FAULT TOLERANCE AND RELIABILITY PATTERNS

By

Ingrid A. Buckley

A Thesis Submitted to the Faculty of the

College of Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, Florida

December 2008

FAULT TOLERANCE AND RELIABILITY PATTERNS

By

Ingrid A. Buckley

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. E.B. Fernandez, Department of Computer Science and Engineering, and has been approved by the members of the supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Eduardo B .Fernandez, Ph.D.
Thesis Advisor

Maria M. Larrondo Petrie, Ph.D.

Michael VanHilst, Ph.D.

Borko Furht, Ph.D.
Chair, Department of Computer Science and Engineering

Karl K. Stevens, Ph.D., P.E.
Dean, College of Engineering and Computer Science

Barry T. Rosson, Ph.D.
Dean, Graduate College

11-21-08
Date

ACKNOWLEDGEMENTS

I am grateful to my advisor Dr. Eduardo B. Fernandez and the committee members Maria M. Larrondo Petrie and Dr. Michael VanHilst and also the members of the Secure Systems Research Group for the feedback they provided in writing the patterns included in this manuscript. I wish to express deep appreciation to my parents, siblings and close friends for their support and encouragement throughout this masters program. I dedicate this degree to my mother, Monica Buckley for her continuous support and encouragement throughout the years; without which I would be lost, and so I am unconditionally appreciative.

I wish to express sincere appreciation to Dr. Borko Furht for his assistance and encouragement to pursue the Masters of Science in Computer Science at Florida Atlantic University. I am most grateful to God, for all my achievements and accomplishments, I am truly grateful to all those that provided me with assistance and support.

ABSTRACT

Author: Ingrid A. Buckley

Title: Fault Tolerance and Reliability Patterns

Institution: Florida Atlantic University

Dissertation Advisor: Dr. Eduardo B. Fernandez

Degree: Master of Science

Year: 2008

The need to achieve dependability in critical infrastructures has become indispensable for government and commercial enterprises. This need has become more necessary with the proliferation of malicious attacks on critical systems, such as healthcare, aerospace and airline applications. Additionally, due to the widespread use of web services in critical systems, the need to ensure their reliability is paramount. We believe that patterns can be used to achieve dependability. We conducted a survey of fault tolerance, reliability and web service products and patterns to better understand them. One objective of our survey is to evaluate the state of these patterns, and to investigate which standards are being used in products and their tool support. Our survey found that these patterns are insufficient, and many web services products do not use them. In light of this, we wrote some fault tolerance and web services reliability patterns and present an analysis of them.

TABLE OF CONTENTS

TABLES

FIGURES

1 INTRODUCTION

Dependability is that property of a system that allows one to rely on its service. Dependability is comprised of four cornerstone aspects, namely fault tolerance, reliability, safety, and availability. Dependability is of utter importance in critical systems today, the need to identify varying ways for achieving a dependable system is imperative. Reliability measures the success with which the system conforms to its specification. Fault Tolerance as it relates to systems, software, and hardware is the ability to remain operable in the presence of faults. Safety is the prevention of catastrophic effects on the environment or the users of the system. Availability is the ability of a system to perform its functions when needed.

Applications and web services that are used for example in the medical, aerospace and airline industries can have catastrophic consequences when they malfunction. These systems can become very complex for their designers to implement. This increases the probability of failure in the presence of logic and system errors. Therefore, it is important to improve the reliability and fault tolerance of critical systems. Patterns offer one solution to this problem, because they are designed to aid developers in applying dependable features to their design. We propose a solution to the problem of achieving

dependability in systems by categorizing and evaluating dependability patterns which can be used to build critical systems. Additionally, due to the widespread use of web services by enterprises, the need to ensure their reliability has become crucial. There are several standards that intend to govern how web services are designed and implemented. Such designs include protocols to which they must adhere. Web Services are an important area where reliability is increasingly needed; we analyze some related products and standards and provide two patterns.

One objective of our work is to evaluate dependability patterns with a view of identifying areas for which there are no appropriate patterns. We intend to provide recommendations and solutions to improve and create current dependable mechanisms in applications and web services.

The goals of this research are:

- To conduct a survey of dependability patterns, in particular reliability and fault tolerance patterns with a view of identifying areas that have not yet been covered or are insufficiently covered.
- To rewrite some dependability patterns in a more complete form. These include the Acknowledgement and Active Replication patterns.
- To conduct a survey of the existing reliability supports for Web Services products and tools.
- Write patterns that describe the following standards and clarify their use:
    - WS-Reliability

    - WS-Reliable Messaging
- Identify relationships between dependability patterns

The goals listed are important in the overall development and implementation of critical systems. Surveying current dependability patterns makes it easier to identify reliability and fault tolerance areas for which no patterns exist. It also allows for the enhancement of current patterns thus making them more effective and useful to designers and helps support their best practices. Our work broadens the cross section of dependability patterns scope by adding new patterns.

The reliability and fault tolerance patterns which are written in this research follow the POSA [Bus96] template and are similar to the style of the patterns written in [Sch06].

Chapter 2 provides necessary background information. Chapter 3 presents a survey of reliability and fault tolerance patterns. Chapter 4 presents a survey of web services products and tools. Chapter 5 illustrates a set of fault tolerance patterns. Chapter 6 illustrates two web service reliability patterns. Chapter 7 presents an analysis and summary of results for the patterns developed in chapters 5 and 6 respectively. Chapter 8 presents conclusions and highlights potential areas for future work.

2 BACKGROUND

This chapter gives a brief introduction to patterns and fault tolerance concepts.

2.1 Patterns

A *pattern* is an encapsulated solution to a recurrent problem that solves a specific problem in a given context and can be tailored to fit different situations. Patterns are very useful in the development of software systems. Patterns allow for reusability, maintainability, and a systematic approach to defining and implementing mechanisms. They also serve as a guideline for application developers and help support best practices.

2.2 Security Patterns

Security patterns describe mechanisms that can control treats. Security patterns join the extensive knowledge accumulated about security with the structure provided by patterns to provide guidelines for secure system construction and evaluation. Security has had a long trajectory, resulting in a variety of approaches to analyze security problems and to design security mechanisms. It is intrinsic to try to codify this expertise in the form of patterns [Fer07].

2.3 Dependability

The dependability of a system is its ability to deliver specified services to the end users so that they can justifiably rely on and trust the services provided by the system. Dependability has several attributes, including reliability, availability, maintainability, and safety, we discuss this in more detail in [Buc07].

2.4 Fault Tolerance

The basic definition of fault tolerance as it relates to systems, software, and hardware is being operational when some parts of the system are not functioning properly [Sar02]. Fault tolerance is an important and relevant area for critical systems and as such, patterns for fault tolerance are being written to increase the reliability, availability and safety of critical systems. Fault tolerance is particularly important in critical infrastructures such as water systems, electricity generation systems, airports, hospitals, transportation (traffic lights) and emergency services (fire brigade, police). Fault tolerance is a fundamental aspect of safety. It is important that when a fault occurs in these types of systems, serious malfunction and failure are minimal and do not propagate to other areas of the system, which could have catastrophic repercussion for those who depend on such systems.

A *fault* is a defective value in the state of a component or in the design of a system which is the manifestation of an error. An *error* is a defective value in an erroneous state of a system. A system failure occurs when there is a deviation from the system specification; a failure is the manifestation of an error.

A system that can detect, mask and recover from the effects of a fault and continue operating correctly is said to be *fault tolerant.*

There are five fault tolerance techniques that are used to confront faults and their consequences in a system [Sar02]. They are:

- Detection - Detecting the occurrence of an error.
- Diagnostic - Locating the unit or component where the error has occurred.
- Masking – Masking errors so as to prevent malfunctioning of the system if a fault occurs.
- Containment – Confining or delimiting the effects of the error.
- Recovery - Reconfiguring the system to remove the faulty unit and erasing the effects of the error.

A fault tolerant system may require different mechanisms that implement different fault tolerance techniques. Some of these mechanisms are:

Redundancy - The duplication of critical components in a system with the intention of increasing the reliability of the system. This mechanism is often applied to chemical, power, nuclear applications, and aerospace applications.

Diversity – Requires having several different implementations of software or hardware specifications, running in parallel to cope with errors or failures that could arise directly from a specific implementation or design.

Graceful degradation – This mechanism is essential in systems where, in the event of a failure, a system crash is highly unacceptable. Instead, some functionality should remain in the event of a failure. If the system's operating quality decreases, the decrease should be proportional to the severity of the failure.

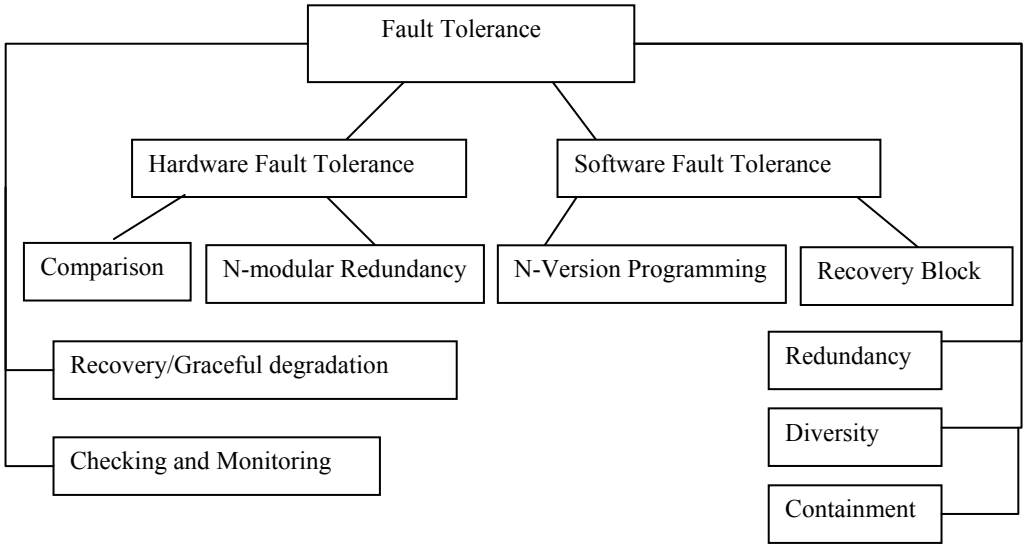
Checking and monitoring – Constant checking of the state of a system to ensure that specifications are being met is critical in detecting a fault. This mechanism, while very simple, plays a key role in achieving a fault tolerant system.

Containment – Faults are contained within some specific execution domain, which prevents error propagation across system boundaries.

N-modular Redundancy, N-Version programming, recovery block and comparison mechanisms will be discussed in detail in chapter 3.

These approaches can be divided into hardware and software based mechanisms. Figure 1 shows a classification of these mechanisms. The mechanisms not in the hardware or software branches apply to both hardware and software.

Figure 1. Fault Tolerance Mechanisms Tree



3 A SURVEY OF RELIABILITY AND FAULT TOLERANCE PATTERNS

This chapter presents a survey of fault tolerance patterns. Fault tolerance has been used over the last three decades to aid in applying fault tolerance to critical systems, the aerospace and medical industry has adopted them in the use of air traffic and control systems and networked infusion pumps respectively. Several persons have created a classification scheme of fault tolerant patterns with a view of discovering their relationship with each other [Sar02]. The notion is that these relations could be used to create design frameworks that could possibly achieve greater complexity and efficiency in critical systems. We examine here, different varieties of fault tolerance patterns appropriate for hardware and software, including their possible use and features.

3.1 Hardware Fault Tolerance

Hardware fault tolerance applies unit replication to enhance the system availability/reliability in the presence of hardware faults. Comparison and triple modular redundancy mechanisms [Avi85, Das07] are often used to handle hardware faults. We enumerate several possibilities below.

Dual Modular Redundancy (DMR)

The output of two or more components can be compared to ascertain the correct output [Avi85].

Triple Modular Redundancy (TMR)

This is a fault tolerant mechanism in which three systems perform the same functions and the results of each are processed by a voting system to produce a single output. If any one of the three systems fails, the other two systems can correct and mask the fault [Avi85]. This is often used in avionics and aerospace applications.

N-modular Redundancy (NMR)

This is a fault masking mechanism used to prevent the input and output of hardware errors in real-time systems, and transient faults are also masked by majority voting. This approach permits redundant systems to be robust with respect to failures in redundant processors [Avi84].

These mechanisms can be implemented in different ways. Several specialized mechanisms are used to support these two basic approaches.

A **Watchdog** is a specialized mechanism which primarily provides protection from time-based faults by creating an alarm whenever liveness messages are not received in a given time frame [Ljs06, Ham07]. This approach can also be used to improve deadlock

detection in a system [Ljs06]. Similar mechanisms include I am alive, Are you alive and Acknowledgment.

An **Acknowledgement** mechanism primarily detects crash failures and is based on acknowledging the reception of input within a given time interval [Sar02]. This mechanism is similar to the Watchdog mechanism because they both check and monitor a component based on a specified time interval; however, the watchdog uses a separate independent measure of time. The acknowledgement pattern has its own built in timer. Similar mechanisms include the Riding over transients and Leaky bucket counter.

A **fail Stop Processor** mechanism automatically halts in response to any internal failure and does so before the effects of that failure becomes visible in the system [Sch83]. This mechanism is based on replication and comparison of each replica's output for agreement [Sar02]. A similar mechanism, which evolved from the fail stop mechanism, is the Active Replication mechanism.

Roll Forward is a failure recovery mechanism which detects and recovers from a fault by monitoring two replicas for errors and masking errors when detected [Sar02]. One replica will process the input to the system and if no error occurs then the second replica will roll forward to the new state. The operation of rolling forward is based on the first replica exporting the new systems' state and the second replica importing it [Sar02]. This mechanism is similar to the comparison dual modular redundancy (DMR) mechanism.

The **Rollback** mechanism [Sar02] has similar functionality as the roll forward mechanism; however, one replica rolls back to the last error-free state if a failure occurs.

The **Input Guard** stops erroneous input from propagating an error inside a component by placing a guard at every access point of the component to check the validity of each input [Sar03]. This mechanism supports fault containment by stopping the propagation of an error from the outside to the inside of the guarded component. Its counterpart is the Output Guard.

The **Fault Container** provides the same benefits as the combination of the Input Guard and the Output Guard mechanisms. It prevents an error from being propagated inside and outside a given component, including preventing environmental errors from entering a component [Sar03].

3.2 Software Fault Tolerance

Software fault tolerance applies diversity to any phase of the software development life cycle to tolerate faults. We enumerate some approaches below:

N-Version Programming (NVP)

This is a method where multiple functionally equivalent programs are independently generated from the same initial specifications [Avi85].The concept is that the independence of programming efforts will greatly reduce the probability of identical software faults occurring in two or more versions of the program. NVP has been applied

to software systems that perform flight control computations on modern airliners, electronic voting, and train switching.

Recovery Block (RB)

This method uses a set of versions where one of them acts as the primary and applies an acceptance test to its results. Control exits from the recovery block if the execution result passes the acceptance test, otherwise the next alternate version is executed. This process is repeated until some version passes the acceptance test or all versions fail it [Kim91].

Reliable Hybrid

This mechanism provides a general object-oriented framework for fault tolerance which can range from approaches such as NVP or RB [Dan97].

Fault Injector

Provides a useful technique for evaluating the behavior of a system in the presence of faults [Lem01]. This technique produces or simulates faults during an execution of the system under test. The behavior of the system is then observed.

Monitor

This pattern is generally used with the Fault Injector pattern and used to capture the behavior of a system while it is being tested [Lem01].It can also be used to monitor the behavior of a system in production.

Riding Over Transients

This pattern detects transient errors and overload conditions in system applications. It provides increased successive escalation and recovery strategies depending on the severity of the error [Ada95].

Master-Slave

The Master-Slave design pattern provides accurate parallel computation to detect errors in a system [Bus96]. A master component distributes work to identical slave components and computes a final result from the results the slaves return.

Object Group

Provides a local surrogate for a group of objects distributed across networked machines [Maf96]. All objects within the group can view the state of each other, and each knows when an object joins or leaves the group. It enables easy implementation of replicated objects, for load sharing, and efficient multicast communications.

Protected Entry Points

Allows processes within a central or distributed system to use the services of another process or to collaborate in the computation of an algorithm thus allowing processes to share data and other resources [Fer08]. This pattern forces a call from one process to another to go through only pre-specified entry points where the correctness of the call is checked and other access restrictions may be applied.

Protection Rings

Prevents processes that may be malicious or contain errors from performing process execution in a centralized environment [Fer08]. This pattern applies only to centralized environments. It assigns each process to a set of hierarchical rings that control how processes call other processes and how they access data.

Multilevel Secure Partitions

Confines execution of a process in a centralized or distributed environment system partition which has been assigned to a specific confidentiality or integrity level [Fer08]. Access from one process to other partitions (processes or data) is restricted according to the rules of a multilevel security model, where processes have sensitivity levels.

3.3 Analysis of Fault Tolerance Patterns

Table 1 describes an analysis of dependability patterns that implement some of the fault tolerance mechanisms that we discussed in the previous sections.

Table 1 Legend:

Complete Pattern: Uses the template of POSA or GOF, includes UML diagrams that describe the structure and dynamics of a pattern.

Incomplete pattern – Does not completely conform to the POSA or GOF pattern templates.

Insufficient UML diagram – Does not include a class and, or sequence UML diagram to describe the pattern.

No UML diagram – No UML diagrams were given to structure or dynamic of the pattern.

Insufficient detail – limited detail and explanation given about the scope of the pattern.

Table 1. Analysis of Fault Tolerance

Fault Tolerance Mechanism	Pattern	Intent	Assessment
▪ Checking and Monitoring	Acknowledgement	Detects errors in a system by acknowledging the reception of an input within a given time interval [Sar02].	▪ Incomplete pattern ▪ Insufficient UML diagram
	I am Alive	Detects errors that lead to crash failures in a system by receiving regular time interval notifications from the system as a sign that no errors are present [Sar02].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Containment ▪ Checking and Monitoring	Input Guard	Stops erroneous inputs from propagating inside a guarded component by placing a guard at every access point of the component. The validity of the input is checked against the system's specifications [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Containment ▪ Checking and Monitoring	Output Guard	Confines an error inside the component where the error occurred by placing a guard at every exit point of the component. The guard checks each output against the system's specification [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Containment ▪ Checking and Monitoring	Fault Container	Provides a wrapper that embraces the system by preventing errors from entering or exiting the wrapper. The wrapping acts like a container which checks inputs and outputs against the specifications of the system [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Checking and Monitoring ▪ Redundancy ▪ Comparison	Fail-Stop Processor	Solves the problem of transforming Byzantine failures to fail-stop failures by comparing the output of system replicas for unanimity [Sar02].	▪ Incomplete pattern ▪ Insufficient UML diagram

▪ Redundancy ▪ Error masking ▪ Containment	Passive Replication	Masks a detectable error in a system by having two replicas a primary and a backup. The backup is activated when the primary fails [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Redundancy ▪ Error masking ▪ Containment	Semi-Passive Replication	Masks a detectable error in a system by having two replicas a primary and a backup. The backup imports the state of the primary when the backup requests a checkpoint [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Redundancy ▪ Error masking ▪ Containment ▪ Comparison	Semi-Active Replication	Masks the occurrence of a detectable error in a system by using a group of two replicas to process input independently in parallel. The primary replica will deliver the output to the system [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Redundancy ▪ Error masking ▪ Containment ▪ Comparison	Active Replication	Masks the occurrence of a detectable error in a system by having a group of system replicas. The replicas receive an input and they all deliver an output which is compared by majority voting for the correct output [Sar03].	▪ Incomplete pattern ▪ Insufficient UML diagram
▪ Comparison ▪ Redundancy ▪ Containment Checking and monitoring ▪ Recovery Block	Roll Forward	Recovers from errors in a system by using two replicas in the system. One replica will process an input and if no errors occur the second replica will roll forward to that new error free state [Sar02].	▪ Incomplete pattern ▪ Insufficient UML diagram.
▪ TMR ▪ Comparison ▪ Redundancy ▪ Error Masking ▪ Containment	Triple Modular Redundancy	Provides protection against single point failures by replicating the channel three times and running all three in parallel. If one channel breaks, then the other two will come to an agree [Tic06]	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given
▪ N-version ▪ Recovery Block	Reliable hybrid	Provides a general object-oriented framework for fault-tolerance which can range from basic approaches (e.g. NVP and RB) to complex hybrids (CRB, AV, NSCP) [Dan97].	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given

▪ N-version	Master-Slave design	Supports fault tolerance parallel computation and computational accuracy. A master component distributes work to identical slave components and computes a final result from the results these slaves return [Dan97].	▪ No UML diagram. ▪ Insufficient detail given
▪ Error masking ▪ Containment ▪ N-version	Homogeneous Redundancy	Increases reliability by using identical redundant channels to simultaneously perform the same computation to improve reliability of the system against random faults [Dou07].	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given
▪ Error masking ▪ Containment ▪ N-version	Diverse Redundancy	Provides the benefits of the homogeneous redundancy pattern but also protects against systematic errors, such as software errors. It does this by providing multiple channels which are identical in semantics and interface but differ in their implementation [Dou07].	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given
▪ Comparison ▪ Redundancy ▪ Checking and monitoring	Fault Injector	It provides a useful technique for evaluating the behavior of a system in the presence of faults [Lem01]. The technique tries to produce or simulate faults during an execution of the system under test, the behavior of the system is then observed [Lem01].	▪ No UML diagram
▪ Comparison ▪ Redundancy ▪ Checking and monitoring	Monitor	Documents the behavior of a system while it is being tested to reveal possible errors [Lem01].	▪ No UML diagram
▪ Comparison ▪ Checking and monitoring	Riding Over transients	Detects transient errors and overload conditions in applications and provides successive escalation steps and recovery strategies [Ada95].	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given
▪ Redundancy ▪ Error masking ▪ Containment ▪ Comparison ▪ Checking and monitoring	Master-Slave	Provides accurate parallel computation to detect errors in a system [Bus96].	▪ Incomplete pattern ▪ No UML diagram. ▪ Insufficient detail given

▪ Redundancy ▪ Error masking ▪ Comparison ▪ Checking and monitoring	Object Group	Provides a local surrogate for a group of objects distributed across networked machines [Maf96].	▪ No UML diagram. ▪ Insufficient detail given
▪ Comparison ▪ Redundancy ▪ Checking and monitoring	Protected Entry Points	Forces a call from a process to another to go through only pre-specified entry points where the correctness of the call is checked and other access restrictions may be applied. [Fer08].	▪ Complete Pattern
▪ Comparison ▪ Redundancy ▪ Checking and monitoring	Protection Rings	Assign processes to a set of hierarchical rings that control how processes call other processes and how they access data. Crossing of rings is done through *gates* that check the rights of the crossing process [Fer08].	▪ Complete Pattern
▪ Comparison ▪ Redundancy ▪ Checking and monitoring	Multilevel Secure Partitions	Confines execution of a process in a system partition which has been assigned to a specific confidentiality or integrity level [Fer08].	▪ Complete Pattern

We classified most of the patterns in table 1 as incomplete because they did not conform to the POSA [Bus96] or GOF [Gam94] pattern templates. Some of the patterns surveyed did not explicitly give the pattern's intent, and did not state clearly the problem the pattern intended to solve. In others, no clear solution was provided. Additionally, some of the patterns lacked detail and no UML diagrams were used to describe their solutions (in particular class or sequential diagrams).

3.4 Summary

Due to the importance of fault tolerance in systems today, particularly in critical systems, our analysis has shown that many fault tolerance patterns that have been written are incomplete and lack detail. In particular, many of the patterns illustrated in table 1 do not

conform to the POSA [Bus96] or GOF [Gam94] pattern templates. We believe that the use of UML diagrams is necessary to describe patterns, especially their structure and dynamics, to guide a designer trying to apply its solution in an actual system design.

The analysis of these patterns showed that we need more complete fault tolerance patterns before they can be useful to a designer, and that we need to rewrite many of the existing patterns to make them more concise, complete and useful. We intend to increase the usage of fault tolerance patterns in critical systems, by making them more complete in their description and representation with the use of UML diagrams and by using the pattern templates outlined by POSA [Bus96] and or GOF [Gam94]. Adding more detail to the patterns will make them more understandable to system designers, and may also increase the use of them in critical systems.

4 SURVEY OF WEB SERVICES PRODUCTS AND TOOLS

A web service is defined by the W3C 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. Web services communicate using XML messages that typically follow the SOAP standard.

Due to the proliferation of business and services over the internet, web services are playing a greater role in the evolution of the web. They offer a dynamic means of rendering service to customers on a larger platform which is the internet. Due to the nature of the internet there exist a great challenge for web service developers to ensure that they implement web services that are secure, and have a high level of availability and reliability. As a result, there are a large number of products and tools that are now available on the market, which were designed with the objective of giving greater levels of functionality whilst conforming to industry standards and specifications.

Web services are generally used in two ways, for remote procedure calls (RPC), and document style. Several products are available on the market that offer one or more of

these functionalities. In this chapter we evaluate some of the products and development tools available to create web services including their capabilities; with a view of identifying areas that either have no support or which can be better enhanced to increase the overall efficiency of the products and tools used in the development of web services.

4.1 Web Services Products

We enumerate a list of web service products that are currently available on the market, which provide cutting edge capabilities and key features that are required when implementing a reliable web service.

Actional - Actional XMS

This XML Firewall provides access control to web services, while filtering SOAP messages based on their content [Act02]. Some supported standards include XML Signatures, SAML, WS-Security 1.0, SOAP and WSDL.

BEA - BEA WebLogic Enterprise Security

WebLogic Enterprise Security provides access control to applications based on policies [Bea98]. It includes policy-based delegation management, authentication with single sign-on, consolidated auditing, and dynamic-role and policy-based authorization with delegation. It supports SAML and WSDL 1.1 standards.

Cerebit – InnerGuard

InnerGuard provides access control to applications (including web services), based on policies [Cer04]. Its features include policy management and enforcement, identity management, privacy, provisioning and PKI capabilities [Cer04]. It conforms to the SOAP standard.

DataPower XS40 XML Security Gateway and XS40 XML Firewall

DataPower provides an XML Firewall that parses, filters, validates schemas, decrypts, verifies signatures, transforms, signs and encrypts XML message flows [Ibm04b]. It includes an XML Security Gateway that can be used to control access to web services [Ibm04b]. DataPower supports WS-Security, SOAP 1.2 and the SAML standards.

Digital Evolution XML VPN (acquired Flamenco networks)

Allows service consumers to securely connect to authorized web services [Soa04]. These web services expose consumers to applications as local services. It provides central policy, rights management and transaction auditing services. It supports the WS-security standard.

Entrust Secure Transaction Platform

Uses a set of security services that provide security capabilities that enable secure transactions [Ent03]. These services provide a foundation for integrating authentication, authorization, digital signatures, and encryption into transactions. These trust services are provided through web services interfaces to allow for integration and deployment. It

supports SAML, XACML, XML Digital Signatures, XML Encryption, XKMS, and WS-Security.

Forum XWall Web Services Firewall

Forum XWall is a web services firewall equipped with data authentication as well as XML intrusion prevention to protect against XML viruses, data corruption and denial of web service attacks [For04a]. It offers data level authentication, XML intrusion prevention and interoperability enforcement that protects enterprises against XML viruses, denial of web service attacks and unauthorized data access. It supports WS-I Basic Profile, XML 1.0, SOAP 1.1/1.2 and WSDL standards.

Forum Sentry Web Services Security Suite

Forum Sentry enables trusted information sharing using XML data and web services across different security domains and business processes [For04a].It supports XML Digital Signature, XML Encryption, WS-Encryption, WS-Digital Signatures, WSDL 1.1/1.2, WS-Security, SAML, XKMS and WS-I Basic Profile standards.

Forum Vulcon Vulnerability Containment Service

Forum Vulcon is an early warning system for known and impending XML related vulnerabilities and countermeasures [For04b]. Forum Vulcon is an automated online service that delivers threat intelligence reports and antivirus updates, software and policy revisions and recommendations to help enhance a system's defense [For04b]. It conforms to XML, SOAP and WSDL standards.

Forum Presidio OpenPGP Security Gateway

Forum Presidio is a content exchange platform that allows enterprises to comply with government information and privacy regulations without complexity with the use of the ubiquitous OpenPGP standard [For03]. Forum Presidio can be used as a legacy-to-XML security bridge for a smooth migration to XML web services.

GXS Technology

GSX technology focuses its usage of web services on the SOAP and WSDL standards [Gxs08]. The current interfaces complement one or more additional protocols used for bulk data transfer [Gxs08]. Current development work involves the use of WS-Security for the GXS Trading Grid Business Services APIs, as well as SAML.

IBM Tivoli Identity Manager and Tivoli Access Manager

Tivoli Identity Manager is a policy-based user management system [Ibm07a]. While Tivoli Access Manager is a policy based access control system [Ibm07a].They provide authentication and authorization APIs that allow integration with application platforms such as J2EE [Ibm07a]. It supports WS-Federation and SAML standards.

IONA Artix

Artix is an extensible Enterprise Service Bus (ESB). It enables an enterprise to integrate and expose its applications as web services [Ion07]. Its security features include a role based access control mechanism, authentication, support of WS-Security, SSO, LDAP

plug-in, Active Directory Plug-In and SAML.

MapPoint Web Service

MapPoint is a programmable web service hosted by Microsoft and used by enterprises and independent software developers [Mic08d]. It can be used to integrate location based services such as maps, driving directions and proximity searches into software applications and business processes [Mic08d]. It uses SOAP and XML standards.

Microsoft Trust Bridge

Allows different organizations using the Windows operating system to exchange user identities and interoperate in heterogeneous environments [Mic08c]. It uses industry standard XML Web services protocols that include Kerberos, WS-Security and forthcoming protocols in the WS-Security family.

Netegrity TransactionMinder

TransactionMinder provides centralized authentication, policy-based authorization, and audit for web services transactions [Net02]. By intercepting requests made to web services, it analyzes them and communicates with its Policy Server [Net02]. Netegrity conforms to the SOAP, WSDL, SAML and XML Digital Signature standards.

ORACLE SOA Suites

Oracle Web Services Manager provides a means for governing the interactions with shared services through security and operational policy management and enforcement to

ensure service reuse remains under control [Ora07].

Oracle Management Pack for SOA provides management service for applications and infrastructure in a service-oriented architecture (SOA). Administrators are able to associate events and activities for all components across the SOA environment. This aids in resolving performance and availability issues [Ora07]. It includes a set of services and system level dashboards that can be used by administrators to view service levels for key business processes, and SOA infrastructure components from a central location.

Oracle Service Registry provides a standard base interface for SOA runtime infrastructure that dynamically discovers and binds to deployed service end points [Ora07]. It helps to bridge the gap between the design time and runtime environments through automated synchronization with Oracle Enterprise Repository and Oracle SOA Suite.

Oracle Enterprise Repository provides a base for delivering governance throughout the service-oriented architecture (SOA) lifecycle. It acts as the single source for information surrounding SOA assets and their dependencies [Ora07]. It provides a communication channel for the automated exchange of metadata and service information between service consumers, providers, policy decision points, and additional governance tooling.

Progress Actional for Active SOA Policy Enforcement

Separates the SOA policy lifecycle from the service lifecycle to centralize the creation and management of SOA policies [Pro07]. It provides security and compliance while ensuring distributed SOA policy enforcement. This allows security and compliance experts to author policies once, then apply them consistently across the SOA. This helps to guarantee complete coverage for reduced risk [Pro07]. It conforms to the for WS-Security standard.

Reactivity Gateway, Reactivity Manager and Reactivity Gatekeeper

The Reactivity Gateway enforces XML web services security policies [Cis05]. The Reactivity Manager is used to define policies. It conforms to SAML 2.0, XML Digital Signature, and WS-Security 1.0 standards.

Rohati's Transaction Networking System (TNS)

Transaction networking system platform is a network-based entitlement control (NBEC) system; that extends control across an enterprise's applications and resources [Roh08]. It enables enterprises to authenticate sessions and authorize each transaction based on business context [Roh08]. It supports the XACML standard.

Sarvega - XML Guardian Gateway

XML Guardian Gateway filters incoming and outgoing XML or SOAP messages based on the security policies [Sar04]. It conforms to XML encryption, XML signature standards and XPath.

Securent Entitlement Management

Entitlement Management provides deployment, management, and auditing application security [Sec07]. It separates fine-grained authorization policy from core application logic and delivers it as a XACML standard based service [Sec07].

Sun Microsystems Sun Identity Management

Sun Identity Management streamlines the process of managing user identities across a variety of applications [Sun08c]. It provides provisioning and secure access, to ensure ongoing compliance and enable federation for sharing beyond boundaries [Sun08c]. It supports Service Provisioning Markup Language (SPML).

Vordel XML Gateway

Uses a network device for offloading processor intensive tasks from applications running in general purpose application servers [Vor06]. The Vordel XML Gateway performs application networking by routing traffic based on content, based on sender, and performing XML content screening [Vor06]. It supports SSL, WS-Security, WS-Trust XML, SOAP, and SAML 2.0[Vor06].

Vordel XML Firewall

It shields XML applications from malicious attacks and allows them to be deployed in safety and confidence [Vor07]. The XML Firewall forms an integral component of an enterprise's SOA security infrastructure and can be deployed as part of a strategic architecture of XML firewalls, gateways and run time governance products [Vor07]. It

complements other application security and network security products by providing the XML data screening. It supports SOAP and XML Schemas and WS-I Basic Profile.

Xtradyne's WS-DBC

The Web Services Domain Boundary Controller (WS-DBC) is an XML Firewall. It provides protection against malformed messages and malicious content, encryption/decryption of XML messages, XML digital signatures, as well as provide authentication, authorization, and auditing [Xtr05]. It supports WS-Security, SAML WSDL and XML Digital Signature.

Table 2 describes some security features and standards used in the web services security products discussed previously.

Table 2. Web Services Products Security Features

Functionalities or Standards	IBM	IONA	BEA	Cerebit	DataPower	Netegrity	Vordel	Reactivity	Actional	Sarvega	Xtradyne	Forum Systems	Digital Evolution
XML schema validation					X		X	X	X	X	X		
XML Encryption					X			X	X	X	X		X
XML Signature					X			X	X	X	X		X
WS-Security		X			X						X		X
Web Services access control	X	X	X	X		X	X	X		X	X	X	X
User Authentication	X	X	X	X	X	X	X	X	X	X	X	X	X
Audit			X	X		X	X	X	X		X		
Alert				X			X		X				
Content Inspection							X	X	X		X	X	
Conformance validation									X				X
Integrity checks							X						

Functionalities or Standards	Forum Sentry	Forum Vulcon	Forum Presidio	Vordel Gateway	Vordel Firewall	Actional Enforcement	Entrust	Forum Xwall	Sun Identity Management	GXS Technology	Trust Bridge
XML schema validation						X					X
XML Encryption	X						X				X

XML Signature	X						X				X
WS-Security	X			X		X	X			X	X
Web services access control	X		X	X	X		X	X	X	X	X
User Authentication	X		X	X	X	X	X	X	X		
Audit		X	X	X	X						
Alert		X									
Content Inspection		X						X			
Conformance validation			X		X		X			X	
Integrity checks	X							X			

4.2 Web Services Development Tools

There are several commercial and open source tools which are now available that enable designers to create web services. These tools leverage and incorporate cutting edge technologies that include implementation of many web services standards. Web services development tools arm programmers with the control needed to create powerful web services while saving them time. They include key features that can be combined transparently with other architectures, environments and methodologies. We enumerate some of these tools below:

Eclipse Web Tools Platform (WTP)

Eclipse web tools platform extends the Eclipse platform with tools for developing web and Java EE applications [Ecl08]. It includes source and graphical editors for a variety of languages, wizards and built in applications, tools and APIs to support deploying, running, and testing web applications.

GlassFish

GlassFish is an open source application server which implements some new features in the Java EE 5 platform [Sun08b]. The Java EE 5 platform includes the latest versions of technologies such as JavaServer Pages (JSP) 2.1, JavaServer Faces(JSF) 1.2, Servlet 2.4, Enterprise JavaBeans 3.0, Java API for web services(JAX-WS) 2.0, Java Architecture for XML Binding(JAXB) 2.0 and web services metadata for the Java Platform 1.0.

MapForce

Altova MapForce 2008 Professional Edition is a graphical data mapping, conversion, and integration tool. The data mapping tool maps between combinations of XML, databases, flat file, EDI, and/or web services [Alt08]. It transforms data or auto generates a royalty free data integration application for execution of recurrent conversions. MapForce features include XML editing, validation and integration into Microsoft Visual Studio .NET C++/C#/Java/XLT.

Metro

Web services can be developed using java technology APIs and tools provided by an integrated web services stack called Metro [Sun02]. The Metro stack consists of JAX-WS, JAXB, and WSIT, which enable the designer to create and deploy secure, reliable, transactional, interoperable web services and clients. The Metro stack is a part of project Metro and is a part of GlassFish, java platform, enterprise edition (Java EE), and partially in java platform standard edition [Sun08a]. Metro enterprise features fall into four main categories: messaging, metadata, security, and quality-of-service (QoS).

MissionKit

MissionKit for XML Developers provides XML data integration, and style sheet design capabilities [Alt08]. It includes XML, XSD, XSLT, XQuery, semantic web development, WSDL and SOAP web services development.

Stylus Studio

Stylus Studio® 2008 XML Enterprise Suite provides a set of XML tools and features for working with XML, XQuery, web services, XML publishing, and other XML technologies [Sty08]. Stylus Studio includes the following features:

- **Apache Axis** is an implementation of the W3C SOAP standard [Sty08]. Stylus Studio uses Apache Axis to query and retrieve data through web services; it also generates code for web services. Additionally, with the support of the XML converters, web services can be built into applications, called and executed through XSLT and/or XQuery, and used in XML pipelines and reports [Sty08].
- **Integrating Web Services using XQuery** provides a flexible means for data abstraction.
- **Web Service Data Mapping**

  Stylus Studio allows the use of web services as XML data sources to conduct live XML mapping projects [Sty08].

XMLSpy

XMLSpy 2008 is an XML editor and development environment for modeling, editing, transforming, and debugging XML related technologies [Alt08]. It includes an XML

editor, a graphical schema designer, code generator, file converters, debuggers, profilers, and database integration [Alt08]. It supports XSLT, XPath, XQuery, WSDL, SOAP, and Visual Studio .NET and Eclipse plug-ins.

Web Matrix

Web Matrix is a WYSIWYG application development tool for ASP.NET. A WYSIWYG editor or program is one that allows a developer to see what the end result will look like while the interface or document is being created. Web Matrix features adds XML web service support to applications [Mic08a]. It allows developers to expose a SOAP based XML web service, as well as the calling and usage of XML web services hosted on another server.

Web Services Enhancements (WSE)

Web Services Enhancements (WSE) 3.0 for Microsoft® .NET is an add on to Microsoft Visual Studio 2005 and the Microsoft .NET Framework 2.0 [Mic08b]. It enables developers to build web services based on the latest web services protocol specifications [Mic08b]. It allows developers and administrators to apply security policies to web services running on the .NET Framework 2.0.

4.3 Summary

Many of the web service products and tools discussed offer several of the same features; also many conform to a few of the most common web services standards such as SOAP,

XQUERY and XML. However, the problem being faced is how to select the right web services product or tool which best suit a designer's needs. This can be very hard to determine at present since many companies do not explicitly state the features and standards which are supported by their products or tools, more over it is very time consuming to acquire this type of information.

Additionally, many of the products may only conform to a few web services standards. For example, our evaluation of the survey shows that many products are not compliant with the WS-Reliability and WS-Reliable Messaging standard among others. Table 2 shows that many of the web services products and tools surveyed do not leverage dependability mechanisms in them. In general, most of these products do not adopt fault tolerance and reliability features in their designs. Most of the products surveyed are cutting edge products that were recently introduced into the market place within the last two years.

One possible solution in overcoming this problem is using web services reliability patterns in the implementation and design of web services products and tools. More web service patterns could be written to conform to a more of the web services industry standards. In so doing, consumers can readily make an informed decision regarding a particular tool or product. Since patterns are used to solve recurrent and general problems, they are flexible in how they can be used in different products and tools for different purposes and needs.

In addition, there is a need for a comprehensive web service methodology that dictates how we should model a web service prior to its implementation. Such a methodology will allow designers to have a realistic picture of what is required before they begin the actual implementation of web services. This can also serve as a guide to developers, to show how to effectively design a web service; this approach can encourage and stimulate best practices.

5 FAULT TOLERANCE PATTERNS

The Acknowledgement and Active replication patterns were rewritten to improve upon the deficiencies that were found in the survey discussed in chapter 3 [Buc07]. Because these two patterns are useful in providing simple, yet powerful fault tolerance mechanisms in any critical application; we have made them more complete to address the deficiencies found in our survey. We present below the enhanced versions of these patterns.

5.1 Acknowledgment

5.1.1 Intent

Detects errors in a system by acknowledging the reception of an input within a specified time interval.

5.1.2 Context

Applies to systems in which the errors that the monitored system may experience cause it to exhibit omission or crash failures. The frequency of interactions between the

monitored system and monitoring system may vary significantly. The time it takes for the monitoring system to contact the monitored system is bound and known.

5.1.3 Problem

How to achieve timely detection of an error within a set time interval before system failure occurs. How to reduce the time overhead in detecting such an error. The solution to this problem is affected by the following forces:

- The time overhead introduced by the detection mechanism should be kept at a minimum.
- The information that the monitored system has failed is of importance only when that system is in use (i.e. it has received some input and it is expected to produce the corresponding output).
- The communication between the monitored system and the monitoring system must not increase unnecessarily, e.g. there must not be any communication overhead when the monitoring system does not wish to interact with the monitored system.

5.1.4 Solution

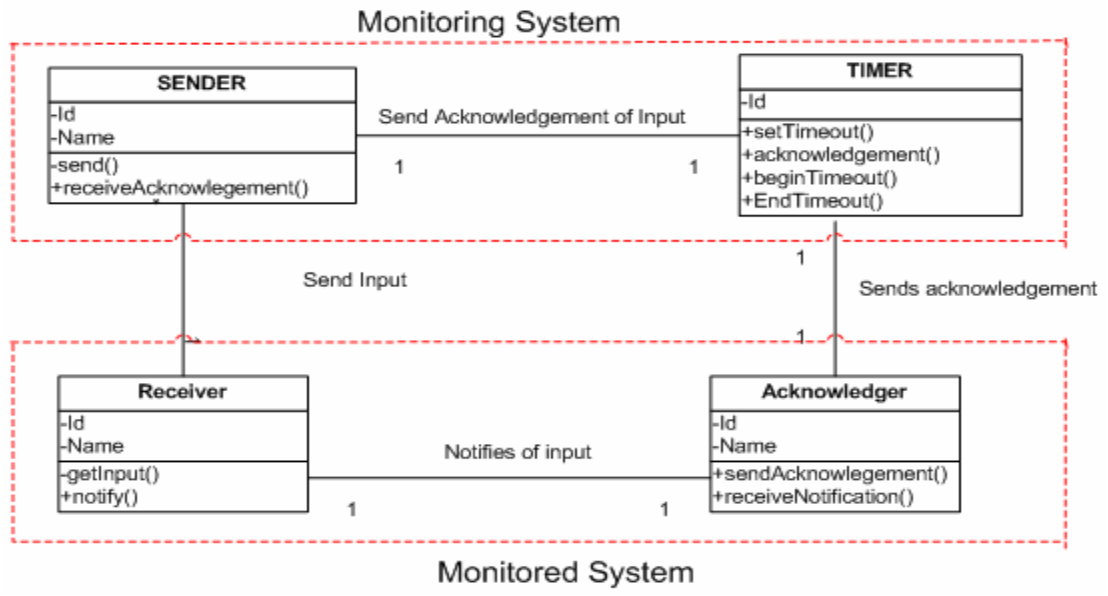
Acknowledge receipt of an input within a specified time interval without increasing the time overhead significantly [Sar02]. Once the monitoring system provides input to the monitored system, it sets a local timer to a predefined timeout. While the timer is counting down from the timeout period, the monitoring system waits to receive an

acknowledgment regarding the reception of the input by the monitored system [Sar02]. If the acknowledgment arrives before the timeout has expired then the monitored system is considered to function correctly, otherwise it assumes that an error has occurred on the monitored system [Sar02].

Structure

Figure 2 shows a class diagram for the acknowledgement pattern's solution. The sender in conjunction with the timer constitutes the **Monitoring system**. The **Sender** is responsible for contacting the monitored system. The **Timer** is responsible for counting down the timeout period every time an input is provided to the monitored system [Sar02]. When the timeout period expires for N consecutive times without receiving an acknowledgement from the monitored system, the timer detects an error on the **Monitored system** and notifies the sender [Sar02]. The receiver in conjunction with the acknowledger entity constitutes the monitored system. The **Receiver** is contacted by the sender regarding an input. The **Acknowledger** is responsible for sending an acknowledgement to the timer every time the monitored system receives an input.

Figure 2. Class Diagram for the Acknowledgment Pattern



Dynamics

We describe the dynamic aspects of the Acknowledgement pattern using a sequence diagram for the use case "get an acknowledgement for an input".

UC: get an acknowledgement for an input (Figure 3).

Summary: An acknowledgment is required for a given input.

Actors: Sender, Receiver

Description:

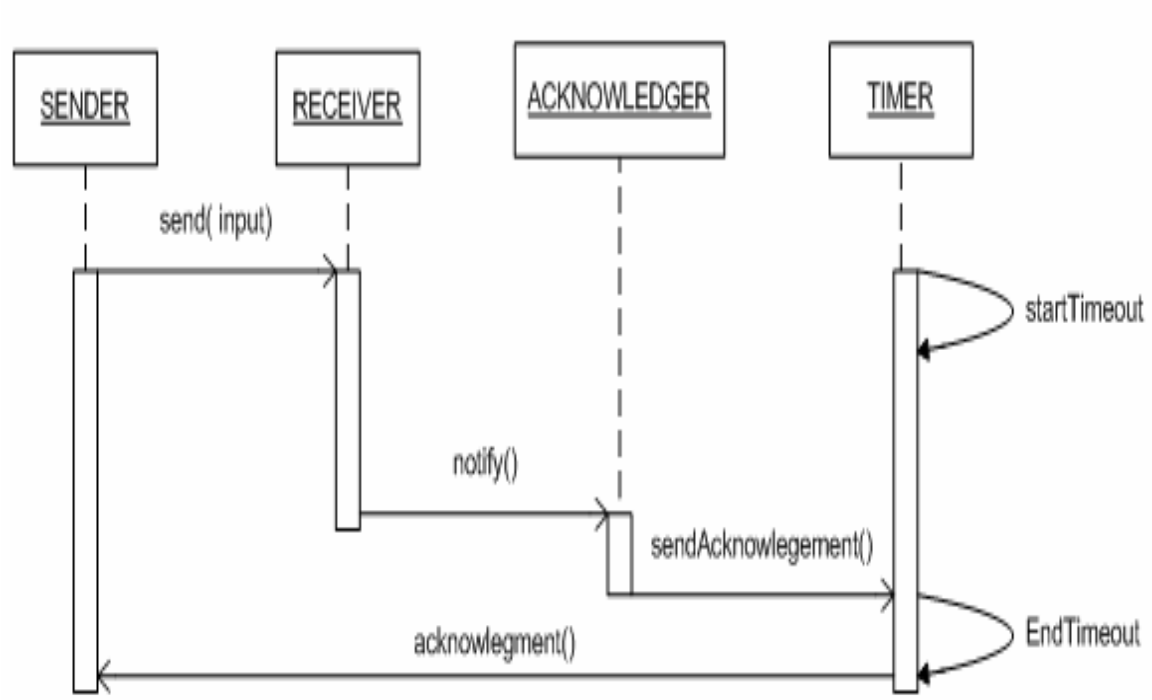
1. The sender sends an input to the receiver; and immediately the timer begins to count down from the timeout period.
2. The receiver notifies the acknowledger relaying that it has received an input.

3. The acknowledger sends an acknowledgment to the timer, which ends the count down of the timeout period.
4. The timer sends an acknowledgment to the sender.

Post Condition:

An acknowledgement is sent to the sender.

Figure 3. Sequence Diagram for the UC "get an Acknowledgment for an Input."



5.1.5 Implementation

To implement the Acknowledgement pattern the following is required:

1. Design error handling so as to allow the system to mitigate faults that may occur in the environment or at runtime.

2. Design authentication and authorization mechanisms for the sender, receiver and acknowledger. This helps to ensure that only authorized entities can send an input to the monitored system.
3. Utilize a public key and a secure channel when sending an input to the monitored system. This helps to secure the input, and reduce the probability of a malicious entity intercepting and modifying the input before it reaches the receiver.
4. Include access control in the timer in order to reduce unauthorized access to the timer.
5. Define the data format for the input that is sent from the sender to the receiver.
6. Define the timeout period used, setting an imprecise timeout period can result in an elevated turn around time to the sender.

5.1.6 Consequences:

The Acknowledgement pattern presents the following advantages:

- Introduces a low space overhead. Both the timer and the acknowledger entities do not map to additional architectural or software components. Instead, they describe some additional functionality associated with the monitoring and the monitored system respectively [Sar02].
- In the case of OO systems, where the interaction among subsystems (objects) is based on method invocations, the acknowledgment can be merged with the reply to an invocation, reducing further the design complexity.
- Introduces a low time overhead to set the timer.

The pattern also has some possible liabilities:

- An error on the monitored system is detected only after some input has been issued to it. This means that although an error might have already occurred on the system a long time ago, it will remain undetected until the moment when some input is sent to the monitored system [Sar02]. Hence, the time overhead for detecting an error can be significant.
- The timeout must be set based on the time it takes for the input to reach the monitored system plus the time it takes for the acknowledger to reach the monitoring system. In many cases it is very difficult to have an exact figure for the aforementioned times, and the timeout is usually the result of estimations based on statistical observations. Hence, configuring an optimum timeout is not a trivial task [Sar02].
- The timer and the acknowledger represent functionalities that are added to the monitoring and the monitored system respectively. Consequently, the application of this pattern on legacy systems might be very demanding in terms of implementation effort and space overhead [Sar02].

5.1.7 Known Uses

This pattern is used in client server peer-to-peer communication networks including telecommunications networks such as AT&T, Cable and Wireless Jamaica limited and in many microcontroller based systems.

The following protocols utilize the Acknowledgement pattern:

Transmission Control Protocol (TCP) provides reliable, ordered delivery of a stream of bytes from one program on one computer to another program on another computer. Besides the web, other common applications of TCP include e-mail and file transfer [Kes07].

Internet Control Message Protocol (ICMP) is an adjunct to the Internet Protocol (IP) that notifies the sender of IP datagrams about abnormal events. This collateral protocol is particularly important in connectionless environment of the Internet Protocol [Kes07].

WS-Acknowledgement is a protocol designed to support reliable message exchange between services by providing for at-least-once and exactly-once SOAP message transfer guarantees [BEA03].

5.1.8 Related Patterns

The Riding over Transients [Sar02] and Leaky Bucket Counter pattern [Sar02].

5.2 Active Replication

5.2.1 Intent

Active replication masks hardware errors that can lead to a failure in a system. It uses a set of replicated processors that take in the same input and conduct independent and

concurrent processing of that input [Sar02]. The outputs from all replicas are compared to ascertain the correct output. One processor error can be masked.

5.2.2 Example

Consider an aircraft which has to perform critical operations based on a defined input. This input is sent to a processor for processing and an output is returned. This output is then used to change the path or direction of the airplane. If the output given by the processor is incorrect this could result in a system failure (a crash).

5.2.3 Context

Applications or critical systems that are deterministic and which can experience errors while processing inputs. These errors may lead to system failures.

5.2.4 Problem

How to mask hardware errors in a system so as to avoid incorrect outputs. Some applications such as aircraft and health care systems require the masking of errors so as to avoid system failure that can lead to serious consequences.

How do we mask the occurrence of hardware errors to avoid incorrect results? The solution to this problem is affected by the following forces:

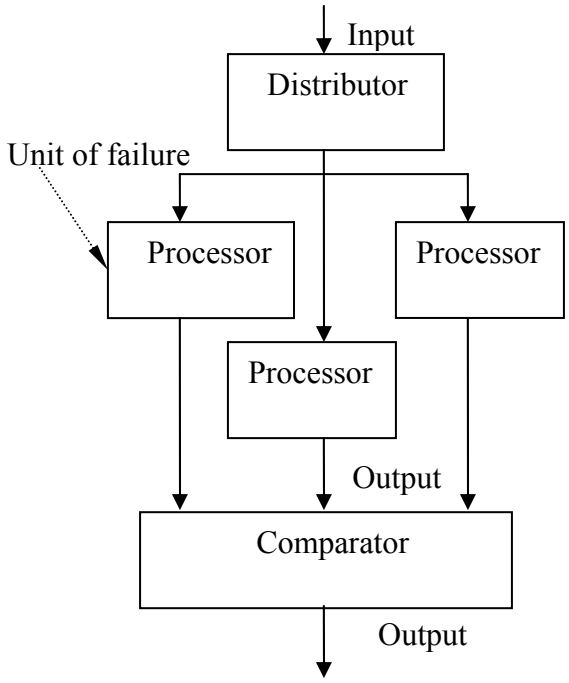
- The inputs received by the system must be processed and the system must deliver the corresponding output independently of whether an error occurs in one of the processors.
- The error-free execution of the system must suffer minimum time penalties [Sar02].
- The time penalty introduced by the solution in the presence of errors must be kept very low.

5.2.5 Solution

Use a set of processors which receive the same inputs in the same order and process their inputs independently and concurrently. The outputs from all processors will be compared using some algorithm and the correct output will be selected and delivered to the receiver.

The active replication technique typically utilizes five main components as illustrated in Figure 4.

Figure 4. Structure and Data Flow of the Components involved in Active Replication Technique



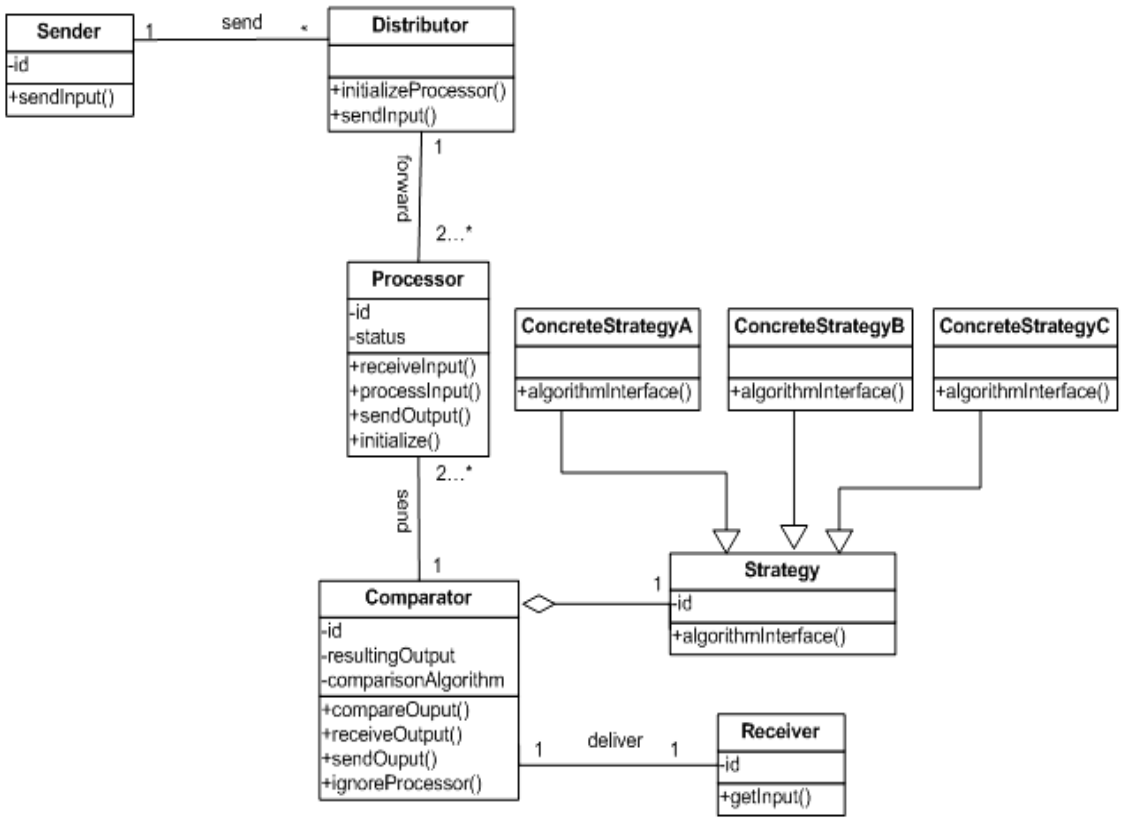
The distributor forwards the input to three separate processors for processing. Each replica is initialized at the same time and begins processing on the input independently and concurrently [Sar02]. Each replica forwards its output to a comparator. The comparator compares the three outputs delivered by the replicas and uses some algorithm (e.g. majority voting) to select the result. The output selected by the comparator is then forwarded to the receiver. The replica that sends an incorrect output is ignored [Sar02].

Structure

Figure 2 shows the class diagram for the solution. The **Sender** sends an input to the Distributor, which initializes all the Processors and delivers the same input to each

processor for processing. The **Processors** must be two or more and perform the same operation on the input and deliver their independent outputs to the Comparator when they have completed processing. The **Comparator** receives the outputs from the processors and compares them in order to deliver the correct output to the Receiver. The **Receiver** represents the application or environment that gets the correct output from the comparator. The comparator may use the **Strategy** [Gam94] pattern to select the most suitable algorithm for comparison. The comparison algorithm must be able to determine the processor sending a result different from the selected output.

Figure 5. Class Diagram for the Active Replication Pattern



Dynamics

We describe the dynamics aspects of the Active Replication pattern using a sequence diagram for the use case "obtain the correct output from a set of equal inputs".

UC: obtain the correct output from a set of equal inputs (Figure 3).

Precondition:

The inputs sent by the distributor to the processors must be equal.

Summary: An input value needs to be processed correctly.

Actors: Sender, Receiver

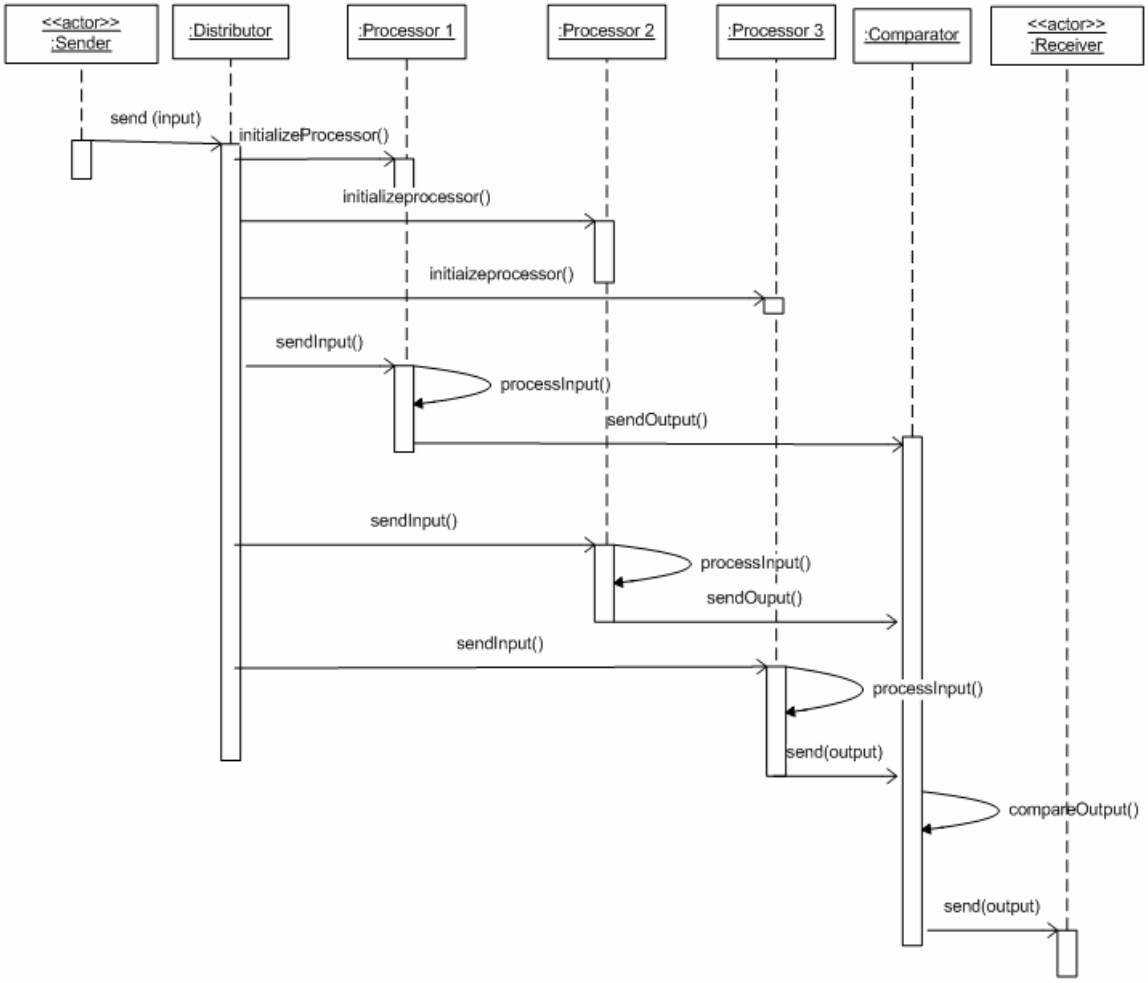
Description:

1. The sender sends an input to the distributor.
2. The distributor forwards the input to the processors.
3. Each processor processes the input independently and sends its output to the comparator.
4. The comparator gets the output of each processor and compares the results. The comparison is done using an algorithm that is selected by a strategy pattern.
5. The comparator uses the strategy pattern to select the most suitable algorithm that will be used to conduct the comparison of the outputs.
6. The comparator selects the correct output and delivers it to the receiver.
7. If a processor delivers an incorrect output the comparator ignores all of its subsequent outputs.

Post condition:

The correct output (according to the comparator) is delivered to the receiver.

Figure 6. Sequence Diagram for the UC "Obtain the correct output from a set of equal inputs.



5.2.6 Implementation

To implement the Active Replication the following is required:

1. In order to deal with N simultaneous hardware errors, 2N+1 processors are required to produce the correct output [Sar02]. With even number of processors we may not be able to determine the correct output but we can detect errors. To

2. The distributor must ensure that all processors receive exactly the same input in terms of content and delivery order. The distributor should be more reliable than all the processors; for example by using redundant logic or high quality VLSI.
3. The comparator receives output from each processor and should use some algorithm (e.g. Majority voting) to determine the correct output [Sar02]. The comparator should have a more reliable implementation than all the processors; for example by using redundant logic or high quality VLSI.

5.2.7 Consequences

The Active Replication pattern presents the following advantages:

- The time overhead introduced in error-free system execution is low because it amounts to the time it takes the comparator to select the correct output [Sar02].
- The design complexity introduced is relatively low, because the comparator and distributor have simple functionalities [Sar02].
- No internal synchronization overhead is incurred in processing because the processors operate independently of each other [Sar02].

The pattern also has some possible liabilities:

- The space overhead is high, because it takes 2N+1 processor to mask N errors [Sar02].

- The comparator and distributor do experience errors that can be experienced by the processors because they do not have error semantics as in the processors.
- The distributor and comparator introduce single points of failure in the system. They can be rendered fault tolerant by applying other fault tolerance approaches to them.

5.2.8 Known Uses

- The Boeing 777 airplane manufactured by Boeing Commercial Airplanes uses this approach [Boe90].
- The HP Integrity NS16200 Server, manufactured by Hewlett Packard, is a server that provides scalable operational data stores for real time processing with high transaction volumes [hp05].
- The Lunar Landing Research Vehicle (LLRV) was created by Bell Aerosystems and was used in the Appolo 13 project for landing on the moon's surface [Bel64].

5.2.9 Related patterns

- The Active replication pattern is the enhancement to the Fail-Stop Processor pattern [Sar02]. The Fail-Stop Processor pattern is used for masking errors that lead to Byzantine failure.
- The Object Group pattern is an object behavioral pattern for group communication and fault tolerance in distributed systems [Maf96]. This pattern can be used to dynamically manage and synchronize the processors.

- The Strategy Pattern [Gam94] can be integrated with the comparator and it can be used to select at runtime the algorithms most suitable to compare the outputs received by the comparator.

5.3 Summary

The Acknowledgement and Active Replication patterns were rewritten to make them more useful to developers. Our survey [Buc07] showed that both patterns lacked detail and did not conform to any of the standard pattern templates. Both patterns introduce simple models that can be used by developers to leverage fault tolerance in an application.

6 WEB SERVICES RELIABILITY PATTERNS

Web services have become the most popular means used by enterprises to offer services to their customers and to interoperate with business partners. These services are accessed through messages. Since messaging is crucial to the enterprise in terms of the services and transactions that are exchanged between businesses and customers, it has become essential to ensure reliable messaging. Reliable messaging, as used in this context, is the act of sending a message without duplication, ensuring guaranteed delivery as well as message ordering and message state disposition [Oas04, Oas07]. The implications of a failure in this respect can have a damaging impact on businesses that rely on the availability and reliability of the services offered to customers [Smi08]. In general, standards defined by committees are rather complex and their descriptions are given in only one level of detail, which together with their length, make understanding of the standards rather difficult.

In particular, web services standards are specifically complex and lengthy. By expressing these standards as patterns, and including precise UML models, we attempt to make them more understandable and easier to compare to other standards with similar objectives.

The WS-Reliability and WS-Reliable Messaging Standards are defined by OASIS and the former has borrowed from the ebXML Message Service Specification 2.0 technology [Ebx02]. WS-Reliability is a SOAP based (uses SOAP 1.1 and SOAP 1.2 Part 1) [W3c07] specification that fulfills reliable messaging requirements critical to some applications of web services. The WS-Reliability standard utilizes quality of service (QOS) contracts, and uses conditions attached to the invocation of a set of operations; namely deliver, submit, respond and notify [Oas04]. To perform reliable delivery it uses the concept of a Reliable Message Processor (RMP).

The WS-Reliable Messaging standard provides guaranteed delivery, message ordering, duplicate elimination and state disposition [Oas07]. The WS-Reliable Messaging "protocol is described in this specification in a transport-independent manner allowing it to be implemented using different network technologies" [Oas07]. To support interoperable web services, a SOAP binding is defined within this specification. However the protocol depends upon other web services specifications for identification of service endpoint addressing and policies [Oas07]. We present below two new patterns for web services reliability.

6.1 WS-Reliability

6.1.1 Intent

WS-Reliability ensures that a notification [Buc08a] is always sent in response to a failure, it also provides guaranteed message delivery, message ordering, and duplicate elimination whenever messages are sent from one entity to another.

6.1.2 Example

Consider a remote SCADA system used in electricity production and distribution where several operations are carried out across the internet from different locations. Due to the nature of such a service, a great degree of reliability is expected. Since computer operators send their commands in the form of messages, it is important that messages are delivered and their corresponding acknowledgments produced. In the case where a computer operator sends maintenance operations which are highly critical to the service offered, via messages, it is necessary to have a mechanism in place that can give some acknowledgement that the messages were delivered, and ensure that the order in which they were sent is maintained without any duplication. To illustrate this idea further, the following messages are typically sent to the remote SCADA system used by an electricity generation company: 1. Open valves, 2. Update system and 3. Shutdown system. In the event that the messages above are executed in reverse order, the system would be shutdown before the other two operations are carried out. This could result in loss of power to some areas. In contrast if the messages were not delivered at all, unknowingly to the person that sent them, this could also result in loss of money and services to the

business and consumers respectively. Similarly, messages that are delivered twice may be executed more than once and may create an undesirable outcome.

6.1.3 Context

Institutions, B2B applications, and critical infrastructure systems that need to send and receive messages in real-time.

6.1.4 Problem

Some applications need reliable messaging in order to fulfill their business operations effectively and successfully. Many people communicate via the internet, thus creating heavy network traffic. Many companies offer services to consumers across the internet, which give rise to bandwidth and availability problems. Enterprises are concerned about how to achieve reliable messaging given some of the factors mentioned previously; more specifically, to ensure that messages are delivered with acknowledgment of receipt, in the order sent, and without duplication.

How do we ensure that messages that are sent are delivered, acknowledged, sent in order, without duplication? The solution to this problem is affected by the following forces:

1. Dissimilar internet connection speed used by both sides (receiving and sending parties) can affect how quickly messages are sent and received.
2. Network traffic affects the time it takes a message to reach a recipient; this may increase the delay time for the messages and may change their order.

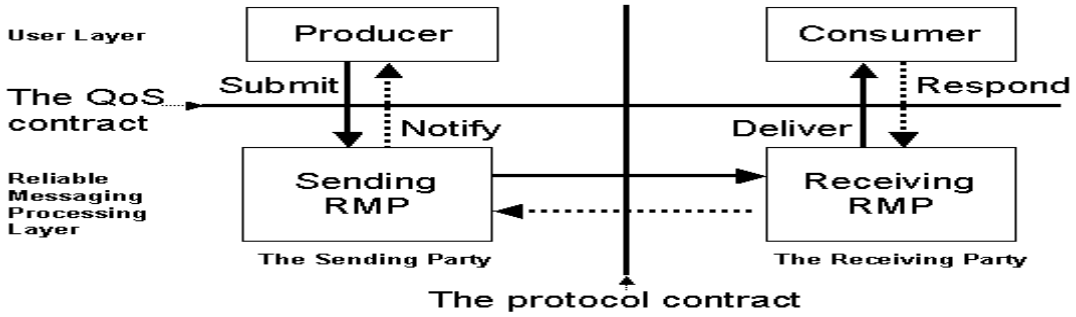
3. The receiving or sending party may become unavailable and some or all messages may not get sent or received.
4. Unordered and delayed messages can lead to problems for online transactions especially in banking systems and critical infrastructures.
5. The response time to messages contributes to delay; when messages get lost or arrive to a recipient unordered, the recipient may take more time to respond, thus increasing the delay time.

6.1.5 Solution

Use a protocol that performs guaranteed message delivery with acknowledgement of delivery or failure, message ordering, and duplicate message elimination. This is achieved by first having an enforceable contract between the sending and receiving parties, and the use of sending and receiving reliable message processors (RMPs) that send, deliver, order and eliminate duplicate messages.

The WS-Reliability standard utilizes four primary conceptual units as illustrated in figure 1. The Producer creates and submits messages to the Sending RMP. The Consumer receives messages delivered by the Receiving RMP and sends an acknowledgement. The Sending RMP submits messages to and receives acknowledgements from the Receiving RMP. The Receiving RMP is responsible for delivering messages to the consumer and receiving and sending notification from the consumer to the Sending RMP. A QoS Contract binds the agreement made between the consumer and producer. A protocol contract binds the Sending and Receiving RMP.

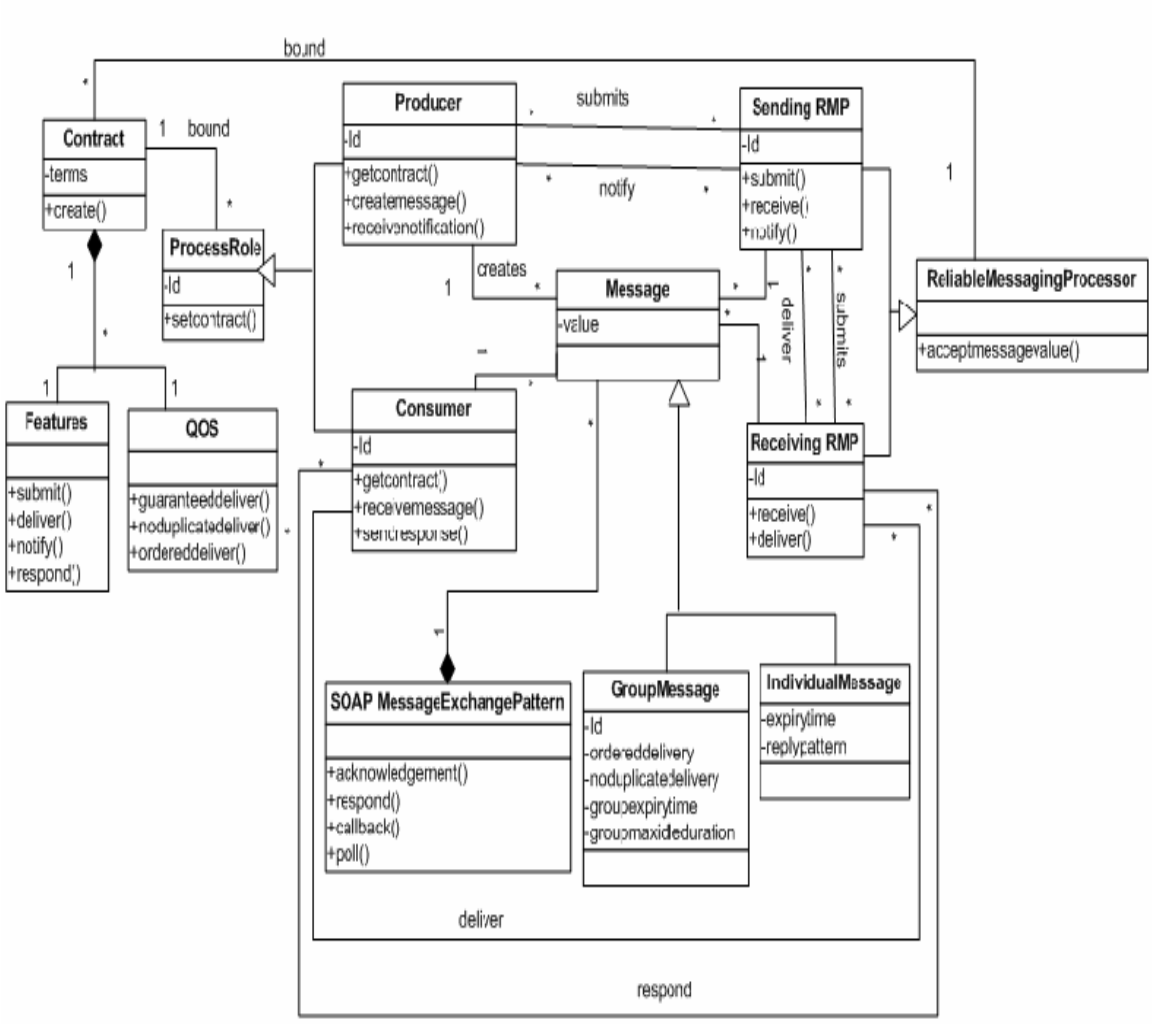
Figure 7. Structure and Dataflow of the Components involved in the WS-Reliability Standard



Structure

A **contract** (Figure 7) defines the quality of service expected between the sending and receiving RMP as well as the terms of the relationship between the **Producer** and the **Consumer.** The contract includes a specification of the expected quality of service (**QoS**), which determines the quality of messaging service to the communicating parties, and the **Features** which define the operations and rules which are expected. The **Reliable Messaging Processor (RMP)** [Oas04] handles messages that are sent between a producer and a consumer and perform messaging as outlined in the contract in the form of requirements such as guaranteed delivery, duplicate message elimination, and message ordering. The implementation of the RMP is not specified by the standard, and can be implemented in many different ways (see implementation).

Figure 8. Class Diagram for the WS-Reliability Pattern



Processes may have two **ProcessRoles**, the **Producer** role creates messages and sends them to the **Sending RMP**. The **Consumer** role consumes messages that have been processed by the **Receiving RMP**. A **Message** can be a **Group Message** or an **Individual Message** with varying attributes depending on the type of message. The **SOAP MessageExchangePattern (MEPs)** defines different modes of response which can be sent from the Consumer to the Receiving RMP in response to a previously received message. The SOAP MEPs used here is defined in SOAP 1.2 [W3c07].

Dynamics

We describe the dynamics aspects of the WS-Reliability pattern using a sequence diagram for the use case "Sending a reliable message".

UC: Sending a reliable message (Figure 9).

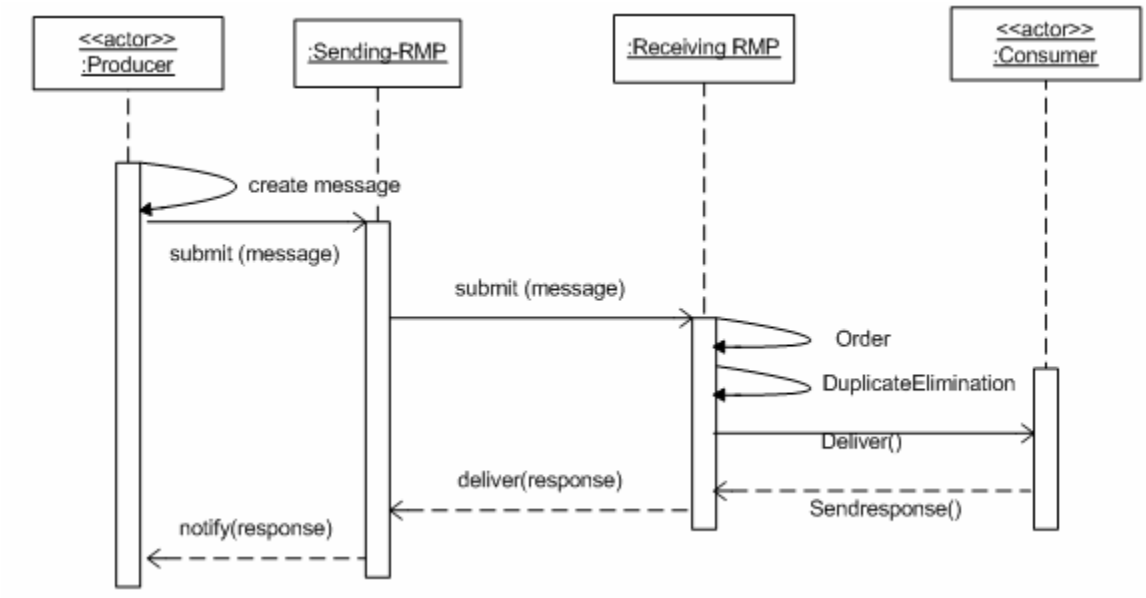
Summary: A producer wishes to send a reliable message to a consumer.

Actors: Producer, Consumer.

Precondition:

The producer and the consumer must have a defined contract prior to communicating with each other.

Figure 9. Sequence Diagram for the UC "sending a Reliable message"



Description:

1. The producer creates and submits one or more messages (group messages) to the Sending RMP to be sent to a consumer.
2. The Receiving RMP receives the message from the Sending RMP and delivers it to the consumer.
3. The consumer acknowledges reception of the message by responding back to the Receiving RMP.
4. This response is delivered to the Receiving RMP which sends it to the sending-RMP which notifies (sends acknowledgement) the producer.

Post condition:

The message has been delivered and the sender has been notified of its delivery or failure.

The use case "Establishing an agreement" should be performed before sending messages.

UC: Establishing an agreement (Figure 10).

Summary: A consumer and a producer establish a contract for their message communication.

Actors: Consumer, Producer.

Description:

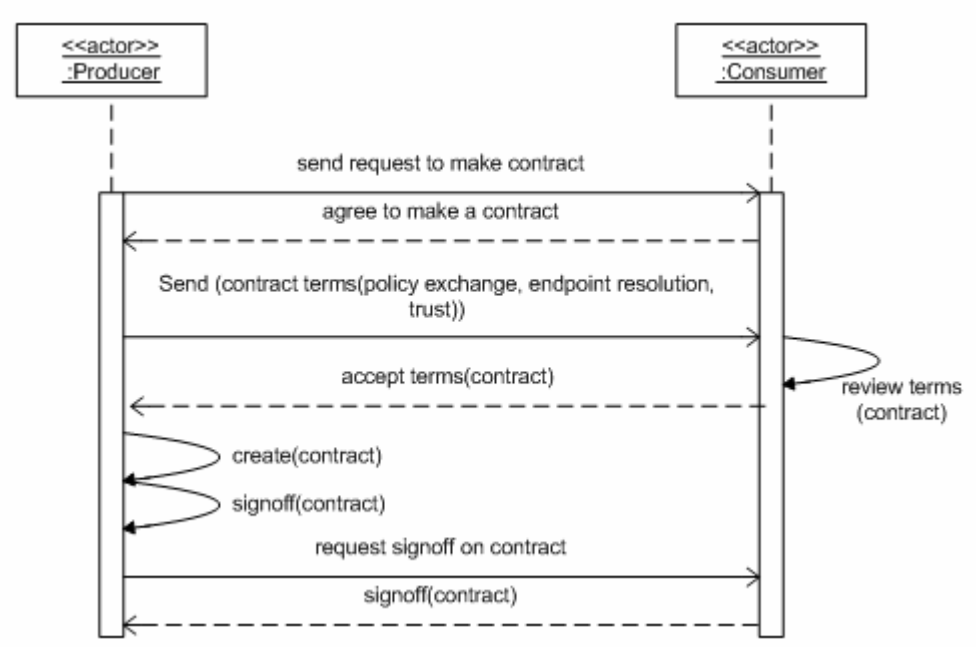
1. The consumer sends a request to define a contract to the producer who agrees to negotiate such.
2. The consumer sends the terms of the contract to the producer who reviews the terms and accepts the terms of the agreement.

3. The producer signs off on the contract and requests a signoff from the consumer to make it binding.

Post condition:

A contract has been defined by the Consumer and Producer about their future message communication.

Figure 10. Sequence Diagram for the UC "Establishing an Agreement"



6.1.6 Implementation

To implement the WS-Reliability, the following is required:

1. Select a RMP that can enforce a contract defined by communicating parties. The RMP processor has to be a SOAP processor [Ebx02]. This type of processor specializes in handling soap messages, and includes an infrastructure such that is

2. Delivery assurance for each message can be achieved based on either AtleastOnce, AtMostOnce, ExactlyOnce and in order assertions.
3. Depends on the use of the SOAP 1.1 and SOAP 1.2 part 1 [W3c07] specifications to function.

6.1.7 Consequences

The WS- Reliability pattern presents the following advantages:

- Messages sent between end points can be controlled by means of the RMP which ensures delivery with acknowledgment, ordering, and duplicate elimination of messages within the limits imposed by the network.
- Enterprises are able to obtain a higher degree of reliability for network communication because the sender and receiver confirm reception by an acknowledgment each time they communicate via a message.
- Quality of service defined by contracts can be maintained between businesses thus increasing reliability and supporting the accountability of business partners. Policies can be attached to the contracts that govern the modus operandi agreed to by all communicating parties.
- The RMP sends an acknowledgment if the consumer becomes unavailable during the transmission of a reliable message.

The pattern also has some possible liabilities:

- Because the message and its response are passed between several components and not directly to the recipient or producer of the message; this process increases the time it takes the message to be delivered to the recipient and the time it takes to send the corresponding notification back to the producer of the message.
- The WS-Reliability pattern increases complexity in the system.

6.1.8 Known Uses

Since this is a recent standard we could find only one known use.

- Reliable Messaging for Grid Services (RM4GS) [Fuj04], is an open source middleware application produced by Fujitsu, Hitachi, and NEC Corporation in Japan.

6.1.9 Related Patterns

The WS-Reliability sends acknowledgements to confirm receipt of a message as a part of its solution, which can be done using the Acknowledgment [Buc08a] pattern. The Strategy pattern [Gam96] can be used as a part of the implementation of the RMP to select algorithms to implement the reliable messaging requirements.

Ordering and Duplicate Elimination are possible patterns which could be used as a part of the implementation of the RMP to order and eliminate duplicate messages.

6.2 WS-Reliable Messaging

6.2.1 Intent

WS-Reliable Messaging ensures guaranteed receipt in response to each message received; it also provides, message state disposition, ordered delivery, and duplicate elimination whenever messages are sent between endpoints.

6.2.2 Example

See section 6.1.2

6.2.3 Context

Institutions, B2B applications, and critical infrastructure systems that need to send and receive messages in real-time.

6.2.4 Problem

Many errors can interrupt communication, messages can get lost, duplicated, reordered, the host system may experience failures and lose volatile state and messages may also experience lost in state during transmission.

Some applications need to have reliable messaging in order to fulfill their business operations effectively and successfully, therefore lost, unordered and duplicate messages can have a negative affect on successful business operations. How do we ensure ordered

delivery, guaranteed receipt, duplicate elimination and state disposition of messages? The solution to this problem is affected by the following forces:

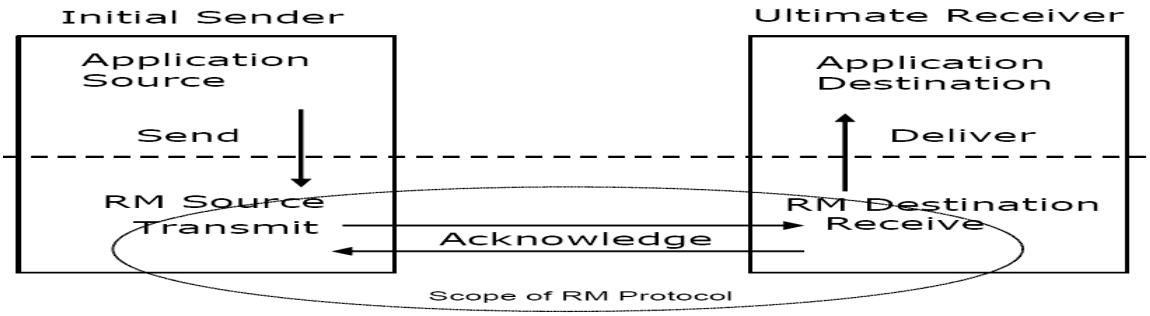
- The receiving or sending host may become unavailable and some or all messages may not get sent or received.
- Messages may get lost during transmission.
- Unordered and delayed messages can lead to problems for online transactions especially in banking systems and critical infrastructures.
- The response time to messages contributes to delay in sending a receipt; when messages get lost or arrive to a recipient out of order, it may take more time to respond, thus increasing the response time.
- Dissimilar internet connection speed used by both sides (receiving and sending parties) can affect how quickly messages are sent and received.
- Network traffic affects the time it takes a message to reach a recipient; this may increase the delay time for the messages and may change their order.

6.2.5 Solution

Use a protocol that performs guaranteed receipt, ordered delivery, state disposition, and duplicate elimination of messages. This is achieved by first having an agreement which includes a policy exchange, endpoint resolution and establishment of trust between end points.

The WS-Reliable Messaging standard utilizes four primary conceptual units as illustrated in Figure 11.

Figure 11. Structure and Dataflow of the Components involved in the WS-Reliable Messaging Standard



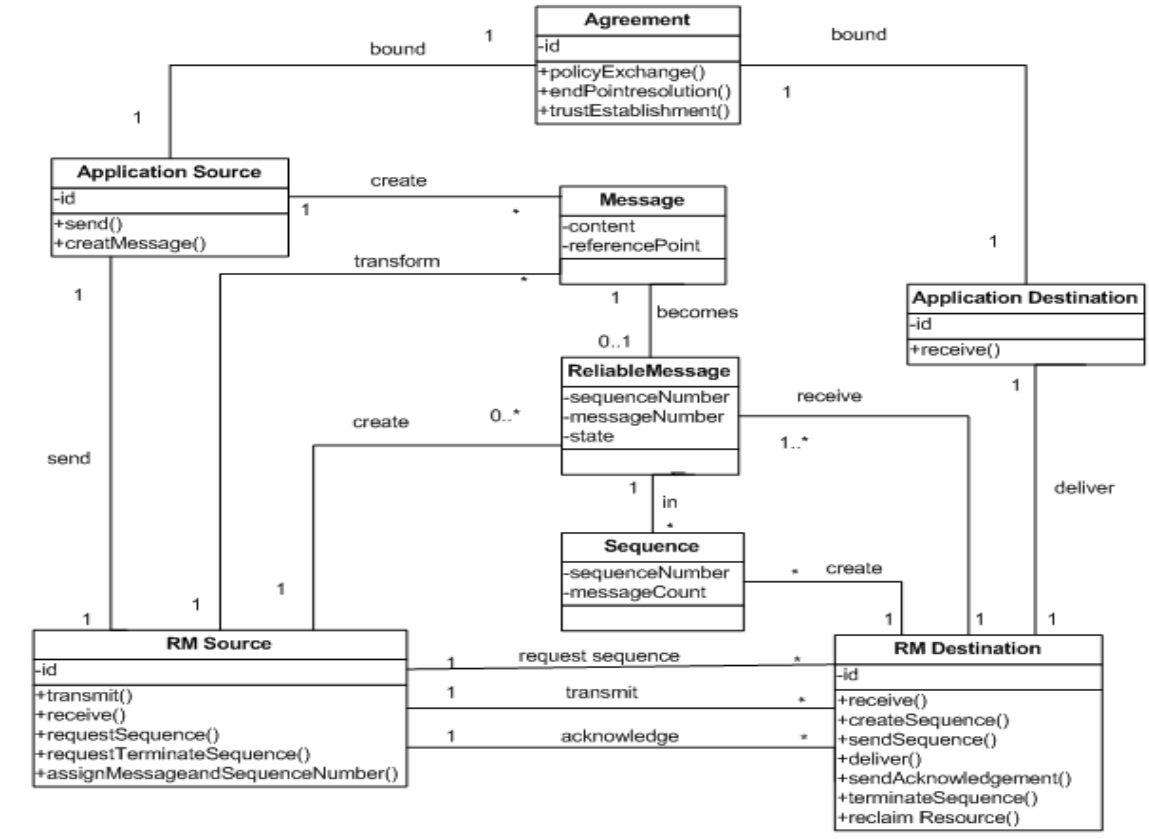
The application source creates and sends messages to the RM Source. The RM Source transmits messages to the RM Destination. The RM Destination receives messages transmitted from the RM source and sends a corresponding receipt of acknowledgement; the message is then delivered to the application destination/receiver.

Structure

An **Agreement** enforces policy exchange, end point resolution, and trust establishment between the **Application Source** and the **Application Destination**. The **Application Source** creates and sends messages to the RM source (Figure 6), **A Message** consists of content and information about where it is supposed to be delivered. The RM Source transforms a message into a **Reliable Message** by adding new properties to the message. A **Sequence** (created by the RM destination at the request of the RM Source) acts like an envelope in which a Reliable message is placed before it is transmitted. The **RM Source** accepts messages and acknowledgements from the Application Source and

RM Destination respectively, and transmits reliable messages to the RM destination. The **RM Destination** receives messages sent from the RM Source, sends a corresponding acknowledgement of receipt to the RM Source, and delivers the reliable message to the destination application. The **Application Destination** receives reliable messages from the RM Destination.

Figure 12. Class Diagram for the WS-Reliable Messaging Pattern



Dynamics

We describe the dynamics aspects of the WS-Reliable Messaging pattern using a sequence diagram for the use case “Sending a reliable message”.

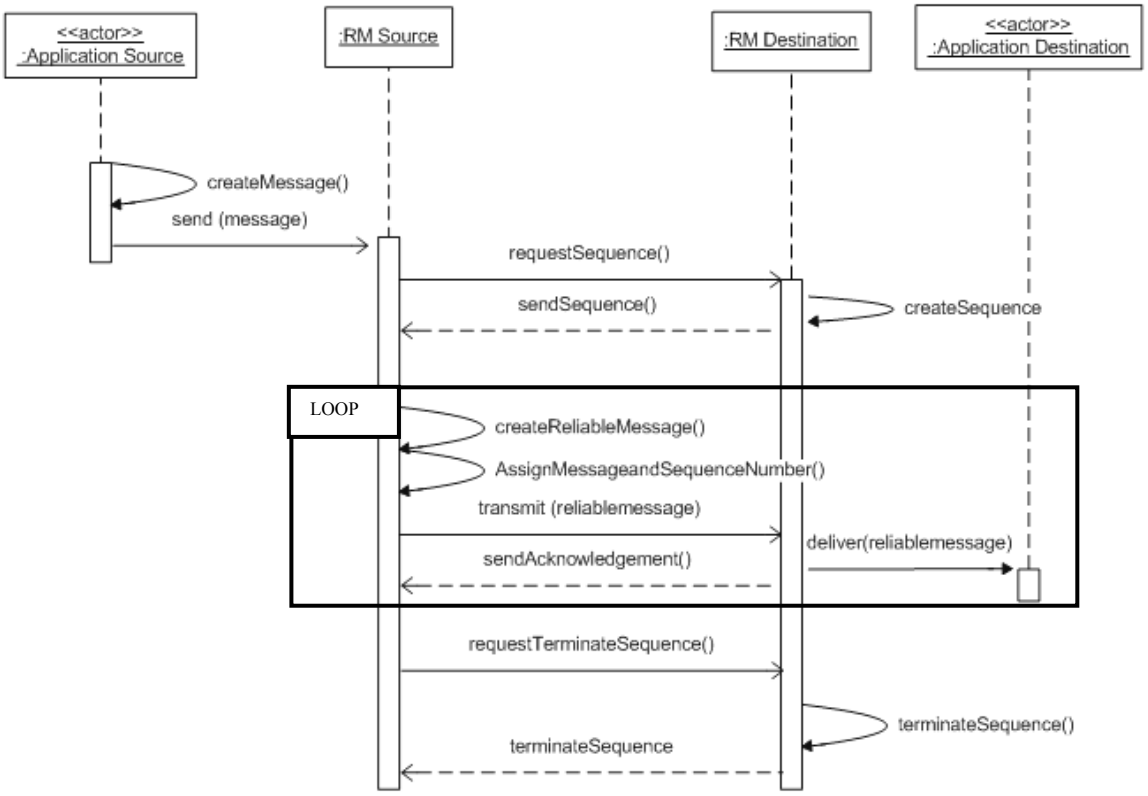
UC: Sending a reliable message (Figure 13).

Summary: An Application Source wishes to send a reliable message to a Destination Source.

Actors: Application Source, Application Destination

Precondition: The Application Source and Destination Source must establish an agreement prior to communicating with each other.

Figure 13. Sequence Diagram for the UC "sending a reliable message"



Description:

1. The Application Source creates and sends a message to the RM Source.

2. The RM Source requests a new sequence from the RM Destination and adds new properties to the message to be sent, thus transforming the message into a reliable message.
3. The RM Source transmits the reliable message wrapped in this new sequence to the RM Destination.
4. The RM Destination delivers the reliable message to the Application Destination.
5. The RM Destination sends an acknowledgement to the RM Source.
6. The RM Source sends a terminate sequence request to the RM Destination when it has no more messages to transmit in that sequence.

Post condition:

All reliable messages have been delivered and an acknowledgement is sent for all messages received by the RM Destination.

The use case "Establishing an agreement" must be completed before sending messages.

UC: Establishing an agreement (Figure 14):

Summary: An Application Source wishes to establish an agreement with the Application Destination.

Actors: Application Source, Application Destination.

Description:

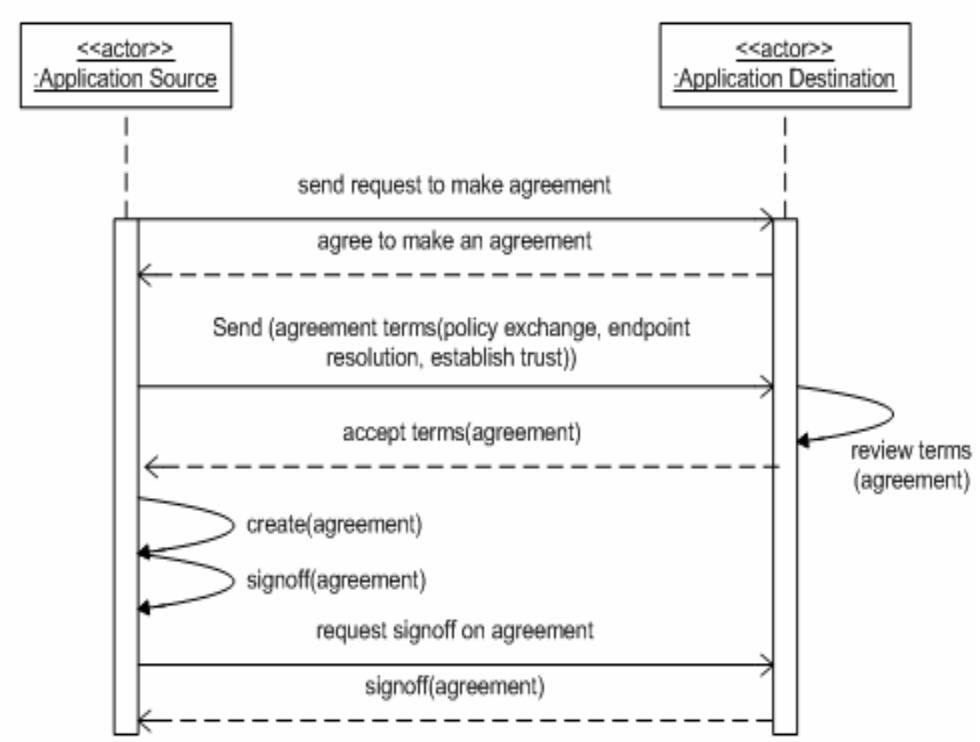
1. The Application Source sends a request to make an agreement to the Application Destination who agrees to make such agreement.
2. The Application Source sends the terms of the agreement to the Destination Application who reviews the terms and accepts the terms of the agreement.

3. The Application Source signs off on the agreement and requests a signoff from the Application Destination to make it binding.

Post condition:

An agreement has been defined by the Application Source and Application destination about their future message communication.

Figure 14. Sequence Diagram for the UC "Establishing an agreement"



6.2.6 Implementation

To implement the WS-Reliable Messaging, the following is required:

1. Use an adaptive mechanism in the RM Source that can dynamically adjust the retransmission time and the back-off intervals that are appropriate to the nature of

2. Delivery assurance for each message can be achieved based on either AtleastOnce, AtMostOnce, ExactlyOnce and in order assertions.
3. The sequence number of messages starts at one (1) and is incremented by one (1) for all other subsequent messages in a given sequence. However this may present some implementation problems since most systems expect elements or processes to begin numbering at zero. The programmer therefore has to ensure that sequence numbers begin at the same starting point between endpoints to avoid discrepancies.
4. Endpoint referencing that uniquely identifies the RM Destination must be obtained by the RM Source.

6.2.7 Consequences

The WS- Reliable Messaging pattern presents the following advantages:

- Enterprises are able to obtain a higher degree of reliability for network communication because endpoints create and terminate message sequences. In

addition a receipt of acknowledgement is sent every time a message is sent and retransmission of messages is done for messages that were not received.

- Quality of service defined by agreements can be maintained between businesses, thus increasing reliability and supporting the accountability of business partners.
- The WS-Policy standard is used to govern policies that can be attached to the agreements that govern the operations agreed to by communicating endpoints, therefore leveraging the use of other web service standards.
- WS-Addressing is utilized to achieve endpoint referencing. This specifies the endpoint reference to where the receipt of acknowledgement is to be sent in response to a message. In this way messages cannot be intercepted easily because the destination is known prior to their transmission.
- Terminate message sequence requests are sent to the RM destination to notify when no more messages will be sent using a given sequence. Therefore the system resources attached to a sequence can be freed and used to conduct other operations.

The pattern also has some possible liabilities:

- Introduces a high time overhead with the retransmission of messages and acknowledgements. The RM Source will retransmit messages for which no receipt of acknowledgments was received. This could result in high volume requests flooding the RM Destination depending on the retransmission and back-off interval set (see implementation).
- There is a high demand on the resources used to track the state of each message transmitted as required by the RM Source.

6.2.8 Known Uses

The following commercial products utilize the WS-Reliable Messaging Pattern:

- SAP NetWeaver Process Integration 7.1 is a platform for process integration based on the exchange of XML messages [Sap07].
- Apache Sandesha Apache Axis is an implementation of the Web Services Reliable Messaging (WS-Reliable Messaging standard), published by IBM, Microsoft, BEA and TIBCO Software as a joint specification, on top of Apache Axis (The Next Generation SOAP) [Apa05].
- WebSphere MQ is an application integration tool used for passing messages between applications and Web services [Web08].

6.2.9 Related Patterns

The WS-Reliability is an alternative used to send reliable messages [Buc08b]. The WS-Reliable Messaging sends a receipt of acknowledgements to confirm reception of each message received as a part of its solution. This can be done using the Acknowledgment pattern [Buc08a]. The WS-Policy and WS-Addressing Patterns can be written and used to manage policies and endpoint referencing respectively.

The Strategy pattern [Gam94] can be used as a part of the implementation of the retransmission of messages to ascertain the best retransmission time and back-off interval to be used for lost messages and receipts of acknowledgment.

6.3 Summary

The WS-Reliability and WS-Reliability patterns describe the WS-Reliability and WS-Reliable Messaging standards defined by Oasis [0as04, Oas07].WS-Reliability and WS-Reliable Messaging are two standards intended to specify the reliable delivery of message between web services. We have provided patterns for these standards. The original standards are complex and hard to understand; we hope to have clarified their structure and behavior. Since both standards apply to the same problem we have used our patterns to make clear the differences and applicability.

7 ANALYSIS

We present our analysis of the fault tolerance and reliability patterns surveyed and written in the previous chapters.

7.1 Fault Tolerance Patterns

Due to the importance of fault tolerance in any system, particularly in critical systems our analysis has shown that many fault tolerance patterns that have been written are incomplete and lack detail. In particular most patterns illustrated in Table 1 do not conform to the POSA [Bus96] or GOF [Gam94] pattern templates. We believe that, the use of UML diagrams is necessary to describe the structure and dynamics of patterns to guide a designer in applying them as a solution.

Our analysis of these patterns showed that we need more fault tolerance patterns. We also need to rewrite many of the existing patterns to make them more concise, complete and useful. We intend to increase the usage of fault tolerance patterns in critical systems, by making them more complete in their description and representation with the use of UML diagrams, by using the pattern templates outlined by POSA [Bus96] and or GOF [Gam94]. Adding more detail to these patterns to make them more understandable may also increase the use of them in critical applications. Additionally, our survey showed

that this area is in its infancy and has not been adopted or used much in web services products and tools by the leading bodies in software development.

7.2 WS-Reliability and WS-Reliable Messaging Patterns

WS-Reliability and WS-Reliable messaging specifications offer the same service, which is sending messages in a reliable manner. However, the two protocols utilize different means of performing this service. WS-Reliability has a binding to HTTP whereas WS-Reliable Messaging is transport independent allowing it to be implemented using different network technologies. In order to support interoperable web services, a SOAP binding is defined within both patterns. The specifications mandate that an agreement or contract be made before communication can be done between endpoints. However the WS-Reliable Messaging explicitly states that endpoint referencing, establishment of trust and policy exchange be included in the agreement. Endpoint referencing explicitly states the address where a reliable message should be sent. Establishment of trust is achieved with an enforced agreement and policy exchange is to facilitate the updating of quality of service terms and conditions. WS-Reliability does not explicitly dictate the terms of the contract.

WS-Reliability engages the producer and consumer of a message in the entire cycle of sending a reliable message; due to the fact that WS-Reliability ensures that a guaranteed acknowledgement is sent to the producer of a reliable message. The producer specifies the mode of response that is required from the consumer and waits until an acknowledgement is received. This acknowledgement ends the cycle. In contrast, WS-

Reliable Messaging ensures guaranteed receipt; The RM Source and RM Destination components control the execution of a reliable message between each other. Once the initial message is obtained, a guaranteed receipt is sent between these two components, not directly to the initial sender. WS-Reliable Messaging does not require that the sender listen for a guaranteed receipt which is dealt with in the RM Source.

Additionally, WS-Reliable Messaging must use a sequence to transmit all messages (individual and series), while WS-Reliability uses the Sending RMP and Receiving RMP to send messages either individually or in groups, assigning a unique group id and a unique sequence number for group messages. However, the sequence number is optional for individual messages. Another stark contrast between the two specifications is that WS-Reliable Messaging mandates that all sequences be ended when no further messages will be sent using that sequence. This allows resources that are attached to each sequence to be reclaimed. WS-Reliability uses a GroupExpiryTime to terminate group messages and an ExpiryTime to terminate an individual message.

Another difference between the two specifications is that WS-Reliability uses SOAP message exchange patterns (MEPs), which specifies the mode of response to be used by the recipient of a reliable message. The message exchange patterns used are poll, respond and callback. However, WS-Reliable Messaging does not explicitly ask for a particular response mode from the recipient of a reliable message. In fact WS-Reliable Messaging does not require a response from the recipient of a reliable message, because the RM Destination sends a receipt of acknowledgement to the RM Source directly. Additionally

WS-Reliable Messaging allows a receipt of acknowledgment to be sent with or without using the SOAP body.

In summary WS-Reliability will only send an acknowledgement when a reliable message is delivered to the recipient, this supports real time communication using messaging. However, WS-Reliable Messaging sends a receipt of acknowledgment once the RM destination receives a reliable message which can be done before, after or simultaneously to delivering the reliable message to its destination. In the case of group messages WS-Reliable Messaging can hold messages at the RM Destination until all messages are received and send them all at once to the recipient. Therefore the concept of guaranteed acknowledgment and guaranteed receipt is different between the two specifications.

WS-Reliable Messaging supports the use of other transport protocols including HTTP; however WS-Reliability is HTTP dependent because it is SOAP dependent. SOAP uses HTTP to exchange information between applications. This may be a disadvantage for endpoints that wish to use different transport protocols to communicate. In the event that two established companies wish to communicate, WS-Reliability would require that they use HTTP. This may require some implementation change to their applications to facilitate communication, which may be costly.

WS-Reliable Messaging is dependent on WS-Policy and WS-Addressing for policy and identification of endpoints respectively. WS-Reliability and WS-Reliable Messaging do not explicitly state how to achieve ordering and duplicate elimination of messages.

Instead, they state that both features can be implemented in different ways. WS-Reliable Messaging includes message state disposition, The RM Source tracks each reliable message until a receipt is received from the RM Destination. WS-Reliability utilizes the use of a message number to keep track of each message sent.

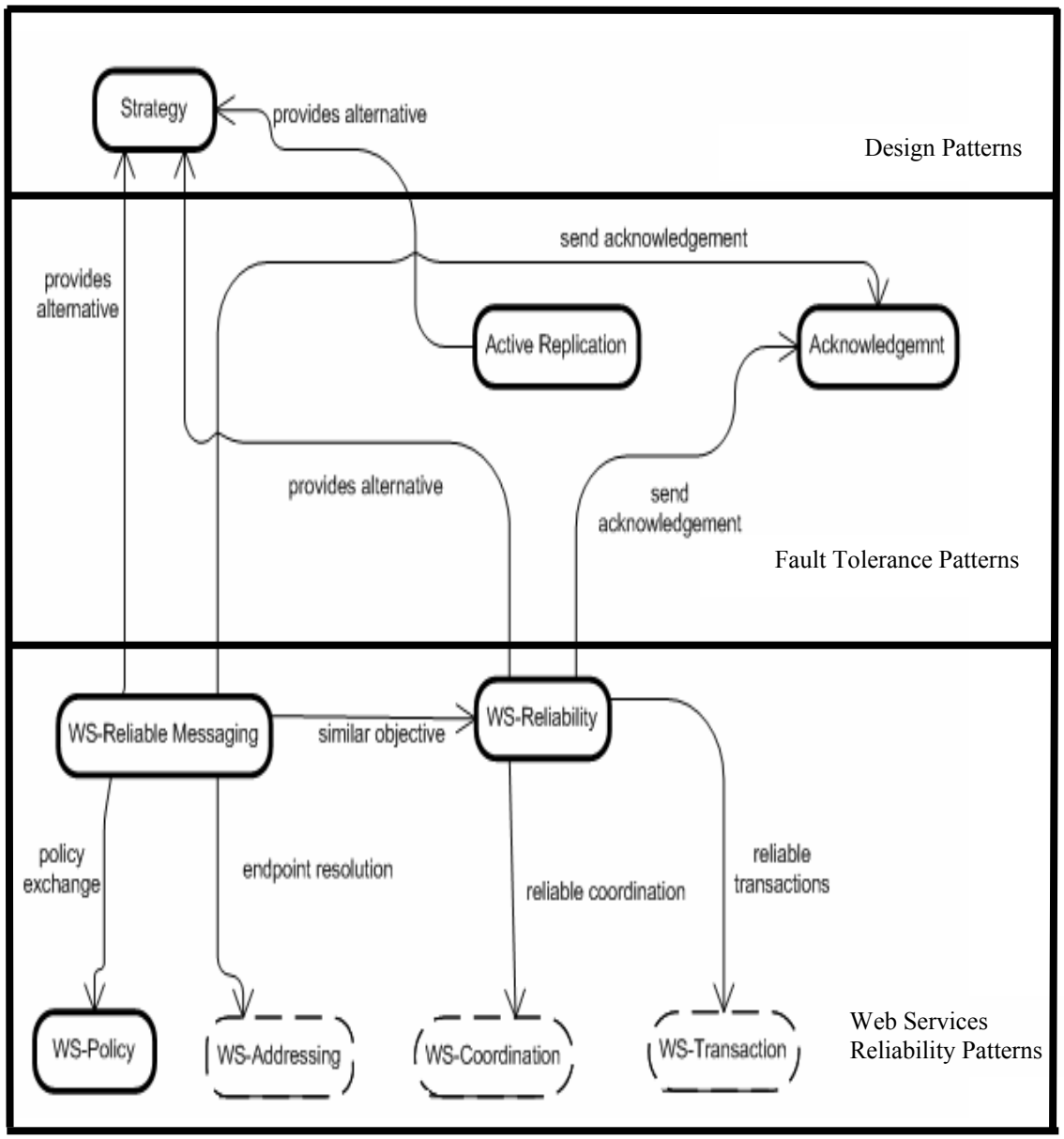
WS-Reliability requires a contract while WS-Reliability requires an agreement before communication can begin between endpoints. The Patterns are similar in this regard however, the agreement includes establishment of trust, policy exchange and endpoint resolution. The structure of the key components in the WS-Reliability and WS-Reliable Message patterns are similar see Figure 8 and 12. However, the WS-Reliability pattern uses the Sending and Receiving RMP to provide acknowledgement, ordered, duplicate elimination and guaranteed delivery of messages. Where as, the WS-Reliable Messaging pattern uses the RM Source and RM Destination to provide ordered, duplicate elimination, state disposition and guaranteed receipt and delivery of messages.

7.3 Fault Tolerance and Reliability Patterns

We present a classification of the fault tolerance and reliability patterns presented earlier in chapter 5 and 6. A pattern diagram of dependability patterns serves to help system designers find appropriate patterns that suit their needs and show their relationship with each other. In identifying the relationship between patterns an application developer can readily make a decision about which patterns fits best with their existing or desired system. Our classification scheme of dependability patterns (See figure 15) is not exhaustive and includes some of the dependability patterns that were discussed in

previous sections. Figure 15 shows patterns that have already been written, these are drawn with solid unbroken lines. The patterns that have not yet been written are illustrated with a broken line. The classification separates this group into design patterns [Gam94], fault tolerance patterns and Web Services reliability patterns.

Figure 15. A Classification of Dependability Patterns



8 CONCLUSIONS AND FUTURE WORK

Our research on fault tolerance and web services reliability patterns illustrates the need to advance this area. Our survey showed many significant deficiencies. One of the most common weaknesses with current fault tolerance patterns is the fact that most pattern writers do not follow any standard method of presenting a pattern [Buc08a]. As a result, there are fault tolerance patterns that are useful but lack the necessary details as highlighted in chapter 3. The advancement of this area in particular is important because we depend on reliable critical infrastructures. The dependability patterns presented here are geared at supporting best practices that developers can adopt. They also provide a concise model which can be used by developers to leverage the properties of fault tolerance and reliability in applications.

Due to the rapid increase in the use of web services applications in critical infrastructures [Fer07a], there is an urgent need to make them more dependable. Oasis [Oas04, Oas07] has several standards which define properties that can be leveraged in a web service. These include reliability and security. Our concern here is reliability.

We wrote new patterns that describe the WS-Reliability and WS-Reliable Messaging standards. Our objective here is to provide an easy way for web services developers to leverage the use of web services standards in web services to make them more reliable

whilst conforming to industry standards. These patterns offer a detailed and convenient means for developers to include these reliability aspects in web services.

Our comparison of the WS-Reliability and WS-Reliable Messaging standards was crucial in understanding the differences between the two standards, and ultimately our patterns, because they solve the same problem and provide the same services. Our analysis highlighted in detail the differences between the two standards, which is useful to allow software developers to choose between the standards depending on the nature of their applications. One pattern may be more useful depending on the protocols that have to be used in a web service and its functionality.

In rewriting and creating new patterns for fault tolerance and reliability, we observed that the patterns could be combined to form new patterns. Combining fault tolerance and reliability patterns could add more dependability properties in a critical system. The pattern diagram of dependability patterns is very useful in showing which combination of patterns can be leveraged. It also illustrates patterns that could be added; and highlights the relationship among them.

Additionally, our survey found patterns that are useful but have not yet been written, and this may pave the way for our future work. We intend to write the patterns of Figure 15 that have not yet be written. These patterns include WS-Coordination, WS-Addressing and WS-Transaction. Future work will include the development of further patterns so as to provide the designer with a catalog of patterns that can be used when developing web-services-based systems.

REFERENCES

[Act02] Actional. (2007, May 1). Actional XMS. Retrieved March 26, 2008, from http://www.actional.com/products/active-policy-enforcement/.

[Ada95]Adams, M., Coplien, J., Gamoke, R., Hanmer, Keeve, F., & Nicodemus, K. (1995). Fault-Tolerant Telecommunication System Patterns. Retrieved May 2008, from http://users.rcn.com/jcoplien /Patterns/PLoP95_telecom.html.

[Alt08] Altova.(2008, March 29).XMLSpy XML Editor. Retrieved October 2, 2008, from http://www.altova.com/products.html.

[Avi84]Avizienis, A.A., and Kelly, J.K.J. (1984).Fault tolerance by design diversity: concepts and experiments. IEEE Computer, 17(8), 67-80.

[Avi85] Avizienis, A.A. (1985).The Methodology of N-version Programming Software Fault Tolerance. IEEE Transactions in Software Engineering, 11(12), 1491- 501.

[Axw07] Axway. (2007). Synchrony Gateway. Retrieved October 5, 2008, from http://www.axway.com/products/synchrony_gateway.php, 2007.

[Bea03] BEA Systems.(2008). Retrieved October 5, 2008, from http://dev2dev.bea.com/webservices/WS-Acknowledgement-0_9.html.

[Bea98] BEA Systems.(1998). Web logic. Retrieved June 5, 2008, from http://www.bea.com/framework.jsp?CNT=homepage_main.jsp&FP =/content.

[Bea07] BEA Systems. (2008).BEA weblogic Integration 10.2. Retrieved June 15, from http://www.bea.com/framework.jsp?CNT=overview.htm&FP/ content/products/weblogic/integrate/.

[Bel64] Bell Aerosystems.(1964).Lunar landing research vehicle (LLRV) in flight. Retrieved October 5, 2008, from http://www.dfrc.nasa.gov/Gallery/ Photo/LLRV/HTML/ECN-541.html.

[Boe90] Boeing.(1990).777 Family. Retrieved October 5, 2008, from http://www.boeing .com/commercial/777family/background.html.

[Buc08a] Buckley, I. & Fernandez E.B. (2008a).A survey of fault tolerance patterns 1-16. Unpublished manuscript.

[Buc08b] Buckley, I. & Fernandez E.B. (2008b).Web services reliability patterns.1-17. Unpublished manuscript.

[Buc08c] Buckley, I. & Fernandez E.B. (2008). Web services security products and Tools .Section 2, 1-25. Unpublished manuscript

[Bus96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M.(1996). Pattern oriented software architecture: A system of patterns. West Sussex, England:Wiley.

[Cer04] Cerebit. (2006).InnerGuard. Retrieved April 19, 2008, from http://www.cerebit .com/Product_innerGuard.htm.

[Cis05] Cisco. (2005).Reactivity Gateways. Retrieved September 10, 2008. from .cisco.com/dlls/2007/corp_032107.html.

[Dan97] Daniels, F., Kim, K. &.Vouk, M. A .(1997.)The reliable hybrid pattern: A eneralized software fault tolerant design pattern. PLoP. 1-5. Retrieved April 9, 2008, from http://st-www.cs.uiuc.edu/users/hanmer/PLoP-97.html

[Dou03] Douglas,B.P.(2003).Real-time design patterns.Boston Ma: Addison Wesley Ltd.

[Ebx02] Oasis. (2002). ebXML messaging service TC 2.0. Retrieved May 29, 2008, from http://www.oasis-Open.org/committees/download.php /272/ebMS_v2_0.pdf.

[Ecl08] Eclipse. (2008).Web tools platform (WTP) project. Retrieved September 21, 2008, from http://www.eclipse.org/webtools/

[Ent03] Entrust. (2003).Entrust secure transaction platform. Retrieved November 1, 2008. from http://www.entrust.com/web-services-security/.

[Fer07a] Fernandez, E.B., & Larrondo-Petrie, M. M. (2007a). A methodology to build secure systems using patterns. 22nd Annual Computer Security Applications Conference (ACSAC):Works in Progress. 1-3.

[Fer07b] Fernandez, E.B., Yoshioka, N., Washizaki, H., & Jurjens, J. (2007b). Using security patterns to build secure systems.14th Asia-Pacific Software Engineering Conference (APSEC).1- 2.

[Fer08] Fernandez, E.B., & LaRed, M. D. (2008). Patterns for the secure and reliable execution of processes using security patterns to build secure systems. PLoP '08, 1-16. Retrieved November 6, 2008,http://hillside.net/plop/2008/

[Ferr98] Ferreira, L. L., Rubira, C. M. F.(1998) Reflective design patterns to implement fault tolerance. PLoP 98. 1-2. Retrieved February 20, 2008, from http://www.hillside.net/plop/plop98/final_submissions/.

[Feu05] Feuerlicht, G., & Meesathit, S.(2005).Towards software development methodology for web services. 3.Unpublished manuscript.

[For03] Forum Systems Inc. (2003).Forum presidio. Retrieved February 20, 2003, http://www.forumsys.com/products/ftpgateway/features.php.

[For04a] Forum Systems Inc. (2004a). Forum xwall is now forum sentry. Retrieved April 21, 2008, from http://www.forumsys.com/ products/xwall.php.

[For04b] Forum Systems Inc.(2004b).Forum sytems vulnerability containment.Retrieved April 21, 2008, from http://www.computerlinks.com/res/File/forum _systems/VulCon_Datasheet.pdf.

[Fuj04] Fujitsu Limited, et al. (2004). RM4GS overview. Retrieved June 12, 2008, from http://xml.coverpages.org/RM4GS-Overview20040305.pdf.

[Fuj08] Fujitsu Limited.(2007). Interchange. Retrieved January 30, 2008, from http://www.fujitsu.com/global/services/software/interstage, 2007.

[Gam94] Gamma, E.,Helm, R., Johnson, R.,& Vlissides.J. (1994). Design patterns: elements of reusable object-oriented software, Boston, Mass:Addison-Wesley.

[Gxs08]Gxs. (2005). Gxs. Retrieved September 30, 2008, from http://www.gxs.co.uk /products/technology/application_integrator.htm.

[Ham07] Hammer, R. S. (2007).Patterns for Fault tolerant Software. West Sussex, England: John Wiley & Sons Ltd.

[Hp08] Hewlett Packard.(n.d). HP Integrity nonstop servers. Retrieved September 9, 2008, from http://h20223.www3.hp.com/NonStopComputing/cache/ 307953-0-0-0-121.html.

[Hyf07] Hyfinity.(n.d.). Products. Retrieved September 11, 2008, from http://www. hyfinity.com/Products.html.

[Ion07] Iona.(2007).Artix. Retrieved September 11, 2008, from http://www.iona.com /products/artix/whats_new/welcome.htm.

[Ibm04] IBM.(2004).WebSphere datapower XML security gateway XS40. Retrieved September 11,2008,from http://www-.ibm.com/software/integration/data power/xs40/.

[Ibm07] IBM.(2007a). Tivoli. Retrieved September 11, 2008, from http://www01.ibm. com/software/tivoli/?pgel=ibmhzn&cm_re =masthead-_-products-_-sw-tivoli.

[Kes07] Kessler, G. (2007). An overview of TCP/IP protocols and the internet. Retrieved June 10, 2008, from http://www.garykessler.net/library/tcpip.html#TCP.

[Kim91] Kim, K.H, Hecht, M., Agron, J., & Hecht, H.(1991). A distributed fault tolerant architecture for nuclear reactor and other critical process control applications. 21st FTCS. 462-470.

[Lem01] Leme, N., Matins, E. & Rubira, C. (2001) .A software fault injection pattern system. PLoP 2001 Conference, 2-5. Retrieved May 10, 2008, from http://www.hillside.net/plop/plop2001/.

[Ley01] Leymann, F. (2001). Web services flow language (WSFL 1.0). Retrieved February 10, 2008, from http://xml.coverpages.org/WSFL-Guide-200110.pdf

[Ljs06] Ljosland, I. (2006).BUCS: patterns and robustness -experimentation with safety patterns in safety-critical software systems. Retrieved from Norwegian Universityof Science and Technology (NTNU).

[Maf96] Maffeis, S.(1996). The object group design pattern.USENIX Conference on Object Oriented Technologies, 4-7. Retrieved April May 21, 2008, from USENIX database.

[Mic08a] Microsoft. (2008a). Web matrix. Retrieved February 10, 2008, from http://msdn.microsoft.com/en-us/library/aa479004.aspx.

[Mic08b] Microsoft.(2008b).Web services enhancements. Retrieved February 10, 2008 http://msdn.microsoft.com/en-us/netframework/aa663324.aspx.

[Mic8c] Microsoft. (2008c).TrustBridge. Retrieved February 10, 2008 from http://articles .techrepublic.com.com /5100-10878_11-1054084.html.

[Mic08d] Microsoft. (2008d).MapPoint web services. Retrieved February 10, 2008 from http://msdn.microsoft.com/en-us/library/aa286513.aspx.

[Net02] Netegrity Inc. (2002).Netegrity announces TransactionMinder for authentication and authorization of web services. Retrieved February 12, 2008, from http://xml.coverpages.org/NetegrityTransactionMinder.html.

[Oas04] Oasis. (2004). Web services reliable messaging TC ss-reliability 1.1", Retrieved May 18, 2008, from http://docs.oasis-open.org/wsrm/ws-reliability/v1.1 /wsrm- ws_reliability-1.1-spec-os.pdf.

[Oas05] Oasis. (2005). Web service implementation methodology. Retrieved May 18, 2008, from http://www.oasis open.org/committees/download.php/13420/ fwsi-im-1.0-guidlines-doc-wd-publicReviewDraft_files.

[Oas07] Oasis. (2007).Web services reliable messaging (WS-Reliable Messaging) version1.1. Retrieved May 18, 2008, from http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01-e1.pdf.

[Ora07] Oracle.(2007).SOA suite. Retrieved July 18, 2008, from http://www.oracle.com/ technologies/soa/management- pack-soa.html, 2007.

[Pro07] Progress Actional. (2007).Progress actional for active SOA policy enforcement. Retrieved May 18, 2008, from http://www.actional.com/products/active-policy- enforcement/.

[Roh08] Rohati. (2008).Rohati products overview. Retrieved May 18, 2008, from http://www.rohati.com/products/index.php .

[Sap07] SAP.Community Network.(2007).SAP netweaver process integration 7.1. Retreived October18, 2008, from http://www 01.ibm.com/software /integration/wmq /index.html.

[Sar02] Saridakis, T. (2002) .A system of patterns for fault tolerance. EuroPLoP. 1-30. Retrieved May 19, 2008, from http://www.hillside.net/patterns/EuroPLoP 2002/.

[Sar03] Saridakis, T. (2003). Design patterns for fault containment. EuroPLoP. 1-5. Retrieved May 19, 2008, from http://hillside.net/europlop/europlop2003/ contents.

[Sar04] Sarvega. (2004).XML guardian gateway. Retrieved March 19, 2008, from http://www.itslabs.com/tests/its04003.jhtml.

[Sch06] Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F.& Sommerlad, P. (2006).Security Patterns: integrating security and systems engineering. West Sussex, England: John Wiley & Sons.

[Sch83] Schlichting, R. D. & Schneider, F. B. (1983). Fail-Stop processors: an approach to designing fault tolerant computing systems. ACM Transactions on Computing Systems, 1(3), 222-238.

[Sec07] Securent. (2007).The leader in entitlement management. Retrieved March 10,

2008, from http://www.securent.com/products/overview/.

[Smi08] Smith, G. E. (2008).Introduction to SOA Gateways: best practices benefits & Requirements. Retrieved June 30, 2008, from http://soanetworkarchitect. com/files/6585-6409/Best_Practices_SOA_Gateway_v2.pdf.

[Soa04] SOA Software. (2004).Digital evolution XML VPN appliances. Retrieved March 10, 2008, from www.soa.com.

[Sty08] Stylus Studio. (2008). Stylus studios 2008. Retrieved January 6, 2008, from http://www.stylusstudio.com/.

[Sun02] Sun Microsystems Inc. (2002). Web service processing and interaction models. Retrieved March 17, 2008, from http://java.sun.com/blueprints/using/ webservbp2.html.

[Sun08a] Sun Microsystems Inc. (2008).Metro web services overview. Retrieved March 17, 2008, from http://java.sun.com/webservices/.

[Sun08b] Sun Microsystems Inc. (2008).Glass fish. Retrieved March 17, 2008, from https://glassfish.dev.java.net/.

[Sun08c] Sun Microsystems Inc. (2008). Sun identity management. Retrieved March 17, 2008, from http://www.sun.com/software/products/identity/ds_solutions _your_environment.pdf

[Tic06] Tichy, M. (2006).Pattern based synthesis of fault tolerant embedded systems. 14th ACM SIGSOFT international symposium on foundations of software engineering, 2-4.

[Vor06]Vordel.(2006).Vordel XML gateway. Retrieved July 9, 2008, from com/products /vx_gateway/.

[Vor07] Vordel.(2007).Vordel XML firewall. Retrieved July 9, 2008, from http://www. vordel.com/products/vx_firewall/index.

[Web08] IBM.(2008).WebSphere MQ. Retrieved July 30, 2008, from http://www /software/integration/wmq/index.html.

[W3c07] W3C.(2007).SOAP version 1.2 part 1: messaging framework (second edition). Retrieved September 23,2008, from http://www.w3.org/TR/soap12-part1/.

[Xtr05] Xtradyne.(2005).Xtradyne WS-DBC-XML/SOAP Firewall. Retrieved February 3, 2008,from http://www.xtradyne.de/products/ws-dbc/ws-dbc.htm.