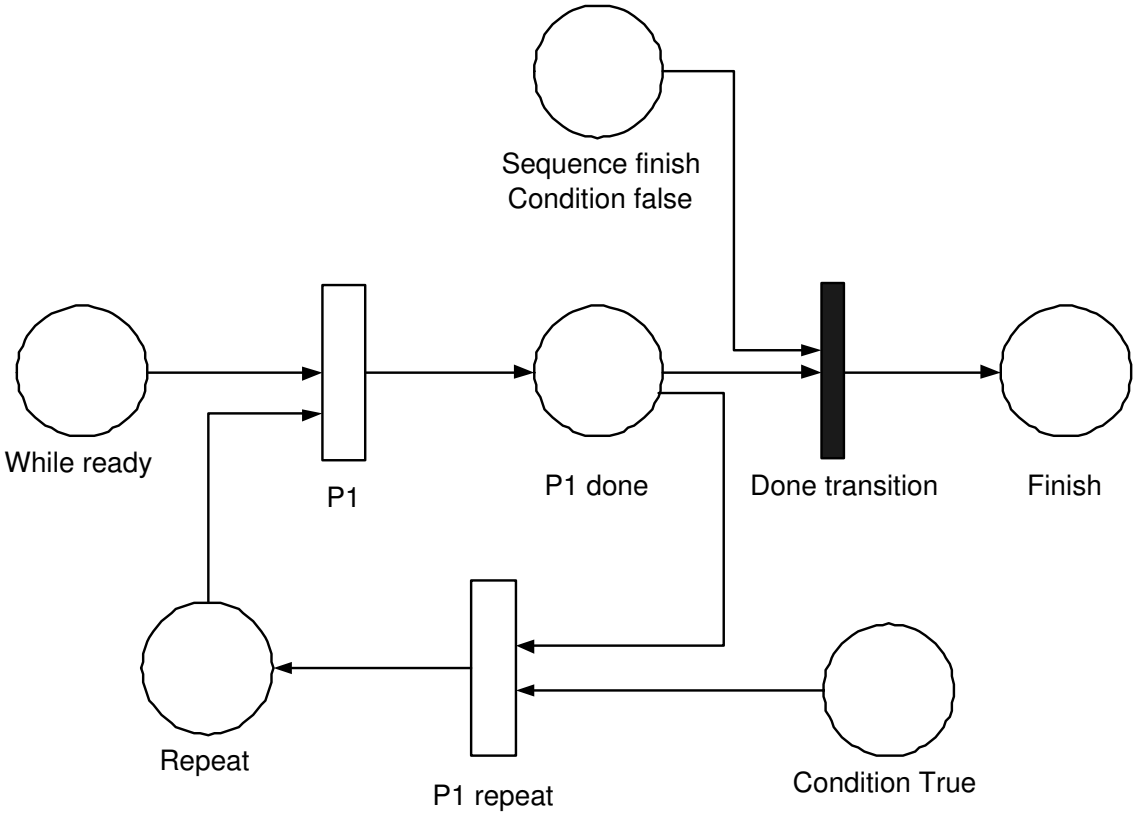

Figure 4.11: Structure Petri-Net building block – While.

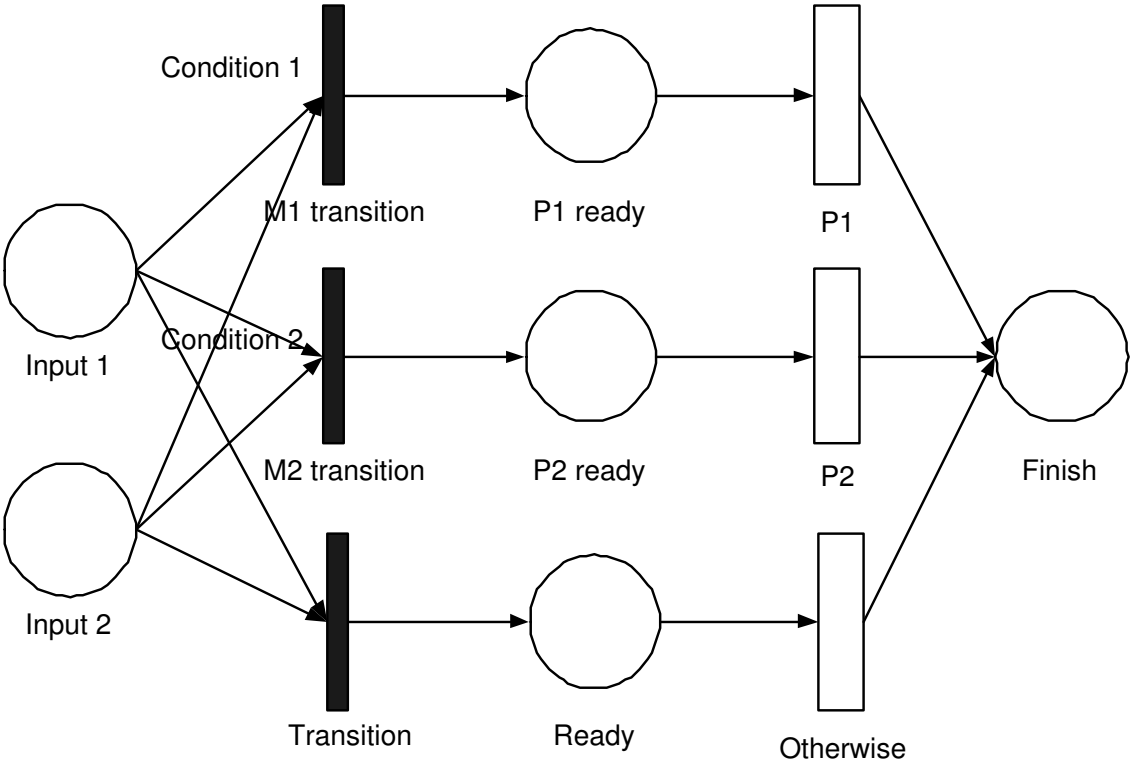

Figure 4.12: Structure Petri-Net building block – Switch.

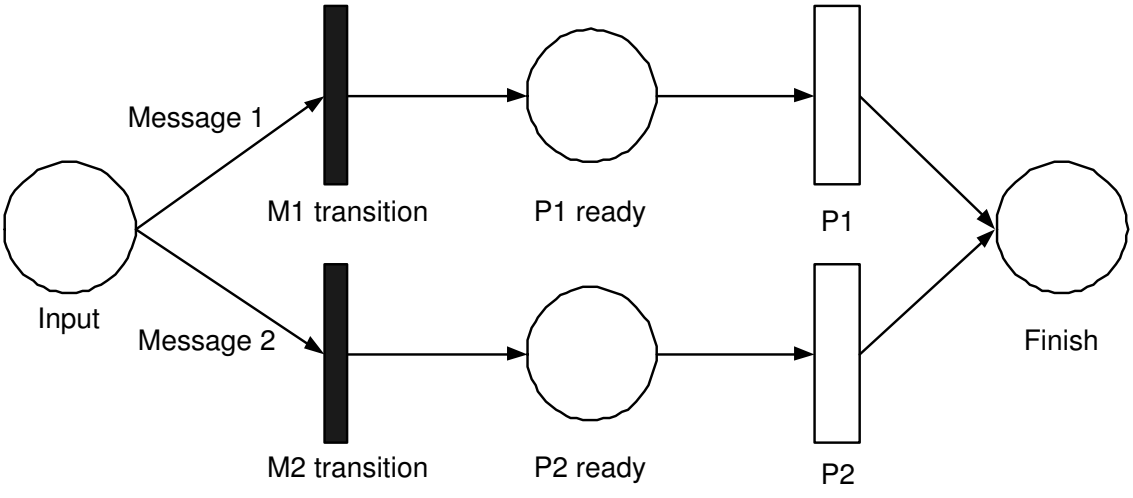

Figure 4.13: Structure Petri-Net building block – Flow.

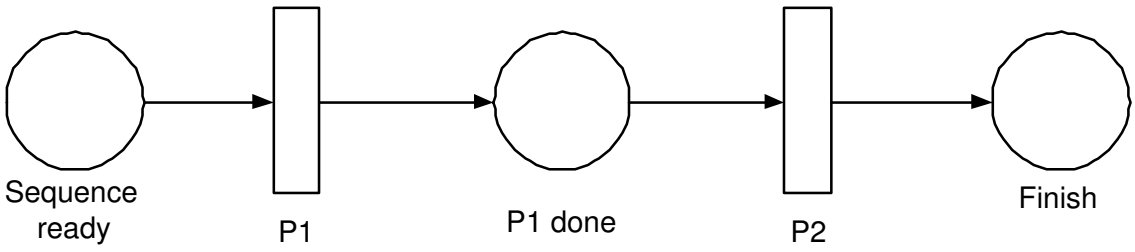

Figure 4.14: Structure Petri-Net building block – Sequence.

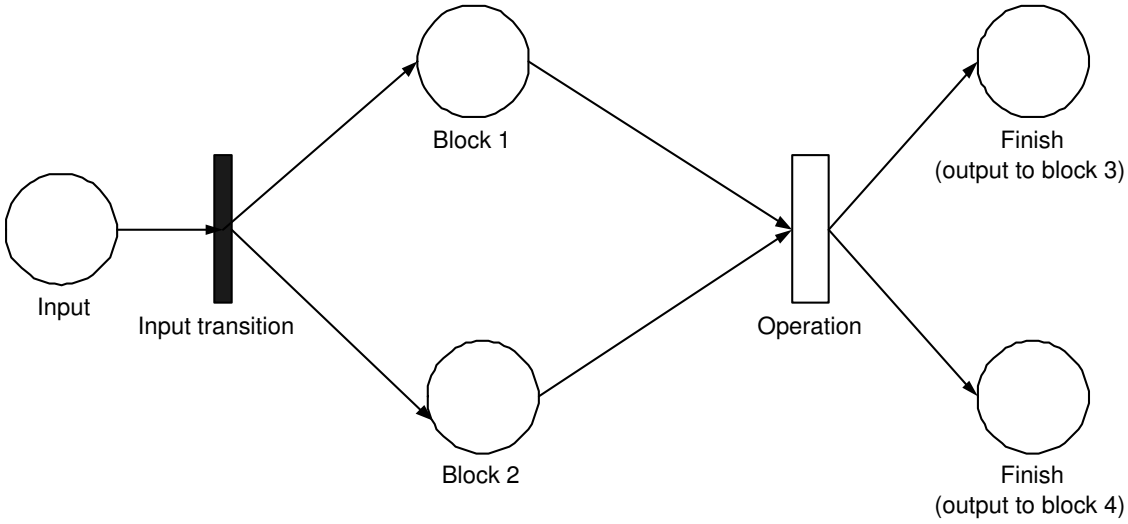

Figure 4.15: Composed Petri-Net building block graph.

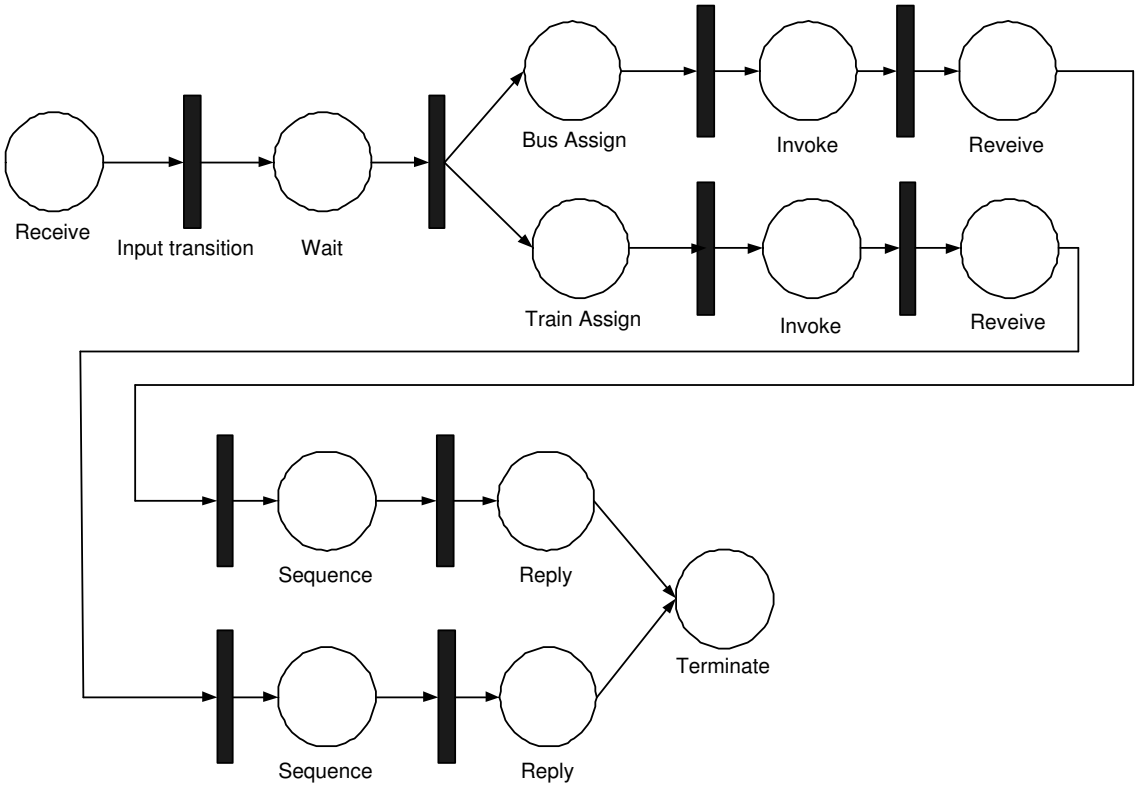

Figure 4.16: The Petri-Net of a BRF.

test and the deadlock-free is verified by Petri-Net. We also analyze the algorithm with case study.

□ **End of chapter.**

Chapter 5

Reliability Modeling

5.1 Introduction

We develop reliability models of the proposed Web service paradigm using Petri-Net [53] and Markov chains [24]. The models are described in this section. The reliability models are analyzed and verified using the SHARPE tool [61]. Petri-Net is built to evaluate the performance of the system, and the Markov chains model is developed to analyze the system reliability.

Furthermore, we develop a mathematical model for each approach applied in the reliable Web service paradigm. Thus, the reliability of the system can be evaluated and compared systemically.

5.2 Modeling with Petri-Net

We analyze different approaches with Petri-Net. With the models, we evaluate the performance and the throughput of different approaches in the proposed reliable Web service paradigm.

5.2.1 Round-robin

In Figure 5.1, we model a system composed of two Web services. When the messages arrive at the first Web service, a message queue is formed and the Web service will handle the requests in the message queue. In the model, the messages are queued up in the *queue* state. Next, the *token* will be passed to the *service* and arrive at the *finished job* state. Then, the *token* is passed to *queue2*, which is for the second Web service.

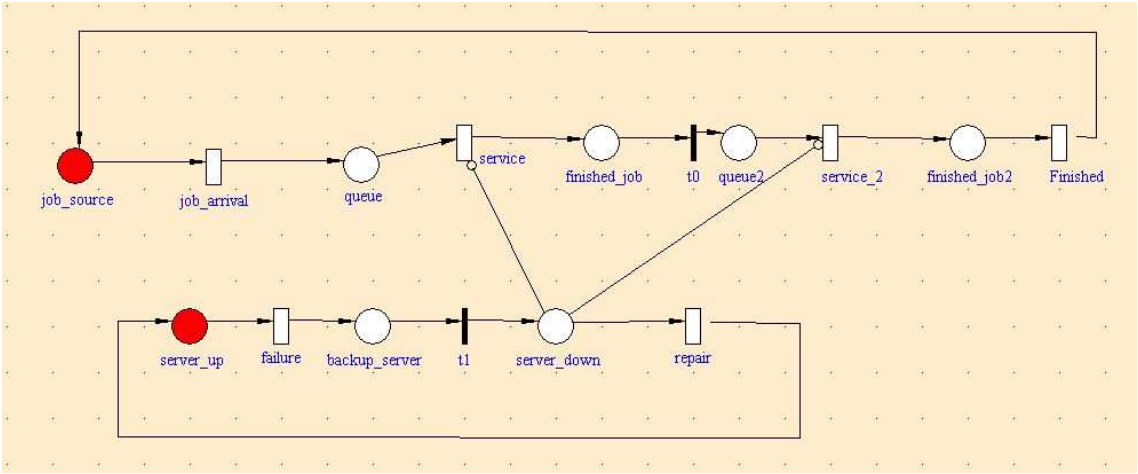

Figure 5.1: Petri-Net based reliability model for the proposed system with Round-robin algorithm

The reliability of the service is directly affected by the availability of the Web server. When the server is down, the two Web services will not be available. In our system, when the primary server fails, the backup server will be invoked.

5.2.2 N-version Programming

In Figure 5.2, we model the system with N versions of the Web service. The messages are queued up in the *queue* state. Then, the *token* will be passed to services $servicev1$ to $servicev5$. When the job is finished, it arrives at the *finished job* state. Finally, the *token* is passed to the *voting* state.

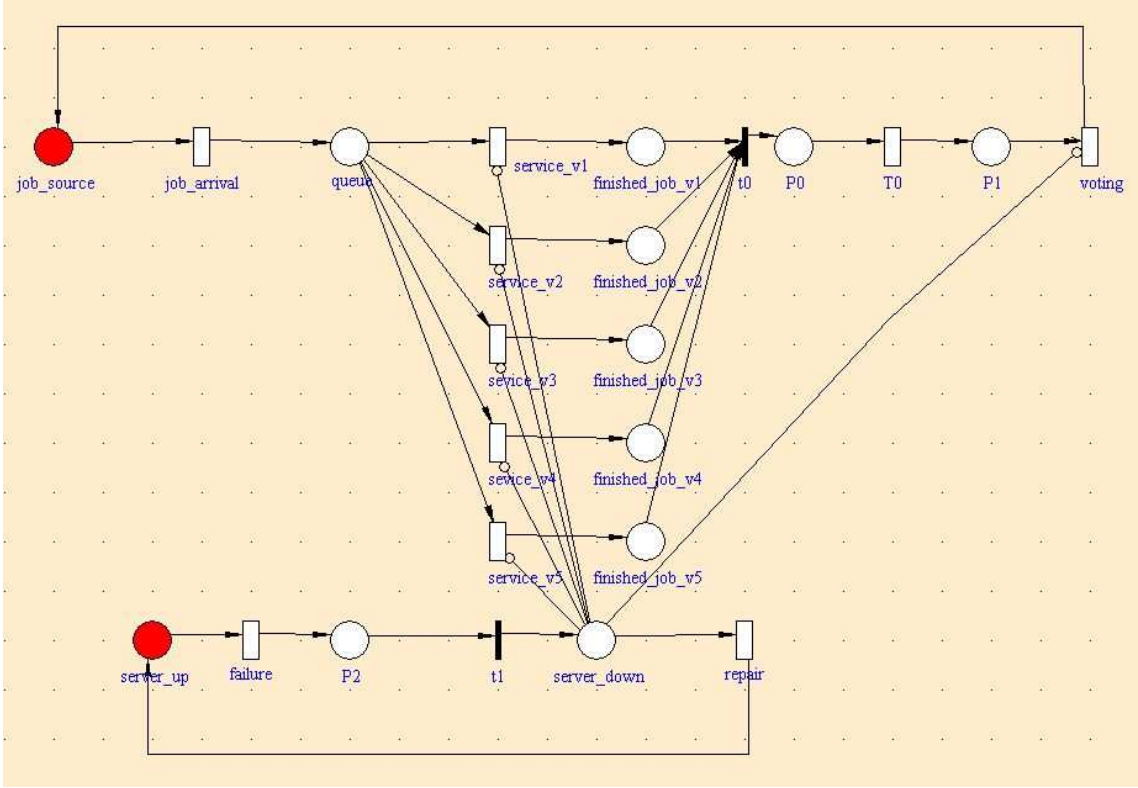

Figure 5.2: Petri-Net based reliability model for the proposed system with N-version programming

5.2.3 Recovery Block

In Figure 5.3, we model the Web service system with recovery block and checkpointing. The messages are queued up in the *queue* state.

Then, the *token* will be passed to services $servicev1$. When the job is finished, it arrives at the *finished job* state. After that, the *token* is passed to the *Acceptance test* state. If it is correct, it finally goes to the *finished* state. Otherwise, it will goes to the *checkpoint* state and *servicev2* state. If it is still failed, the token is passed to the *server down* state and *repair* transition is reached.

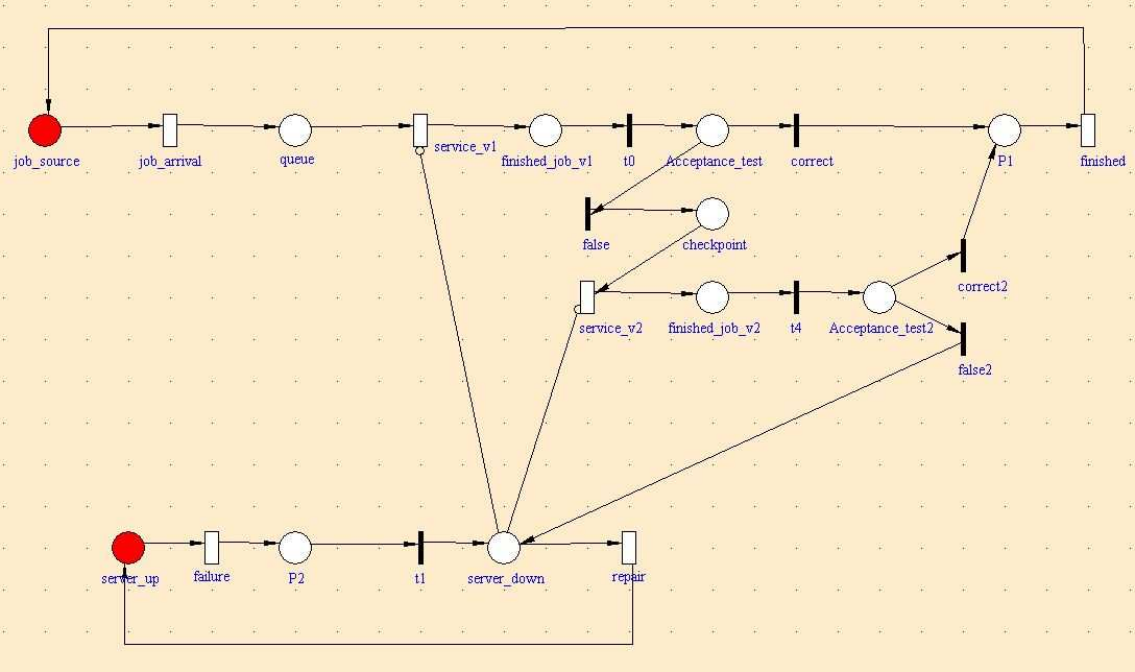

Figure 5.3: Petri-Net based reliability model for the proposed system with recovery block

All the models are developed and verified through the tool SHARPE [61] to be deadlock-free. Afterwards, we perform experiments on the proposed system and feed the parameters to the models. Thus we are able to predict the throughput and performance of the system. The result and analysis are shown in the next chapter.

5.3 Modeling with Markov Chain

In this section, we model our proposed system with Markov chain. In Figure 5.4(a), the state s represents the normal execution state of the system with n Web service replicas. In the event of a fault causing the primary Web service to fail, the system will either go into the other states (i.e., $s - j$, which represents the system with $n - j$ working replicas remaining, if the replication manager responds on time), or it will go to the failure state F with conditional probability $(1 - C_1)$. λ^* denotes the failure rate , i.e. the rate of occurrence of failures from which recovery cannot be completed, and C_1 represents the probability that the replication manager responds in time to switch to another Web service.

When the failed Web service is repaired, the system will go back to the previous state, $s-j+1$. μ^* denotes the rate at which successful recovery is performed in this state, and C_2 represents the probability that the failed Web server reboots successfully. If the Web service fails, it goes to another Web service. When all Web services fail, the system enters the failure state F. λ_n represents the network fault rate.

States $(s - 1)$ to $(s - n)$ in Figure 5.4(a) represent the working states of the n Web service replicas and the reliability model of each Web service is shown in Figure 5.4(b). There are two types of faults simulated in our experiments: P_1 denotes a temporary fault and P_2 denotes a permanent fault. If a fault occurs in the Web service, either the Web service can be repaired with μ_1 (to enter P_1) or μ_2 (to enter P_2) repair rates with conditional probability C_1. If the fault cannot be recovered, the system goes to the next state $(s - j - 1)$ with one

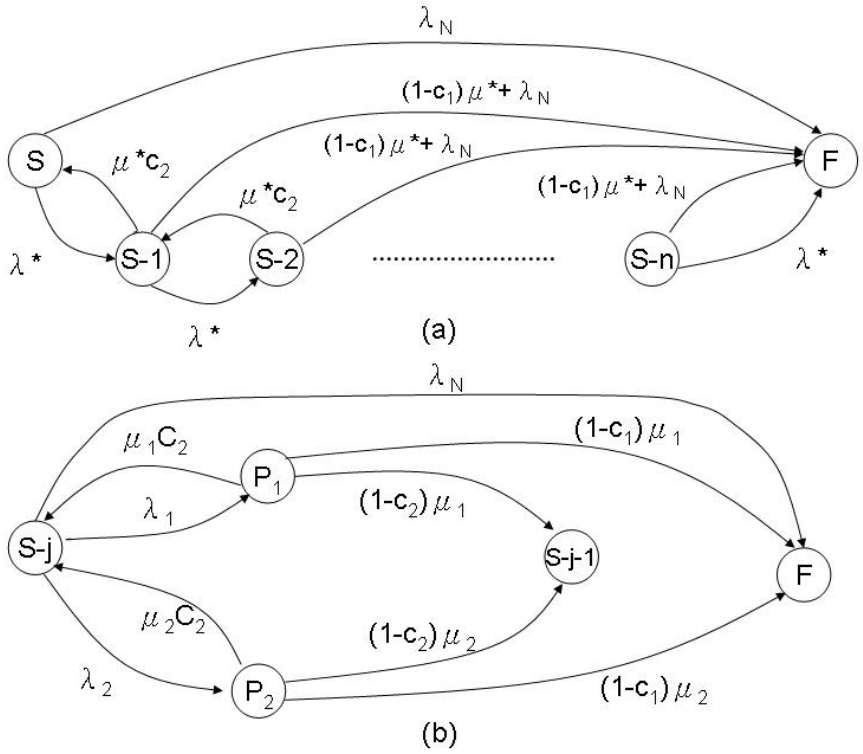

Figure 5.4: Markov chain based reliability model for the proposed system

less Web service replica available. If the replication manager cannot respond in time, it will go to the failure state. From the graph, two formulae can be obtained:

$$\lambda^* = \lambda_1 \times (1 - C_1)\mu_1 + \lambda_2 \times (1 - C_2)\mu_2 \tag{5.1}$$

$$\mu^* = \lambda_1 \times \mu_1 + \lambda_2 \times \mu_2 \tag{5.2}$$

5.4 Mathematical Models

We analyze the different approaches of the proposed system with mathematical models in this section. With the models, we evaluate and compare the performance and the correctness of different approaches in the proposed reliable Web service paradigm.

5.4.1 Round-robin

In Round-robin approach, there are numbers of replicas. We define the reliability of each replica be

$$R(n) = \chi \tag{5.3}$$

We define the probability the n^{th} replica is choosen be $Q(n)$. Thus, the reliability of Round-robin is

$$p(n) = \chi Q(n) \tag{5.4}$$

When retries are allowed in the system, the reliability becomes

$$p(n,r) = \sum_{j=0}^{r} \chi Q(n) \tag{5.5}$$

where r is the number of retries.

5.4.2 N-version Programming

In N-version programming, there are numbers of versions in the system. We define the reliability of a version as the probability that the version does not cause to fail on an input. The reliability of a program version is

$$R_{1vp} = \chi^m \tag{5.6}$$

where χ is the reliability of each version and m is the number of success versions in the system [6].

We define $p_N(i)$ as the probability that selected i of N versions failed and other versions succeed on an input. If the versions fail independently,

$$p_N(i) = (1-\chi)^i \chi^{N-i}, i = 0, 1, \ldots, N \tag{5.7}$$

Thus, this is the reliability of the approach with N-version programming.

When retries are allowed in the system, the reliability becomes

$$p_N(i,r) = \sum_{j=0}^{r} (1-\chi)^i \chi^{N-i}, i = 0, 1, \ldots, N \tag{5.8}$$

where r is the number of retries.

5.4.3 Recovery Block

There are numbers of versions in the system. We define the reliability of a version as the probability that the version does not cause to fail on an input. The reliability of a program version is

$$R_1 = \chi \tag{5.9}$$

In the recovery block, there is an acceptance test in the system. We define the probability that the testing segment rejects a correct result as t_1 and probability that the testing segment accepts an incorrect result as t_2 [8, 9].

We define $p_N(k)$ as the probability that the recovery block failed k times and it must be smaller than N which is the number of versions in the system. If the versions fail independently,

$$p_N(k) = \{(1-\chi)t_1\}^k \chi(1-t_1)(1-t_2), k < N \tag{5.10}$$

When retries are allowed in the system, the reliability becomes

$$p_N(k,r) = \sum_{i=0}^{r} \{(1-\chi)t_1\}^{k(i)} \chi(1-t_1)(1-t_2), k < N \tag{5.11}$$

where r is the number of retries and $k(i)$ is the number of fails in the i^{th} trial.

5.5 Summary

In this chapter, we develop the model of the proposed reliable Web service paradigm with Markov chain and Petri-Net. In the following chapter, we will perform a series of experiments to evaluate and

compare the models developed in this chapter. With the models, we will compare and analyze the results obtain from the experiments. The detail will be described in the next chapter. Also, through the models, we demonstrate the properties of the Web service, including the reliability, performance and throughput.

□ **End of chapter.**

Chapter 6

Experiments

6.1 Introduction

In this chapter, we describe the various approaches and experiments in more detail. We formulate several additional quality-of-service parameters from the viewpoint of service customers. We propose a number of fault injection experiments showing both dependability and performance with and without diversified Web services. The outlined roadmap to fault-tolerant services leads to ultra reliable services where hybrid techniques of spatial and time redundancy can be used for optimal.

6.2 Optimal Parameters

There are numbers of parameters can be varied in the system. Before evaluating the reliability of the system, we try to find the optimal parameters of the system. To evaluate the parameters in the system, we perform a set of experiments. Through the experiments, we obtained a set of optimal parameters setting for the Web service system. The

Table 6.1: Number of failure with varying number of tries

Number of retries	Number of failure in temporary fault	Number of failure in permanent fault
0	9978	8994
1	25	34
2	0	0
3	0	0
4	0	0
5	0	0

Table 6.2: Number of failure with varying timeout period for retry

Timeout period for retry (s)	Number of failure in temporary fault	Number of failure in permanent fault
0	270	8974
2	45	562
5	0	4
10	0	0
20	0	0

parameters we examined include: number of retries, timeout period for retry, polling frequency, number of replicas and load of server. The types of fault injected include permanent fault (the server is down permanently once the fault occurs) and temporary fault (the fault only occurs randomly). The results are shown in Table 6.1 to 6.6, respectively. In each experiment, a total of 1,000,000 requests are handled in a 10-days duration.

In Table 6.1, we found that the number of retry depends on the failure rate of the Web service. If the failure rate of the Web service is large, the number of retry needed for the application is increased.

Table 6.3: Number of failure with varying timeout period for retry in a single server

Timeout period for retry	Number of failure in temporary fault	Number of failure in permanent fault
0	9826	857998
2	168	846782
3	12	842136
5	0	821634
6	0	324853
7	0	89421
8	0	7124
9	0	687
10	0	2
12	0	0
14	0	0

In Table 6.2, as the timeout period for retry is too short, the replica cannot reboot on time, causing the number of failures to increase. Also, another cause of the failure is that the replication manager cannot respond on time to switch the primary Web service. There are five tries, and the reboot time for a server is around 50 seconds.

Table 6.3 shows the number of failures varies with timeout period for retry in a single server and the result is plotted in 6.1. If the timeout period for retry is too short, the server does not reboot on time to provide the service again.

Table 6.4 shows the number of failures varies with polling frequency and Figure 6.2 shows the plotted graph. If the polling frequency is low, the replication manager cannot respond on time and

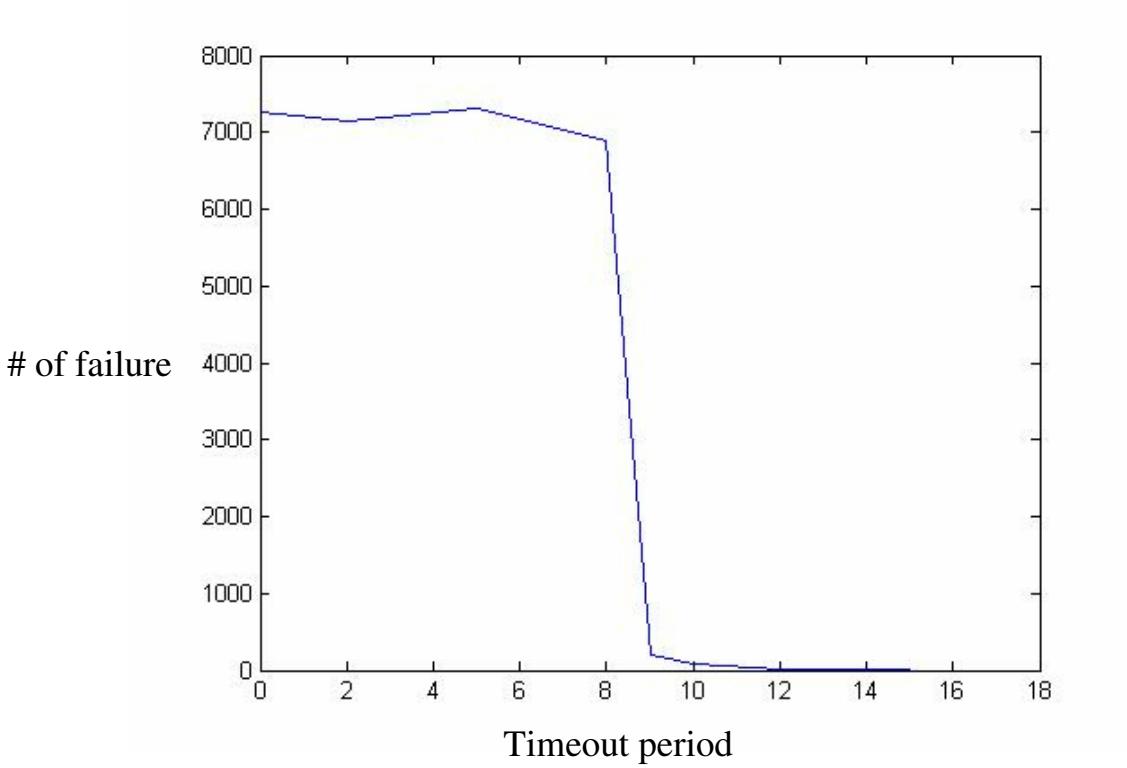

Figure 6.1: Number of failure with varying timeout period for retry in a single server

Table 6.4: Number of failure with varying polling frequency

Polling frequency (number of requests per mins)	Number of failure in temporary fault	Number of failure in permanent fault
0	0	712489
1	0	10911
2	0	3045
5	0	523
10	0	1
15	2218	2652
20	14746	12536

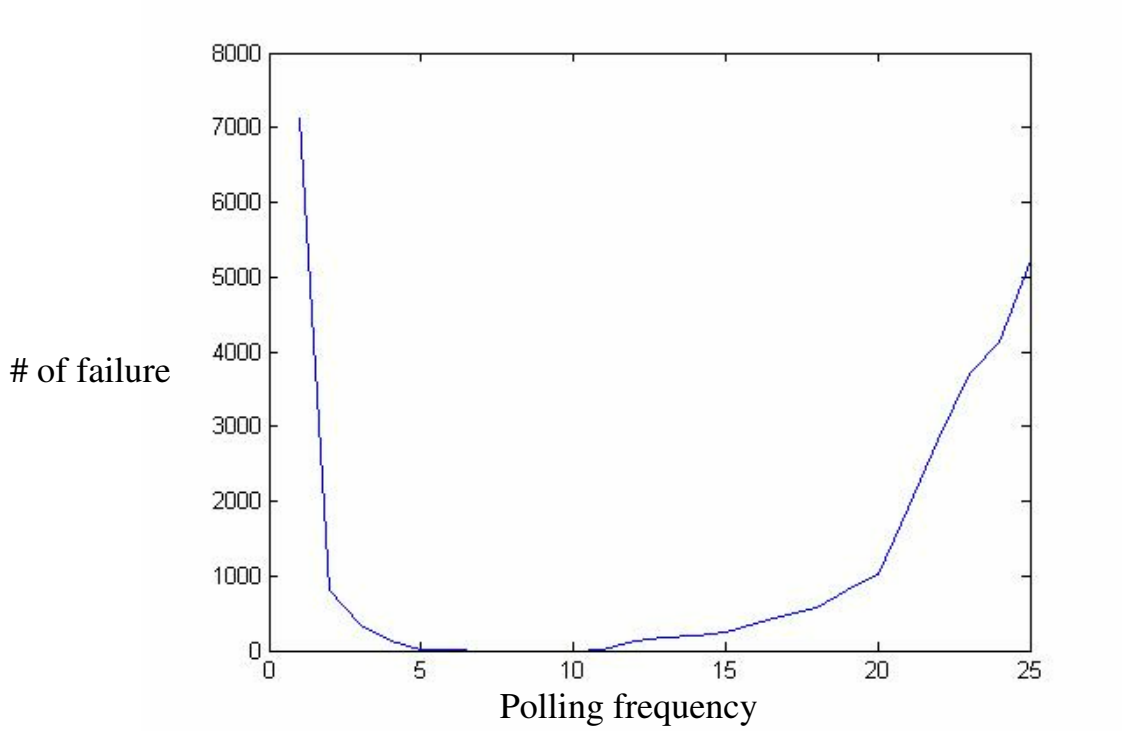

Figure 6.2: Number of failure with varying polling frequency

Table 6.5: Number of failure with varying number of replicas

Number of replicas	Number of failure in temporary fault	Number of failure in permanent fault
No replica	91	8152
2	2	356
3	0	0
4	0	0

Table 6.6: Number of failure with varying load of the server

Load of the server	Number of failure in temporary fault	Number of failure in permanent fault
70%	0	0
75%	0	0
80%	26	39
85%	104	141
90%	5128	5258
95%	12149	11258
99.9%	867525	886951

the request will still be sent to the failed server causing the failures in the system. When the polling frequency increases, the situation improves. The replication manager can respond on time and reduce the number of failures.

From Table 6.6, the optimal load of the Web server is 75%. If the load of the server is too high, the server is not able to handle the requests, which increases the failure rate of the system.

Also, we performed experiment on the number of failures changes with the number of replicas where the result is shown in Table 6.5. From the experiments, we found that three replicas are sufficient to

Table 6.7: Summary of the experiments

Experiment ID	1	2	3	4	5	6	7	8
Spatial replication	0	0	0	0	1	1	1	1
Reboot	0	0	1	1	0	0	1	1
Retry	0	1	0	1	0	1	0	1

reduce the number of failures nearly to zero.

6.3 Experiments for the Web Service Paradigm

A series of experiments are designed and performed for evaluating the reliability of the Web services. In the system, we apply retry, reboot and spatial replication with Round-robin, N-version Web services or recovery block. We perform the experiments with different combinations. Table 6.7 shows all the combinations of the experiments.

6.3.1 Round-robin

The Web servers work concurrently and a Round-robin algorithm [64] is employed for scheduling the work among the Web services. For different round, different Web service will be the primary server which provides the service. In our experiment, the Web service is replication on different machines as shown in Figure 6.3(a). In the experiment, five replicas are used for providing the service, which run on different machines with same configuration.

Table 6.8: Program metrics of the five versions of Web services

ID	Lines	Line without comment	Number of function	Complexity
01	2372	1982	47	87
02	2582	2033	26	45
03	3223	3029	78	124
04	5874	5275	80	124
05	4578	4187	47	113

6.3.2 N-version Programming Web Services

In this N-version Web service approach, five different versions of the Web service are employed. The five different Web services are implemented by different programming teams with different approaches and run in different machines with same configuration. The programming metrics of the five different versions are shown in Table 6.16.

As shown in Figure 6.3(b), the requests from the clients are forwarded to all versions of the Web services. The voting algorithm is run on the primary Web server which is selected by the replication manager. Once all the results in different versions are ready, the voting algorithm is applied to obtain the majority result and return the answer to the corresponding client.

6.3.3 Recovery Block

In the recovery block approach, different versions of the Web service are employed. In our experiment, five different versions of the Web services are used. The architecture is shown in Figure 6.3(c).

Checkpoints are defined in the Web services and the checkpoints are stored in the recovery cache.

The requests from the clients are sent to the primary Web services. The result will be examined by the acceptance test. If it passes the test, the result will be sent to the client. Otherwise, the alternative service will perform the service by rolling back to the checkpoint of the previous service.

6.4 Experimental Setup

In our experiments, we run a variety of Web services in the system to evaluate the reliability of the proposed fault tolerant techniques under different situations. Faults are injected in the system using fault injection techniques similar, for example, to those in [44, 51]. A number of faults may occur in the Web service environment [72]. The types of fault injected include permanent fault (the server is down permanently once this fault occurs), temporary fault (the fault only occurs randomly), Byzantine fault [14, 31] and network fault [43]. A Byzantine fault is an arbitrary fault that occurs during the execution of an algorithm by a Web service. In our experiment, several teams implement various versions of the Web service using a number of algorithms, in which faults are triggered. To generate a network fault, WS-FIT fault injection is applied. The fault injector decodes the SOAP message and can inject faults into individual RPC parameters, rather than randomly corrupting a message, for instance by bit-flipping.

Our experimental environment is defined by a set of parameters. Table 6.9 shows the parameters of the Web services in our experi-

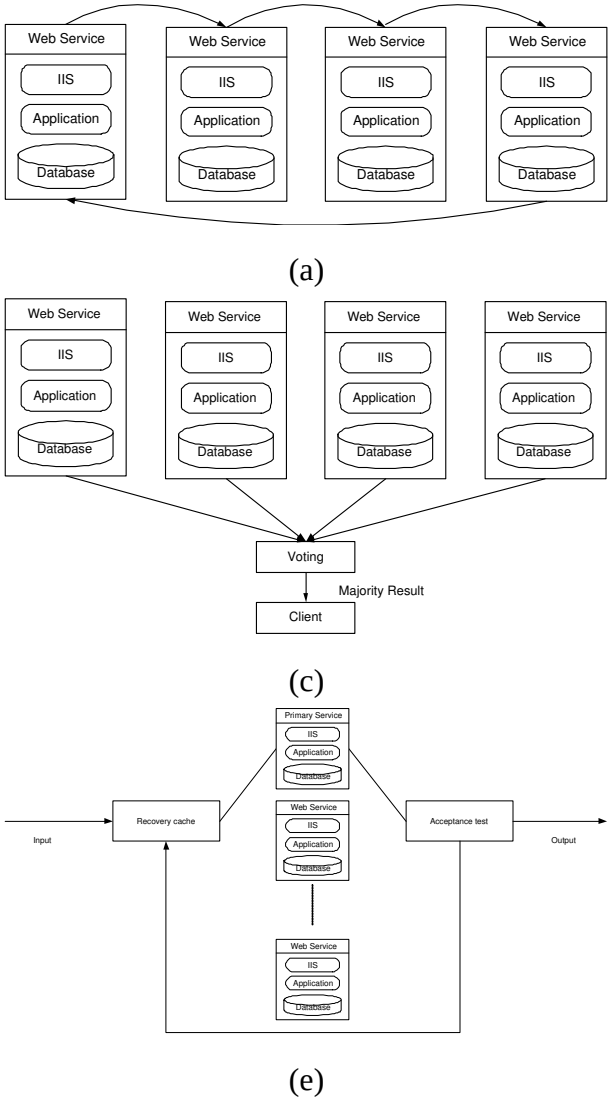

Figure 6.3: Summary of different approaches

Table 6.9: Parameters of the experiments

	Parameters	Current setting/metric
1	Request frequency	1 req/min
2	Polling frequency	10 per min
3	Number of replicas	5
4	Client timeout period for retry	10 mins
5	Max number of retries	5
6	Failure rate λ	number of failures/hour
7	Load (profile of the program)	78.5%
8	Reboot time	10 min
9	Failover time	1 s
10	Communication time to	
	Computational time ratio	10:1
11	Round-robin rate	1 s
12	Temporary fault probability	0.01
13	Permanent fault probability	0.001

ments.

For each of the approaches described in the pervious section, several experiments are performed. In each approach, we compare eight approaches, as shown in Table 6.7, for providing the Web services. The details of the experiments are as follows:

1. Single server without retry and reboot. The Web service is provided by a single server without any replication. No redundancy technique is applied to this Web service.

2. Single server with retry. The Web server provides the service and the client tries another Web service when there is no response from the original Web service after timeout.

3. Single server with reboot. The Web service provides the service and the Web server will reboot when there is no response from the Web service. Clients will not retry after timeout where there is no response from the service.

4. Single server with retry and reboot. The Web service provides the service and the Web server will reboot when there is no response from the Web service. Clients will retry after timeout when there is no response from the service.

5. Spatial replication with Round-robin / N-version / Recovery block. We use a generic spatial replication: For the first approach, the Web service is replicated on different machines and the requests are transferred to different Web services according to the scheduling of the workload by the replication manager. The replication manager carries out a failover in case of a failure. For the second approach, five different versions of the Web service are employed. The requests are sent to all versions and the majority answer is chosen by voting and sent back to the client. The client will only submit the request once and will not retry. For the third approach, different versions of Web service is run on different servers and the request is sent to the primary Web service and the result is pass to the acceptance. If the result is failed, the backup Web service will be take up the request and rollback to the checkpoint.

6. Spatial replication with Round-robin / N-version / Recovery block and retry. This is a hybrid approach which is based on the approach in experiment 5. However, the clients will retry

after timeout when there is no response from the service.

7. Spatial replication with Round-robin / N-version / Recovery block and reboot. This is similar to the experiment 5 where the Web service is run on different machines and the request is transferred to the server scheduled by the replication manager. In addition, the Web server will reboot when there is no response from the Web service.

8. Hybrid approach with spatial replication, retry and reboot. This is the fully hybrid approach. The Web service is run as described in experiment 5. The server will reboot when there is no response from the Web service, and the client will retry after timeout.

Our experimental system is implemented with Visual Studio .Net and runs with a .Net framework. The Web server is run on different machines and the Web service providing the service is chosen by the replication manager.

6.5 Experimental Results

The Web services were executed for 7 days for each experiment, generating a total of 10000 requests from the client. A single failure is counted when the system cannot reply to the client. For the approach with retry, a single failure is counted when a client retries five times and still cannot get the result. A summary of the results with the Round-robin algorithm, N-version programming and recovery block approach are shown in the following tables (Table 6.10 to Table 6.13).

Table 6.10: Experimental results without spatial redundancy

Experiments (number of failure / response time(s))	1	2	3	4
Normal case	5/186	3/192	2/190	3/187
Temporary	1025/190	4/223	1106/231	4/238
Permanent	8945/3000	8847/3000	1064/3000	5/1978
Byzantine failure	315/188	322/208	314/186	326/205
Network failure	223/187	2/227	239/193	3/231
Average	2102/730	1835/770	541/220	68/568

Table 6.10 to Table 6.13 show the improvement of the reliability of the system with our proposed paradigm. In the normal case, no failures are introduced into the system. For the other cases, we insert various kinds of faults into the systems.

When no redundancy techniques are applied on the Web service system (Exp 1), it is clearly seen that the average failure rate of the system is the highest. The results from the two different ways of improving reliability investigated here, i.e. spatial redundancy with replication and temporal redundancy with retry or reboot, are described below.

6.5.1 Single Server with Retry

When the system is under temporary fault and network fault, the experiment shows that the temporal redundancy helps to improve the reliability of the system. For the Web service with retry (Exp 2), the number of failures is reduced to zero. This shows that the temporal

Table 6.11: Experimental results with Round-robin

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	4/188	2/195	3/193	2/190
Temporary	1044/187	3/233	1057/188	2/231
Permanent	5637/3000	5532/3000	213/187	3/191
Byzantine failure	152/189	5/219	187/192	3/194
Network failure	237/193	3/213	206/197	2/192
Average	1415/751	1109/772	333/191	2/199

Table 6.12: Experimental results with N-version programming

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/189	0/190	0/188	0/188
Temporary	0/190	0/190	0/189	0/187
Permanent	3125/191	3418/192	197/189	0/191
Byzantine failure	0/190	0/191	0/190	0/188
Network failure	0/190	0/192	0/188	0/187
Average	625/190	683/191	40/189	0/188

Table 6.13: Experimental results with recovery block

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/191	0/189	0/193	0/188
Temporary	0/205	0/203	0/204	0/201
Permanent	3478/215	3245/208	201/211	0/201
Byzantine failure	0/194	0/198	0/195	0/194
Network failure	0/195	0/197	0/198	0/194
Average	696/200	649/199	40/200	0/198

redundancy with retry approach can significantly improve the reliability of the Web service. When a fault occurs in the Web service, on average, the clients need to retry twice to get the response from the Web service. However, the response time of the Web service is increased. Also, when there is a permanent fault, this scheme cannot reduce the number of failures in the system.

6.5.2 Single Server with Reboot

Another temporal redundancy approach is Web service with reboot (Exp 3). Our results show that the failure rate of the system is reduced when there is a permanent fault, in which case the server will try to reboot. Once the server finishes rebooting, it can provide the service again. The resulting failure rate is reduced from 85.3% to 14.0%. For temporary faults, the improvement is not as substantial as that of the temporal redundancy with retry. This is due to the fact that, when the Web service cannot be provided, the server will take

time to reboot.

6.5.3 Single Server with Retry and Reboot

With retry and reboot, the failure rate of both temporary and permanent cases are reduced. This approach enjoys the advantages of both algorithms. For temporary faults, the number of failures is reduced to zero. For permanent faults, the number of failures is significantly reduced from 85.3% to 1%; however, the response time is also greatly increased.

6.5.4 Spatial Replication with Round-robin

Refer to the Table 6.14, with the spatial replication approach in Exp 5 with Round-robin, the failure rate in the permanent fault is reduced from 85.3% to 78.3%. The failure rate is reduced because there are more servers in the system. When a server has failed, the replication manager will update the availability list and forward the requests to other servers. However, when all the servers have failed, the system will not be able to handle the requests from the clients. For the Byzantine failure, the failure rate is also reduced. This is because, with the Round-robin algorithm, different servers are employed for different requests, and so, the failure rate is reduced.

6.5.5 Spatial Replication with N-version Programming

Refer to the Table 6.12, with the spatial replication approach in the N-version programming of Exp 5, the failure rate of the Web service is greatly reduced. When a fault occurs in a Web service, other Web

services are still operating, from which the majority result will be selected and returned to the client. Thus, the fault of a Web service will be tolerated in the system. When permanent faults occur, the failure rate is reduced from 85.3% to 21.4% with this scheme. For Byzantine and network faults, the N-version approach can even reduce the failure rate to zero in our experiment. It is noted that in the N-version approach, the failure rate is much lower than that of the Round-robin approach. This shows the majority results are normally more reliable than the results produced by an individual version.

6.5.6 Spatial Replication with Recovery Block

Refer to the Table 6.13, with the spatial replication approach in the recovery block of Exp 5, the failure rate of reduced greatly. When the fault occurs in a Web service, it is detected by the acceptance test. Thus, the system will rollback to the checkpoint and perform the task again with another service. This approach successfully reduce the failure rate in most of the situations.

6.5.7 Spatial Replication, Retry or Reboot with Round-robin

In Exp 6 and 7, hybrid approaches with retry (Exp 6) or reboot (Exp 7) are conducted. We find that the failure rate is not much improved compared with that in Exp 4. However, the average response time of the Web service is reduced.

6.5.8 Spatial Replication, Retry or Reboot with N-version Web Service

In Exp 6 and 7, hybrid approaches with retry (Exp 6) or reboot (Exp 7) are conducted. We find that the failure rate is reduced to zero. It shows that when the N-version programming is applied to the Web service system together with the temporal replication techniques, the reliability of the system is improved a lot. This approach is efficient also. The average response time for the Web service is around 190 seconds, which is more or less the same as the normal case.

6.5.9 Spatial Replication, Retry or Reboot with Recovery Block

In Exp 6 and 7, hybrid approaches with retry (Exp 6) or reboot (Exp 7) are conducted. We find that the failure rate is reduced to zero. The reliability of the system is improved by the recovery block approach with temporal replication. However, the response time is a bit longer than the approach with N-version programming. This is because all the requests need to pass through the acceptance test before it returns to the client.

6.5.10 Spatial Replication with Round-robin / N-version / Recovery Block, Retry and Reboot

After performing the above experiments, we propose a hybrid approach for improving the reliability of Web services, including spatial redundancy, retry and reboot. The reliability of the system is improved most significantly by this approach: The failure rate of the system is reduced from 85.3% to 0 and the average response time is

shortened. The replication manager keeps checking the availability of the Web services. When there is a server fault, other servers are responsible for handling the requests. At the same time, the failed server will reboot. Thus, the response time for handling the requests is greatly reduced. In Exp 8, it is demonstrated that this approach results in the lowest failure rate. This demonstrates that combining spatial and temporal redundancy in a hybrid approach achieves the highest gain in reliability improvement of the Web service.

6.5.11 Comparing the Three Approaches

Table 6.14 shows the comparison between different approaches, including no spatial replication, Round-robin scheduling approach, N-version programming and recovery block. *X* stands for there are failures in that approach, *V* stands for there is no failure in that approach. It is clear that spatial replication helps to improve the reliability of the system. N-version programming and recovery block approach even achieve zero failure case when it is combined with the temporal replication.

The detailed performance and reliability comparisons are described in the following sections with the Petri-Net and Markov chain modelings.

6.6 Verification with Models

In the pervious chapter, we built the Petri-Net and Markov chain model for the proposed reliable Web services paradigm. We are now going to use the parameters obtained in the experiments to predict

Table 6.14: Comparing the three approaches

Failure	No spatial replication	Round-robin	N-version	Recovery block
Normal case	V	V	V	V
Temporary	X	V	V	V
Permanent	X	X	V	V
Byzantine failure	X	X	V	V
Network failure	X	X	V	V

the throughput and reliability of the system.

6.6.1 Petri-Net

According to the parameters obtained from the experiments, the performance of the system over time is obtained with the tool SHARPE and Figure 6.4 shows the result.

6.6.2 Markov Chain Model

Based on the experiments described in the pervious section, we obtain the fault rates and the repair rates of various components in the system; the results are shown in Table 6.15. The reliability of the system over time is further calculated with the tool SHARPE. Figure 6.6 shows the reliability over time at different fault rates λ^*; the repair rate is (set at) 0.421 faults/s. Note that the fault rate obtained from the experiments is 0.03 failure/s. This failure rate is measured under an accelerated testing environment.

Through the models developed, the characteristic of the system proposed is shown.

By considering the test compression factor [47], the failure rate of

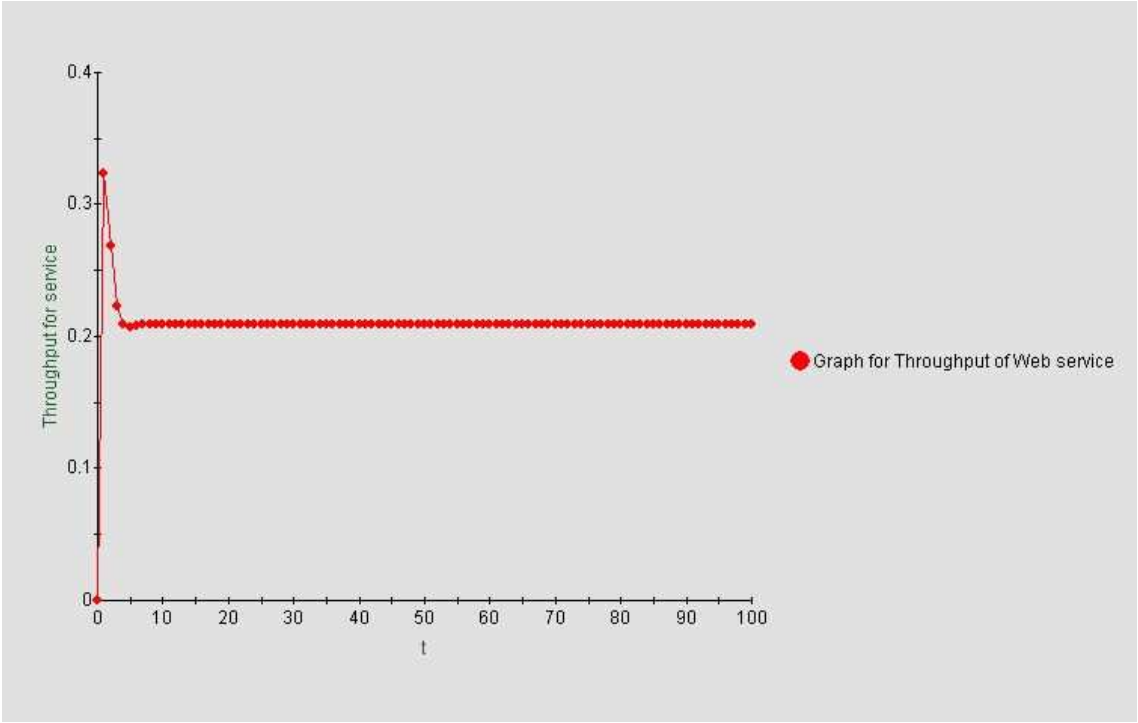

Figure 6.4: Throughput of the Web service

Table 6.15: Model parameters

ID	Description	Value
λ_N	Network fault rate	0.05
λ^*	Web service fault rate	0.03
λ_1	Temporary fault rate	0.01
λ_2	Permanent fault rate	0.001
μ^*	Web service repair rate	0.421
μ_1	Temporary fault repair rate	0.995
μ_2	Permanent fault repair rate	0.995
C_1	Probability that the RM response is on time	0.978
C_2	Probability that the server reboots successfully	0.978

the system in a real situation will be much less. A similar reliability curve is plotted with in Figure 6.5 with repair rate μ^* equal to 0.572 failure/s.

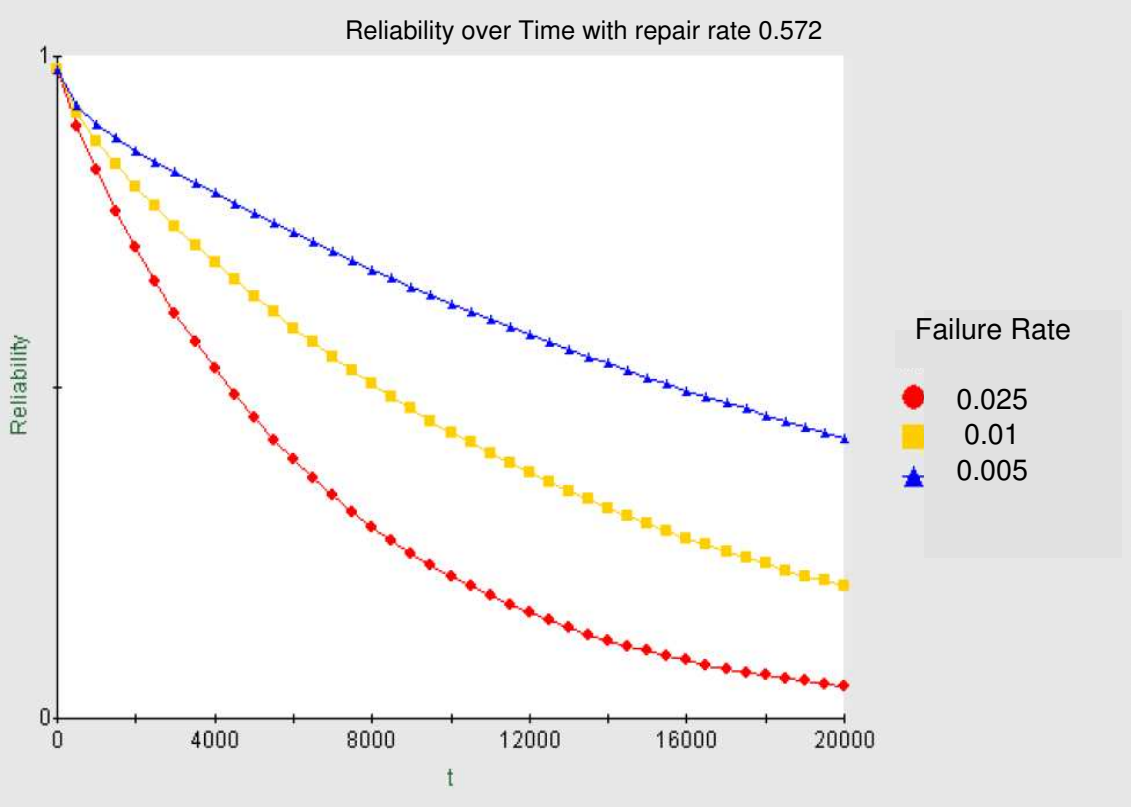

Figure 6.5: Reliability with different failure rate and repair rate is 0.572

Through the model, we confirm that the analysis result agreed with the experiment in the pervious section. Also, we demonstrates the properties of the Web service with models, including the reliability and performance.

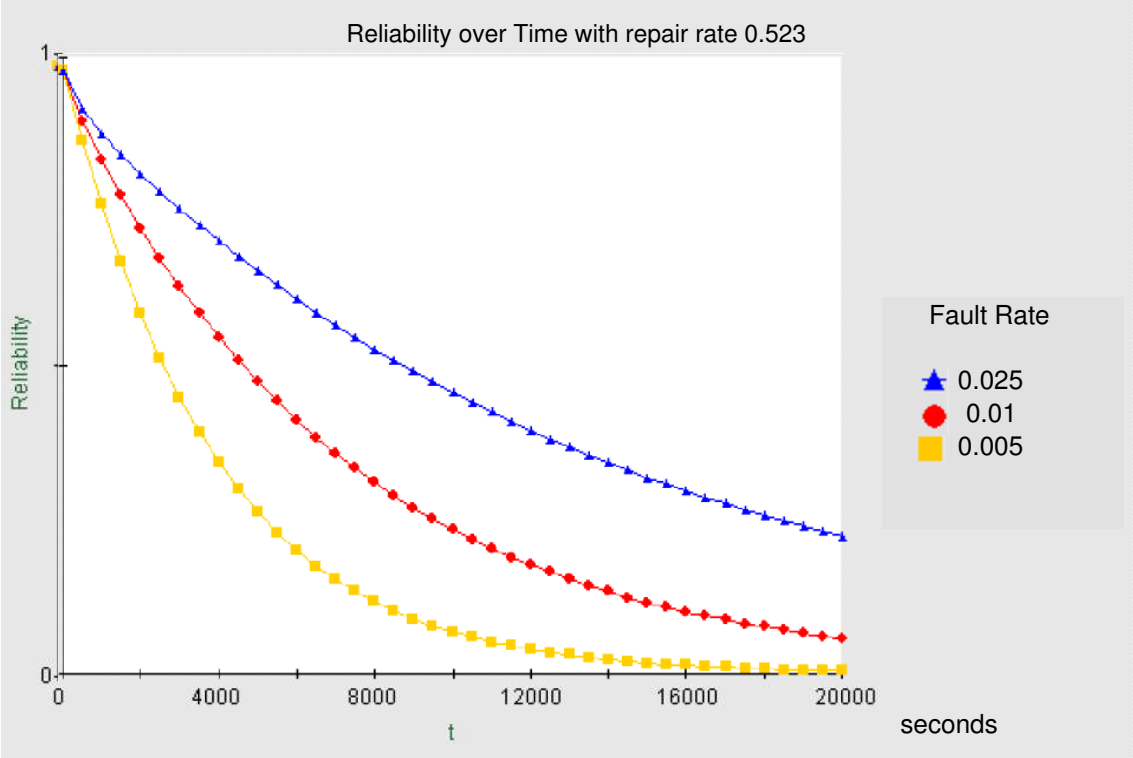

Figure 6.6: Reliability with different fault rates and repair rates

6.7 Experiments for the Web Service Composition Algorithm

In this section, we preform experiments to evaluate the properties, correctness and performance of the proposed Web service composition algorithm. We generate different versions of BRF with the Web service composition algorithm and evaluate with program metrics. Furthermore, we perform an acceptance test on the composed versions to evaluate the correctness of the algorithm.

6.7.1 Different Versions of Best Route Finding

We obtained several versions of BRF, which are implemented by different teams using different components. Also, the Web service components may differ between parties; thus, in this experiment, we try to compose the Web services from different versions with the WSDL and WSCI provided to create new versions.

According to the Web service composition algorithm described in Section 3, different versions of BRF are composed. The program metrics for 15 versions of BRF are shown in Table 6.16, where the first 11 versions are implemented by different teams and the rest are composed by the proposed algorithm. In the experiment, we recorded the following information of each versions: number of lines in the program, number of lines without comments, number of functions, complexity [49], composition time (the time for composing the version), and deadlock-free checking result.

Table 6.16: Program metrics of the 15 versions

ID	Lines	Line without comment	Number of function	Complexity	time for composition (s)	Deadlock free	Acceptance test
01	3452	3052	59	64	-	yes	pass
02	2372	1982	47	87	-	yes	pass
03	2582	2033	26	45	-	yes	pass
04	3223	3029	78	124	-	yes	pass
05	2358	2017	34	89	-	yes	pass
06	4478	3978	56	107	-	yes	pass
07	1452	1320	38	46	-	yes	pass
08	5874	5275	80	124	-	yes	pass
09	3581	3214	45	74	-	yes	pass
10	4578	4187	47	113	-	yes	pass
11	2364	2015	36	76	-	yes	pass
12	2987	2336	65	147	1.48	yes	pass
13	4512	3948	75	155	1.74	yes	pass
14	3698	3247	60	192	1.58	yes	pass
15	4185	3856	34	88	1.62	yes	pass

6.7.2 Verification with Petri-Net

Petri-Net of each version are constructed and verified with the SHAPRE tool. In each version, the Petri-Net is constructed and the results are shown in 6.16.

6.7.3 Acceptance Test

To test the correctness of the composed Web service, an acceptance test is prepared. Once the composition is finished, 100 test cases are run on the system. For the BRF, the acceptance test is designed as follows.

There are 100 tests cases. In each test, we provide a start point and a destination. The system will give out the best route with the transit points, total price and time. For each test case, we have the solution with which we compare the output from the system.

6.7.4 Experiments on the Proposed Reliable Paradigm

With the composed versions of the Web service, we perform a series of experiments on our proposed reliable Web service paradigm. We employed 10 versions Web service in our experiment.

Our experimental environment is defined by a set of parameters. Table 6.17 shows the parameters of the Web services in our experiments. We have modified some of the parameters and new experiments are performed.

Table 6.17: Parameters of the experiments

	Parameters	Current setting/metric
1	Request frequency	1 req/min
2	Polling frequency	10 per min
3	Number of versions	15
4	Client timeout period for retry	10 mins
5	Max number of retries	5
6	Failure rate λ	number of failures/hour
7	Load (profile of the program)	78.5%
8	Reboot time	10 min
9	Failover time	1 s
10	Communication time to	
	Computational time ratio	10:1
11	Round-robin rate	1 s
12	Temporary fault probability	0.01
13	Permanent fault probability	0.001

Table 6.18: Experimental results without spatial redundancy

Experiments (number of failure / response time(s))	1	2	3	4
Normal case	5/186	3/192	2/190	3/187
Temporary	1025/190	4/223	1106/231	4/238
Permanent	8945/3000	8847/3000	1064/3000	5/1978
Byzantine failure	315/188	322/208	314/186	326/205
Network failure	223/187	2/227	239/193	3/231
Average	2102/730	1835/770	541/220	68/568

6.7.5 Experimental Results

The Web services were executed for 7 days for each experiment, generating a total of 10000 requests from the client. A single failure is counted when the system cannot reply to the client. For the approach with retry, a single failure is counted when a client retries five times and still cannot get the result. A summary of the results with the Round-robin algorithm, N-version programming and recovery block are shown in Table 6.18 to Table 6.21.

Table 6.18 to Table 6.21 shows the improvement of the reliability of the system with our proposed paradigm. In the normal case, no failures are introduced into the system. For the other cases, we insert various kinds of faults into the systems.

6.7.6 Discussion

After composition, a BPEL for that version of Web service would be created and the corresponding Petri-Net is constructed for dead-

Table 6.19: Experimental results with Round-robin

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	5/216	3/225	3/224	1/220
Temporary	1114/215	2/281	1072/218	3/284
Permanent	5682/3000	5362/3000	222/217	3/224
Byzantine failure	142/219	6/259	177/222	2/224
Network failure	229/223	2/253	211/227	2/222
Average	1434/775	1075/804	328/222	2/235

Table 6.20: Experimental results with N-version programming

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/219	0/220	0/216	0/217
Temporary	0/221	0/222	0/219	0/216
Permanent	3136/221	3427/223	189/229	0/221
Byzantine failure	0/221	0/219	0/220	0/218
Network failure	0/220	0/222	0/218	0/217
Average	627/220	685/221	38/218	0/217

Table 6.21: Experimental results with recovery block

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/221	0/219	0/224	0/219
Temporary	0/235	0/231	0/237	0/231
Permanent	3473/241	3250/238	201/242	0/231
Byzantine failure	0/224	0/230	0/225	0/224
Network failure	0/225	0/226	0/228	0/224
Average	695/231	650/229	40/231	0/228

lock free-checking. The result is also shown in Table 6.16. With our proposed composition algorithm, the average Web service composition time for different versions is 1.605 seconds and all the versions are deadlock-free. Compare with the existing algorithms, it is 0.3 seconds faster and deadlock-free guaranteed. According to the acceptance test results, the correctness of our algorithm is good. All the test cases are passed. Moreover, another advantage of our algorithm is dynamic. Whenever there are new components, our algorithm can be applied to generate new version without rewriting the specification.

Also, from the experiment, the results agree with the results performed in the pervious section. The proposed paradigm improves the reliability of the system.

6.8 Summary

In this chapter, we describe the details of experiments on the proposed reliable Web service paradigm and the Web service composition algorithm. We perform a series of experiments to evaluate the reliability and performance of the Web services when the proposed paradigm is applied. Through the experiments, we verify that our Web service paradigm is reliable when both spatial and temporal replication is employed. Also, we propose to apply Round-robin scheduling algorithm, N-version programming and recovery block to our scheme. They further improve the reliability of the system. Also, we perform the correctness check on the proposed Web service composition algorithm. We generate different versions of the BRF system and perform the acceptance test and deadlock-free test with Petri-Net.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

In this thesis, we surveyed the applicability of replication and design diversity techniques for reliable Web services and proposed a hybrid approach for improving the reliability of Web services. Furthermore, a dynamic service composition algorithm was proposed to enhance the readiness of different versions of Web services available on the Web. The correctness of the proposed algorithm is verified through Petri-Nets. Also, we described a series of experiments to evaluate the performance and reliability of the proposed Web service system. From the experiments, we concluded that both temporal redundancy and spatial redundancy are important for reliability improvement of Web services, especially when applying the N-version programming techniques. Thus, modeling techniques by Markov chains provide further insights into reliability of Web service systems using the proposed fault tolerant mechanisms, which can be deployed in various industry applications.

7.1 Contributions

1. We performed a survey on the current state-of-the-art Web service techniques, fault tolerance technologies and Web service composition algorithm.

2. We proposed a reliable Web service paradigm. We describe the methods of dependability enhancement by redundancy in space and redundancy in time which improve both the reliability and performance of the system.

3. We proposed three approaches, Round-robin [64], N-version programming [45] and recovery block [64], to be intergraded with our system to increase the reliability of the system,

4. We proposed an algorithm for Web services composition. In the algorithm, WSCI [4] and BPEL [3] are employed to enable the compositability of the Web services. The composition algorithm are verified through Petri-Nets. The correctness and deadlock-free are checked in our proposed scheme. Also, the performance of the algorithm is better then the existing ones.

5. We performed a series of experiments. A series of experiments are designed and performed for evaluating the reliability of the Web service. Also, according to the Web service composition algorithm proposed, different versions of the experiment system are composed and the program metrics are measured. Thus, the correctness and the performance of the proposed algorithm is evaluated.

6. Finally, we developed reliability models of the proposed Web

service paradigm using Petri-Net [53] and Markov chains model [24]. The correctness, reliability and throughput of the proposed paradigm are evaluated with the models built.

7.2 Future Work

In this thesis, we developed a reliable Web service paradigm and a Web service composition algorithm. Through the experiments, we verified the correctness and reliability of the system. We can further enhance the system by including more Web service characteristics and perform more solid experiments.

Also, Web Services can be applied to many different areas, such as e-business, aerospace applications, teaching... etc. Different applications have different requirements, such as reliability, performance, correctness... etc. There are more researches need to be done to enable them to be realized in the Internet. In this thesis we only discuss the reliability issue, more work can be done on the varies areas.

□ **End of chapter.**

Bibliography

[1] http://www.ebpml.org/bpml.htm.

[2] http://www.ebxml.org/.

[3] A. Alves and e. al. Web services business process execution language version 2.0. In *http://www.oasis-open.org/committees/documents.php*, 2006.

[4] A. Arkin, S. Askary, S. Fordin, W. Jekeli, and e. al. *Web Service Choreography Interface (WSCI) 1.0*. W3C, http://www.w3.org/TR/wsci/, 2002.

[5] A. Avizienis. The methodology of n-version programming. In M. R. Lyu, editor, *Software Fault Tolerance*, pages 23–43, New York, 1995. Wiley.

[6] A. Avizienis and L. Chen. On the implementation of n-version programming for software fault-tolerance during program execution. In *Proc. of First International Computer Software and Applications Conference*, pages 149–155, 1977.

[7] A. Avizienis and J. Kelly. Fault tolerance by design diversity: Concepts and experiments. *IEEE Transactions on Computer*, pages 67–80, Aug 1984.

[8] F. Belli and J. Jedrzejowicz. Reliability modelling of fault tolerant programs. In *Proc. of Fifth IASTED International Conference V Reliability and Quality Control*, pages 53–56, Lugano, Switzerland, 1989.

[9] O. Berman and U. Kumar. Optimisation models for recovery block schemes. *European Journal of Operational Research*, 115(2):368–397, 1999.

[10] R. Bilorusets and A. Bosworth. Web services reliable messaging protocol ws-reliablemessaging. Technical report, EA, Microsoft, IBM and TIBCO Software, Mar 2004.

[11] P. G. Bishop. Software fault tolerance by design diversity. In M. R. Lyu, editor, *Software Fault Tolerance*, pages 211–227, New York, 1995. Wiley.

[12] X. Cai. *Coverage-Based Testing Strategies and Reliability Modeling for Fault-Tolerant Software Systems*. PhD thesis, The Chinese University of Hong Kong, Aug 2006.

[13] D. Caromel, A. Costanzo, D. Gannon, and A. Slominski. Asynchronous peer-to-peerweb services and firewalls. In *Proc. 19th IEEEInternational Conference Conference on Parallel and Distributed Symposium*, page 183, Apr 2005.

[14] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[15] P. Chan. *Best Route Finding specification*. http://www.cse.cuhk.edu.hk/pwchan/BRF.doc, 2006.

[16] P. Chan, M. Lyu, and M. Malek. Making services fault tolerant. In *Proc. of the 3rd International Service Availability Symposium*, volume 4328, pages 43–61, Helsinki, Finland, 15-16 May 2006. Springer.

[17] P. Chan, M. Lyu, and M. Malek. Reliable web services: Methodology, experiment and modeling. In *Proc. of IEEE International Conference on Web Services*, Salt Lake City, Utah, USA, 9-13 Jul 2007.

[18] Z. Chen, J. Ma, L. Song, and L. Lian. An efficient approach to web services discovery and composition when large scale services are available. In *Proc. of IEEE Asia-Pacific Conference on Services Computing (APSCC '06)*, pages 34–41, 2006.

[19] W. Chou, G. Wang, L. Li, and F. Liu. Web service enablement of communication services. In *Proc. of IEEE International Conference on Web Services (ICWS 2005)*, pages 393–400, Jul 2005.

[20] W.-L. Dong, H. Yu, and Y.-B. Zhang. Testing bpel-based web service composition using high-level petri nets. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06)*, pages 441–444, Oct. 2006.

[21] D. E. Eckhardt, Caglavan, Knight, Lee, McAllister, Vouk, and Kelly. An experimental evaluation of software redundancy as a strategy for improving reliability. *IEEE Transactions on Software Engineering*, 17(7):692–702, Jul 1991.

[22] S. ECMA-348. Web services description language (wsdl) for csta phase iii. In *Proc. 19th IEEEInternational Conference Conference on Parallel and Distributed Symposium*, page 2nd Edition, Jun 2004.

[23] A. Erradi and P. Maheshwari. A broker-based approach for improving web services reliability. In *Proc. of IEEE International Conference on Web Services*, volume 1, pages 355–362, 11-15 Jul 2005.

[24] K. Goseva-Popstojanova and K. Trivedi. Failure correlation in software reliability models. *IEEE Transactions on Reliability*, 49(1):37–48, Mar 2000.

[25] M. Haines. Web service as information systems innovation: A theoretical framework for web service technology adoption. In *Proc. of IEEE International Conference on Web Services (ICWS 2004)*, pages 11–16, Jul 2004.

[26] S. Jones. Toward an acceptable definition of service [service-oriented architecture]. *IEEE Transactions on Software*, 22(3):87–93, May-Jun 2005.

[27] B. Kim. Reliability analysis of real-time controllers with dual-modular temporal redundancy. In *Proc. of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 364–371, 13-15 Dec 1999.

[28] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion program-

ming. *IEEE Transactions on Software Engineering*, 12(1):96–109, Jan 1986.

[29] J. Kwon, K. Park, D. Lee, and S. Lee. Psr : Pre-computing solutions in rdbms for fastweb services composition search. In *Proc. of IEEE International Conference on Web Services, 2007. ICWS 2007.*, pages 808–815, 9-13 Jul. 2007.

[30] J. Lala and R. Harper. Architectural principles for safety-critical real-time applications. In *Proc of IEEE*, volume 82, pages 25–40, Jan 1994.

[31] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[32] J. C. Laprie, J. Arlat, C. Beounes, and K. Kanoun. Architectural issues in software fault tolerance. In M. R. Lyu, editor, *Software Fault Tolerance*, pages 47–80, New York, 1995. Wiley.

[33] J. C. Laprie, J. Arlat, C. Beounes, K. Kanoun, and C. Hourtolle. Hardware and software fault tolerance: definition and analysis of architectural solutions. In *Proc. of the 17th International Symposium on Fault-Tolerant Computing (FTCS-17)*, pages 116–121, Pittsburgh, PA, 1987.

[34] J. C. Laprie and K. Kanoun. Software reliability and system reliability. In M. R. Lyu, editor, *M. R. Lyu, editor, Handbook of Software Reliaiblity Engineering*, pages 27–69, New York, 1996. McGraw-Hills.

[35] F. Lecue, A. Delteil, and A. Leger. Applying abduction in semantic web service composition. In *Proc. of IEEE International Conference on Web Services, 2007. ICWS 2007.*, pages 94–101, 9-13 Jul. 2007.

[36] D. Leu, F. Bastani, and E. Leiss. The effect of statically and dynamically replicated components on system reliability. *IEEE Transactions on Reliability*, 39(2):209–216, 1990.

[37] F. Leymann. Web services flow language (wsfl 1.0). Technical report, Member IBM Academy of Technology, IBM Software Group, May 2001.

[38] D. Liang, C. Fang, and C. Chen. Ft-soap: A fault-tolerant web service. Technical report, Institute of Information Science, Academia Sinica, 2003.

[39] D. Liang, C. Fang, and S. Yuan. A fault-tolerant object service on corba. *Journal of Systems and Software*, 48:197–211, 1999.

[40] F. Liu, G. Wang, L. Li, and W. Chou. Web service for distributed communication systems. In *Proc. of IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 1030–1035, Jun 2006.

[41] P. Liu and M. Lewis. Uniform dynamic deployment of web and grid services. In *Proc. of IEEE International Conference on Web Services (ICWS 2007)*, pages 26–34, Jul 2007.

[42] N. Looker and M. Munro. Ws-ftm: A fault tolerance mechanism for web services. Technical report, University of Durham, 19 Mar 2005.

[43] N. Looker, M. Munro, and J. Xu. A comparison of network level fault injection with code insertion. In *Proc. of the 29th Annual International Computer Software and Applications Conference 2005*, volume 1, pages 479–484, 26-28 Jul 2005.

[44] N. Looker and J. Xu. Assessing the dependability of soap-rpc-based web services by fault injection. In *Proc. of the 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, pages 163–170, 2003.

[45] M. Lyu and A. Avizienis. Assuring design diversity in n-version software: A design paradigm for n-version programming. In H. Pham, editor, *Proc. of Fault-Tolerant Software Systems: Techniques and Applications*, IEEE Computer Society Press Technology Series, pages 45–54, 1992.

[46] M. Lyu and V. Mendiratta. Software fault tolerance in a clustered architecture: Techniques and reliability modeling. In *Proc. of 1999 IEEE Aerospace Conference*, volume 5, pages 141–150, Snowmass, Colorado, 6-13 Mar 1999.

[47] M. R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1995.

[48] M. R. Lyu, editor. *Software Fault Tolerance*. John Wiley and Sons Inc, Apr 1995.

[49] McCabe and J. Thomas. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, Jan 1976.

[50] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proc. of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pages 482–493, 2002.

[51] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service application. In *Proc. of IEEE Symposium on Reliable Distributed Systems*, Orlando, FL, Oct 2005.

[52] M. Mukarram and e. al. Introducing dynamic distributed coordination in web services for next generation service platform. In *Proc. of IEEE International Conference on Web Services (ICWS 2004)*, pages 296–305, Jul 2004.

[53] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[54] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *Proc. of International Conference on World Wide Web 2002*, 2002.

[55] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, 1(2):220–232, 1975.

[56] B. Randell. Fault tolerance in decentralized systems. In *Proc. the 4th International Symposium on Autonomous Decentralized Systems (ISADS '99)*, pages 174–179, Tokyo, Japan, 20-23 Mar 1999. IEEE Computer Society Press 1999.

[57] B. Randell and J. Xu. The evolution of the recovery block concept. In M. R. Lyu, editor, *Software Fault Tolerance*, pages 1–21, New York, 1995. Wiley.

[58] J. Rao and e. al. Logic-based web services compositions: from service description to process model. In *Proc. of IEEE International Conference on Web Services (ICWS 2004)*, pages 446–453, Jul 2004.

[59] W. Rao, A. Orailoglu, and R. Karri. Fault tolerant approaches to nanoelectronic programmable logic arrays. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DSN '07*, pages 216–224, 25-28 Jun. 2007.

[60] R. Riter. Modeling and testing a critical fault-tolerant multiprocess system. In *Proc. of the 25th International Symposium on Fault-Tolerant Computing*, pages 516–521, 1995.

[61] R. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems. An Example-BasedApproach Using the SHARPE Software Package*. Kluwer Academic Publishers, Boston/London/Dordrecht, 1996.

[62] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. In *Proc. of Workshop on Internet Server Performance 98*, Madison, WI, Jun 1998.

[63] K. Shen and M. Xie. On the increase of system reliability by parallel redundancy. *IEEE Transactions on Reliability*, 39(5):607–611, Dec 1990.

[64] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/AMC Transactions on Networking*, 4(3):375–385, Jun 1996.

[65] A. Shye, T. Moseley, V. J. Reddi, J. Blomstedt, and D. A. Connors. Using process-level redundancy to exploit multiple cores for transient fault tolerance. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DSN '07*, pages 297–306, 25-28 Jun. 2007.

[66] B. Srivastava and J. Koehler. Web service composition — current solutions and open problems. In *ICAPS 2003*, 2003.

[67] S. Thatte. Xlang: Web services for business process design. Technical report, Microsoft, Jun. 2001.

[68] P. Townend, P. Groth, N. Looker, and J. Xu. Ft-grid: A fault-tolerance system for e-science. In *Proc. of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, Sept 2005.

[69] W. Tsai, Z. Cao, Y. Chen, and R. Paul. Web services-based collaborative and cooperative computing. In *Proc. of Autonomous Decentralized Systems*, pages 552–556, 4-8 Apr 2005.

[70] P. Wang, J. Zhang, and Z. Chang. Fault tolerance of multiprocessor-structured control system by hardware and software reconfiguration. In *Proc. of International Conference*

on Mechatronics and Automation, 2007. ICMA 2007., pages 3745–3749, 5-8 Aug. 2007.

[71] Wikipedia. *BPEL*. http://en.wikipedia.org/wiki/BPEL.

[72] Y. Yan, Y. Liang, and X. Du. Controlling remote instruments using web services for online experiment systems. In *Proc. of IEEE International Conference on Web Services (ICWS) 2005*, 11-15 Jul 2005.

[73] T. Zhou, X. Zheng, and D. Chen. Personalized web service composition model for non-diploma distance learning. In *Proc. of 11th International Conference on Computer Supported Co-operative Work in Design, 2007. CSCWD 2007.*, pages 962–966, 26-28 Apr. 2007.