

**A Reengineering Approach to
Reconciling Requirements and
Implementation for Context - Aware
Web Services Systems**

Ph.D. Thesis

Jianchu Huang

**Software Technology Research Laboratory
De Montfort University**

2012

To my wife, Jing Wu and my daughter, Phoebe Huang,

my parents, Guiwen Huang and Yuyun Li,

for their love and support

Acknowledgement

I am grateful to everyone who has made this thesis possible. First and foremost, I would like to thank my first supervisor Prof. Hongji Yang, who has given me enormous support, guidance and encouragement. It has been an honour and a pleasure to be his PhD student.

Thanks must go to Prof. Hong Zhu, Dr. Francois Siewe, and Dr. Ali Al-Bayatti for being my examiners and providing many valuable suggestions on my PhD thesis. My research career will benefit tremendously from the research insight that they grant me.

I also want to thank the colleagues at De Montfort University providing me with their feedback, and establishing such a stimulating and friendly working atmosphere. Especially, Prof. Hussein Zedan for being my advisor, Dr. Martin Ward for being my second supervisor, Prof. Andrew Hugill, Dr. Feng Chen, Dr. Shaoyun Li, Dr. Jian Kang, Dr. Helge Janicke, Miss Yingchun Tian. It is fortunate for me to work and study with them.

On the personal front, I am thankful to my delightful wife Jing Wu for her patience, love, understanding and support, who takes good care of me and my little daughter – Phoebe, who never fails to bring joy to me and the family. When I was stuck in my research, Jing has been always there to comfort me and encourage me to go forward. Thanks to my father Guiwen Huang and mother Yuyun Li who have never failed to support to me throughout my life. Thanks also to my parents in law for their encouragement and patience for the past few years.

Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University, UK from May 2006 to January 2012. Apart from the degree that this thesis is currently applying for, no other academic degree or award was applied by me based on this work.

Abstract

In modern software development, the gap between software requirements and implementation is not always conciliated. Typically, for Web services-based context-aware systems, reconciling this gap is even harder. The aim of this research is to explore how software reengineering can facilitate the reconciliation between requirements and implementation for the said systems. The underlying research in this thesis comprises the following three components.

Firstly, the requirements recovery framework underpins the requirements elicitation approach on the proposed reengineering framework. This approach consists of three stages: 1) Hypothesis generation, where a list of hypothesis source code information is generated; 2) Segmentation, where the hypothesis list is grouped into segments; 3) Concept binding, where the segments turn into a list of concept bindings linking regions of source code.

Secondly, the derived viewpoints-based context-aware service requirements model is proposed to fully discover constraints, and the requirements evolution model is developed to maintain and specify the requirements evolution process for supporting context-aware services evolution.

Finally, inspired by context-oriented programming concepts and approaches, ContXFS is implemented as a COP-inspired conceptual library in F#, which enables developers to facilitate dynamic context adaption. This library along with context-aware requirements analyses mitigate the development of the said systems to a great extent, which in turn, achieves reconciliation between requirements and implementation.

Table of Contents

Acknowledgement	i
Declaration.....	ii
Abstract.....	iii
Table of Contents	iv
List of Tables	xi
List of Figures	xii
List of Acronyms	xiii
Chapter 1 – Introduction	1
1.1 AREA OF STUDY	1
1.2 PROBLEM STATEMENT.....	2
1.3 RESEARCH OBJECTIVES	6
1.4 RESEARCH METHODOLOGIES	7
1.5 RESEARCH QUESTIONS AND PROPOSITIONS	8
1.6 ORIGINAL CONTRIBUTIONS.....	11
1.7 CRITERIA FOR SUCCESS	12
1.8 THESIS ORGANISATION	13
Chapter 2 – Background and Related Research	15

Table of Contents

2.1 SOFTWARE REENGINEERING	15
2.1.1 <i>Legacy System</i>	16
2.1.2 <i>Reengineering Phases</i>	17
2.2 REQUIREMENTS ENGINEERING	18
2.2.1 <i>Requirements</i>	18
2.2.2 <i>Requirements Engineering Process</i>	19
2.2.3 <i>Requirements Engineering Challenges</i>	20
2.3 GOAL-ORIENTED REQUIREMENTS ENGINEERING.....	21
2.4 SERVICES EVOLUTION VS. REQUIREMENTS EVOLUTION.....	22
2.5 WEB SERVICES-BASED CONTEXT-AWARE SYSTEMS	25
2.5.1 <i>Context-Awareness</i>	25
2.5.2 <i>Web Services</i>	27
2.5.3 <i>Survey of Web Services-Based Context-Aware Systems</i>	28
2.6 CONTEXT-ORIENTED PROGRAMMING.....	30
2.7 SUMMARY	32
Chapter 3 – Proposed Framework	34
3.1 OVERVIEW	34
3.2 THE PROPOSED REENGINEERING FRAMEWORK AND APPROACH.....	36
3.2.1 <i>Services Candidate Discovery</i>	40

Table of Contents

3.2.2 Services Reimplementation	42
3.2.3 Services Integration	46
3.2.4 Forward Engineering in Proposed Reengineering Framework	47
3.2.5 The Differences and Consequences of the Proposed Reengineering Framework ...	50
3.3 SUMMARY	52
Chapter 4 – Requirements Recovery Framework for Services Candidate Discovery	55
4.1 OVERVIEW	55
4.2 CONTEXT-AWARE SYSTEM FRAMEWORK	56
4.2.1 The Problem	56
4.2.2 Layered Conceptual Framework for Context-Aware Systems	57
4.3 REQUIREMENTS RECOVERY FRAMEWORK AND APPROACH	65
4.3.1 Requirements Recovery – In a Nutshell	65
4.3.2 Requirements Recovery Framework	66
4.3.3 Requirements Elicitation Approach	69
4.3.4 Requirements Elicitation Approach for A Location-Aware System - A Brief Example	70
4.4 SUMMARY	73
Chapter 5 – Context-Aware Services Requirements Model and Requirements Evolution Model	75
5.1 OVERVIEW	75

Table of Contents

5.1.1 <i>The Problem</i>	75
5.1.2 <i>Background</i>	76
5.2 CONTEXT-AWARE SERVICES REQUIREMENTS MODEL.....	79
5.2.1 <i>Concepts for Context-Aware Services</i>	79
5.2.2 <i>Customised Derived Viewpoints</i>	81
5.2.3 <i>Requirements Model for Context-Aware Services</i>	84
5.3 REQUIREMENTS EVOLUTION MODEL FOR CONTEXT-AWARE SERVICES EVOLUTION.....	90
5.4 THE RELATION BETWEEN REQUIREMENTS EVOLUTION AND SERVICES EVOLUTION	94
5.5 AN EXAMPLE	95
5.6 SUMMARY	97
Chapter 6 – Context-Aware Web Services Reimplementation with ContXFS Support.....	99
➤ <i>To describe the requirements for the reimplementation</i>	99
➤ <i>To describe the architecture design for the reimplementation</i>	99
➤ <i>To discuss the reimplementation concerns and strategies</i>	99
➤ <i>To introduce the F# language and the development tools</i>	99
➤ <i>To introduce context-oriented programming and F# library ContXFS</i>	99
➤ <i>To demonstrate an example of such services reimplementation within the proposed reengineering framework and the application of ContXFS</i>	99
6.1 OVERVIEW	99
6.1.1 <i>The Problem</i>	99

Table of Contents

6.1.2 <i>The Background</i>	100
6.2 REIMPLEMENTATION REQUIREMENTS.....	102
6.2.1 <i>Requirements for Implementing Context-Aware Web Services</i>	103
6.2.2 <i>Requirements Mapping</i>	106
6.3 ARCHITECTURE DESIGN.....	108
6.3.1 <i>Client-Side</i>	108
6.3.2 <i>Web Services Applications</i>	109
6.3.3 <i>Server-Side</i>	110
6.4 REIMPLEMENTATION CONCERNS AND STRATEGIES	113
6.4.1 <i>Reimplementation Concerns</i>	113
6.4.2 <i>Reimplementation Strategies</i>	117
6.5 INTRODUCTION OF F# AND DEVELOPMENT TOOLS	121
6.5.1 <i>Background</i>	123
6.5.2 <i>Most Appreciated Features in F#</i>	124
6.6 CONTEXT-ORIENTED PROGRAMMING AND MAIN FEATURES	143
6.6.1 <i>Context-Oriented Programming</i>	144
6.6.2 <i>COP Main Features</i>	144
6.7 CONTEXT-ORIENTED PROGRAMMING IN F#.....	146
6.7.1 <i>Overview of ContXFS Development</i>	146

Table of Contents

6.7.2 Behavioural Variations in ContXFS	147
6.7.3 Context Switching On-The-Fly.....	150
6.7.4 Layers in ContXFS.....	151
6.7.5 Layers Composition in ContXFS.....	152
6.8 AN EXAMPLE OF REIMPLEMENTATION	153
6.9 SUMMARY	155
Chapter 7 – Case Study	158
7.1 OVERVIEW	158
7.2 OPENMOBSTER	161
7.2.1 Overview of Requirements Recovery Framework (RRF) Approach	161
7.2.2 RRF Approach on Openmobster.....	162
7.3 THE GEOLOCATION API	166
7.3.1 Overview of CASRM and Requirements Evolution Model.....	166
7.3.2 Requirements Evolution Model for The Geolocation API Applications	168
7.4 GEOCLUE	171
7.5 CONTEXTCHAT.....	173
7.5.1 Overview of COP and F# Agent-Based Programming Model.....	174
7.5.2 Reimplementation via ContXFS and F# Agent-Based Messaging Techniques	175
7.5.3 Quantitative Experiments.....	176

Table of Contents

7.6 DEVELOPMENT TOOLKIT	179
7.7 SUMMARY	180
Chapter 8 – Conclusions and Future Work.....	182
8.1 SUMMARY OF THESIS	182
8.2 ORIGINAL CONTRIBUTIONS REVISITING	184
8.3 EVALUATION	186
<i>8.3.1 Answering Research Questions.....</i>	<i>186</i>
<i>8.3.2 Research Proposition Revisiting.....</i>	<i>189</i>
<i>8.3.3 Revisiting Criteria of Success.....</i>	<i>191</i>
8.4 LIMITATIONS	193
8.5 FUTURE WORK	194
References	196
Appendix A Prototype Implementation of ContXFS and Its Test Samples	212
Appendix B List of Publications	217

List of Tables

Table 3.1 A Sample of Refined Context-Aware Web Services Requirements	43
Table 4.1 Content of SCI and REQ in Services Pattern Module.....	71
Table 4.2 An Updated Knowledge-Based Library (KBL).....	72
Table 5.1 Requirements Modelling Techniques	78
Table 5.2 Definitions of Context-Aware Service from Both Perspectives.....	80
Table 5.3 Customised Derived Viewpoints from Users and Developers	82
Table 5.4 A fragment of Knowledge-Based Library (KBL)	96
Table 5.5 Services Requirements of ContextChange	96
Table 6.1 A Sample of Requirements for Context-Aware Web Services	105
Table 6.2 Reflected Requirements for Development	107
Table 6.3 F# Features and Advantages.....	116
Table 7.1 Attributes of Each Case Study.....	160
Table 7.2 A Snapshot of Initial SPM for Openmobster.....	163
Table 7.3 A Snapshot of Updated Content of KBL.....	164
Table 7.4 Services Requirements of PositionInformation.....	169

List of Figures

Figure 2.1 A General Model for Software Reengineering [124].....	18
Figure 2.2 A High-Level General Framework for Context-Aware Systems....	26
Figure 3.1 CAWSRF Approach.....	39
Figure 4.1 The Proposed Layered Conceptual Framework for Context-Aware Systems	60
Figure 4.2 Requirements Recovery Framework	68
Figure 5.1 Context-Aware Services Requirements Model (CASRM)	85
Figure 5.2 Requirements Evolution Model	91
Figure 6.1 Proposed Architecture Design.....	112
Figure 6.2 Events as First-Class Values	126
Figure 6.3 WPF in F#	142
Figure 6.4 Variations Composition of Variation_A and Variation_B.....	152

List of Acronyms

<i>ARRE</i>	<i>Associate Requirements Repository Engine</i>
<i>AST</i>	<i>Abstract Syntax Tree</i>
<i>CASRM</i>	<i>Context-Aware Services Requirements Model</i>
<i>CAWSRF</i>	<i>Context-Aware Web Services Reengineering Framework</i>
<i>CCO</i>	<i>Communication Computation Overlap</i>
<i>CLR</i>	<i>Common Language Runtime</i>
<i>ContXFS</i>	<i>A Context-Oriented Programming Library in F#</i>
<i>COP</i>	<i>Context-Oriented Programming</i>
<i>DSLs</i>	<i>Domain-Specific Languages</i>
<i>FCA</i>	<i>Formal Concept Analysis</i>
<i>HB-CA</i>	<i>Hypothesis-Based Concept Assignment</i>
<i>HTTP</i>	<i>Hypertext Transfer Protocol</i>
<i>KBL</i>	<i>Knowledge-Based Library</i>
<i>LINQ</i>	<i>Language Integrated Query</i>
<i>LIS</i>	<i>Legacy Information Systems</i>
<i>MPI</i>	<i>Message Passing Interface</i>
<i>P2P</i>	<i>Peer-to-Peer</i>
<i>QoS</i>	<i>Quality of Service</i>
<i>RPC</i>	<i>Remote Procedure Call</i>
<i>RRF</i>	<i>Requirements Recovery Framework</i>
<i>SaaS</i>	<i>Software-as-a-Service</i>

List of Acronymss

<i>SCI</i>	<i>Source Code Information</i>
<i>SOA</i>	<i>Service-Oriented Architecture</i>
<i>SOAP</i>	<i>Simple Object Access Protocol</i>
<i>SPM</i>	<i>Services Pattern Module</i>
<i>UML</i>	<i>Unified Modeling Language</i>
<i>WPF</i>	<i>Windows Presentation Foundation</i>
<i>WSDL</i>	<i>Web Services Description Language</i>
<i>XML</i>	<i>Extensible Markup Language</i>

Chapter 1 – Introduction

Objectives

- To set the area of study and introduce the problem statement
- To describe the research objectives and the research methods
- To raise research questions and develop research propositions
- To state the thesis contributions and the criteria for success
- To outline the structure of the thesis

1.1 Area of Study

Any successful system is subject to evolution so that it survives beyond its normal environments. Continuous modifications to software system have to perform in order that new software functional and non-functional requirements are fulfilled. Typically, the changes of non-functional requirements may recur when the legacy software system entails an adaptation of a new computing environment. The ever-increasing cost associated with software maintenance vindicates the fact that software is difficult to maintain. Nevertheless, software evolution [13, 72, 73, 124] is a way out. Being a preferred name to software maintenance, software evolution can be seen as a sequence of software reengineering [13] that embraces reverse engineering [27], functional restructuring, and forward engineering. Specifically, during the conventional activities in a software reengineering process, software is firstly comprehended

to create a higher level representation via identifying system's components and their relationships from the code level; secondly, depending on the categories of solution for legacy software system problems, program transformation might be carried out via restructuring or refactoring; finally, the software will be implemented by resisting the traditional software development lifecycle. In fact, further techniques may be utilised during the three steps above. For instance, programming comprehensive and formal methods could be used to assist reverse engineering work such as specifying and verifying the legacy software systems.

Emerging computing requirements drive the demand of software evolution. In the recent years of research, typically, context-awareness and Web services-based computing post a great challenge for software evolution. In a largely scalable Web services-based context-aware environment, context-awareness is concerned with reasoning about the surrounding well-defined context and adapting the interpreted services accordingly (almost) on the server-side, and finally distributing the services to clients in a reliable way through trustworthy network protocols. The underlying development challenge of such system lies in not only the agile yet concise implementation of context-awareness that entails well-defined context and context modelling (what functional requirements the system will perform when context information varies), but also the development of Web services-based computing that requires high reliability and performance (how non-functional requirements the system will meet).

1.2 Problem Statement

Conventionally, reverse engineering focuses on the code level analysis with little further investigation on recovering stakeholders' goals or requirements

towards the subject system [128, 129]. The basic aim of reverse engineering is to identify system's components and their relationships, and create a higher abstract representation of the systems from source code. In general, the artifacts, extracted via e.g., program slicing [119, 120, 121] and concept assignment [15], are code segments with little implication of functional and non-functional requirements behind them because code-related segments are not always kept 'in one piece' which breaks the internal link of stakeholders' requirements. As software continues to evolve, increasing software reengineering activities will eventually deepen the understanding curve of the evolved requirements, and which in turn leads to increasing difficulty in eliciting the obscure requirements behind the modified code fragments.

On the other hand, requirements engineering [26, 67, 85, 102, 131] accommodates many sound requirements elicitation methods for this issue, for example, goal-oriented requirements elicitation method [28, 66] and scenario-based requirements elicitation method [78]. In spite of a great deal effects that have been made on exploring the questions such as "why the software is needed", tiny attention spans are concentrated on constraints (e.g., implementation requirements) [58]. Inevitably, software developers will potentially face a choice of selecting proper programming languages for reimplementing. Instead of choosing those mainstream languages within the object-oriented programming paradigm, a general purpose programming language with attributes that facilitate Domain Specific Language (DSL) design may be more appropriate than the former. For example, an implementation of context-oriented programming [56] in Erlang can be used to address implementation issues of context-awareness at run time. The fact that many existing Web services-based context-aware systems are implemented in object-oriented languages motivates us to seek alternative.

In effect, amidst software evolution of Web services-based context-aware systems, the gap between software requirements and implementation is becoming less likely conciliated. This proposed work is to identify the hidden issues that lead to this fundamental gap. With regard to Web services-based context-aware systems, functional and non-functional requirements are needed to be clarified in the first place. Such requirements consist of the recovered requirements from the source code and new requirements. Code-related artifacts recovered from the source code can be used to assist the examination whether the existing software system fulfils the current requirements as well as the investigation whether current programming languages are capable of addressing the programmatic problems without plethoric convoluted development. Once the combination of requirements and code-related artifacts are available, reimplementation will be carried out to mark once software evolution. During this process, implementation issues and strategies are taken into account. In practice, language choice is one of the most critical implementation issues along with required platform and standard, because the programming language itself may deeply impede software developer's time and effects on tackling the development tasks. Implementation strategies may vary depending on the specific requirements and architectural design. It is preferable for those general propose programming languages that embrace desired programming models which can fulfil the relevant implementation strategies. For instance, to reduce communication overhead of Web services and applications, Message Passing Interface (MPI) [82] programming model is an efficient choice. Therefore, those languages which embrace similar programming models are very good candidates.

Thus, on top of the traditional reengineering framework, a novel software reengineering framework for context-aware Web services-based systems is introduced. The ultimate goal of the proposed framework is to reconcile the

underlying gap between requirements and implementation for the said systems.

The proposed software reengineering process comprises the following core steps:

- Legacy System Assessment which decides if the legacy system is applicable.
- Services Candidate Recovery where requirements and code segments are recovered.
- Services Reimplementation where a context-oriented programming approach is applied.
- Services Integration where existing and newly built services are composed.

This proposed framework is founded on a subset of frameworks and models, as well as language support of a context-oriented programming approach, i.e., a requirements recovery framework which underpins the requirements elicitation approach, a context-aware service requirements model that is a users' and developers' derived viewpoint, a requirements evolution model which manages the evolved requirements for services evolution, requirements mapping for finding the right programming language candidates and an architectural design model on which services reimplementation is based, and a context-oriented programming library – ContXFS implemented in the language F# [45].

1.3 Research Objectives

The objectives of the thesis can be summarised as follows:

- To build a reengineering framework for Web services-based context-aware systems
- To create a requirements recovery framework for requirements elicitation approaches
- To design a context-aware service requirements model and a requirements evolution model to support context-aware services evolution
- To develop a functional-first, context-oriented programming library to facilitate the reimplementations concerns and strategies

The basic idea of the proposed research is to create a reengineering framework for Web services-based context-aware systems to mitigate the increasing gap between requirements and implementation during software evolution. It is comprehensive, which covers redevelopment in the software reengineering process; it is inspiring, where contributions can be made by developing a new theory, framework, model or methodology. Nevertheless, considerable software reengineering works remain in reverse engineering or restructuring steps without further carrying out reimplementations to fulfil the whole reengineering process. Our development of the said system adds empirical research to the forward engineering step. Hence, this proposed work is a completed software reengineering research that is practical and academically rigorous.

1.4 Research Methodologies

The research field in this thesis falls into software engineering which aims to generate the successful production of software. As the natural characteristics of software engineering, constructive research is predominantly applied in computer science. However, the new solution to problems always entails empirical research. In other words, the combination of constructive and empirical research enables academically rigorous and industrially practical.

Therefore, the basic research methodologies employed in this thesis are classified as follows:

- **Process:** A process methodology is utilised to understand the processes applied to accomplish tasks in software engineering. This methodology is widely used in the areas of software reengineering where a typical reengineering framework often consists of several phases to fulfil the relevant reengineering tasks.
- **Model:** The model methodology, a means to defining an abstract model for a real system, plays a central role in this work. Modelling allows better understandings of the system. The proposed work develops a requirements model for recovering requirements. Specifically, it can guide requirements elicitation, provide a measure of completeness of the elicitation, and visualise the requirements.
- **Classification:** The concept of classification underpins considerable tasks related to this work. In software engineering, various reengineering approaches are employed in terms of the functionalities and features the system may own. For example, approaches may be either concerned with architecture design or related to programming

language support during the forward engineering step. In this work, language support is investigated further.

- **Quantitative and Qualitative Methodologies:** This research work mirrors qualitative methods by discussing wh-related questions (e.g., why and what) in an exploratory phase, whilst how-related questions (e.g., how many) reflect quantitative methods in an evaluation phase.

1.5 Research Questions and Propositions

Research questions motivate this proposed work and guide the structure of the research work. The principle research question in this work is described as follows:



How can a software reengineering approach be developed in order to reconcile the gap between requirements and implement for Web services-based context-aware systems?

To answer the above question, a subset of smaller research questions is defined below:

REQ1: *What does the context-aware Web services candidate discovery recover?*

- What is the common architectural design of context-aware systems?
- How may requirements be extracted from source code in legacy systems?

- How may other reengineering tasks benefit from the recovered requirements-related and code-related artifacts?

REQ2: *Why non-functional requirements (i.e., qualities and constraints) are so important?*

- How may software evolution be hindered by not fully evaluating implementation decisions during the reengineering process?
- How may constraints be discovered during the reengineering process?
- How may software developer's time and effects be impeded by inappropriate language choice?

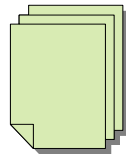
REQ3: *How is services reimplementation carried out?*

- What are the requirements for services reimplementation?
- How may the architectural design model be developed?
- What are the reimplementation concerns and strategies?

REQ4: *How may domain specific language help in the reimplementation process?*

- Which language and language paradigm may be suitable for building a domain specific language?
- Why may context-oriented programming be able to address the need for context-aware adaption?
- How may context-oriented programming library be developed?

A range of research propositions is developed to address these research questions. The underlying proposition of this thesis can be presented as follows:



Requirements elicitation during reverse engineering and domain specific language support during forward engineering can be combined in order to reconcile requirements and implementation for the said systems.

The principle proposition above is examined by requirements recovery and services reimplementing in the course of the overall software reengineering process. A subset of more detailed propositions can be further described as follows:

PRO1: A combination of viewpoints-based requirements, as well as code-related artifacts can be recovered from legacy systems. This proposition can be tested by building a requirements recovery framework along with a set of well-established reverse engineering techniques.

PRO2: The language choice makes a profound impact on the structure of the development solutions as well as how software developers think of the implementation issues. This proposition can be tested by examining the said legacy systems implemented in mainstream languages belonging to object-oriented programming paradigm. It is the fact that intricate code excessively exists due to convoluted development for the fulfilment of non-functional requirements particularly.

PRO3: Raising the importance of choosing language(s) for implementation.

This proposition can be tested via a comparison of various programming features, requirements mapping, realising the architectural design model, finally taking into account the reimplementation concerns and strategies.

PRO4: DSL allows software developers to quickly and efficiently develop a software system which lead to easy understanding and reasoning about, as well as low maintenance cost. This proposition can be tested by ContXFS, a context-oriented programming library in F#, enables software developers to facilitate the implementation of context-awareness at run time especially.

1.6 Original Contributions

A novel reengineering framework approach is proposed with a set of frameworks and models including requirements recovery framework, a context-aware service requirements model, a requirements evolution model, and a context-oriented programming library. The primary contributions of this thesis are:

C1: A novel software reengineering framework is created to mitigate the increasing gap between requirements and implementation for the Web services-based context-aware systems.

C2: Methodologies for eliciting context-aware service requirements in the requirements recovery framework.

C3: A context-aware service requirements model is proposed to extract existing requirements from source code and allows for conveniently reconstructing new context-aware service requirements primarily based on users' and developers' customised derived viewpoints.

C4: A requirements evolution model is built to manage evolved requirements in order to support context-aware services evolution.

C5: A requirements mapping and a technique of choosing a programming language candidate are presented.

C6: A functional-first programming approach to context-oriented programming library is implemented in F# that natively supports concurrent and parallel programming in distributed environments.

C7: An investigation of the effectiveness of the functional approach that supports context-aware adaption at run time.

C8: A set case studies is carried out to evaluate the overall framework approach.

1.7 Criteria for Success

The following criteria are given to judge the success of the research work proposed in this thesis:

- The proposed approach should be able to reconcile the underlying gap between requirements and implementation for the said systems.
- The requirements recovery framework approach should be able to elicit users' requirements and constraints that reflect the original requirements.
- The context-aware service requirements model should be able to reconstruct new requirements combining with existing requirements.

- The requirements evolution model should be able to manage the services requirements and context in a way to support services evolution.
- The architectural design model should be able to uncover reimplementations concerns and strategies.
- The ContXFS should be able to address the reimplementations issues and provide programmatic supporting for development.
- The implementation of a Web services-based context-aware system should be able to realise the architectural design and meet the combined requirements such as context-awareness, concurrency, reliability, and scalability etc.

1.8 Thesis Organisation

The rest of the thesis is structured as follows:

Chapter 2 provides an overview of a wider related research background in software reengineering and requirements engineering, and reviews a relevant literature on goal-oriented requirements engineering, requirements elicitation, requirements modelling, services evolution and requirements evolution, Web services-based context-aware systems, and context-oriented programming.

Chapter 3 introduces the overall reengineering framework and its approach, as well as further depicts the services candidate discovery, services reimplementations, and services integration.

Chapter 4 firstly, describes the layered conceptual framework for context-aware systems; secondly, describe the requirements recovery framework and

the associated approach; and finally, demonstrates an intermediate result of the framework approach on a location-aware system.

Chapter 5 firstly, depicts the context-aware service requirements model; secondly, presents the requirements evolution model; thirdly, discusses the relation between requirements evolution and services evolution; and finally, shows an example of the model of requirements evolution.

Chapter 6 firstly, discusses the requirements for the reimplementation; secondly, describes the architecture design for the reimplementation; thirdly, discusses the reimplementation concerns and strategies; then introduces F# and its related programming features for reimplementation and ContXFS as a library in F# is developed to allow for context-oriented programming; and finally, demonstrates an example of such services reimplementation with ContXFS support via the proposed reengineering framework approach.

Chapter 7 presents four case studies with different focuses to evaluate the overall proposed reengineering approach.

Chapter 8 draws a conclusion in terms of the proposed frameworks and approaches, as well as outlines the limitations. The prospective further work is also discussed. Typically, the research questions will be revisited and answered.

Appendix A is the prototype implementation of ContXFS and its testing samples as a guide to demonstrate the ways of using this library to facilitate the implementations of other more sophisticated agents. ContXFS suggests that the reimplementation strategies embrace an agent-based programming model and ContXFS applications.

Appendix B lists all the associated publications written by the author in the course of the PhD study.

Chapter 2 – Background and Related Research

Objectives

- To provide an overview of software reengineering and requirements engineering
- To survey literature on goal-oriented requirements engineering
- To survey literature on services evolution and requirements evolution
- To survey literature on Web services-based context-aware systems
- To survey literature on context-oriented programming

2.1 Software Reengineering

On the day new a software system is put to work, it is certain that it will become a legacy system one day. Legacy system poses many conventional challenges [12, 81, 97, 100] to software maintainers. Nevertheless, in order to reduce cost of software development, organisations have to maximise the benefits from legacy assets (software system). Thus, maintaining functionalities and keeping up with changing business or technical conditions are considered as two important and urgent tasks.

2.1.1 Legacy System

As legacy software systems no longer meet the needs from customer's requirements, emerging operating software and hardware environments, they are subject to evolve. The maintenance scope has to cover not only maintenance of the existing functions, but also modifications to the current architecture and functions so that adding requirements will be fulfilled. In addition to such changes, non-functional changes may also be considered typically when software system entails adaption of a new computing environment.

Bennett defined legacy systems informally as “large software systems that we don't know how to cope with but that are vital to our organization [12]”, while Brodie defined it as “any information system that significantly resists modification and evolution” [18]. Whichever, a legacy system is the one that is still valuable, but is difficult to maintain.

Legacy Information Systems (LIS) are currently posing numerous problems to their host organisations. The most serious of these problems are [16]:

- LISs usually run on obsolete hardware that is slow and expensive to maintain.
- Software maintenance can also be expensive, because documentation and understanding of system details is often lacking and tracing faults is costly and time consuming.
- A lack of clean interfaces makes integrating LISs with other systems difficult.
- LISs are also difficult, if not impossible, to extend.

Bennett [13] et al. pointed out: 1) current software with middleware support or even within an enterprise framework is likely to be far more difficult to address; 2) legacy software is not so much a technological problem as an organisational and management problem: solutions need to be addressed at a higher level of abstraction than the software.

2.1.2 Reengineering Phases

Software reengineering, motivated by the need for new user-required functionalities, is an important and promising approach to tackle legacy system evolution problems. It is widely accepted that the process of software reengineering generally includes three stages: 1) reverse engineering, 2) functional restructuring, and 3) forward engineering. Each step carries out different tasks and purposes. At large, software evolution can be regarded as a process of conducting repeated software reengineering.

According to [27], Chikofsky et al. gave the following definitions:

- **Reengineering** is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.
- **Reverse engineering** is the process of analysing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or higher level of abstraction
- **Restructuring** is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behaviour (i.e., functionality and semantics); Refactoring [87] is an object-oriented variant of restructuring that the transformation happens at different abstraction levels, i.e., "the process

of changing a object-oriented software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure” [44]. Having said that, refactoring can be also used for other programming language paradigms [80].

- **Forward engineering** is the traditional process of moving from high-level abstractions and logical, implementation-independent design to the physical implementation of a system.

In terms of the above definitions, Figure 2.1 shows a general model for software engineering in the course of software evolution.

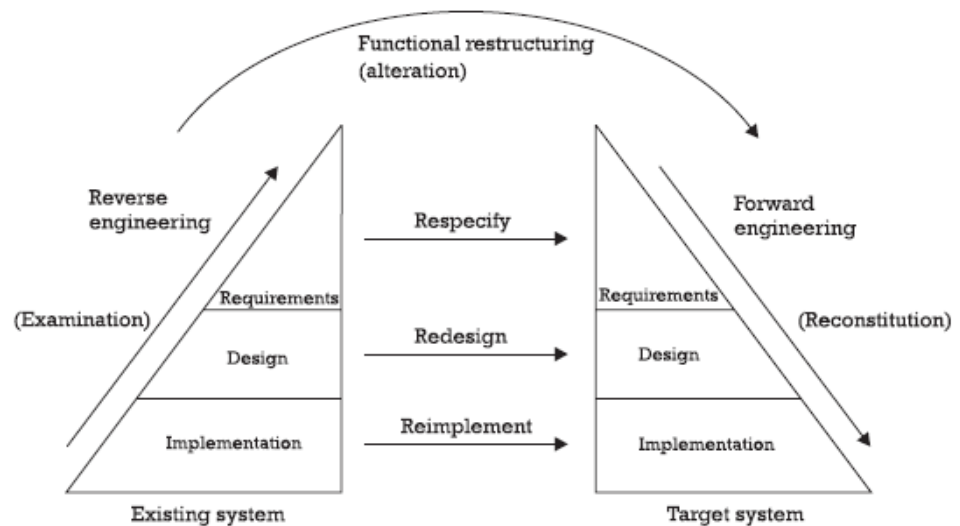


Figure 2.1 A General Model for Software Reengineering [124]

2.2 Requirements Engineering

2.2.1 Requirements

For many years, software systems were successfully created without the participation of requirements engineers. However, with the increasingly rapid

software development, software specification or requirements engineering is becoming more and more important. The success of a software system is subject to how well it meets the needs of its users and its running environment. Requirements analysis is the first phase in the software development life cycle to study software requirements, i.e., what the system will do. The IEEE Computer Society defines a requirement [62] as “a condition or capability needed by a user to solve a problem or achieve an objective”, or “a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents”. The set of all requirements establishes the foundation for subsequent development of the system.

2.2.2 Requirements Engineering Process

The scope of requirements engineering [131] is “the branch of systems engineering concerned with real-world goals for, services provided by, and constraints on a large and complex software-intensive system. It is also concerned with the relationship of these factors to precise specifications of system behaviour, and to their evolution over time and across system families.” The core processes involved requirements engineering is composed of the following steps [67]:

- **Domain Analysis:** the existing system in which the software should be built is studied. The relevant stakeholders are identified and interviewed. Problems and deficiencies in the existing system are identified; opportunities are investigated; general objectives on the target system are identified.
- **Elicitation:** alternative models for the target system are explored to meet such objectives; requirements and assumptions on components of such models are identified, possibly with the help of hypothetical

interaction scenarios. Alternative models generally define different boundaries between the software-to-be and its environment.

- **Negotiation and agreement:** the alternative requirements/assumptions are evaluated; risks are analyzed; ‘best’ tradeoffs that receive agreement from all parties are selected.
- **Specification:** the requirements and assumptions are formulated in a precise way.
- **Specification analysis:** the specifications are checked for deficiencies (such as inadequacy, incompleteness or inconsistency) and for feasibility (in terms of resources required, development costs, and so forth).
- **Documentation:** the various decisions made during the process are documented together with their underlying rationale and assumptions.
- **Evolution:** the requirements are modified to accommodate corrections, environmental changes, or new objectives.

2.2.3 Requirements Engineering Challenges

Cheng and Atlee draw attention to nine requirements engineering research hotspots [26], and claim that the solutions to those hotspots are likely to have the greatest impact on software-engineering research and practice. Six of them are future grand challenges, the other three hotspots focus on extending and maturing existing technologies to improve requirements engineering methodologies and requirements reuse and on increasing the volume of evaluation-based research. Software scale is the first future challenge that highlights the ‘scale factors’ such as complexity, degree of heterogeneity, sensor numbers, and decision-making nodes and so on. These factors are

becoming common in the Web services-based context-aware systems in the Cloud; for example, complexity can be referred to services implementation of parallelism or asynchronous computations; Cloud provides a high variety of services for heterogeneous users and devices; context-awareness entails a large scale of sensor deployment; finally, decentralised decision-making nodes share part of the burden from server-side computations and in turn, deliver faster services to end-users.

2.3 Goal-Oriented Requirements Engineering

Conventionally, requirements elicitation, as a means to identifying system boundaries, is one of the most important activities in requirements engineering. Requirements elicitation process consists of data interpreting, analysing modelling and validating, whereas, goal-oriented requirements engineering is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analysing, negotiating, documenting, and modifying requirements [69].

Oyama et al. [89] develop a context-aware goal elicitation process by exploring the aspects of data, information, knowledge and wisdom. The goal elicitation process is composed of conceptualisation for a service problem, goal identification, and conceptualisation for a service issues. In their work, stakeholders' intentions are defined as constrained sequences of user events to achieve a goal, whereas goals in the context are defined as the steady states of the system. A healthcare system is given to show the feature of intention changes as the users' intention is highly relevant to their health situation, which is observable from the contexts of physiological data.

Finkelstein et al. [43] refer changing context and changing requirements as two main challenges of requirements engineering in context-aware services. They

propose a reflection-based framework for requirements engineering for context-aware services. This framework views reflection as a mechanism instead of a goal. The mechanism is for manipulating highly-dynamic services in a clean and consistent way and eventually able to dynamically adapt themselves to changing context and changing requirements. To summarise, this work focuses on maintaining context representation of system behaviour at runtime.

Yu et al. [128] apply reverse engineering to source code to recover requirement structures. In their framework, the first step of their approaches relies heavily on well-structured comments during the code refactoring; the second step is converting the refactored code into an abstract structured program; the third step is extracting a goal model from abstract syntax tree (AST); the fourth step is identifying soft goal (i.e., non-functional requirements). But this framework does not extend to other reengineering activities other than reverse engineering.

Tun et al. [112] present an approach by applying concept assignment to recover structures in the problem context. Their approach contains four steps: extracting solution structures from sources; performing problem structures analysis; computing a similarity metrics between problem structures, and finally assessing new requirements based on the similarity metrics. This work does not take into account the potential conflicts between users' requirements and constraints.

2.4 Services Evolution vs. Requirements Evolution

Software evolution consists of a series of software reengineering tasks. Its aim is to implement and revalidate the possible major changes to the system to

satisfy new requirements. On the other hand, services are subject to changes in order to meet new requests. Services of context-awareness are able to adapt themselves to changing context. Services users can perceive the behaviours of software system physically. To distinguish the terms of software evolution and service evolution can be from various stakeholders' perspectives [20], for instance, from users' perspective, service evolution is used to highlight the nature of computer based applications to prevail nowadays; and from developers' perspective, software evolution is used to emphasise the mechanism for evolving computer programs as enabler of service evolution.

Papazoglou [90] classifies two types of service changes, i.e., shallow changes, where the change effects are localised to a service or are strictly restricted to the clients of that service, for example, changes on the structural level and business protocol changes; and deep changes, these are cascading types of changes which extend beyond the clients of a service possibly to entire value-chain, i.e., clients of these service clients such as outsourcers or suppliers, for instance, operational behaviour changes and policy induced changes.

Chang et al. [20] present a situation-theoretic approach to human-intention-driven service evolution in context-aware service environments. Other than giving a definition of situation which is rich in semantics and useful for modeling and reasoning human intentions and a definition of intention that is based on the observations of situations, they also distinguish software evolution and services evolution in terms of stakeholders' perspectives. To model and infer human intentions, they also propose a computational framework that supports detecting the desires of an individual and capturing the corresponding context values through observations.

Requirements evolution is still a research topic that somehow is not drawn much attention in requirements engineering community [38], even though

Cheng and Atlee [26] suggest that its importance is rising. Much research on evolving requirements still remains at the initial stage in software lifecycle.

The challenge of requirements evolution had been first comprehensively discussed by Harker et al [55]. They concentrate on the structure of requirements and classify stable and changing requirements into the followings types: Enduring Requirement (technical core of the business origin), Mutable Requirement (environmental turbulence origin), Emergent Requirement (stakeholder engagement in requirements elicitation origin), Consequential Requirement (system use and user development origin), Adaptive Requirement (situated action and task variation origin) and Migration Requirement (constraints of planned organisational development origin).

Adopting Formal Concept Analysis (FCA), Fabbrini et al. [40] depict an approach to improving requirements evolution management by making more systematic and effective the identification of semantic inconsistencies between different stages of requirements evolution. The process for validating evolving requirements is done via an FCA-based requirements consistency assessment. In the process, they focus on the source-outcome relationship between requirements belonging to two different evolutionary stages of the specifications.

Ernst et al. [38] predict that software of the future will consist of not only code and documentation, but also requirements and other types of models representing design, functionality and variability. They also point out important reasons why requirements evolution is about to become a focal point for research activity in requirements engineering.

By investigating the uncertain validity of requirements reengineer's assumptions as another cause of requirements evolution which can be divided into types, i.e., autonomic and designer-supported requirements evolution, Ali

et al. [4] describe an approach to monitor the assumptions in a requirements model at runtime and to evolve the model to reflect the validity level of such assumptions. They view requirements evolution as a continuous movement from assumptions-based requirement to reality-based ones.

Felici [42] investigates the current understanding of requirements evolution and propose a formal framework for requirements evolution.

Lormans [76] develop a requirements engineering framework that structures the process of requirements evolution, and a methodology that improves the traceability and monitoring of requirements.

2.5 Web Services-Based Context-Aware Systems

2.5.1 Context-Awareness

Context-awareness is concerned with reasoning about the surrounding well-defined context and adapting the interpreted services accordingly. Depending on the running environment the context-aware system is in, the services can be distributed via a network protocol or locally. Each of context-aware system architectures comes with a context-model of representing and sharing data. The architecture of a context-aware system is mainly shaped by the context acquisition approach.

Research with respect to context-awareness may include the followings: definitions of context [1, 34], context acquisition and representation [133], context modelling and reasoning [14], context interpreting [9]. Most modern context-aware architectures are middleware-based or context server-based. With such architecture, context-aware systems can be implemented in many ways. Figure 2.2 represents a layered conceptual framework for modern

context-aware systems [9]. Although it is a conceptual framework, it contains the majority of the key research topics within context-awareness.

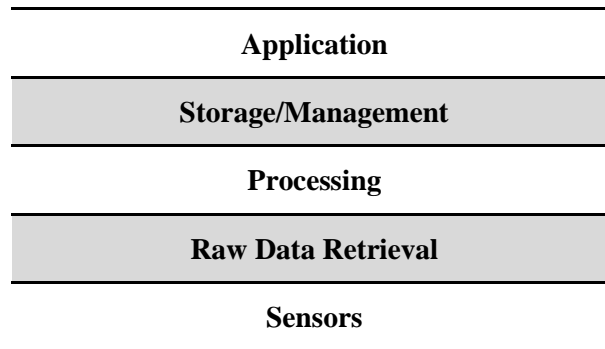


Figure 2.2 A High-Level General Framework for Context-Aware Systems

From low to high level, there are the following layers in the above framework:

- **Sensors** – To capture raw context retrieval, the sensors can be physical, virtual, or logical.
- **Raw data retrieval** – Appropriate drivers are chosen for physical sensors and APIs for virtual and logical sensors
- **Processing** – Responsible for reasoning and interpreting contextual information.
- **Storage/Management** – Responsible for handling client's requests. In the majority of cases the asynchronous approach is more appropriate than the synchronous approach due to rapid changes in the underlying context.
- **Application** – The implementation of actual reaction on various events and context instances. Agents may be used for communicating with the context server and acting as an additional layer between the pre-processing and the application layer [24].

2.5.2 Web Services

The current best option for supporting Software-as-a-Service (SaaS) and Service-Oriented Architecture (SOA) is Web services technologies [71]. The composite concepts from context-awareness and Web services provide enriched properties for future software. Ideally, a context-aware Web services system can understand surrounding context information and share that context information with other services. When comparing to context-aware systems, the concept of Web services is a relatively new one. In terms of The World Wide Web Consortium (W3C), a Web service “is a software system designed to support interoperable machine-to-machine interaction over a network.”

All Web services communicate with other applications and Web services in a machine-processable format (e.g., HTTP, SOAP and WSDL). Even though some context-awareness techniques could be potentially sought-after in Web services-based environments, it is not clear to which extent they are related and how to apply them [109]. In fact, the research on identifying where context-awareness techniques can be feasibly and applicably exercised is worthy investigating.

Inspired by the traditional forward engineering methods, the fundamental development method of Web services can be classified as bottom-up and top-down approach, where bottom-up approach starts with existing systems, discovers service interfaces from APIs, builds service contracts and compose them together in terms of the business process requirements; whereas, top-down approach starts with the business model, decomposes it into smaller models until the models can be easily defined.

2.5.3 Survey of Web Services-Based Context-Aware Systems

The following surveys are relevant to context-aware architecture and system: an early survey can be found in the study from Chen and Kotz [23] that they survey comprehensive types of context and models of context information, and discover that systems are responsible for context collecting and disseminating, whilst the changing context whereby applications adapt their behaviour. Baldauf et al. [9] survey common architecture principles of context-aware systems and a layered conceptual framework-based context-aware middleware and frameworks. Bettini [14] surveys a variety of context modelling and reasoning techniques and discuss a description and comparison of these techniques. Focusing on model-driven and aspect-oriented approaches to context adaptation, Prado et al. [92] survey a set of relevant approaches in such area. Truong et al. [109] compare the state of the art of context-aware systems and their environments, and claim that a survey of techniques and methods suitable for the development of context-aware Web services is missing by that time. Their survey concentrates on studying and analysing current techniques and methods for context-aware Web services systems, discussing future trends and proposing further steps on making Web services systems being context-aware. Beside the above surveys, some individual research works are close related to ours as well.

Ailisto et al. [2] present a five-layer model for structuring context aware application, i.e., layers are physical, data, semantic, inference and application. Many applications are built based on this five-layer model afterwards.

Keidl et al. [65] implement an open distributed Web service platform - ServiceGlobe within a generic framework that accommodates development support for context-aware adaptable Web services. ServiceGlobe provides users with client services based on personalised behaviour. Context process is

within a SOAP message. In this framework, two types of context processing are depicted, i.e., explicit processing by Web services or clients, and automatic processing by the context framework.

Omnipresent [5] is a service-oriented architecture (SOA) for context-aware applications. Technically, it is mainly a location-aware service system based on Web services. Users are able to access the location information via either mobile devices or Web browsers. In addition to the primary location information services, a reminder tool is offered.

Waldburger et al. [116] develop Akogrimo, which aims to radically advance the pervasiveness of grid computing across regions by leveraging the large base of mobile users. Akogrimo concentrates on core context that is related to mobile users' situations, such as user presence and location, and environmental information. The core component is the context manager responsible for collecting contextual information and delivers it to applications. It was implemented in Java and C# within the object-oriented paradigm. It is not clear whether Akogrimo is able to render its context manager responsive with increasing users.

Athanasopoulos et al. [8] create CoWSAMI, a middleware-based context-aware system that utilise Web services as interfaces to context sources. Context collectors are responsible for acquiring context information. Reliability and performance are subject to enhance.

The ESCAPE framework [108] is a P2P Web services-based context management system designed for emergency/crisis situations. ESCAPE services are composed of front-end and back-end systems which support context sensing and sharing between Web services within the ad-hoc network, and context information storage respectively. The context information executed

in this P2P Web services are largely restricted by its domain specific application.

Han et al. [54] present Anyserver, a client-proxy-server based architecture which supports context-awareness in mobile Web services. The Anyserver architecture utilises various types of context information, such as device information, networks, and application type. Application specific proxy tailors the original resource in terms of the mobile user's context information.

Chen et al. [25] develop a Context-aware Service Oriented Architecture (CA-SOA) which is a context model-based architecture. It is composed of three parts: Web services based on surrounding contexts, an agent platform with three types of agent: service, broker, and request agent; a service repository that contains service profile and service ontology; and a semantic matchmaker for context management. However, their work does not address the possibility of deploy their services to the Cloud and potentially massive users will not able to access appreciable context information in a responsive way.

2.6 Context-Oriented Programming

A domain-specific language (DSL) has a potential to make software maintenance simpler [32]. A DSL provides a notation tailored towards an application domain and is based on the relevant concepts and features of that domain [33]. Although context-aware system development is becoming one of increasingly important hotspots in software engineering community, there is little DSL support for building an application with regard to context.

Now that context belongs to a domain-specific concept and its dependency is a crosscutting concern for a system. As the context environment changes, the applications need to behave differently accordingly. Many research works in

this area still focus on architecture level, but when it comes to the implementation, mainstream programming languages do not support mechanisms that allow programs to dynamically adapt their behaviours due to the changing context. In effect, context-aware adaptation can be greatly facilitated by using programming languages that natively support high-level features to deal with contexts, context changes, and context-aware behaviours [51]. To avoid spreading over the application code with excessive conditional statements, this demands a new programming paradigm or high-level native language features to solve such issues.

Costanza and Hirschfeld [31] propose ContextL, an extension to the Common Lisp Object System that enables programmers to do context-oriented programming (COP) [56]. ContextL provides means to associate partial class and method definitions with layers and these layers can be activated and deactivated during run-time. Whether partial definitions belong to program depends on the activation status of a layer. This implies that a program's behaviour can be varied in terms of the change in context. To summarise, COP treats context explicitly, and provides mechanisms to dynamically adapt behaviour in reaction to changes in context, even after system deployment at runtime.

Lowis et al. [77] extend Costanza and Hirschfeld's work by introducing two additional language concepts: implicit activation of method layers, and the introduction of dynamic variables.

Since COP extensions have been implemented for several languages, Appeltauer et al. [6] represent a comparison of eleven COP implementations according to their designs and performance.

To relieve programmers from explicitly specifying and managing context awareness and the associated adaptation mechanisms particularly in pervasive

computing environment, Rakotonirainy [93] proposes a context-oriented programming approach that implemented in Python for pervasive systems. While COP allows context as a first-class construct of a programming language, the requirements for COP are discussed in [64].

As SaaS applications are becoming popular in Web services, Truyen et al. [110] claim that cross-tier tenant-specific software variations can be easily integrated into the single-version application code base via a COP model. They give a case study based on a Cloud platform for building multi-tenant Web applications to suggest that COP can be helpful for providing software variations in SaaS.

2.7 Summary

- ❑ The software reengineering phases that consist of reverse engineering, functional restructuring, and forward engineering are discussed. A general model for software reengineering diagram has been described.
- ❑ The definitions of requirements and requirements engineering are introduced. The general requirements engineering process is discussed, i.e., domain analysis, elicitation, negotiation and agreement, specification, specification analysis, documentation, and evolution. A brief discussion of the challenges for requirements engineering is included.
- ❑ Rolland [94] pointed out that the dominant concern in requirements engineering is to move from requirements to code. This trend poses a great challenge for requirements engineering and reverse engineering research respectively. Fortunately, in recent years, some researchers leap out of their boundaries and explore a wider range of investigations

into requirements behind the source code. For example, reverse engineering techniques can be applied to source code to assist the recovery work of requirement structures. Research on recovering requirements from source code via reverse engineering is presented.

- ❑ The processes of services evolution and requirements evolution complement each other: requirements evolution helps to offer guidance for evolution of context-aware services, and context-aware service evolution helps to genuinely revalidate requirements evolution. In fact, the failure in disclosing these hidden requirements will hamper services evolution. The relation between the said evolutions has been discussed.
- ❑ Context-aware system architecture in general can be envisioned as a hierarchical layer-based structure and is driven by context acquisition models. Components in different layers perform individual tasks and communicate with components in other layers. A layered conceptual framework is introduced. Web services-based context-aware systems deliver appropriate services accordingly whilst treating applications as a service. Such systems can be seen as a special type of context-aware systems, yet they are so crucial that they will pave the way toward ubiomp and Cloud computing development. A comprehensive survey on Web services-based context-aware systems has been covered.
- ❑ Increasing software developers are now dealing with context-dependent behaviour at run-time, yet many mainstream programming languages have not been created for such propose. This leads to convoluted programming, where programmers have to bend the languages to facilitate the difficulties of run-time context flexibility. Thus, COP is a promising approach to address such potential issues.

Chapter 3 – Proposed Framework

Objectives

- To introduce the overall reengineering framework and its approach
- To depict the services candidate discovery
- To depict the services reimplementatation
- To depict the services integration

3.1 Overview

Software reengineering is the primary technique for successful evolution of software systems. In general, software reengineering is mainly composed of a series of phases that further specific techniques are exercised in the course of tasks of reverse engineering, functional restructuring, and forward engineering. For instance, program comprehension techniques such as program slicing or formal concept assignment may be utilised in reverse engineering step; refactoring could be adopted during functional restructuring to achieve program transformation; domain specific language extension/library can be chosen for alleviating implementation tasks. Those traditional techniques evolved in software reengineering process are fairly promising within software-engineering community, yet reconciliation of requirements and implementation remains one of the main issues within requirements engineering community. Cheng and Atlee [26] suggest that the distinction

between the problem space, i.e., requirements and solution space, i.e., implementation resides primarily in the fact that requirements descriptions are written entirely in terms of the environments whilst other software artifacts are written in the light of internal software entities and properties.

In effect, software evolution is partially impeded by the gap between what the software is to do and how the proposed software is to do for the following main reasons:

- Few research works on recovering requirements via reverse-engineering related techniques;
- Many legacy software systems are written in inappropriate languages;
- Constraints are given less attention than other stakeholders' particularly during implementation stage.

Therefore, given such issues to address and the fact that software reengineering, as a well-established and well-accepted technique, plays a key role in a software lifecycle, a novel reengineering framework becomes necessary and it is worthy investigating relevant approaches within the framework to aid the software reengineering process for successful software evolution.

The success of a software system depends on not only how well it satisfies its requirements but also how well it fits into its running environments. In essence, change to environments can trigger software evolution. For example, deploying a legacy information system to the Cloud entails a series of software reengineering works. This research focuses on two environment changes, i.e., changes of context-awareness and changes of Web services. A services system of context-awareness is capable of adapting its services to changing context environments, while a Web service, in terms of The World Wide Web Consortium, is a software system designed to support interoperable machine-

to-machine interaction over a network. Nowadays, Web services-based context-aware systems are more complex and heterogeneous distributed than ever before. Such systems are composed of not only the essential components – sensors, applications, and context (reasoning) managers, but also various types of lightweight Web services that behave like agents. It is this property that poses a series of great challenges for programming models in the reimplementation stage in reengineering activities.

3.2 The Proposed Reengineering Framework and Approach

Generally speaking, software reengineering comprises understanding the existing software (i.e., what the system does) to decide what to modify in the software in terms of the new requirements and environments, and how to implement such modifications. To bridge the gap between these two tasks, novel approaches are needed to leverage the traditional reengineering framework and approach. For instance, many earlier systems may only contain a vague requirements specification or may not have it at all. To recovery such design documentation, e.g., requirements specification is essential. It will in turn facilitate the evolution of the software system.

Figure 3.1 demonstrates the proposed Context-Aware Web Services Reengineering Framework (CAWSRF). Other proposed frameworks and models found on this overall reengineering framework will be briefly introduced in this chapter; detailed discussion about them will be described in the following chapters.

Typically, the proposed framework approach consists of the following core phases:

- **Legacy System Assessment:** This assessment of legacy systems from imperative and OO language paradigms is responsible for judging the applicability of CAWSRF approach and deciding if other reengineering approaches should be performed.
- **Services Candidate Discovery:** It is carried out based on the proposed Requirements Recovery Framework (RRF). The two core reverse engineering techniques used are discussed as follows:
 - **HB-CA Performance:** Applying Hypothesis-Based Concept Assignment (HB-CA) [52], as one of plausible reasoning techniques, onto the qualified source code with the content from Services Pattern Module (SPM) found on Requirements Recovery Framework (RRF). This will generate a list of event-linked concepts as potential requirements each time and the hypothesis source code information is not necessarily executable.
 - **Program Slicing:** Static program slicing [107] techniques are applied to decompose qualified source code reflected from the results of SPM. This indicates code segments of interest for further reengineering activities.
- **Services Reimplementation:** It is this stage that context-awareness requirements are fulfilled in the Web services system. The avenues of achieving context-awareness can be via a novel architectural design and language programming support. In this thesis, the latter is adopted. Requirements mapping will be exercised during requirements analysis for context-aware Web services. Depending on domain specific requirements, for example, some sought-after services will entail implementing communication computation overlap (CCO) strategy, or handling massive asynchronous requests (e.g., 1000 simultaneous

requests), requirements in the specific implementation domain are taken into account when choosing programming languages to implement the desired services. Library – ContXFS is developed to assist the implementation tasks by offering software developers with efficient libraries to build the services components for further integration.

- **Services Integration:** In this integration process, legacy services and newly-built functional services are composed via connectors in order to construct the target system. This can be implemented via wrappers and code gluing techniques.

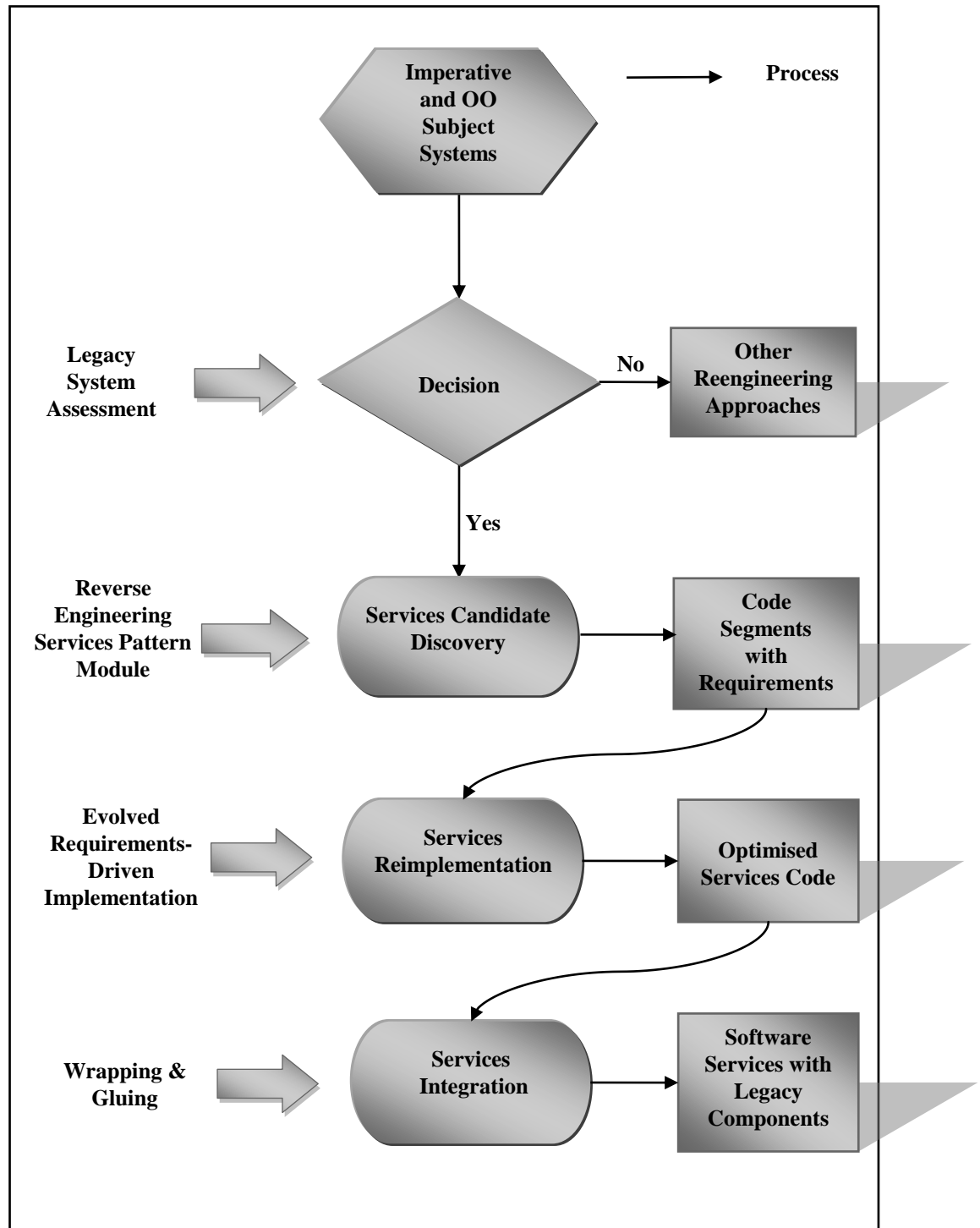


Figure 3.1 CAWSRF Approach

3.2.1 Services Candidate Discovery

The Services Candidate Discovery can be divided as two types of discovery:

- **Code-related Artifacts Discovery** (shallow recovery): Traditional reverse engineering techniques are applied. Understanding a software system generally requires comprehension of the domain specific knowledge and the application itself. This task may embraces analysis of detailed code knowledge, i.e., code algorithms, structures and documentation including comments. Broadly speaking, program slicing and concept assignment have been proposed as source code extraction techniques [53]. Specifically, program slicing may be used to decompose source code into segments of interest given the extraction criteria are specified. Depending on the slicing techniques [122], the extracted code segments may not be executable. Concept assignment, on the other hand, may be use to relate information regarding the problem and application domains, e.g., structures of the program to fragments of source code. Other techniques such as clustering analysis [35] and ontology [21] can be adopted for such proposes.
- **Requirements-related Artifacts Discovery** (deep recovery): Comparing to code-related artifacts discovery, requirements-related artifacts discover tries to recovery deeper information from the source code, i.e., recovering requirements behind the code typically when requirements specification is not available during the reverse engineering phase. For this kind of discovery, conventional reverse engineering techniques are still applied, however techniques from other research fields, e.g., requirements engineering are also taken into

consideration. Essentially, recovering requirements from source code is becoming far more important than ever before [38].

In practice, shallow recovery is more suitable for software migration when legacy systems is to be moved to new environments that allow information systems to be easily maintained and adapted to new business requirements, while retaining functionality and data of the original legacy systems without having to completely redevelop them [16]. In contrast, deep recovery is more appropriate for dramatic redevelopment when the gap between existing code-related artifacts and new requirements is too big or a new programming language (model) is available for higher abstraction, which leads to more concise implementation.

Our proposed services candidate discovery is based on the Requirements Recovery Framework (RRF) [58]. The holistic framework composed of the following parts is briefly described:

- **Services Pattern Module (SPM):** This module contains Knowledge-Based Library (KBL), Source Code Information (SCI) including comments, identifiers and keywords, and Requirements (REQ) covering functional requirements and non-functional requirements.
- **Concept Generator:** It takes source code and SPM as input and applies Hypothesis-Based Concept Assignment (HB-CA) method onto them.
- **Event Concepts:** When concepts are available, concepts will be linked with events (in the source code) as a tuple <Concept, Event>.
- **Source Code Information (SCI):** It embraces information directly reflected from the source code including identifiers, comments, and keywords. It is initially created along with requirements.

- **Requirements:** It consists of functional requirements and non-functional requirements.
- **Knowledge-Based Library (KBL):** It comprises lists of intermittently enhanced tuples: <Concept, Event>.

The details of the requirements elicitation approach based on the above framework will be discussed in the Chapter 4.

3.2.2 Services Reimplementation

Forward engineering is last step in the reengineering activities, yet it is the final stage that software evolution is completely reflected and embodied. In most cases, forward engineering revisits the traditional software engineering processes based on the recovered code-related artifacts or requirements-related artifacts from the source code. Typically, the forward engineering in our proposed reengineering framework is the step of implementing the sought-after software system against the evolved requirements with reusable components from the legacy system.

Requirements analysis is the first step to understand “what the services are to do”. In the context of the environments that are context-aware and Web services-based, for example, some core requirements can be summarised to be satisfied in terms of Galster’s taxonomy [47] for non-functional requirements in a service-oriented context. Table 3.1 depicts a sample of detailed corresponding requirements of a subject services system to meet.

Requirements Types	Concrete Requirements		
CFR	Context-Awareness	Concurrency & Parallelism (distributed system)	
NFSR	Reliability	Scalability	Evolvability
PR	Implementation Requirements	Composition Requirements	

Table 3.1 A Sample of Refined Context-Aware Web Services Requirements

In Table 3.1, Core Functional Requirements (CFR) that consist of context-aware, and concurrency & parallelism; from non functional requirements perspective, Process Requirements (PR) covers implementation requirements (e.g., .NET Framework), composition requirements (e.g., composable Web services); Non Functional Service Requirements (NFSR) contains reliability, scalability, and evolvability. These requirements are the result of evolved requirements that synthesise the new requirements and the recovered requirements. Further activities will not perform until these requirements are available.

In most cases, the evolved requirements are generated by Context-Aware Service Requirements Model (CASRM) [59] which is a derived viewpoints based requirements model. The results of CASRM, from context-aware services evolution perspective, are the requirements that mingle functional requirements, non-functional requirements, interface requirements and context requirements, which in turn, can be considered as the initial input of the said

requirements evolution. The fast dynamic changes of context-aware Web services system entails a requirements management model that is to address the changes and impacts on the original services systems. Thus, a requirement evolution model for context-aware service requirements evolution [59] is proposed. By separating context-aware Web service requirements into Web services requirements and context requirements, two possible triggers of changing requirements are highlighted, i.e., changing Web services requirements and context requirements. The components and steps to construct context-aware service requirements and the details of the requirements evolution model will be represented further in Chapter 5.

Emphasis of constraints can potentially reduce the costs and risks of re-implementing a complete existing system. For instance, alleviating developers' burdens can be done by providing developers with tools support, a guide as to how to choose the appropriate programming languages, domain specific libraries for efficient development support and so on. Therefore, different from conventional requirements analysis, these requirements analyses carry out two major analyses from users' perspective and developers' perspective in terms of the requirements in SPM. Thus, both functional and non-functional requirements from both perspectives will be traded off by stakeholders involved. In practice, although this work is almost impossible to be automatic, and users' requirements are conventionally considered solely in this stage, constraints from developers' side should be primarily heard and considered as valuable knowledge for implementations. Being distinct from some other research, our approach offers more voice to developers as they also need to ease their development burdens. It is in this stage that requirements and implementation can be reconciled and evolved requirements and services code segments are generated therefore.

When the requirements analysis completes and the evolved requirements are available, for most of cases, reimplementation is necessary. Although migration and wrapping benefit from avoiding the long, costly and risky process of implementing an entire legacy system, the target services system will barely satisfy the continuing changing requirements, which leads to unsuccessful services evolution eventually.

Since reimplementation is a must, choosing a programming language becomes important. Most of current context-aware Web services-based systems were built in mainstream object-oriented languages, e.g., Java and C#. Notwithstanding, some critical and essential implementation techniques that are other paradigm languages' sweet pot are missing. Instead of excessively adopting Design Pattern of 'Gang of Four' from object-oriented programming paradigm, functional programming language [61] is embraced. It has been a long history that Domain Specific Languages (DSLs) are conveniently created in a functional programming language. DSLs enable software developers to more concisely describe a problem itself, and use this custom language to solve the problem. As few mainstream programming languages directly enable software entities to adapt their behaviour dynamically to the current execution context. Software developers will end up spending more time and effects on bending the languages harsh enough to 'hit the point' by convoluted development. Such time and effects can be saved by introducing a new programming language with support of Context-Oriented Programming (COP) [31] that facilitates implementation tasks. COP treats context explicitly, and provides mechanisms to dynamically adapt behaviour in reaction to changes in context, even after system deployment at runtime [56]. In the services reimplementation, F# [104] library – ContXFS is developed to assist the development. The reasons why F# is a better candidate to build the context-oriented programming library will be expounded, along with the overall services reimplementation will be discussed in Chapter 6.

To summarise, the services reimplementation lies in the heart of services evolution. The proposed approach at this stage employs conventional methods in the course of software lifecycle with the evolved requirements and emphasis on constraints, especially on design requirements and implementation requirements. The evolved requirements consist of not only the context-awareness requirements, but also requirements for Web services computing, where ContXFS is developed to address the former issues, and an appropriate programming language is selected to support an asynchronous agent-based programming model in the concurrency and parallel computing environment. Services evolution is incarnated through services reimplementation.

3.2.3 Services Integration

In general, Web services integration is fairly straightforward. The extracted reusable services code can be wrapped and integrated into preferable service architecture, e.g., Service-Oriented Architecture (SOA). In our case, legacy services and newly-built context-aware Web services are composed via connectors in order to construct the target system. This can be implemented via wrappers and other code gluing techniques. The detailed integration will not be discussed in this thesis as this is more about implementation platform issues (Web services in Microsoft ASP.Net and Java) rather than the core issues that cause the underlying gap between requirements and implementation. In other words, one of the implementation platform issues is that constructing Web services and clients in the .NET Framework and in Java so that they are able to interact with each other. Namely, a .NET Framework-based Web service is invoked with a Java client or vice versa.

3.2.4 Forward Engineering in Proposed Reengineering Framework

Traditionally, software reengineering process focuses on reverse engineering and functional restructuring; there is much research on how to extract artifacts as well as to identify their internal relationships, and to apply relevant methods to fulfil program transformation. Whilst in this thesis, the methodologies and technologies of forward engineering within the proposed reengineering framework are different from those applied in the conventional object-oriented software process. From software developers' prospective, the choice of implementation languages during forward engineering largely affects their abilities to solve the software problems [114]. A functional-first hybrid programming language (e.g., F# [104]) is selected to address the specific issues that Web services-based context awareness brings.

The differences of the proposed forward engineering can be summarised as follows:

- **Requirements:** Requirements are achieved by synthesising the recovered requirements from the legacy system and new requirements from users for further evolution. Along with discussion about shallow and deep recovery in Section 3.2.1, the recovered requirements can be used by domain experts as a gauge to measuring the gap between current and new requirements, which implies whether the current system is more suitable for migration or redevelopment.
- **Design:** In object-oriented design, a software design can be represented as a set of communicating objects. In other words, object-oriented design process involves defining object classes and setting up their relationships. In effect, design patterns are widely applied in

object-orientation design/development. They are considered as descriptions of interactive customised classes and objects that solve a generic design problem in a specific context. Based on object-orientation, UML is always adopted to specify their internal relationships. While in functional design, functions play a key role. It is not necessarily that functions must be wrapped into a class. The independent existence of functions along with other constructs and features from F# allow for more flexibilities. This implies many concepts of design patterns could literally disappear (e.g. Lazy Initialization and Builder) or be just idioms of that language (e.g., Factory Method is essentially a function returns an object.).

- **Implementation:** Object-oriented programming languages provide constructs to design object classes, whilst F# accommodates constructs with much higher abstraction that the importance of design patterns fade away. For example, Lazy Initialization can be achieved via F# lazy value or lambda function. Builder can be implemented by passing optional arguments to a constructor in a class type definition. Specifically, F# features asynchronous programming support with its ‘Async’ library. This can address many issues of requests from Web services. Context-awareness is typically achieved through F# constructs – discriminated union types and pattern matching without spreading conditional statements. Obviously, other approaches such as context-oriented programming can be implemented in F# for deeper requirements that context-awareness entails.
- **Evaluation/Maintenance:** Object-oriented programming languages often come with ‘high ceremony’ that OO programmers are so customised to that suggests they do not realise how inefficient their OO code is. On the other hand, F# is a succinct and expressive functional-

first language. Thus, the F# code is always shorter than OO code for a same implementation. Less code infers lower maintenance. Moreover, F# is so expressive that it maps the problem solving process of human being more directly into the implementation with appropriate constructs.

In summary, the benefits of using such as F# for forward engineering are twofold. The differences can be depicted from design problems and implementation problems respectively. From design prospective, design patterns are commonly applied to abstract the way of factoring object into classes, defining class interfaces and inheritance hierarchies, and establishing relationships among them in a particular context. In F#, functions can take any argument as an input and return an object; it provides developers with more flexibilities. On the other hand, from implementation prospective, design patterns can be used to specify object interfaces and object implementations. F# does not have the object-oriented constraints that everything is wrapped into a class. Function can fulfil many of similar tasks. For example, F#'s constructs – discriminated union type and object expression and the feature of pattern matching are pleasant combination of completing many of programming tasks.

Eventually, from maintenance prospective, by building up with less code lines, components will be much easier to maintain than those implemented via inheritance. The implementation inheritance will often make the supper types more complex and it is against the maintenance essence. Nevertheless, F# partially implementation types can be implemented via delegation with object expressions within a concrete type.

3.2.5 The Differences and Consequences of the Proposed Reengineering Framework

The main differences of this proposed framework and approach can be classified as follows:

- **Requirements Recovery in Services Discovery:** Significant research works of reengineering merely focus on code segments extraction [53, 70, 112, 132], whilst the proposed framework is designed to further recover the underlying requirements. In effect, recovered code segments are more suitable for migration, while redevelopment entails new requirements. The availability of recovered requirements along with the code segments can provide a better understanding of the legacy components and their relationships, which in turn assists the reimplementation during later activity – forwarding engineering.
- **Forward Engineering:** In software reengineering, not much research works investigate the implementation details in forward engineering. Even a reengineering framework approach that clearly embraces a forwarded engineering phase, the methodologies and technologies applied to implementation are still based on object-oriented platform [134]. Fixed programming paradigm hinders efficiency. This proposed framework approach however highlights the importance of implementation languages choices and implies that the method of selecting appropriate programming languages.

Based on the differences discussed above, the primary advantages of this proposed framework and approach can be depicted as follows:

- **Complete Requirements:** Requirements recovery is the centre of requirements analysis in the holistic framework approach. Extracted

code segments should be consistent with the legacy technologies applied in forward engineering, yet the proposed framework is designed to recover the relevant requirements to be analysed and synthesised with new requirements for further redevelopment. Extracted code segments may be not always coherent with new technologies of design and implementation. Therefore, requirements recovery can be a complement.

- **Different Reimplementation:** Choosing appropriate methodologies and technologies to fulfil the requirements of context-awareness and Web services is crucial. For example, functional programming techniques can be used to better express the problem domain and map it into the salutation domain. Thus, software developers might be able to spend more time in focusing on the hardest parts of the development (e.g., asynchronous and parallel programming) than only arranging classes and objects into an appropriate abstract level.

The primary disadvantages of this proposed framework and approach can be described as follows:

- **Manually Generated Requirements:** Automated and semi-automated mechanisms in software reengineering always attract lots of research attention. Although some works can be implemented in a (semi-) automated way, the majority of stages involves fairly much manual work from domain or software engineering experts. This because recovering deep requirements and managing requirements evolution during services evolution is a systematic process. In order to obtain correct and practical results, recovering and maintaining frequently changing requirements may make manual work inevitable.

- **Cost of Reimplementation:** In terms of the assumption of the need of migration, conventional reengineering approaches mainly highlight reverse engineering and functional restructuring. Reimplementation during the forward is the last option for reengineering work due to the high cost and risk that it may pay for. Nevertheless, in the case of reengineering context-aware Web services-based systems, reimplementation may be a better solution to reuse the legacy system yet be able to deliver the sought-after services. For example, it is almost certain that there are few design patterns specific for functional programming paradigm because of a number of historical reasons [115].

3.3 Summary

In this chapter, a novel software reengineering framework for Web services-based context-aware systems has been proposed. The core reengineering steps can be summarised as follows:

- Services candidate recovery, traditionally, this is the process of identifying reusable code-related artifacts from the legacy systems, e.g., a set of class structures or the algorithm of certain code segments. Notwithstanding, without further recovery of requirements behind the source code, this will hinder services evolution in future. The proposed services candidate discovery belongs to requirements-related artifacts discovery (deep recovery) whose process is a requirements elicitation-based approach. One of the main reasons for such a deep recovery is that requirements of context-aware Web services change as the environment (context information) changes dynamically.

- ❑ Services reimplementation corresponds to forward engineering during the proposed reengineering framework approach. To guarantee the evolved requirements are completed requirements, a context-aware services requirements model is proposed. Furthermore, a requirements evolution model is built to maintain the requirements evolution during the reengineering process.
- ❑ F# is functional first and object-oriented second programming language that enables developers to more directly map their implementation process into the relevant language constructs. Furthermore, F# code is much shorter than OO code which in turn reduces the potential cost of maintenance and evolution in the future.
- ❑ Typically, emphasis on constraints during services reimplementation stage is crucial since services evolution will be impeded when inappropriate programming languages are chosen for implementation. Moreover, to facilitate development task, an F# library – ContXFS that allow for context-oriented programming is developed. It embraces efficient libraries for building context-aware Web services-based components.
- ❑ Services integration is the finally stage that newly built services and the existing services integrate together to deliver the entire services to customers.
- ❑ The proposed framework involves fairly much manual work relying on the knowledge from domain or software engineers. Qualitative methods of this work are mainly reflected by the requirements recovery step where ‘why’ and ‘what’ related questions are discussed, whilst quantitative methods of this work are primarily suggested by the

reimplementation step of the said systems where programming language support is discussed.

Chapter 4 – Requirements Recovery Framework for Services Candidate Discovery

Objectives

- To discuss the layered conceptual framework for context-aware systems
- To describe the requirements recovery framework
- To discuss the framework approach
- To show an intermediate result of the framework approach on a location-aware system

4.1 Overview

In modern software development, software requirements and implementation are not always reconciled. This leads to difficulties for software evolution tasks in future. Typically, for modern Web services-based context-aware systems, changes of stakeholders' requirements and context environments imply that existing services system is subject to modifications as current implementation is no longer sufficient to meet the new requirements. On the other hand,

reverse engineering, a well-known method used within software engineering community, aims to understand the functions and behaviour of a subject system from source code. In effect, many reusable code-related artifacts are extracted without recovering the system's requirements behind the source code. However, the fact that requirements recovery from source code is necessary and has far-reaching implications is being recognised. Therefore, a requirements recovery framework is built. Based on this framework, a requirements elicitation approach is developed. This framework approach is further claimed to reconcile the gap between software requirements and implementation for context-aware Web services evolution within the overall reengineering framework explored in Chapter 3.

4.2 Context-Aware System Framework

4.2.1 The Problem

Context awareness is a key property of ubicomp systems that reasons about the surrounding information to adapt applications accordingly. Since the concept of context-awareness [95, 117] debuted, several models, conceptual frameworks, and architecture have been developed to represent, process and model context. For example, context model is designed to define and store context data. In fact, many existing context models are constrained by their pre-defined requirements. Nevertheless, as user intentions can change at arbitrary time and context models may not be capable of handling all possible circumstances. In other words, context models have limited capability in involving human intentions for self-adaptability [89].

Specifically, this research focuses on the context-aware systems where context-aware middleware or context server is the software that provides services of

context-awareness. Stakeholders' intentions are not always well captured in the early stage of development process due to lack of formal languages support. For instance, at design stage, system customers may not be articulate enough to express all the functionalities they need, or at implementation stage, same will happen when revisiting conventional software development lifecycle in the process of reengineering; software engineers may choose a programming language that is not abstract enough to express the common programming patterns to support the implementation of customers' functionalities, which causes software maintainers have to spend much more time and efforts on understanding the convoluted programming that is not necessary provided an appropriate programming language were chosen for development in the first place.

In consequence, these discrepancies will hamper the context-aware service evolution tasks. In order to conciliate these, this thesis proposes a context-aware requirements elicitation approach to reconcile the gap between software requirements and implementation for context-aware service evolution based on the proposed reengineering framework approach discussed in Chapter 3. Hence, in this chapter, the paramount job is to further our traditional practice [124] on reengineering activities to recover the requirements from source code in order for navigating other reengineering activities, e.g., functional restructuring and forward engineering.

4.2.2 Layered Conceptual Framework for Context-Aware Systems

Context-aware systems can be implemented within various frameworks and every framework owns its context models. Although context model is responsible for representing and sharing context data, in many cases, the

architecture style of a context-aware system is shaped by the architectures of context acquisition and context management. For instance, by summarising the design approaches to existing context-aware systems, Baldauf et al. [9] conclude that the architectural style of a context-aware is mainly driven by the context acquisition method. Thus, the approaches to context acquisition and context management are vital during the design period of context-aware systems.

While context acquisition is an important topic, this chapter focuses on context management because it exposes itself to functional requirements and non-functional requirements. Apparently, abstraction functionality can be implemented by an application directly in a context-aware system; nevertheless, the frequent changes of user's requirements and context make such assumption unreasonable. As software developers are unable to predict what changes would be made. Thus, the implementation of abstraction functionality should be encapsulated and put into a middleware, in this case, a context server. The overall benefit of such arrangement is that the development of applications in the client side can be facilitated.

Although middleware-based or context server based context-aware systems can be implemented in different ways, the hierarchy of a common framework often consists of the following parts from low level to high level: sensors; raw data retrieval; pre-processing; storage/management; application. Various layers are responsible for different tasks, for example, sensors in the sensor layer are to capture the raw context information before further aggregation and interpretation; context storage in the management layer is to maintain a database of context information for user query; application is developed in application layer to fulfil the functional requirements, whilst context server in the middleware layer achieve the non-functional requirements.

With over nearly two decades' evolution of context-aware systems, nowadays, context-aware systems become far more complex. For example, Web services-based context-aware system poses great challenges of not only context-awareness, but Web services-related issues, such as concurrency, parallelism, scalability and so on. Amidst this evolution, a layered conceptual framework for this kind of systems primarily based on [2, 9, 14, 19, 57] is proposed. This framework is depicted in Figure 4.1 as below:

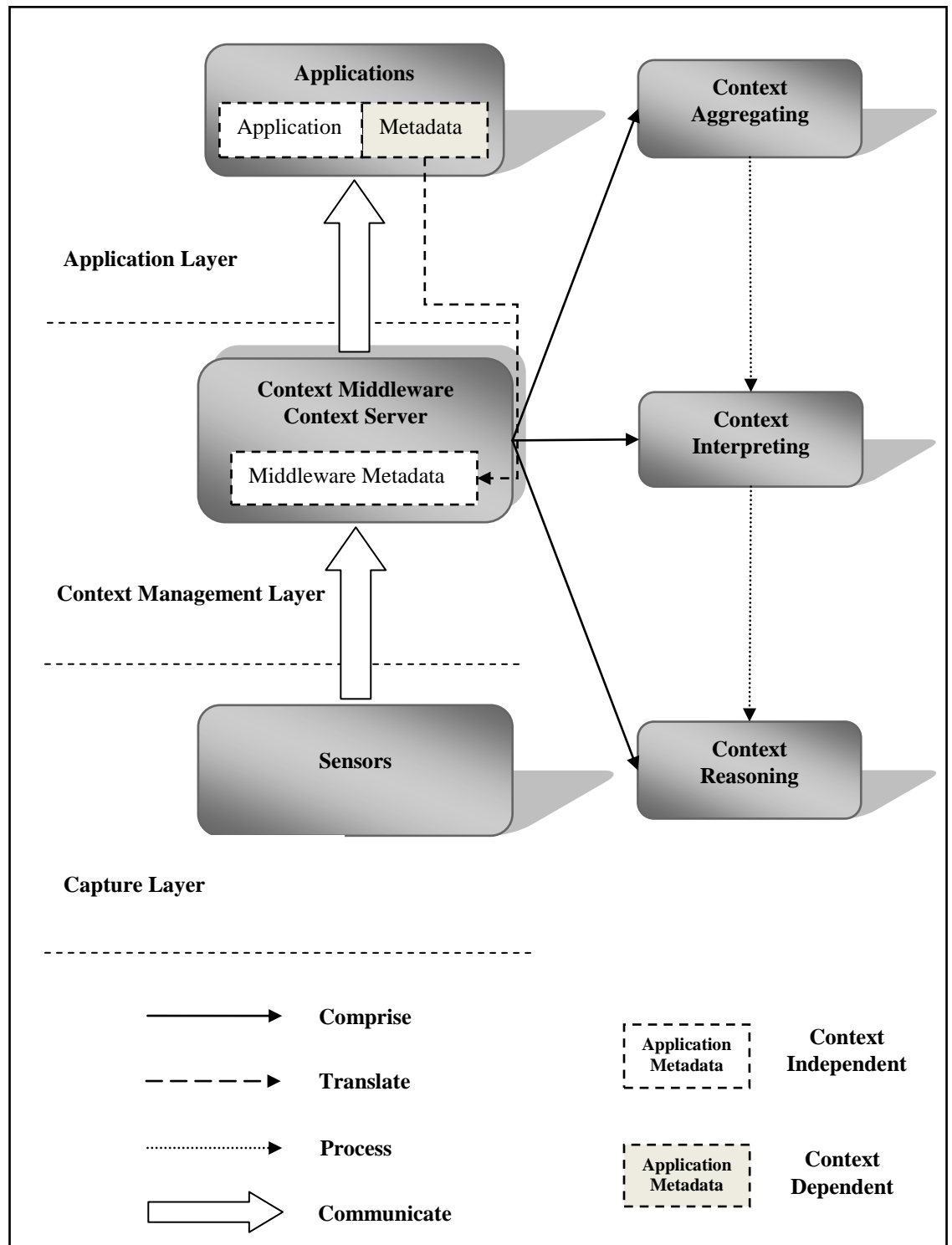


Figure 4.1 The Proposed Layered Conceptual Framework for Context-Aware Systems

Figure 4.1 demonstrates a layered conceptual framework for middleware-based context-aware systems. The components of this framework are introduced as follows:

- **The Sensors in Capture Layer:** They are responsible for raw context retrieval. Although sensor is tightly associated with sensing hardware, it may include every data source that provides appropriate context information. The context information will be sent to upper level for use. These sensors can be further classified as follows:
 - Physical Sensors, which are capable of capturing physical data, such as image, motion, light, audio, temperature, touch, and location and so on. In practice, the communication between physical sensors and context middleware/server accounts for the main input to realise context-awareness.
 - Virtual Sensors, which detect the context data captured from software applications or services. For instance, by querying a login user account on a computer system, sensors can tell who are using the computer systems in office although it might be not as accurate as physical sensors. For example, a user account could be misused or stolen.
 - Logical Sensors, which synthesise and analyse raw physical data and virtual data to reason about higher abstract level tasks. For example, a logical sensor can be deployed to infer a person's social hobbies by analysing the history of location information of where (s)he has been and activities on their personal devices, e.g., laptop, smart phone etc.

- **The Middleware/Server in Context Management Layer.** Context server facilitates the stored information needed for performing synchronous or asynchronous computations. Predefined short-running requests may usually complete in a synchronous manner, i.e., it sends a request for some kind of data and pauses until it receives the server's response, whereas the asynchronous approach may be more preferable because of continuous changes in the underlying context and increasing requests from users. Thus, from development prospective, context-aware applications that respond to events raised on main thread or worker thread by registering event handlers with their context-aware middleware or context server, i.e., event handlers are registered in middleware/server that detects the environment changes and dispatches message to application to perform actions to respond to the underlying context changes. This also implies that context-aware application is a concurrent program.

Furthermore, a typical context server consists of the following components to account for the context management functions:

- Context Aggregating, where an aggregation of context atoms either to combine all context data relevant to a particular entity or to create a higher level context object. This process is essential as a simple individual sensor value is always not useful, whilst combined context data may contain wider context information that is of interest. This the first stage where raw context data is captured and combined for next stage.
- Context Interpreting, as sets of context data are available, they will be interpreted in a form that the clients can understand. Context interpretation, along with Application Metadata (shown

in Figure 4.1), builds a mechanism to provide the relevant processed context information for applications' polling. This is the second stage where context data is transformed into an interpreted form.

- Context Reasoning, where context is abstracted from low-level context data by building a new model layer that gets the sensor perceptions as input and generates or triggers system actions [14]. This enables services to take a decision whether any adaptation to a change is necessary. This is the third stage where interpreted context data is reasoned in terms of rules within the system. The resulted context data is delivered to clients.
- Middleware Metadata, Middleware Metadata in the context management layer comes from the translation from the context-dependent part of Application Metadata (discussed in the Application Layer below). This metadata relates to the application's non-functional requirements. For instance, in a highly scalable Web services-based context-aware system, context server handle thousands of client requests in an asynchronous computation way rather than a synchronous way. Scalability is a core issue for such kind of system.

Although context servers are now frequently used for acquiring and managing context information, most applications do not make use of any form of support (for instance, programming toolkits or infrastructure) for interpreting and making decisions about context [9]. The context gathering layer acquires context information from sensors and then processes this information, through interpretation and data fusion (aggregation), to bridge the gap between the raw sensor output

and the level of abstraction required by the context management system.

The Applications in Application Layer: where the actual event handling code is implemented to react on specific context changes reported by the context-aware middleware or context server. It is this layer where the client is realised. Application Metadata in this layer consists of two types of metadata, i.e., context independent metadata and context dependent metadata. For example, context independent metadata can be associated with interface requirements while context dependent metadata can relate to specific constraints. Application Metadata is used to instruct the application on how it should behave under what circumstances, in other words, this metadata relates to functional requirements.

In essence, this entire framework suggests that performing long-running computations is inevitable due to the nature of context-aware systems, thus, asynchronous computing is needed, in fact is crucial, otherwise, it may render the middleware unresponsive. It is the middleware that takes control of maintaining a valid representation of the context; whenever a change to user's need and context is detected, the metadata commands the application adapt its computation. Apparently, middleware metadata is dynamic updated as the user's requirements and context change. It is this holistic mechanism that drives the software evolution for context-aware system, and users are able to behold the service evolution as a result of it.

To summarise, the proposed layered conceptual framework is a simplified version of more complex architecture of context-aware systems. This architecture is design for multiple users where simultaneous requests are made. The main drawback of this framework is that the design of this kind of context-

aware systems is based on client-server architecture. In other words, in the case of this research, the implementation of the context-aware systems needs to realise this design architecture during forward engineering.

4.3 Requirements Recovery Framework and Approach

4.3.1 Requirements Recovery – In a Nutshell

In requirements engineering, requirements are often classified as two levels of details in requirements document. Customers need a high-level statement of the requirements, whereas software developers require a more detailed software specification. In fact, a requirement is only one of the possible means to achieving a goal. Compared to requirement, goal is a relatively steady concept. Goals can be referred to as intentions since they are related and complementary concepts. However, although it is possible to recover stakeholders' goals from implementation [69], attempting to elicit their requirements is valuable for software evolution proposes [37, 74].

In reality, requirements documents are always poorly written or out-of-date, even not available. Requirements recovery is an essential task for better understanding a legacy system; navigating the later activities, e.g., re-design and re-development in a reengineering process. The studies to recovery requirements from source code have been carried out for multiple purposes. Yang et al. [125] point out that ontology is a useful source for understanding and reengineering a legacy system; Liu [75] presents a semiotic approach to requirement engineering; recently, Chen et al. [22] depict an ontology-based reengineering approach to recovering requirements from existing systems by matching domain ontology and program ontology.

4.3.2 Requirements Recovery Framework

In this recovery framework, this study focuses on two kinds of requirements, i.e., users' requirements and constraints. Requirements can be divided into functional requirements and non-functional requirements, for instance, user may have this goal: "to search a destination online", which is a very abstract objective. Then, requirements engineers may parse this to "providing a search button on a webpage" as the functional requirement, and "the page should be highly responsive during searching" as non-functional requirements. For example, providing a cancel searching button and a pause searching button could allow users to have more control on search without having to wait the whole search to complete. There are some other elements that may affect our work, such as users may be classified as novice users, advanced users, and professional users, so are developers. But those factors are not considered in this paper.

It is assumed that users' requirements are fused into implementation code and in turn, the code implies them. For services evolution purposes, a requirements recovery framework is created to assist requirements elicitation task based on the framework presented in Figure 4.1. Whilst Figure 4.2 describes this requirements recovery framework as below:

- **Services Pattern Module:** This module contains Knowledge-Based Library (KBL), Source Code Information (SCI) including comments, identifiers and keywords, and Requirements (REQ), i.e., Functional Requirements (FR) and Non-Functional Requirements (NFR). The module underpins the requirements elicitation and an initial service pattern module is created by domain experts and software engineers as a prerequisite. As the requirement recovery framework approach is

exercised, the content of SPM can be updated and improved as a result of enhancements.

- **Concept Generator:** It takes source code and SPM as an ‘input’ and apply Hypothesis-Based Concept Assignment (HB-CA) method [52]. HB-CA is one of plausible reasoning techniques and it is not tailored particularly to certain language, such as COBOL II. It is composed of three stages, i.e., Hypothesis Generation, Segmentation and Concept Binding. Each stage takes the output of the former one as its input. The overall output is a list of concepts, associated with regions of source code. Detailed stages will be described in the Requirements Elicitation Approach discussed later.
- **Event Concepts:** When concepts are available, with tool support, concepts will be linked with events (in the source code) as a tuple <Concept, Event>. Domain experts and software engineers fulfil the enhancement to further enhance the content of services pattern module. These event-linked concepts are the most likely users’ functional requirements.
- **Source Code Information (SCI):** It embraces information directly reflected from the source code including identifiers, comments, and keywords. It is initially created along with requirements.
- **Requirements (REQ):** REQ consists of functional requirements and non-functional requirements.
- **Knowledge-Based Library (KBL):** KBL comprises of lists of intermittently enhanced tuples: <Concept, Event>.

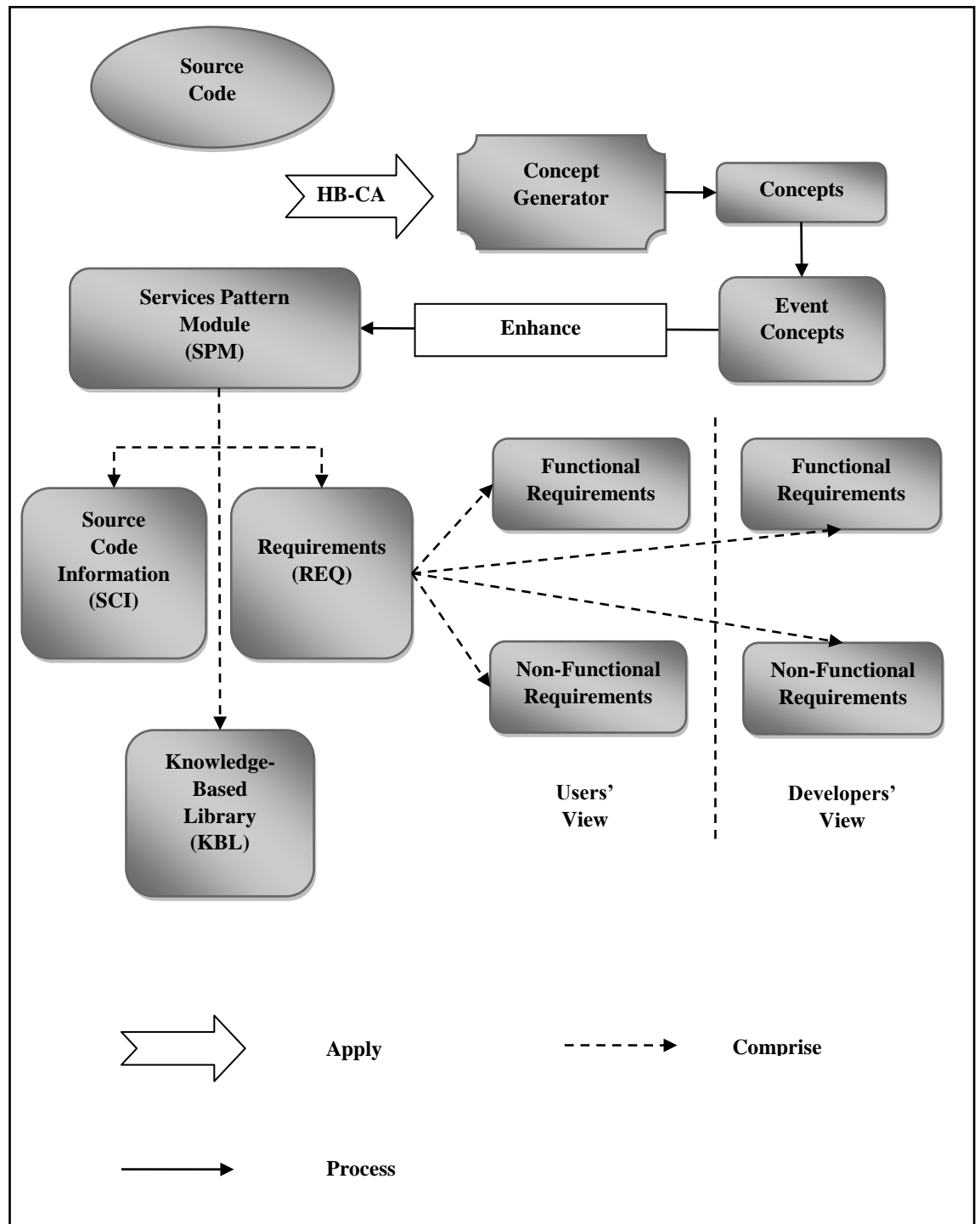


Figure 4.2 Requirements Recovery Framework

4.3.3 Requirements Elicitation Approach

An initial SPM must be created by domain experts and software engineers in order that informal information (comments, identifiers, keywords) and requirements can be stored. In general, concept assignment techniques are applied to relate information about domain problems to portions of source code.

Specifically, applying HB-CA method to the input of the qualified source code and pre-established SPM can generate three stages:

- **Hypothesis Generation:** This involves assigning the source code information tokens; mapping these tokens to correspondences in service pattern module. This step aims to generate the source code information (identifiers, comments, and keywords) in SPM. This hypothesis source code information forms a list and is not necessarily executable.
- **Segmentation:** The hypothesis list is grouped into segments in terms of whether potential exists can form clusters. Selected segments at length form a hypothesis segment list.
- **Concept Binding:** In order to bind the most likely hypotheses concepts, the segments in the list are determined by their occurrence frequencies. When a concept is selected, the segment is labelled with the name of that concept. The result of this stage is a list of concept bindings linking regions of source code.

The event concepts will be generated as a tuple <Concept, Event> by domain experts and software engineers, which in turn will enhance SPM by modifying the existing content.

The framework approach presented in this chapter is based on RRF, and highlights two viewpoints of user and developer. Liu et al. [74] proposed a semiotic approach to recover requirements through studying the legacy system's behaviour. Their approach contains investigation activities at three major stages. SMP lies at the heart of the proposed framework approach, which consists of three stages, i.e., hypothesis generation, segmentation, and concept binding. Chen et al. [22] apply ontology-based reengineering approach to recovering requirements, whilst, hypothesis-based concept assignment is adopted in the proposed framework approach. El-Ramly et al. [37] present a data mining approach called CelLest process in order to discover patterns of frequent similar episodes in run-time traces of user-interface behaviour. The proposed approach to requirements recovery in this chapter emphasise users' and developers' viewpoints as they are the key models for discovering the requirements gap between the legacy system and the subject system. Nevertheless, there are other viewpoints that are not included in the framework, which might contribute the requirements gap, such as viewpoints of system deployment and system integrators.

4.3.4 Requirements Elicitation Approach for A Location-Aware System - A Brief Example

This is a short example performed on UW Campus Navigator (UWCN) [113] which is an open source location-aware application. The application aims to provide new students with location-aware services around The University of Washington campus. It was developed in C# within the Microsoft .NET framework.

An initial services (location-awareness) pattern module is created by domain experts and software engineer after UW Campus Navigator passed the

assessment. The SPM should contain some initial information (historical information related to location-aware systems) and requirements associated with location-aware systems. The following table presents the sample of the content of SCI and REQ in the initial Services Pattern Module:

Source Code Information (SCI)	<i>Identifier</i>	positionIButton
	<i>Keywords</i>	public; class
	<i>Comments</i>	wrapper of iButton
Requirements (REQ)	<i>FR</i>	location-polling
	<i>NFR</i>	high responsiveness

Table 4.1 Content of SCI and REQ in Services Pattern Module

It is assumed that a concept named – Get|CurrentPosition with an event – positionIButton exist. Thus, <Get|CurrentPosition, iButton> as a tuple of <Concept, Event> will be stored in the KBL for further matching and updates. To discover services candidates, firstly this approach creates a SPM, and constructs a KBL accordingly. For instance, 6 instances in the SPM and 6 corresponding tuples of <Concept, Event> in the KBL are created. The list of tuples in knowledge-based library is: [<MapLocation, getCurrentPosition>; <Magnify, getZoomingSize>; <Shrink, getZoomingSize>; <SearchLocation, getDestination>; <Tracking, track_Click>; <POI, getNewDestination>]. Owing to personal independence and preference of concept naming, the final KBL might appear rather different. Based on our research background, the concept terms more related to software engineering are created rather than those from other specific domains.

Once the services pattern module and knowledge-based library both are constructed, HB-CA is applied along with the content of SPM to 4 source files: Map.cs, POI.cs, mainForm.cs and PreferForm.cs. In this stage, strict matching criteria is not adopted, instead, flexible matching is allowed (i.e., sub-string matching or ambiguous matching). The result list of matching tuples of concepts and events is very similar with the one built above, but with an updated tuple, i.e., <Navigation, getGPSInformation>. The results are demonstrated in this stage in Table 4.2 Below:

KBL elements	Identifiers	Events in Source
<MapLocation, getCurrentPosition>	picMap	picMap.MouseDown
<Magnify, getZoomingSize>	lblMagnify	lblMagnify.Click
<Shrink, getZoomingSize>	lblShrink	lblShrink.Click
<SearchLocation, getDestination>	cbSearch	cbSearch. SelectedValueChanged
<Tracking, track_Click>	menuTrack	menuTrack.Click
<POI, getNewDestination>	menuPOI	menuPOI. MenuItems.Add
<Navigation, getGPSInformation>	buttonNav	buttonNav.Click

Table 4.2 An Updated Knowledge-Based Library (KBL)

The content in KBL indicates the location of concept segments. Once this work is done, static program slicing techniques are applied to decompose the qualified source code reflected from the results of SPM further. Slicing is particularly useful when the code segments are too big. This at length generates code segments of interest. For example, the following code could be of our interest:

```
private void picMap_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
    // Stop Tracking When Map is Clicked
    map.Tracking = false;
    menuTrack.Checked = false;
    map.Recenter(e.X, e.Y);
    picMap.Refresh();
}
```

When the target code is available, the phase of services reimplementaion is reached, which will be discussed in Chapter 5.

4.4 Summary

In this chapter, requirements-related artifacts discovery is discussed as code-related artifacts discovery is rather conventional approach within reverse engineering research field. Hence, the methodologies for SPM are explored at the early stage in our proposed reengineering framework. The content covered in this chapter can be concluded as follows:

- ❑ The framework for context-aware systems in general is composed of three layers, i.e., sensors layer for context acquisition, context management layer for fulfilling non-functional requirements by instructing what application should behave upon the changes to user's requirements and context, and application layer to satisfy functional requirements.

- ❑ The context-aware systems framework gives insight of where functional requirements and non-functional requirements reside in the framework. It suggests that functional requirements are satisfied in the application layer, whilst non-functional requirements are met in the middleware/context management layer. This provides a guide for where requirements recovery should be carried out from the legacy system in the requirements recovery framework.
- ❑ The Services Pattern Module (SPM) in the requirements recovery framework consists of knowledge-based library, source code information and requirements. It underpins the requirements elicitation; while concept generator applies methods onto the content in services pattern module and generates event concepts which are the tuples of <Concept, Event>.
- ❑ The requirements elicitation approach uses a Hypothesis-Based Concept Assignment (HB-CA) method to generate a list of event-linked concept in knowledge-based library.
- ❑ A small location-aware system is used to evaluate to show the intermediate result of the requirements elicitation approach at the services candidate recovery stage.
- ❑ The requirements recovery framework provides one possible way of eliciting requirements behind the source code for further reengineering activities. It cannot be implemented (semi-)automatically since this process contains some sub-processes which require manual work for making decisions.

Chapter 5 – Context-Aware Services Requirements Model and Requirements Evolution Model

Objectives

- To describe the context-aware services requirements model
- To describe the requirements evolution model
- To discuss the relation between requirements evolution and services evolution
- To demonstrate an example of the model of requirements evolution

5.1 Overview

5.1.1 The Problem

Typically, services requirements and context are evolving constantly, services providers may not be able to pick up the changing pace, that is, they may fail to satisfy the emerging requests. For instance, with respect to context-aware Web services-based technologies, Web server inevitably needs to perform some

long-run computations. If the server is not implemented in an asynchronous and parallel way, it may be unresponsive due to intensive context changes. On the other hand, changes of services requirements and context are two primary triggers of services evolution. Specifically, services evolution is expressed through the creation and decommission of its services version behind software evolution.

Context-aware services are concerned with reasoning about surrounding context and adapting services accordingly, whereas few research works focus on supporting context-aware services evolution via requirements modelling techniques. Furthermore, requirements engineering conventionally focuses on users' requirements, e.g., elicitation for high-level goals [68], whilst constraints usually do not received much attention in the course of services evolution. In this chapter, to fully discover those constraints, a derived viewpoints-based Context-Aware Service Requirements Model (CASRM) is proposed; to maintain and specify the requirements evolution process, a requirements evolution model is developed for supporting context-aware service evolution. A medium sized open source case study is carried out for evaluation in the end of this chapter.

5.1.2 Background

Requirements analysis is critical to the success of a software system development. Out of question, requirements evolution may result in changes to later artifacts. In fact, economically speaking, defects are cheaper to remove if found earlier, namely, late changes have bigger impacts on work already done. From evolutionary perspective, the changes in requirements occurring after deployment can be referred to requirements evolution. In [41], requirements evolution is pictured as an intermediate viewpoint between architecture evolution and computer-based system evolution in the evolutionary space. In

fact, requirements evolution can be seen as a task in an umbrella concept – requirements management which studies as to how to control the impacts of changes on requirements. Yet, software reengineering for context-aware systems is about re-implementing the current software solution to continuously meet the needs of its stakeholders and the context constraints in a new environment. Hence, managing requirements evolution is an increasingly important research field particularly in requirements engineering. The challenges of requirements engineering in the area of context-aware services are the continuously changing services requirements and context. Comparing to conventional requirements evolution that focus on the evolving users' requirements, constraints must be given sufficient priorities during the process of services evolution.

A good solution to a system can only be developed given the engineer has a correct understanding of the problem. In modern software system development era, modelling and eliciting a large set of essential requirements and context parameters are the fundamental jobs that have to be done before the late activities in software development lifecycle, in other words, requirements modelling plays a very central role in requirements engineering. Nevertheless, in context-aware computing era particularly, the constantly varying context poses a huge challenge to requirements engineering. Not only does context influence software, but it makes an impact on stakeholders' goals and their choices to meet to them [3].

Based on different modelling purposes, some major techniques used in common requirements modelling are covered in Table 5.1. For example, KAOS [66], a goal modelling technique, provides a multi-paradigm specification language and a goal-directed elaboration method. The i* modelling framework [127] introduces some aspects of social modelling and reasoning into information system engineering methods, especially at the

requirements level. The i* modelling can be considered as an organisation modelling. Non-functional requirements modelling can be found in [28].

MODELLING OBJECTIVE	TECHNIQUES
Enterprises Modelling	Goal Modelling
	Organisation Modelling
Functional Requirements Modelling	OO Analysis
	Structured Analysis
	Formal Methods
Non-Functional Requirements Modelling	Quality Tradeoffs
	Specific NFRs

Table 5.1 Requirements Modelling Techniques

Requirements evolution is still a research topic that somehow is not drawn much attention in requirements engineering community, even though Cheng and Atlee [26] mention the rising popularity of it. The challenge of requirements evolution had been first comprehensively discussed by Harker et al [55]. They concentrate on the structure of requirements and categorise requirements into the followings types: Enduring, Mutable, Emergent, Consequential, Adaptive and Migration Requirement. Adopting formal concept analysis, Fabbrini et al. [40] depict an approach to improving requirements evolution management by making more systematic and effective the

identification of semantic inconsistencies between different stages of requirements evolution.

Much research on evolving requirements still remains on the initial stages in software lifecycle, whilst post-development requirements evolution should be paid sufficient attention in order to facilitate software evolution related activities. For instance, changes to requirements may be dictated by new programming languages that require a paradigm shift. Such changes largely are put forward by software developers and they should have their voices for these kinds of changes. Ernst et al. [38] predict that software of the future will consist not only of code and documentation, but also requirements and other types of models representing design, functionality and variability.

5.2 Context-Aware Services Requirements Model

5.2.1 Concepts for Context-Aware Services

Before introducing our definitions to context-aware service, firstly, the concept of services requirements is recurred. A services requirement can be viewed as a requirements collection of functionalities, non-functional properties and interfaces. Functionalities are the basis of services and a set of functions required to perform in a program to accommodate a certain type of service. Non-functional properties indicate the quality of delivered services, yet they are harder to define, e.g., performance, scalability, reliability, security and so on. Interfaces provide users with a customised user-friendly environment although interfaces might not be necessary in many cases, e.g., clients may only want to query the server for certain services only via the publish/subscribe communication paradigm. All the said requirements are composed of a services requirement. Thus, a services system is designed to meet all the sub-requirements to delivery satisfied services.

Utilising the above definition of service requirements based on users' and developers' views, a context-aware service can be defined according to providers' perspective and requesters' perspective as below. Although there are other viewpoints concerned with, for example, system deployment and system integrators, these definitions are created to facilitate the understandings of the relation between context requirements and services requirements particularly during services evolution. Table 5.2 depicts a definition of context-aware service from providers' and requesters' perspectives:

Context-Aware Service	
Perspectives	Definitions
From providers' perspective	a context-aware service is a group of associate functionalities decided to perform subject to current context settings
From requesters' perspective	a context-aware service is an abstract resource that adapts a capability of achieving goals based on current context settings

Table 5.2 Definitions of Context-Aware Service from Both Perspectives

In a nutshell, a service is a software system. According to the definitions above, a context-aware service embraces abstract information that users require and system behaviours that providers offer. In other words, context-aware services enable users to behold the services provided without being aware of the underlying implementation by providers. To summarise, software evolution was reified through services evolution, while context changes to services are incarnated during the services evolution. Services evolution can be referred to the continuous reengineering to services systems through a series of consistent and unambiguous changes. Last but not least, a context change is another trigger to invoke a set of functions to deliver sought after services. This chapter focuses on how the context evolves rather than how they can be described in high-level description languages.

5.2.2 Customised Derived Viewpoints

A basic framework of requirements model that consists of a collection of viewpoints is described in this section. The definition of viewpoints can be found in [84]. Viewpoints are objects that are loosely coupled, locally managed, distributable. Each viewpoint comprises three kinds of software engineering knowledge: representation knowledge, specification knowledge, and software development process knowledge. Moreover, a key principle of viewpoints is that viewpoints organise software development knowledge based on separation of concerns [86]. That is to say, different stakeholders' interests are expressed in different viewpoints, such as a viewpoint that captures a software developer's concern or a viewpoint that expresses interests of user representative. As a viewpoint may represent various areas of concern within a project, the notations for particular stakeholders' perspective vary. Hence, requirements models provide maintainer with guidance and motivation for requirements engineering activities.

In Chapter 4, a requirements elicitation approach has been proposed in a context-aware system software engineering framework. The requirements recovery framework itself contains a SPM which is created by domain experts and software engineers. Its content is dynamically updated and it underpins the requirements elicitation. In this chapter, as the requirements elicitation approach targets users' and developers' perspectives, the intermediate results can be utilised in requirements recovery framework and build an associate customised derived viewpoints based on combination viewpoints from traditional users' and developers' viewpoints.

This novel requirements model focuses on synchrony of users' requirements and constraints in a services evolutionary view. The requirements model is depicted in Table 5.3 below.

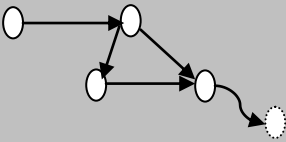
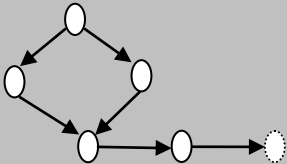
USERS	DEVELOPERS
Service Domain	Implementation Domain
functionalities non-functional properties interfaces	functionalities implementation non-functional properties interfaces implementation
Context	Context
context constrains	predicates
History	History
previous service -> current service	previous implementation -> current implementation
Specification	Specification
	

Table 5.3 Customised Derived Viewpoints from Users and Developers

The model in Table 5.3 presents a combined customised derived viewpoint that consists of derived users' and developers' viewpoints. For users' viewpoints, they comprise the following elements:

- **Service Domain** is a description that expresses the detailed components (i.e., functionalities, non-functional properties and interfaces) of a particular service and its context constraints that trigger the series of functionalities to perform.
- **History** is a work record of changes. It embraces the previous services description and current services description.
- **Specification** used to indicate the contents as the users make changes. Diagrams or other notations may be used.

For developers' viewpoints, they contain the following elements:

- **Implementation Domain** is a description that expresses the detailed components (i.e., implementation of the relative functionalities, non-functional properties and implementation of corresponding interfaces) of a particular service implementation and its predicates that assert performance of associate functionalities.
- **History** is a work record of changes. It embraces the previous implementation description and current implementation description.
- **Specification** used to indicate the contents as the developers make changes. Diagrams or other notations can be used here.

In some cases, the inadequate communications between requirements engineers and end-users leads an increasing gap between requirements and implementation. In fact, different viewpoints can be treated as dialogues

between the relevant stakeholders' to reduce the discrepancy of their communications. While some viewpoints are essential to other actors, e.g., system deployment and system integrators, viewpoints from users' and developers' perspectives are more related to our aim, i.e., to reconcile requirements and implementation during the reengineering activities as services evolve.

5.2.3 Requirements Model for Context-Aware Services

Based on the derived viewpoints discussed above, a context-aware service requirements model is developed. The components and steps to construct context-aware service requirements are represented in Figure 5.1.

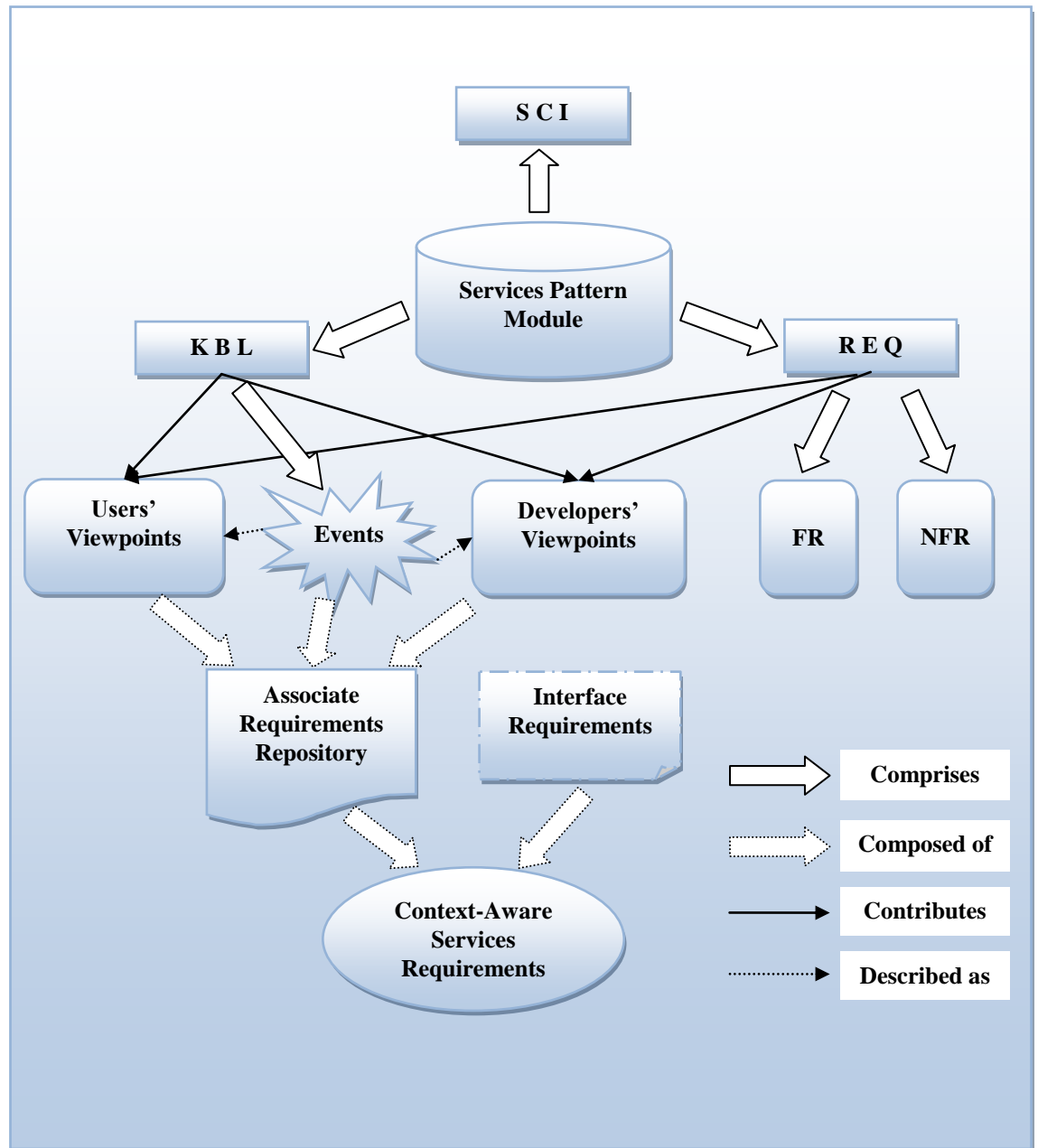


Figure 5.1 Context-Aware Services Requirements Model (CASRM)

Benefiting from our previous work on requirement recovery framework (RRF) that discussed in Chapter 4, this Context-Aware Service Requirements Model (CASRM) extracts current requirements from source code level and reconstructs new context-aware service requirements primarily based on users' and developers' customised derived viewpoints. The main components are detailed as below:

- **Services Pattern Module (SPM)**, this is one of the major components of Requirements Recovery Framework (RRF) that supports requirements elicitation approaches to capturing existing context-aware service requirements. As discussed in Chapter 4, this module embraces the following components (as described in 'Comprises' arrow in Figure 5.1) Source Code Information (SCI), i.e., comments, identifiers and keywords; Knowledge-Based Library (KBL), which holds a list of concept bindings and Requirements (REQ), i.e., functional and non-functional requirements. The overall output of an elicitation approach is a list of concept bindings with linking to regions of source code. The bindings are kept in the format of event concepts, i.e., a tuple of concept and event ($\langle \text{Concept}, \text{Event} \rangle$) in KBL. From developers' perspective for example, this first element of the tuple represents the implemented functionality, while the second element of the tuple indicates the predicate that asserts the performance of this functionality. This entire framework takes advantage of the fact that the implementation of context-aware services most likely is done via event-driven programming.
- **Users' Viewpoints and Developers' Viewpoints** are derived viewpoints and the content items are introduced in Table 5.3. The detailed contents can be found and drawn from KBL and REQ. They are the primary contributors for both viewpoints. Changes may be

dictated not only by those that are caused by users who keep changing their mind, but also by the availabilities of new techniques that developers would raise the needs to consider adopting alternative implementation strategies or paradigms. The two viewpoints need to be in phase.

- **Events** locate in the second element of the desired tuple <Concept, Event> in KBL, which in turn, it is part of the result of RRF approach. Events are triggered whenever context changes. Event can be described as a context constrain or predicate depending on viewpoints from users' or developers' perspective. In context-aware programming, in many cases, short-live computation requests entail services providers to perform asynchronous computations in order that the context-aware server renders in a prompt responsiveness. Technically, when a services provider receives an asynchronous request, the context-aware application responds to the corresponding events by registering event handlers with its middleware or context server. Inherently, this suggests that it is a concurrent application which performs asynchronous computations.
- **FR and NFR**, the abbreviation of Functional Requirements (FR) and Non-Functional Requirements (NFR). FR, such as temperature-awareness in a smart room, location-awareness in a campus; NFR, so called soft goal in requirements engineering. NFR may include performance, scalability, and reliability etc. For example, Web services should be delivered to end-users in a promptly responsive way.
- **Associate Requirements Repository Engine (ARRE)** is a synthesis (as depicted as 'Composed of' in Figure 5.1) of traditional users' and developers' viewpoints, and context constrains and predicates that

assert the requirements are satisfied. Viewpoints, not only conventionally make changes consistent, but build a relation between both viewpoints and reveal constraints to improve the evolving implementation in order to mitigate the pain of software evolution. It is ARRE that constraints are fully discovered and given the same priority as users' functional requirements. For instance, in forward engineering stage, developers would face a choice to select a proper programming language to implement the overall requirements. Instead of choosing mainstream object-oriented languages, a general programming language, which enables programmers to build a domain specific language easily, e.g., an implementation language with abilities of context-oriented programming [56], may be more appropriate than the former. Created by domain experts and seasonal software engineers, ARRE synchronises both derived viewpoints and provides suggestion of modification to functional requirements.

- **Interface Requirements** is one of the traditional requirement elements of context-aware services requirements where user experience needs are expressed. In fact, in the context of a Web services computing environment, these requirements are less necessary to fulfil as clients often access sought-after services via HTTP, SOAP etc without using a Web browser. User interface requirements along with ARRE are composed of the ultimate desired requirements.
- **Context-Aware Services Requirements** are the ultimate desired requirements that the target software system is to meet. For each time the context-aware services requirements are generated, they will serve as the initial requirements input for the proposed requirements evolution model that will be described later.

To summarise, as described in Figure 5.1, SPM comprises of SCI, KBL and REQ. Moreover, KBL and REQ contribute information to two viewpoints. The content of viewpoints and Events are composed of ARRE while Events can be described in the viewpoints. Therefore, in the context of context-aware services evolution, a nourished knowledge-based output is indirectly extracted from RRF which underpins the future composition of context-aware services requirements. These requirements that mingle FR, NFR, interface requirements and context requirements are actually the initial input of the said requirements evolution. Clearly, this work is based on post-development, i.e., after the current services are put into operation or services have been evolving for period of time. The results suggest that requirements evolution model is needed to maintain those evolved requirements and address the impacts on the original services system.

The obvious advantage of the requirements model presented is that focusing on only user' and developers' viewpoints can gain better understanding of the original requirements for the legacy system and in turn disclose the implementation limitation to fulfil those requirements. Separating less important interface requirements allow better implementation of functional requirements and non-functional requirements. One of the disadvantages of this model is the building of SPM, as it evolves manual effects from domain experts and software developers. The other limitation is that further extraction techniques are needed to draw information from the two viewpoints for ARRE to process. Therefore, the future work can be (semi-)automating SPM construction and techniques to fetch details from viewpoints.

5.3 Requirements Evolution Model for Context-Aware Services Evolution

In order to better the explanation as to how CASRM can fit into the holistic picture of context-aware services evolution, a requirements evolution model to specify the evolution process is described in Figure 5.2. In this process model, the context-aware services requirements are distilled into services requirements and context requirements, and investigate two possible triggers, i.e., the changing services requirements and context requirements.

Services requirements and context requirements have different evolution process with different modifying rules. The interaction of these two requirements is indispensable and they influence each other. Separating this services requirements and context requirements in the proposed model allows for applying different modifying rules. Moreover, the feedback system guarantees the quality of requirements via reasonable acceptance criteria and eventually generates combined evolved requirements. In fact, requirements management entails a collection of activities that consists of tasks for such management in details. Zagajsek et al. [130] present a requirements management process model for software development based on legacy system functionalities. In their proposal, the link between requirements and expected software change management is realised mainly by the documentation associated with the requirements. To manage such a broad concept of requirements is difficult. Their proposal does not split the requirements into more specific requirements, for example, services requirements and context requirements rather than more traditional division – functional requirements and non-functional requirements. However, the proposed requirements evolution model in this chapter separates requirements into two types of requirements at the beginning; distilling each of requirement further into more

specific requirements along with modifying rules and a feedback system; finally, these requirements are combined together into guaranteed evolved requirements. While modifying rules are not comprehensive and required new rules in the future, this model clarifies requirements evolution process greatly.

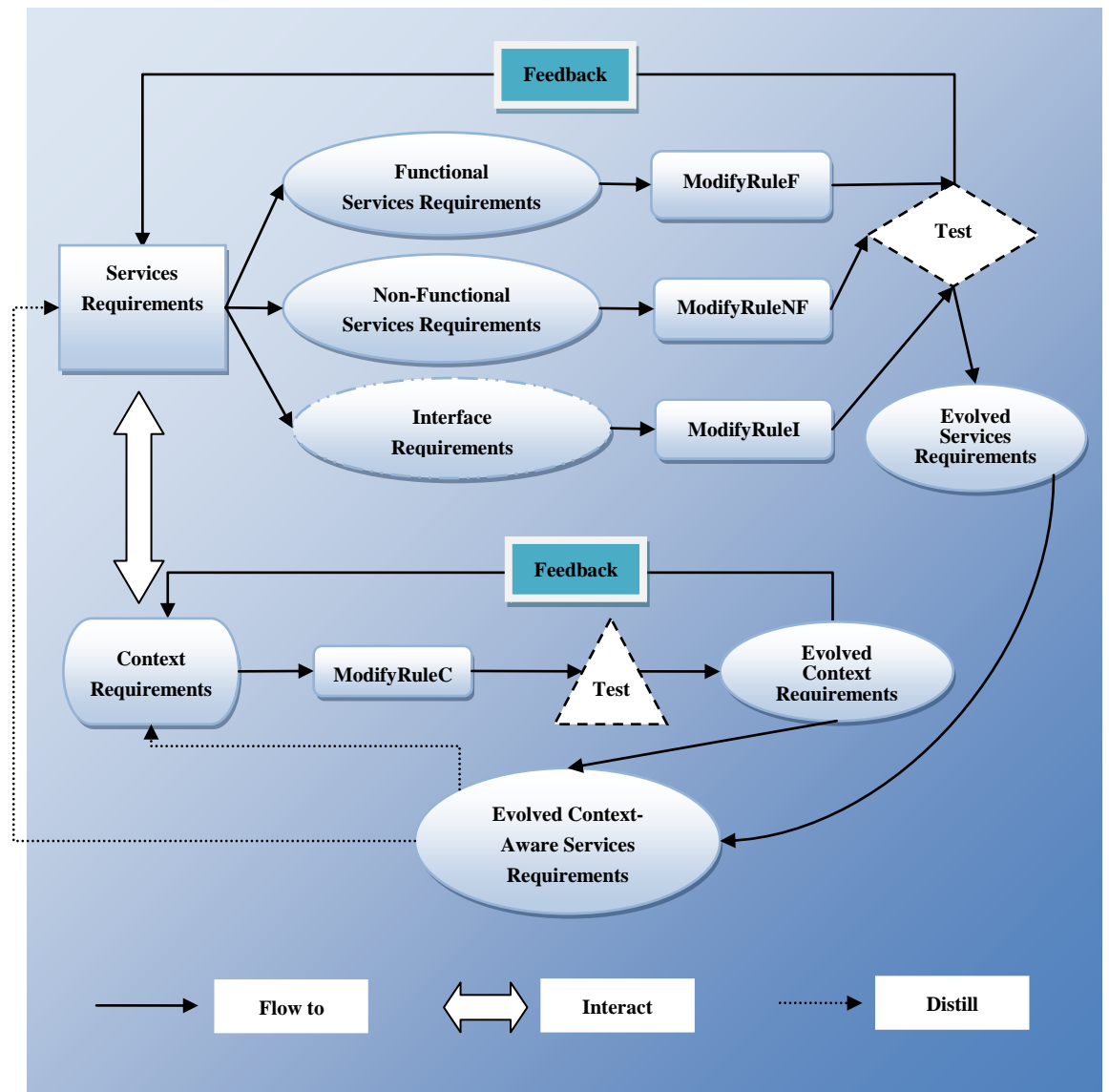


Figure 5.2 Requirements Evolution Model

The model for context-aware services requirements evolution comprises three working stages:

- **Initial Requirements of Services and Context** is the input services and context at initial stage where current context-aware services are discovered by RRF. The requirements of services and context are separated in the first place due to different modification rules for them at a later stage.
- **Defined Requirements of Services and Context** is the key stage of the entire requirements evolution. Services requirements will be divided further into three parts: Functional Requirements, Non-Functional Requirements and Interface Requirements. The corresponding modifying rules are ModifyRuleF, ModifyRuleNF and ModifyRuleI. Basic modifying rules include add, delete, edit, replace, compose and so on. The modified requirements are subject to test with reasonable acceptance criteria before release. Quality of Service (QoS) is adopted for each test case. QoS is defined as $Quality(Q, S) \models Constraint(C)$. Feedback will be sent back to each initial requirement for evaluation. A case study will be performed later to exercise this defining phase.
- **Released Requirements of Context-Aware Services** refer to the final version of the desired context-aware services requirements to be fulfilled in the late services reengineering activities. The requirements combine the evolved services requirements and evolved context requirements. They will be eventually become initial requirements upon the next requirements evolution.

Services requirements and context requirements are closely related. Services requirements are described in a particular context environment, in other words, context requirements constraint services requirements, e.g., in a healthcare context-aware application, patients' appearance (e.g., by image) can be

detected upon the hospital, their names and full health records will be shown on reception. On the other side of the coin, services requirements should be flexible enough to take into account generic context as it evolves. For example, besides patients' images, their fingerprints should be also an acceptable avenue to accessing their health records.

In conclusion, the requirements evolution model aims to reconcile the gap between requested context-aware service requirements and current context-aware service requirements. From services evolution point of view, once the evolved context-aware services requirements are available, software engineer will carry a series of reengineering techniques to fulfil those requirements. The evolved requirements can be exploited for various services evolution frameworks. However, the basic of services reengineering process can be classified as follows:

- **Context-Aware Services Discovery:** This is the first foremost essential task needed to carry out. Reengineering techniques candidates may embrace formal concept assignment, programming slicing, programming refactoring and restructuring and so on.
- **Context-Aware Services Implementation:** Traditional forward engineering techniques will be applied in this phase to fulfil the evolved requirements.
- **Context-Aware Services Integration:** In this integration stage, code gluing and wrapping techniques are often applied to integrate the exiting components (services) and desired services in the new system.

5.4 The Relation between Requirements Evolution and Services Evolution

When it comes to reengineering a system, it is widely accepted that the first most crucial step during the holistic reengineering activities is reverse engineering that understands the subject system's components and creates higher level representations of the system. Essentially, software engineers and developers play a key role at this stage; their expertise and knowledge about the subject system back a sound series of software evolutions. Notwithstanding, along with the traditional top priority in users' needs, constraints (e.g., design requirements and implementation requirements) are always de-emphasised by taking the implementation issues are far more natural for granted.

In some cases, developers are forced to give in their needs to compromise users' needs. Hence, inefficient implementation will make services evolution much more difficult. For example, context-aware services computing always requires sophisticated parallel and asynchronous computing. Many existing modern general programming languages are not primarily designed for such computing issues back to when they were invented. Although great effects have been made to evolve these mainstream programming languages to able to address the said issues in a much concise way, the realistic situation is not that optimistic. Perhaps the object-oriented paradigms restrict themselves too deep to extend to other programming paradigms easier and further. As a result of that, the implementation of asynchronous computing concept for example is unavoidably hard-wired in the languages, yet other relatively new programming languages (e.g., F#, Erlang, and Scala) are capable of addressing those issues via their high-level features from languages themselves without considering adopting some concepts from Design Pattern. Consequently, it will take much longer time to figure out what the ad-hoc code does in the legacy

system implemented in inappropriate languages, which in turn leads to the fact that maintainers have to pay higher cost to maintain this type of legacy systems. At length, it will only exacerbate the severity of the software system heading to service stage [13]. Therefore, conventional emphasis only on users' requirements hinders software evolution directly and services evolution subsequently.

5.5 An Example

This short case study is carried out based on The Java Context Awareness Framework (JCAF) [10], a Java-based open source context-awareness infrastructure and API for creating context-aware software applications. JCAF contains some libraries that facilitate context-aware application development. The class 'ContextEvent' which defines a generic event that indicates that context has been changed. It has the following five bespoke public methods: `getEntity()`; `getEventTye()`; `getItem()`; `getItemType()`; `getRelationship()`. From their examples, a "ContextChanged" service is selected to evaluate our process model. The following code indicates the one of a generic implementation of this service:

```
public void contextChanged(ContextEvent event) {  
    System.out.println("context changed: ");  
    Entity entity = event.getEntity();  
    System.out.println(entity.toXML());  
}
```

In terms of our RRF approach, it is assumed that initially, domain experts or software engineers have a generic SPM in place. For instance, the following table can be seen as a snapshot of Knowledge-Based Library (KBL).

KBL	Identifier	Event
<ContextChange, getCurrentContext>	contextChanged	buttonEve.Click
<Navigation, getGPSInformation>	buttonNav	buttonNav.Click

Table 5.4 A fragment of Knowledge-Based Library (KBL)

When SPM is available, under CASRM depicted in Figure 5.1, a table is created to constitute different services requirements. The table below represents the services requirements of ‘ContextChange’:

CA Services Requirements	Description
Functional Requirements	Concrete Context Changed
Non-Functional Requirements	High Responsiveness (no long delay)
Interface Requirements	Relative Environment Changed
Context Requirements	New Context Accepted

Table 5.5 Services Requirements of ContextChange

The context-aware services requirements then are separated into services requirements and context requirements for different modifying rules. For instance, users may want to conduct a social habit experiment and keep a

record of a series of old context information instead of discarding the pervious context. In this case, the ‘edit’ modify rule is taken, and this related functional requirements will be edited as “Concrete Context Changed” and “Keep a Copy of the Points Where Context is Changed”. Then the modified requirements are subject to test in terms of the formula: $\text{Quality (Q, S)} \models \text{Constraint (C)}$ in a specific context. Finally, feedback will be sent back to initial related requirements with corresponding actors, in our case, the users and developers for ultimate confirmation. As our model is tested with more cases, they suggest some promising results on context-aware services requirements analysis particularly during the reengineering activities. Side of our findings is that constraints tend to be increasing vital, which entails not only a better specification of the system conventionally, but also emerging new techniques that are sought after.

5.6 Summary

In this chapter, a derived viewpoints-based context-aware services requirements model and requirements evolution model are presented. The relationship between context-aware services evolution and requirements evolution is discussed. Finally, an example is described to show the management of requirements evolution.

- ❑ The fact that users’ requirements and constraints are not always given the same priority in requirements evolution hinders software evolution directly and services evolution subsequently.
- ❑ Changes of services requirements and context are two primary triggers of services evolution; evolved services requirements and evolved context requirements are composed of the initial requirements for the requirements evolution model.

- ❑ The derived viewpoints-based Context-Aware Services Requirements Model (CASRM) is proposed to fully discover the importance of constraints, i.e., design requirements, implementation requirements, and interface requirements, which paves the way for the third stages of reengineering process – reimplementation; whereas the requirements evolution model is developed to maintain and specify the requirements evolution process for supporting context-aware services evolution.
- ❑ An example is given to show how the evolved requirements generated by CASRM are managed in the requirements evolution model to achieve requirements evolution.
- ❑ This work in this chapter extends the application of requirements recovery approach and ensures the elicited requirements can be well-maintained with requirements evolution models. One limitation of this chapter's work is that quantitative methods are not discussed with full contents even though qualitative methods are presented in more details with Figure 5.1 and Figure 5.2.

Chapter 6 – Context-Aware Web Services Reimplementation with ContXFS Support

Objectives

- To describe the requirements for the reimplementation
- To describe the architecture design for the reimplementation
- To discuss the reimplementation concerns and strategies
- To introduce the F# language and the development tools
- To introduce context-oriented programming and F# library ContXFS
- To demonstrate an example of such services reimplementation within the proposed reengineering framework and the application of ContXFS

6.1 Overview

6.1.1 The Problem

When the requirements engineering is completed, the design and implementation is the next stage where an executable software system is developed in the overall proposed reengineering framework approach. Based on the recovered requirements and code-related artifacts, along with the new

requirements, software design can be used to identify the software components and their interrelationship, whereas software reimplementation is a process to develop a program to fulfil the combination of requirements by realising the relevant envisioned design. In effect, software design and implementation influence each other. For instance, adopting an object-oriented programming language for implementation can suggest that Unified Modelling Language (UML) would be chosen to document the software design.

Typically, in a largely scalable Web services-based environment, context-awareness is concerned with reasoning about the surrounding well-defined context and adapting the interpreted services accordingly (almost) on the server-side, and finally distributing the services to clients in a reliable way through trustworthy network protocols. Most of Web services-based context-aware systems are either partially or completely implemented within the object-oriented programming paradigm based on their middleware or frameworks [109]. In order to highlight the implementation part in this chapter, an assumption is made that the combined requirements and code-related artifacts have been available and are corresponding to the development, which is discussed in the previous chapters. A functional programming approach is proposed with library support to services reimplementation. The supporting libraries will be discussed in details in Chapter 6. This functional approach in this chapter embraces methods that address functional requirements and non-functional requirements by taking into consideration implementation strategies, e.g., overlapping communication computation on the server-side.

6.1.2 The Background

When it comes to Web services-based context-aware system development, the properties from other emerging software paradigms share a similarity. More recently, a portmanteau term ‘Internetware’ has caught many researchers’ eye,

particularly in China. Internetware [123], a new software paradigm, distils and synthesises the original concept of ‘internet as a computer’. This evolving software paradigm entails some issues to be addressed: firstly, a software model is created to abstract the behaviours of the Internetware entities, that is, these entities should be wrapped as components (servers), acting as agents, interoperating as services and running on demand manner. Secondly, a middleware is designed to seamlessly bind the higher level Web services/applications and lower level supporting components/tools together. Internetware entities are governed by the middleware. Thirdly, an engineering methodology is proposed to develop Internetware entities. Fourthly, a quality evaluation framework [79] is needed to assure the software quality.

Upon the above key properties that Internetware beholds, it is very natural to discover the similarities shared between Internetware and context-aware Web services. Both require a supporting programming model to abstract the detailed implementation complexities; both accommodate a middleware to manage Internetware entities/context information properly; both use a series of (re)engineering methodologies to develop every artifact and maintain the evolvability of the holistic system; and last but not least, both emphasise the non-functional requirements of the overall system to highest level. Nevertheless, the concept of Internetware is expressed in a more abstract way than that of context-aware Web services.

In summary, the similarities between Internetware and context-aware Web services that discussed above are not coincident, but a natural outcome of software evolution [124]. There are not many fully implemented context-aware Web services that truly fulfil some of the core non functional requirements per se. Furthermore, most of the implementation approaches are based on object-oriented techniques as discussed later. By extending our proposed work on

services recovery in Chapter 3, this chapter focuses on server-side development with some implementation strategies and explore how programming languages can assist the development of the Web services-based systems.

6.2 Reimplementation Requirements

In our proposed framework, the redevelopment requirements are composed of the recovered requirements from the source code and the new requirements. This entails requirements analysis that decides the final requirement candidates to be satisfied. The recovered code-related artifacts from the source code can be used as a reference to identify if the current requirements are fully fulfilled or if another programming language is more capable of addressing the implementation issues. For example, to exercise asynchronous programming for concurrency, without asynchronous programming models supporting in many object-oriented programming languages (in fact, in the time of writing, influenced by F#, asynchronous mechanisms will be added in C# 5.0), software developers will end up hard-wiring the chosen languages to carry out convoluted development. Although program comprehension is a well-studied research topic within software engineering community, it will inevitably lead to more difficulties in comprehending the programs in the mentioned systems, let alone extracting reusable code segments from the source code. This will result in higher maintenance costs in the future. Therefore, during the reengineering process, constraints should be given same priority as users' requirements. For instance, a more suitable programming language would be the one with a higher abstraction expression in the language itself even though the language is from a different programming paradigm.

6.2.1 Requirements for Implementing Context-Aware Web Services

In Web services-based context-aware system development, from the context-awareness perspective, a clarified and consistent context definition is a prerequisite that underpins the architecture of context, whilst a context model is designed to reason and interpret all dynamic evolving types of context data, even when encountering undefined context, the model is still able to deliver results in an unobtrusive way. From the Web services perspective, concurrency is a long-time topic studied in programming for distributed system. Scalability, reliability, and evolvability are traditionally symbolised as non functional requirements. In fact, in the context of service-oriented systems development, non-functional requirements are often referred to as Quality of Service (QoS).

In general, the requirements for developing context-aware Web services-based systems can be classified as functional requirements and non-functional software requirements. Functionalities are the backbone of the services systems, that is, a set of functions (in programs) invoked to accommodate some types of services that clients request. While satisfying functional requirements plays a central role in achieving sought-after goals of the Web services systems, the exponentially increasing clients' requests driven by the open and dynamic internet power make non-functional requirements more difficult to meet than functional requirements [28]. It entails its non-functional requirements to tip at the top implementation priority. For example, when more than 10,000 clients are simultaneously requesting the desired context information from services, how the servers ensure the data propagated to the correct clients without letting them wait too long for it. In other words, despite a context-aware Web service fulfil all the essential functional requirements, as long as some of the critical

non-functional requirements are not met, it will fail to satisfy clients' needs and even though context is appreciable per se.

The list of requirements for context-aware Web services development can be carefully drawn from the requirements for context-aware systems and Web services developments respectively, but it is not the best solution as some of them are not close related. Instead, the current characteristics of Web services-based context-aware systems are depicted in Chapter 3 to select the requirements that are crucial but not commonly met or difficult to be met. To categorise these concrete requirements, Galster's taxonomy is adopted, which is only for non-functional requirements in a service-oriented context. The taxonomy implements three main categories of non functional requirements: a process requirements, non-functional external requirements, and non-functional services requirements. Based on the fact that the taxonomy does not cover functional requirements and functional requirements are in general less difficult to fulfil than non-functional requirements, two most prominent of them are chosen, i.e., context-awareness and concurrency. Along with the challenges and requirements studies on [96, 111], a table is created to detail the corresponding requirements as follows:

Requirements Types	Detailed Requirements				
CFR	Context-Awareness	Concurrency (distributed system)	Asynchrony	Parallelism	Reactive Computing
NFPR	Standard Requirements	Composition Requirements	Implementation Requirements	Solution Constraints	Documentation requirements
NFSR	Reliability	Scalability	Performance	Interoperability	Evolvability

Table 6.1 A Sample of Requirements for Context-Aware Web Services

Table 6.1 depicts our analysis results of the current concerned requirements that are crucial yet not fully satisfied. From the table, the following requirements types contain: Core Functional Requirements (CFR), Non-Functional Process Requirements (NFPR), and Non-Functional Service Requirements (NFSR). Specially, Core Functional Requirements (CFR) consist of context-awareness, concurrency, Asynchrony, Parallelism, and Reactive Computing; from non-functional requirements perspective, Non-Functional Process Requirements (NFPR) covers Standard Requirements (e.g., the development of a Web services-based context-aware system has to be ISO9000 conformant), Composition Requirements (e.g., composable Web services), Implementation Requirements (e.g., .NET Framework), Solution Constraints (e.g., the legacy system has to be integrated with newly built the Web services-based context-aware system), and Documentation Requirements (e.g., Documentation has to be created during the ad-hoc programming); Non Functional Service Requirements (NFSR) contains Reliability, Scalability, Performance, Interoperability, and Evolvability. The summarised requirements

over the critical functional and non functional requirements must be addressed as a small case study is carried out in last section in this chapter.

6.2.2 Requirements Mapping

A good design of the said systems prior to implementation can considerably reduce the implementation difficulties, yet the current situation is that the majority of the said services are implemented within the object-oriented paradigm, which some critical and essential implementation issues can be better solved by other paradigm languages which allow software developers to abstract the implementation problems in a higher abstract level. In reality, programming using object-oriented languages to fulfil some of the requirements described in Table 6.1 can be very hard as certain requirements (e.g., concurrency, parallelism, scalability and so on) will inevitably force the object-oriented programming developers to bend the language harsh enough to tackle the implementation issues by convoluted development (it is well-known that mutability is ‘enemy’ of concurrency!). For this reason, it is effective and efficient to map the implementation requirements to the programming language features or properties by comparing the results with the current pervasively used languages can offer.

With the requirements analysis discussed above and comparing the language support from three main programming paradigms (i.e., imperative, object-oriented and functional), Table 6.2 can be created, which contains features which facilitate the implementation issues. In the light of requirements that this table suggests, the desired characteristics of the potential languages can be easily found. For instance, when the context are defined, discriminated union type and pattern matching offered in functional programming languages can be used to easily express the relationship between different strong type of context values and their behaviours.

Desired Characteristics	Terms in Language
Strongly typed	Strongly typed system
Arbitrarily matching any type of value	Pattern matching
Event can be used as value	First-class event
Create types with well-organised behaviour	Discriminated union data type
Immutable value	Immutability
Support asynchronous	Asynchronous programming model
Interoperability	Uniform framework

Table 6.2 Reflected Requirements for Development

Table 6.2 implies that the more terms can be found in a programming language, the higher possibility in general it will become the candidate for implementation language. For example, the asynchronous programming model in F# [105] provides an ‘async’ library to facilitate the asynchronous programming; pattern matching in F# supports arbitrarily matching any type of value; discriminated union data type enables programmers to create types with well-organised behaviour and so on. Further comparison between language choices will be described in later section.

6.3 Architecture Design

Traditionally, the client-server architectural model consists of a set of servers, a set of clients, and the network that underpins the communication between the servers and clients. However, the proposed architecture design for context-aware Web services can be divided as Client side, Web services Application side, and Server side. This architectural model is not comprehensive, while it covers the essence of the evolving Web services-based context-aware systems. Figure 6.1 describes the details of the components in this model.

6.3.1 Client-Side

In our proposed design model from the clients' side, users can access the context-aware Web services via HTTP or mobile devices via SOAP. Other communication protocols may also be supported, e.g., Web Services Description Language (WSDL). The context-aware sensors communicating with sensors on the server side are either embedded into the devices or mounted around users' premises. For example, visitors' smart phones connected to school's local network can be used to as a location-aware system to provide them with the direction in a campus. Hence, clients may have to know the names of the available servers and the services that they offer. In a nutshell, the main function of the applications on client-side is to search the sought-after services that satisfy a range of parameters.

Historically, due to promising language features that JavaScript can provide, JavaScript has been used for client-side Web development for many years. For instance, functions in JavaScript can be passed as arguments to another functions and returned as values; other functional features like, anonymous functions and closures are commonly adopted, combinator operations such as

mapping and folding over lists are also widely used. Moreover, with frameworks and libraries support, JavaScript makes itself a very strong candidate to be chosen for Web development on the client-side. Yet the development of Web services-based context-aware systems also require the language to express content-awareness in a concise way, the combination of Discriminated Union Data Type and Pattern Matching is the way forward. In other words, it enables programmers to arbitrarily match any type of value, whilst such type is with well-organised behaviour that addresses a problem in a concise way. These operations are frequently applied to the functional programming language values.

6.3.2 Web Services Applications

The context-aware Web services application is augmented in the proposed design model in order to highlight the implementation issues. Typically, various Web services applications are asynchronously or synchronously communicating with the context-aware Web services that reside in the server side via given network protocols. Clearly, the variety of functionalities of applications can be implemented within the application themselves and it could empower the capability of the application on client side. However, in order to mitigate the development from client side, such implementation should be moved to the server side where far more computing resources are available. This is one of the reasons why the context management component is placed in the server side as a middleware for encapsulation, which enhances the application reusability. Because the heterogeneity of functionalities pervasively appears in the applications from the client side, it is impossible for software developers to predict such degrees of functionalities. Context server fits well in the client-server architectural design model.

To summarise, although the actual reaction to different events and context instances is implemented on the client side, the context server on the server side manages desired services delivery and fulfils context-awareness requirements behind the scenes. The context management as a conceptual layer has been discussed in Figure 5.1 in Chapter 5. The next section will describe the further components on the server side.

6.3.3 Server-Side

On server side in Figure 6.1, it has the following four main parts:

- **Context Sensor**, its main task is to capture incoming context. Certain sensors may perform some context aggregating work depending on the type of sensors and the context server in the context management layer.
- **Context Management** which is the engine of processing incoming context data before delivering to the end-clients. The tasks include aggregating, interpreting, and reasoning the incoming context data. For instance, to aggregate context data, programmers may find creating a type that represents different type of more concrete context data more appealing than the object-oriented programming techniques such as inheritance. Discriminated union data type exactly accommodates such need.
- **Context Database** is for context data storage and query, and it is connected with context management component. To facilitate the implementation of accessing context database, F# [104] is a good candidate as it is capable of leveraging the functionality provided by Language Integrated Query (LINQ) in .NET Framework and related component for heterogeneous execution [103] in a concise way.

- **Context-Aware Web Services** that act as agents communicating with other Web services from the same side and client side. These heterogeneous Web services along with the context management lie in the heart of the server side. This chapter emphasises the crucial requirements for developing such Web services with open and dynamic nature throughout this chapter. Specifically, the context server on the server side facilitates concurrently a large amount of queries and performs context aggregation, interpretation, and reasoning to ease the computation task from the client side. It is more likely that the context server often handles the demanding requests from the client side in an asynchronous way than the synchronous way. To facilitate the implementation of such type of computations, a language able to carry out asynchronous programming is required. Languages such as F# are good candidates that embrace asynchronous programming model.

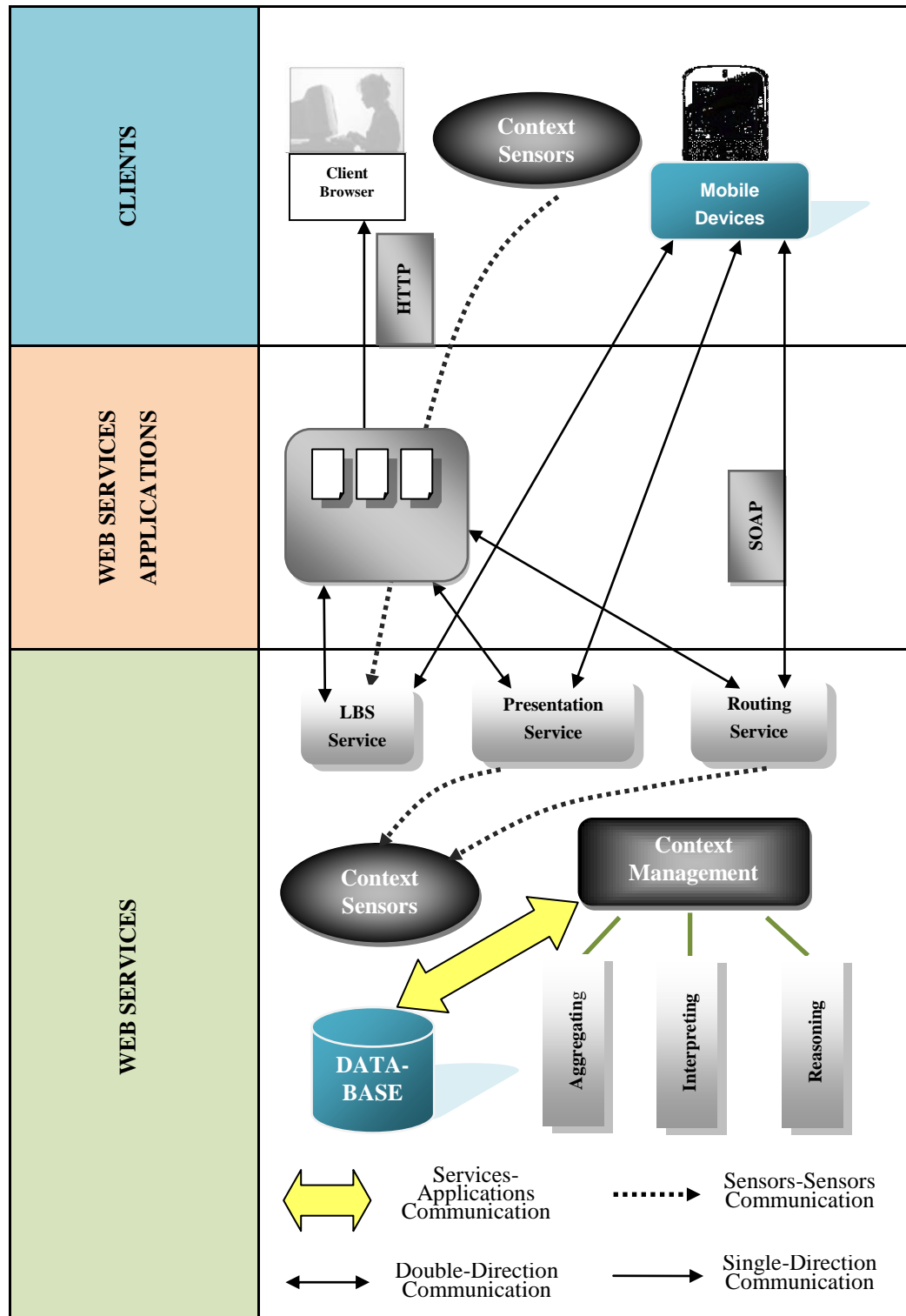


Figure 6.1 Proposed Architecture Design

In Figure 6.1, the black double-arrow lines represent communication between Web services and Web applications. The black single-arrow dashed lines indicate that communications between context-aware sensors to sensors and Web services to sensors (Web services ask context data from sensors). The big yellow double-arrow indicates the context data transfer between context management and context database. Context server is augmented here to emphasise the tasks that it performs. The context server can be implemented either for each service or for multiple services.

6.4 Reimplementation Concerns and Strategies

Following the discussion on requirements and architecture design for context-aware Web services-based systems in the previous Sections 6.2 and 6.3, the concerns and strategies of reimplementation will be discussed in this section. A brief comparison of both object-oriented approach and functional approach for Web services-based context-aware systems will strike out the discussion. To summarise, in the case of the implementation of prospective systems, choosing an appropriate implementation language is a direct and effective way to free software developers from the restrictions by some conventional popular programming languages during redevelopment process.

6.4.1 Reimplementation Concerns

The reimplementation issues discussed in this section focuses the reimplementation on the server side. The concerns can be classified as follows:

- **Performance Issues**, more often, the context server that resides on the server side handles multiple concurrent requests from the client side. This further recurs to the non-functional requirements: scalability and

reliability. Specifically, the throughput of application brings scalability up front to be a crucial issue, as context server has to be designed to handle increasing requests in an appropriate way. Moreover, reliability is a conventional issue that Web services need to ensure. In reality, the context server in Web services can be designed to perform asynchronous computation that deals with such demanding amount of requests. Hence, a programming language is needed to facilitate asynchronous programming.

- **State Sharing Issues**, with a high numbers of clients accessing to the Web services, shared state is evitable. Yet, maintaining mutable state is a notorious programming issue that gives many programmers a headache. Nevertheless, functional programming languages embrace immutability without shared states. This provides software developers a better solution dealing with mutable states.
- **Long-Running Operation Issues**, Web services sometimes need to perform computations that take a relatively long time to complete, e.g., reading a file from a file system. To cut down on the processing time, Web services need to offer a mechanics to processing requests in a parallel way.

The list of issues above is not comprehensive. Clearly, the range of issues depends on the existing code-related artifacts that have been recovered, the recovered requirements and new requirements, as well as the details in architectural design model. Notwithstanding, F# makes three primary contributions to parallel, asynchronous and reactive programming in the context of a VM-based platform such as .NET [106]:

- Functional programming greatly reduces the amount of explicit mutation used by the programmer for many programming tasks.
- F# includes a powerful ‘async’ construct for compositional reactive and parallel computations, including both parallel I/O and CPU computations.
- ‘async’ enables the definition and execution of lightweight agents without an adjusted threading model on the virtual machine.

In the same vein, Bloch [17] points out, “Classes should be immutable unless there is a very good reason to make them mutable. Immutable classes provide many advantages, and their only disadvantage is the potential for performance problems under certain circumstances...If a class cannot be made immutable, limit its mutability as much as possible.” Immutability is a core concept in functional programming languages. In general, shared-memory concurrency is a hard and complex issue. Using immutable values avoids many programmatic issues in parallel and asynchronous computing, e.g., immutable values can be passed between multiple threads without unsafe concurrent access to those values. In other words, race conditions are exempted.

It is the set of functional concepts that functional languages prove themselves as a better candidate than those from object-oriented programming paradigm. Furthermore, F# offers extra yet prominent programming features that are a good fit in our solution domain. Therefore, a more comprehensive table can be drawn, which lists sought-after characteristics in F# and their advantages over other mainstream programming languages. Table 6.3 depicts the details below:

General Language Features	Advantages
Strongly Typed System	Safety
Type Inference	Succinctness/Code Reduction
Immutability	Mitigating Concurrent Programming
Higher Order Functions	Functions as Parameters or Return Results
Closures	Capture Scoped Variables
First Class Events	Events Used as Values
Discriminated Union Types	Creating Types with Well-Organised Behaviour
Pattern Matching	Matching Any Type of Value Arbitrarily
Function Composition	Compositing Functions
General Language Features	Advantages
Asynchronous Programming Model	Supporting Asynchronous Programming
Agent-Based Programming	Supporting Agent-Based Programming
Computation Expressions	Enabling Ad-Hoc Programming
DSLs-Enable	Facilitating DSL implementation

Table 6.3 F# Features and Advantages

In summary, F# functional features facilitate DSL implementation that fulfils the context-awareness requirements, whilst the F# asynchronous related programming models make itself a good fit for handling concurrency and parallel computing in Web or Cloud computing development. In fact, the language chosen affects how software developers think about the

programmatic problems, as well as the structures of the solutions they come up with. Rather than spending considerable time and effects on arranging the classes and objects to the right abstraction in the object-oriented programming paradigm, why not spare the time and effects to address the core issues that really matter such as processing data in parallel. No wonder Vinoski came to understand the impedance and said, “After pondering this problem for years, I finally concluded that our efforts were ultimately most impeded by the programming languages we chose” [114]. Hence, choosing an appropriate programming language can largely alleviate software developers’ programming burdens before reimplementation is carried out.

6.4.2 Reimplementation Strategies

The reimplementation strategies on server side vary in different redevelopments. For example, communication overhead hampers the performance in high-performance computing system [99]. To mitigate the negative impact of communication, overlapping communication and computation [101] via asynchronous communication primitives is a one of the widest accepted approaches. Conventionally, however asynchrony often makes code more intricate and reduces code readability due to much efforts have been put on writing more complex parallel code.

In the implementation domain, the well-studied architecture - Communication Computation Overlap (CCO) is applied in the programming strategy towards our development [60]. The method of overlapping communication computation has been long studied in distributed systems [11] and applied to parallel computing [98] for throughput improvement. The basic idea is to allow CPUs process to perform some independent computational tasks, while communications infrastructure performs I/O request, e.g., message passing. This technique is not new but particularly appreciable in our implementation

domain for it addresses the most fundamental issue of Web services programming – concurrency and parallelism to a large extent.

To implement CCO, many (or hybrid) programming models have been proposed. The Message Passing Interface (MPI) programming model is a good example which has been the widest recognised efficient programming model on distributed-memory architectures. The basic concept of this model is that processes perform computations on their local data and use communication primitives to share data when needed. Asynchronous (also, non-blocking) communication calls are provided in MPI. Essentially, MPI is based on the simpler asynchronous programming model applicable in many communication paradigms. For example, publish/subscribe [39] is the basic communication paradigm that has pervasively been adopted in Web services-based applications due to the loosely coupled nature of distributed systems. Subscribers register their interest in an event, and wait asynchronously until publishers generate the corresponding events.

Since the language chosen for reimplementation is F# [104], which primary is a functional language that supports multi-language paradigms targeting for .NET Framework. One of the prestigious features is the F# asynchronous programming model [105] mentioned above. In the F# ‘async’ library, the module contains primitives to perform asynchronous operations (e.g., to create, execute, and return an asynchronous computation etc). The foundation of F# asynchronous programming is the type: ‘Async<T>’ that indicates an asynchronous computation, that is, it represents a program block that will generate a value of type ‘T’ at some point in the future. ‘Async<’T>’ largely abstracts the complexity of writing continuation-passing or callback programs. With the asynchronous workflow, programmers are able to write a standard control flow code to exercise asynchronous operations without worrying about

the callbacks. For instance, the following code shows how to write an ‘async’ block (asynchronous workflow): an ‘async’ block with ‘async {...}’ is created; within the block, the code firstly prints a string then uses “do!” to perform an asynchronous operation, finally prints the last string (comments start with ‘//’ below).

```
let sleepLoop() = async {  
  
    printfn "Waiting for request.."  
  
    //mock an async operation  
  
    do! Async.Sleep 3000  
  
    printfn "Request received." }  
  
    //To run and wait for results  
  
    Async.RunSynchronously(sleepLoop())
```

To run the ‘async’ block, programmers can use either ‘Async.RunSynchronously’ (to start an asynchronous operation and await the results) or ‘Async.Start’ (to start an asynchronous operation and without await the results). The key to understanding how the workflow works in the ‘async’ block lies in this expression: ‘let! var = expr in body’, which means perform the asynchronous operation ‘expr’ and bind the result to ‘var’ when the operation completes, finally continue by executing the rest of the computation body.

The F# asynchronous workflows are literally designed to allow non-blocking execution of sequential code, but from the ‘async’ library, they also support parallel programming by using ‘Async.Parallel’ and ‘Async.StartAsChild’. In spite of the power that asynchronous workflows offer, they are not part of the

core syntax of the F# language. It actually an instance of more abstract concept called Computation Expressions in F# (or Monads in Haskell).

In F#, it is supported as a type of 'MailboxProcessor<'Msg>' that represents agents. It exists as a class type in the F# control namespace. The body of the agent is written as an asynchronous workflow, in other words, agent-based programming is based on asynchronous programming and agent is lightweight. The following code demonstrates how agent can be written in F# code: firstly this code creates a type abbreviation for 'MailboxProcessor<'T>' which is 'Agent<'T>', then uses static member 'Agent.Start' to create and start an agent, the body of the agent generates an asynchronous operation executed by the agent.

```
//type abbreviation
type Agent<'T> = MailboxProcessor<'T>

let agent = Agent.Start(fun agent -> async {

    while true do

        //agent waits asynchronously until a message arrives

        let! msg = agent.Receive()

        printfn "Hello %s" msg})

//sending a message to agent

agent.Post "hello!"
```

The following explains how code above works in .NET Framework: the body function (fun agent -> async {...}) generates an asynchronous computation executed by the agent. In our case, it repeatedly asynchronously waits for messages, and prints each message when it receives. Upon the asynchronous

computation execution, it starts life as a work item in the .NET thread pool; when the asynchronous computation reaches 'agent.Receive()', a continuations request is made and the continuations are registered as I/O completion actions (callbacks) with some object allocations held by the agent in the .NET thread pool. The thread that runs the agent is released back to the thread pool. In other words, no thread is used during the request is made. Finally, when a message arrives, i.e., a request completes; a callback is triggered in the thread pool. The continuations will carry on but possibly is run on other thread than the original one. Technically, a message processing with agent can be envisioned as a state machine that embraces an initial state and some recursive functions where each of them defines an asynchronous computation.

While the set of the F# language features is a good fit in Web services development, the functional concepts in F# enable software developers easily create a DSL. ContXFS, as a context-oriented programming approach implemented in F#, broadly speaking is a domain specific library. ContXFS will be further described in Chapter 6.

6.5 Introduction of F# and Development Tools

Context-awareness enables systems to dynamically adapt to context changes. Context-awareness techniques have been widely applied in various types of applications although many systems remain in relatively small scale computing environments. As Web services technologies establish wider application, context-aware Web services systems have been capable of exchanging context information in larger scale environments such as Cloud computing and ubiquitous computing environments, that is, it enables Web services-based systems to utilise different types of context information to adapt their services and behaviour to dynamic changes, even at runtime. Until now,

notwithstanding, methods and techniques directly addressing the development issues of Web services-based context-aware are few. In this chapter, from implementation perspective, a new programming approach – Context-Oriented Programming (COP) [56] is introduced and ContXFS, the first programming language library for F#, is developed as an approach to COP. The notion of COP was first presented in the ubiquitous computing research arena [48, 64]. COP treats context explicitly, and provides mechanisms to dynamically adapt behaviour in reaction to changes in context, even after system deployment at runtime [56].

On the other hand, context server or middleware acts as a mediator between services provider and services user. Context information in context server plays a crucial role in the development of system. Thus, modelling context information [34] is an essential research topic. Nevertheless, the said software system must adapt its services to the changing context anytime, and has to change even while it is running. This property entails a novel programming feature due to missing attributes from the mainstream programming languages along with their development environments, in other words, those languages are not competent candidates for such kind of development. This leads to the fact that software developers have been burdened with this development issue. Although some mainstream programming languages such as C# are ‘stretching’ themselves to including some relevant programming models to support this kind of dynamic change, the restriction of their programming paradigm and development environments limits their ability to extend further. This eventually would force software developers to come out with intricate designs and convoluted implementation to address various dimensions of variability.

In this chapter, benefiting from the promises that COP and F# are able to bring, ContXFS is implemented to address the implementation issues for the development of Web services-based context-aware systems.

Before introducing ContXFS, The Microsoft F# programming language and its development tools will be discussed in this section.

6.5.1 Background

Functional programming has long inspired researchers and programmers for its novelty, succinctness and expressiveness power. Yet, for some historical reasons [115], applying functional programming languages into the real world problems has not attracted much attention. However, a new generation of some strongly typed functional languages such as F#, Erlang and Scala is reaching maturity. Nowadays, a decent number of substantial applications implemented in functional languages can be easily encountered. For instance, IntelliFactory's flagship product – WebSharper Platform(TM) [118] implemented in F#, The 'Path of Go' [91] Xbox Live Arcade Game from Microsoft is also written in F#, and Yaws [126] (Yet another Web server) is a Web server written in Erlang.

With the increasing rediscovery of the essence and power of functional languages, real world industry now restarts thinking how they can leverage their legacy systems into a new environment where they can benefit from the sweet spots that functional programming languages bring. The trend is mainly driven by the net software development paradigm such as Cloud computing and context-aware Web services computing.

6.5.2 Most Appreciated Features in F#

6.5.2.1 F# and History

F# is a strongly typed functional programming language for the .NET Framework and it also supports imperative and object-oriented programming. F# was invented in 2002 as a research project in Microsoft, where ML approach was adopted to pragmatic but theoretically-based language design found a high-quality expression for the .NET platform. Influenced by Ocaml from ML family of programming languages, Haskell and C#, F# now is a first-class citizen in Microsoft Visual Studio 2010. As a result of this, F# can call and be called from other .NET languages (e.g., C# and VB) easily within .NET Framework. In a nutshell, F# is a succinct, expressive and efficient functional and object-oriented language for .NET which helps you write simple code to solve complex problems [45].

6.5.2.2 Functions and Events as First-Class Values

The basic building block in F# functional programming is function values. Functions and events can be passed as arguments to other functions and stored in data structures as return values. The following example demonstrates how to use function values to transform one list into another.

```
//first-class function using pattern matching (non-tail recursive version)
let rec map f = function
    | [] -> []
    | h :: t -> f h :: map f t
//function map takes a lambda function and list as arguments
let res = map (fun x -> x.ToString()) [1;2;3;4;5]
//function map's type signature
val map : ('a -> 'b) -> 'a list -> 'b list
```

```
//result
val it : string list = ["1"; "2"; "3"; "4"; "5"]
```

The inferred signature tells us the function ‘map’ accepts a function value *f* as the first argument and a list as the second argument, and it returns a new list of desired elements as a result. The function argument *f* can have any type ‘*a* -> ‘*b*’, and the elements of the input list must have a type ‘*a*. The notations of ‘*a* and ‘*b* are called type parameters, and the functions ‘map’ and *f* that accept type parameters are called generic.

The simple example below is using MouseMove event as a first-class value:

```
//open namespaces
open System.Windows.Forms
open System.Drawing
//first-class event composition
let form = new Form(Visible=true, TopMost=true, Text="First Class Event")
form.MouseMove
//return a new event that passes values transformed by the given lambda function
|> Event.map (fun args -> (args.X, args.Y))
//return a new event that passes values filtered by the given lambda function
|> Event.filter (fun (x, y) -> x > 150 && y > 150)
//run the given lambda function each time when event triggered
|> Event.add (fun (x, y) -> printfn "(%d, %d)" x y)
//form's signature
val form : Form = System.Windows.Forms.Form, Text: First Class Event
val it : unit = ()
```

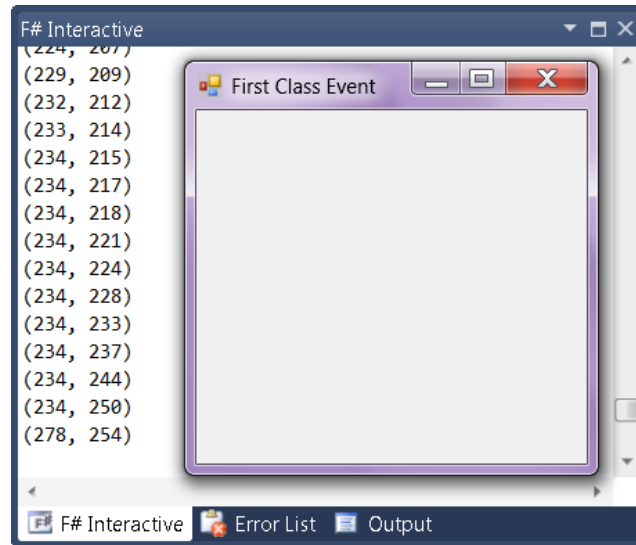


Figure 6.2 Events as First-Class Values

The above example shows that `IEvent<MouseEventHandler, MouseEventArgs>` can be passed around just as any other value. The code above used the standard combinators such as `Event.map`; `Event.filter` and `Event.add` from the `Event` module in `Microsoft.FSharp.Control` namespace.

In fact, the .NET type system when was first designed did not support generics as they would be used by F#. Thus, it uses delegates instead of function types. This leads each kind of function type given cumbersome names. Fortunately, function values to represent functions as first-class values are idiomatically used in F# as the two code samples above suggest. Yet, mainstream languages such as C# where functions are in general not first-class still allow writing higher order functions through delegates. In order to call other .NET languages' APIs that expect delegates, F# enables us to define `Delegate` type and create a delegate that represents a function call as an object where .NET Common Language Runtime (CLR) looks after the transmission. In fact, every .NET delegate type has a corresponding F# function type. For example, the F# function type for the .NET delegate type `System.EventHandler<T>` is `obj -> 'T`

-> *unit*. The following code demonstrates how to define and use a delegate type.

```
//define a delegate type (int -> int option)

type Delegate = delegate of int -> int option

//attach the delegate to static method - AppDele

type ListAssociations =

    static member AppDele (l: int List, d: Delegate) =

        l |> List.tryPick(fun i -> d.Invoke i)

//static method expecting a compatible delegate type

//consumed by a lambda expression as an argument

ListAssociations.AppDele ([1;2;3;4;5;6], (fun i -> if i % 2 = 0 then Some i else None))

//type signatures

type Delegate = delegate of int -> int option

type ListAssociations =

    class

        static member AppDele : l:List<int> * d:Delegate -> int option

    end

//return an option as result

val it : int option = Some 2
```

Delegate types are used in special contexts such as interoperating with other .NET languages, but they have limitations, for example, they do not support compositional operations such as pipelining and forward composition.

6.5.2.3 Computation Expressions (Workflows)

F# accommodates a succinct and compact syntax called Sequence Expressions that specify sequence values. Aggregate operations such as map, filter, and concat can be used to transfer these values. They are also applicable in lists and arrays. A simple example of sequence expressions is depicted below:

```
//a simple sequence expression
seq { for i in 0 .. 3 -> (i, i+i) }
//result
val it : seq<int * int> = seq [(0, 0); (1, 2); (2, 4); (3, 6)]
```

The form of this construct is ‘seq { for pattern in seq -> expression }’. The input seq can be a seq<type> or any type supporting a GetEnumerator method (i.e., flexible type #seq<type>).

In effect, sequence expressions are just one special instance of a more general construct called Computation Expressions (also Workflows). Computation Expressions are the F# equivalent of monadic syntax in the programming language Haskell. Monads are a powerful and expressive design pattern and are characterized by a generic type $M<'T>$ combined with at least two operations:

bind : $M<'T> \rightarrow ('T \rightarrow M<'U>) \rightarrow M<'U>$

return : $'T \rightarrow M<'T>$

These operations: bind and return correspond to the primitives let! and return in the F# computation expression syntax. They are the fundamental primitives that can be used to implement other more ad-hoc operations. In fact, the syntax of computation expressions allows us to construct sequences and other non-standard computations. The general form of a computation expression is ‘builder-expr { comp-expr }’ which translates into [46]:


```
let b = builder-expr in b.Run (b.Delay(fun () -> {/ cexpr /}c))
```

For a fresh variable *b*, the type of *b* must be a named type after the checking of ‘builder-expr’. If no method ‘Run’ exists on the inferred type of *b* when this expression is checked then that call is omitted. Likewise if no method ‘Delay’ exists on the type of *b* when this expression is checked then that call is omitted. This expression is then checked. This translation implicitly places type constraints on the expected form of the builder methods. Some main constructs in computation expressions and their corresponding de-sugaring are available in [104].

Three most important applications of computation expressions in F# programming are as follows [104]:

- General-purpose programming with sequences, lists, and arrays
- Parallel, asynchronous, and concurrent programming using asynchronous workflows
- Database queries, by quoting a workflow and translating it to SQL via the .NET LINQ libraries

The example below is an implementation of a workflows builder called ‘SumOfSquaresMonoid’:

```
// Define SumOfSquaresMonoid type  
type SumOfSquaresMonoid() =  
    // Combine two values  
    // sm.Combine («cexpr1», b.Delay(fun () -> «cexpr2»))  
member sm.Combine(a,b) = a + b
```

```
// Zero value
// sm.Zero()
member sm.Zero() = 0
// Return a value
// sm.Yield expr
member sm.Yield(a) = a
// Delay a computation
// sm.Delay (fun () -> «cexpr»)
member sm.Delay(f) = f()
// For loop
// sm.For (expr, (fun pat -> «cexpr»))
member sm.For(e, f) =
  Seq.fold(fun s x -> sm.Combine(s, f x)) (sm.Zero()) e

// Create one global instance of each such monoid object
let sosm = new SumOfSquaresMonoid()
// Build a SumOfSquaresMonoid value(function)
let sumOfSquares x = sosm { for x in 1 .. x do yield x * x }
// Evaluation
sumOfSquares 5

// The signature of SumOfSquaresMonoid
type SumOfSquaresMonoid =
  class
    new : unit -> SumOfSquaresMonoid
    member Combine : a:int * b:int -> int
    member Delay : f:(unit -> 'b) -> 'b
    member For : e:seq<'a> * f:('a -> int) -> int
    member Yield : a:'c -> 'c
    member Zero : unit -> int
  end
val sosm : SumOfSquaresMonoid
// Signature of SumOfSquaresMonoid function value
```

```
val sumOfSquares : int -> int
// Evaluated result
val it : int = 55
```

From the above example, computation expressions can be used for customising the meaning of a block of code by encapsulating the most complicated logic which might be difficult to construct directly. And the composable nature of computation expressions offers more flexibilities of implementation.

6.5.2.4 F# Asynchronous Workflows

One of the most powerful applications of F# is Asynchronous Workflows, a powerful set of techniques for structuring asynchronous programs in a normal control flow way, i.e., using ‘if’, ‘for’, and ‘while’ and so on. The computation represented by expression runs asynchronously, that is, when asynchronous operations are performed, it will not block the current computation thread. Asynchronous computations are often started on a background thread while execution continues on the current thread. The type of the expression is ‘Async<a>’, where ‘a’ is the type returned by the expression when the return keyword is used. The code in such an expression is referred to as an asynchronous block, or async block.

From object-oriented programming paradigm perspective, ‘Async’ class provides a few methods that support asynchronous programming. The general approach is to create ‘Async’ objects that represent the asynchronous computation(s), and then start these computations by using one of the triggering functions depending on which thread you want to use, whether is a .NET Framework task object or whether to run continuation functions after computation completes.

One of the highly asynchronously examples is an implementation of a ‘fetchWebPageAsync’ function that fetches the html text asynchronously and executes multiple asynchronous operations in parallel.

```
//open namespaces
open System
open System.Net

//define an async funtion that fetches web page contents
let fetchWebPageAsync(name: string, url: string) =
    async {
        let uri = new Uri(url)
        let webClient = new WebClient()
        let! html = webClient.AsyncDownloadString(uri)
        printfn "%s has %d characters" name html.Length
    }

//web page repository, list of tuples (name, url)
let urlList = [ "Google Search", "http://www.google.com"
                "BBC News"      , "http://news.bbc.co.uk"
                "De Montfort U", "http://www.dmu.ac.uk"
                ]

//run the given list of urls asynchronously
let runAllAsync() =
    urlList
    |> Seq.map fetchWebPageAsync
    |> Async.Parallel
    |> Async.RunSynchronously
    |> ignore

//evaluated the results
runAllAsync ()

//the results
De Montfort U has 14992 characters
Google Search has 10782 characters
```

BBC News has 86709 characters

```
val it : unit = ()
```

Within the asynchronous workflow expressions, the language construct ‘let! var = expr’ in body means “perform the asynchronous operation expr and bind the result to var when the operation completes. Then, continue by executing the rest of the computation body [104].”

The following describes what ‘fetchWebPageAsync’ does:

- It gets the instance of Uri with specified uri synchronously.
- It creates the instance of ‘WebClient’ synchronously.
- It downloads the html text asynchronously by calling ‘AsyncDownloadString(uri)’ function after the synchronous Web requests complete.
- After the download completes, it prints the symbols (names) of the Web page and the total number of characters have been downloaded synchronously. Then, a list (i.e., urlLst) of url as the input Web page repository was defined.

Finally, a ‘runAllAsync()’ function was called by composing a series of pipeline operations:

- Firstly, a map function from module ‘Seq’ which maps the given ‘urlList’ of input into the ‘fetchWebPageAsync’ function, sure enough, it returns a sequence of three asynchronous operations (i.e., *seq<Async<unit>>*),
- Secondly, ‘Async.Parallel’ function takes the sequence of the Async objects (i.e., *Async<unit>*) and sets up the code for each Async task object

to run in parallel, and returns an Async object (i.e., *Async<unit []>*) that represents the parallel computation.

- Thirdly, ‘Async.RunSynchronously’ was called to execute an asynchronous operation and wait for its result.
- Finally, used ignore function to throw away the result of the whole computation.

Typically, *Async<'T>* values are essentially a way of writing continuation-passing or callback programs explicitly. *Async<'T>* computations call a success continuation when the asynchronous computation completes and an exception continuation if it fails. They provide a form of managed asynchronous computation, where managed means that several aspects of asynchronous programming are handled automatically [104]:

- Exception propagation is added for free.
- Cancellation checking is added for free.
- Resource lifetime management is fairly simple.

To unveil the techniques used to implement asynchronous computations, consider the following simple async block:

```
async {  
    let uri = new Uri("http://ieeexplore.ieee.org")  
    let webClient = new WebClient()  
    let! html = webClient.AsyncDownloadString(uri)  
    html  
}
```

The above is essentially shorthand for the following code:

```
async.Delay(fun () ->

    let uri = new Uri("http://ieeexplore.ieee.org")

    let webClient = new WebClient()

    async.Bind(webClient.AsyncDownloadString(uri), (fun html ->

        async.Return html))))
```

It is important to note that asynchronous programming library is not built directly into the F# language. Rather, it is implemented by using computation expressions discussed previously as a general purpose feature for writing ‘non-standard’ *computations*.

To wrap up, the values of type ‘Async<T>’ are effectively identical to the following type:

```
type Async<T> = Async of ('T -> unit) * (exn -> unit) -> unit
```

Where, the functions are the success continuation ('T -> unit) and exception continuations (exn -> unit), respectively. Each value of type Async<T> should eventually call one of these two continuations. The async object is of type ‘AsyncBuilder’ and supports the following methods, among others.

The async object is of type ‘AsyncBuilder’ and supports the following methods, among others:

```
type AsyncBuilder with

    member Return : 'T -> Async<'T>

    member Delay : (unit -> Async<'T>) -> Async<'T>

    member Using: 'T * ('T -> Async<'U>) -> Async<'U> when 'T :> System.IDisposable
```

```
member Bind: Async<'T> * ('T -> Async<'U>) -> Async<'U>
```

6.5.2.5 Type Inference

F# is a statically typed and strongly typed language. For statically typed, the type of every value and expression are checked during compile-time before any code is executed, that is, many type errors can be caught early in the development cycle. Although F# is static typed language, the types of values rarely need to be specified explicitly thanks to type inference. The F# compiler analyses the code to collect constraints by assigning types to identifiers as they are defined. The assigned types are based on the type information the compiler already knows. It works through the program from top to bottom, left to right, and outside in. The code below show the result of type inference:

```
//records with type variables

type Car<'a,'b> =

    {Maker: 'a

    Year: 'b

    }

// instantiate the record type

let polo = {Maker = "VW"; Year = 2003}

//the type Car's signature

type Car<'a,'b> =

    {Make: 'a;

    Year: 'b;}

//types are inferred automatically, i.e., Car<string, int>

val polo : Car<string,int> = {Make = "VW";
```



```
Year = 2003;}
```

F# is also strongly typed which leads to type safe code. The type system guarantees that a program cannot contain certain kinds of errors (e.g., you cannot use a function with a value that is inappropriate).

The code below attempts to use function ‘add’ to take two integers as arguments, while the function add’s signature is ‘string -> string -> string’.

```
//constraint argument a to string type leads to string -> string -> string  
  
let add (a: string) b = a + b  
  
//try to add two integers, but the compiler tells us an error in compile-time  
  
//this expression was expected to have type string but here has type int  
  
add 1 2  
  
val add : string -> string -> string  
  
ThesisCode.fs(282,5): error FS0001: This expression was expected to have type  
  
    string  
  
but here has type  
  
    int
```

Although in occasional cases, type annotation is required for clarifying ambiguity of types, type inference and statically typed enabling types of values are automatically inferred during compile time dramatically reduces code clutter and source code size. With strongly typed feature, F# tends to be a language which is safer than many popular statically typed languages (e.g., C, C# and Java) and often more expressive than dynamically typed languages (e.g., Python).

6.5.2.6 Immutability by Default

Immutable data structures are sometimes called persistent or simply functional. Values and data structures in F# programming are completely immutable by default such as tuple values, option values, records, lists, sets, and maps. Immutability offers many advantages: On one hand, code using immutable basic types is often relatively easy to reason about, this eases of the maintenance cost. On the other hand, immutability allows you pass immutable values between multiple threads without worrying about unsafe concurrent access to the values, which in turn makes parallelisation a lot easier.

```
//immutability
type Person =
    {Name: string; Age: int}

//create a new value of type Person
let john = {Name = "John"; Age = 31}

printfn "%s is %d years old." john.Name john.Age

John is 31 years old.

val it : unit = ()
```

Any attempt to modify John's age causes an error:

```
John.Age <- 30
```

```
ThesisCode.fs(51,5): error FS0005: This field is not mutable
```

F# is not purely functional language as mentioned above, so mutability can be applied to values by using the keyword – ‘mutable’. The code below is to make record type Person's age field mutable.

```
type Person =
```

```
//make the field Age mutable

{Name: string; mutable Age: int}

John.Age <- 30

John is 30 years old.

val it : unit = ()
```

Some restrictions related to mutable values are applied.

```
//mutable variables cannot be captured by closures

let generateNumbers() =

    let mutable n = 0

    let incrN() = n <- n + 1

    incrN()
```

Error may be raised for capturing mutable variables by closures:

“ThesisCode.fs(291,23): error FS0407: The `mutable` variable '`x`' is used `in` an invalid way. Mutable variables cannot be captured by closures. Consider eliminating this `use of` mutation `or` using a heap-allocated `mutable` reference cell via '`ref`' and '`!`'.”

As suggested, using a ref cell to store the mutable data on the heap solves the compile error.

```
let generateNumbers =

    //use ref cell to store the mutable data on the heap

    let number = ref 0

    (fun () -> incr number; !number)

generateNumbers()

val generateNumbers : (unit -> int)

val it : int = 1
```

```
val it : int = 2
```

```
...
```

Immutable values are common in functional languages and offer many advantages. For example, by knowing values are immutable, you can pass such values to routines as they are immutable. In concurrent context, passing immutable values among multiple threads will be safe. Even in object-oriented languages such as Java, classes by default should be immutable unless there is a very good reason to make them mutable [17].

6.5.2.7 Interoperability between Other .NET Languages

Although there are many powerful techniques available inside F#, the true value of F# also expands to the connection of the outside world. F# is compiled on .NET Framework and connected to many of the significant programming techniques available on major computing platforms. Therefore, .NET libraries are available in F# (e.g., dot notation (.) and assignment notation (<-) are available) and in turn, you can use F# libraries in any .NET languages (e.g., F# libraries are fully accessible from C# though occasionally small adjustments require in the light of F# Component Design Guidelines [45]).

Using .NET libraries from F#: The code below is use Windows Presentation Foundation (WPF) in F# as depicted in Figure 6.2:

```
open System
open System.Windows
open System.Windows.Controls
//create an application
let app = Application()
```

```
//create a window

let win = Window()

//set width and height

win.Width <- 100.0

win.Height <- 120.0

//create a stackpanel which contains a textblock and button

let sp = StackPanel()

//create a textBlock

let txt = TextBlock(Text = "Hello World")

sp.Children.Add txt |> ignore

//create a button

let bt = Button(Content = "Click me")

//add a callback

bt.Click.Add(fun _ ->

    txt.Text <- "Clicked!")

sp.Children.Add bt |> ignore

//assign the stackpanel to window's content

win.Content <- sp

//Run the application

[<STAThreadAttribute>]

do app.Run(win) |> ignore
```

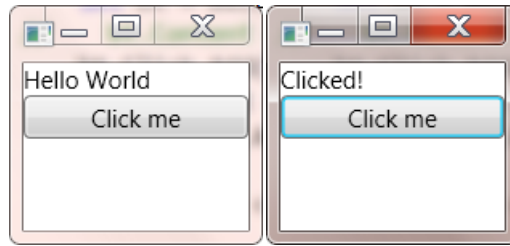


Figure 6.3 WPF in F#

F# function in module 'PhDThesisSampleCode'. Fibonacci

```
namespace PhDThesisSampleCode
```

```
module Fibonacci =
```

```
    let rec fib n =
```

```
        if n <= 2 then 1
```

```
        else fib (n-1) + fib (n-2)
```

Using F# function in C# application:

```
using System;
```

```
namespace PhDThesisCode_InterOp
```

```
{
```

```
    class CSharpInterOp
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            //call F# PerBalTree library
```

```
            Console.WriteLine(PhDThesisSampleCode.Fibonacci.fib(10));
```

```
        }
```

```
    }
```

6.5.2.8 Conclusion

There are many other F# features which are not covered in this chapter, e.g., units of measure, tuples, discriminated union data types, pattern matching, active patterns, composition functions, partially applied functions, collections modules, language oriented programming and so on.

Functional programming languages typically appear to specify ‘what to do’ rather than ‘how to do’. F# is a hybrid programming language that enables developers to choose appropriate programming paradigms for better implementation. In other words, F# allows for encapsulation via object-oriented approaches as well as much more concise code via modern functional approaches. For instance, parallel programming becomes easier because of immutability; delegating members to other underlying objects within a class; uniformly abstracting the complexity of asynchronous operations etc.

To wrap up, F# enables developers to solve complex problems (e.g., Cloud computing, parallel and asynchronous computing) in a more declarative way comparing to other object-oriented languages. It nicely embraces most of the best features from the main programming paradigms, which makes it a better candidate for services reimplementation.

6.6 Context-Oriented Programming and Main Features

Functional programming languages have enjoyed a long-time connection with implementation of domain specific languages since functional features or properties are more suitable for creating parsers and compilers. This relation motivates the implementation of ContXFS.

6.6.1 Context-Oriented Programming

Context-Oriented Programming (COP) is a new programming approach that provides a means of enabling software entities to adapt their behaviour dynamically to the current execution context [56]. In other words, COP allows for the expression of behavioural variation depending on context at run time. While it is largely independent of commitments to programming style, many COP extensions actually have been implemented within object-oriented programming paradigm [6].

COP can be seen as an alternative approach to addressing issues of context-awareness rather than a relatively traditional approach on software architecture level. Although context-awareness in ubiquitous computing environments, software evolution, and execution context dependencies can be considered as the related application domains of COP, in effect, COP are able to address some implementation issues for the development of Web services-based context-aware applications. By adopting a right programming paradigm, distribution, concurrency, and parallelism implementation issues can be better addressed.

6.6.2 COP Main Features

The essential language properties to support COP can be summarised as follows [56]:

- A means to specify **behavioural variations**,
- A means to group variations into **layers**,
- Dynamic **activation** and **deactivation** of layers based on **context**, and
- A means to explicitly and dynamically control the **scope** of layers.

Therefore, approaches to COP should at least address the following properties [56]:

- **Behavioural variations:** Variations typically consist of new or modified behaviour, but may also comprise removed behaviour. They can be expressed as partial definitions of modules in the underlying programming model such as procedures or classes, with complete definitions representing just a special case.
- **Layers:** Layers group related context-dependent behavioural variations. Layers are first-class entities, so that they can be explicitly referred to in the underlying programming model.
- **Activation:** Layers aggregating context-dependent behavioural variations can be activated and deactivated dynamically at runtime. Code can decide to enable or disable layers of aggregate behavioural variations based on the current context.
- **Context:** Any information which is computationally accessible may form part of the context upon which behavioural variations depend.
- **Scoping:** The scope within which layers are activated or deactivated can be controlled explicitly. The same variations may be simultaneously active or not within different scopes of the same running application.

In a nutshell, as an extension to object-oriented programming, COP accommodates means for concise specification as well as dynamic activations and composition of behavioral variations [6]. In the next section, nevertheless, context-oriented programming in F# (ContXFS) will be discussed. Although F# is a multi-paradigm programming language as discussed in previous sections, ContXFS is a COP library implemented within an F# functional model.

6.7 Context-Oriented Programming in F#

6.7.1 Overview of ContXFS Development

Modern development of Web services-based context-aware systems demands dynamic adaptation and context-awareness that poses a great challenge not only to architecture design, but also programming language support. COP addresses the need for applications to behave differently accordingly to the changing run-time context in which they are embedded. This goal is achieved by providing the abstractions that enable application context-awareness without hard-wired conditional statements that spread over the application code, which exempts a need to scatter context dependent behaviours throughout a program.

COP accommodates dynamic activation and composition of behavioural variations. Generally, behavioural variations are grouped in layers and adaption is obtained through layer activation. Asynchronous message passing and processing is a common foundation for concurrent programming in some functional programming languages, e.g., Erlang.

In general, COP can be seen as a language extension for object-oriented programming paradigm. A considerable numbers of COP implementations can be found in [30]. For example, ContextJ [7] is not merely an extension to Java programming language, but also a new compiler and possibly an extension of Java Virtual Machine. It is a compiler-based COP implementation for Java that introduces COP's layer concept into the Java type system. In addition to the implementation by statically typed programming languages, ContextErlang [51] is one of COP that are implemented in dynamically typed programming languages. Benefiting from Erlang type system, it is claimed that applying COP in Erlang known as a language that natively supports distribution and

concurrency can obtain effectiveness of the approach to support dynamic context-aware adaptation.

Inspired by the implementations of ContextJ and ContextEralng, ContXFS is primarily implemented in a functional model while adopting appropriate object-oriented programming techniques for code encapsulation and constructs. In fact, this implementation itself demonstrates one of the most major advantages of choosing F# for COP implementation.

In ContXFS, the notion of ‘context’ can be referred to a complete set of behavioural variations that are dynamically bound to the given application. Because variation activations are implemented when context-enabled module reacts to layer activations. In the following sections, the language constructs, semantics, and implementation of ContXFS will be discussed.

6.7.2 Behavioural Variations in ContXFS

The ability of enabling behavioural variations is one of the core properties of COP. In practice, the means to introducing context dependent behaviour into a program can be obtained via excessively inserting conditional statements, e.g., if statements throughout the program. Alternatively, behaviour variations can be achieved by scattering context dependent behaviour into different objects that can be replaced subject to the changing context. Both low level approaches will lead to high maintenance cost during software evolution in the future. ContXFS is COP-inspired conceptual F# implementation for Web services-based context-aware systems. As F# asynchronous message passing is suitable for no shared memory concurrent systems, F# agent-based programming model is adopted to support agent paradigm.

In practice, message handling, error handling, and fault tolerance are context independent. In other words, an agent must handle every message when they

arrive. Moreover, such message processing shares relatively fixed patterns. Nevertheless, COP entails that the target system is able to change its behaviour at run time. Thus, a component should contain a generic message control which deals with the incoming messages, and upon the arrivals of the messages, a user context-enabled process invokes activated set of functions to implement behavioral variation at execution time. Therefore, a typical component in a ContXFS application contains two conceptual modules, i.e., a generic control module that provides functionalities for message passing, error handling, and fault-tolerance, as well as a server action module that implements specific functionalities the server is to perform at run time upon a context change request. In fact, for a small ContXFS application, both conceptual modules can be included in a single F# module.

In order to expound this approach, a simple example is described here. In a Cloud computing environment, a mobile app can access to a public Cloud and its private Cloud. It is assumed that there are two mobiles apps from different private Clouds where private Cloud₁ is accessible for App₁ and private Cloud₂ is accessible for App₂. For some reasons, App₂ attempts to access private Cloud₁ although it will fail by trying directly. Instead, App₂ can use App₁ as an intermediary for specific services. It can be done under a security agreement. However, such kinds of messages are always ignored as the situation is not always possible. Thus, App₂ will behave differently according to the incoming message and external context, i.e., upon a request message from App₁, App₂ can either accept the message and process the services delivery for App₁ or reject the request by simply dropping all the messages.

In ContXFS, typically, a type extension is implemented for the type `'MailboxProcessor<'Msg>'` with a static member `'SpawnAgent'` which takes two parameters, i.e., message handler and initial state, and optional parameters, e.g., timeout handler and error handler.

```
type MailboxProcessor<T> with
    static member SpawnAgent<'State>(messageHandler: 'T -> 'State -> 'State,
                                     initialState: 'State,
                                     ?timeout: 'State -> int,
                                     ?timeoutHandler: 'State -> AfterError<'State>,
                                     ?errorHandler: exn -> 'T option -> 'State -> AfterError<'State>
                                     ) : MailboxProcessor<T> = ...
```

The message can be classified as user message and control message where user message holds the value of the message while the union – ‘SetAgentHandler’ of discriminated union type – ‘ControlMessage’ holds the state and value of the message. The two types of messages can be easily extended by adding more unions to the union type. The following code describes the concept:

```
type internal ControlMessage<'T, 'State> =
    | Continue
    | Stop
    | Restart
    | GetState of 'State
    | SetState of AsyncReplyChannel<'State>
    | SetAgentHandler of ('T -> 'State -> 'State)
    ...
```

```
type internal Message<'T, 'State> =
    | UserMsg of 'T
    | ControlMsg of ControlMessage<'T, 'State>
    ...
```

```
type AfterError<'State> =
    | ContinueProcessing of 'State
    | StopProcessing
    | RestartProcessing
```

To summarise, server action module enables behaviour variations. A variation contains all the function declarations, these functions are invoked when the variation is activated, i.e., variation activation is bound to the application dynamically (at run time).

6.7.3 Context Switching On-The-Fly

Switching context can be obtained via a ‘SetAgentHandler’ message which is a union of a generic control message represented as discriminated union type. The following code is to demonstrate how to implement context switching on-the-fly.

```
let counterAgent = MailboxProcessor.SpawnAgent((fun msg state -
> printfn "TupleBefore = %A" (msg, state); msg+state), 0)
counterAgent.Post(1)

val it : unit = ()
> TupleBefore = (1, 0)

counterAgent.Post(SetAgentHandler(fun msg state -
> printfn "TupleAfter %A" (state, msg); msg+state))
counterAgent.Post(2)

val it : unit = ()
> TupleAfter = (1, 2)
```

When the counter agent is created via static member ‘SpawnAgent’, it posts a message of integer 1 to a mailbox queue. When counter agent receives a ‘SetAgentHandler’ message, it dynamically switches from a ‘TupleBefore(msg, state)’ agent to a ‘TupleAfter(state, msg)’ agent. In other words, it allows program code to be updated in a running system without turning it down. The

messages that arrive after that switch will perform multiplication of the message and the state. One of the advantages of such switch is that the state is preserved while behavioural variations. Such feature is desirable as Web services-based context-aware systems always need to be at ever running states.

6.7.4 Layers in ContXFS

In object-oriented programming paradigm, layers can be implemented as named first-class entities that can be referred to explicitly at runtime [56]. Behavioral variations can be grouped in layers. The adaptation to a context is achieved by layer activation. In other words, ad-hoc code constructs guarantee that the partial definitions inside a layer are activated at run time and therefore can change the behaviour of the program accordingly.

Two layer declaration strategies have been implemented so far in literature, i.e., ‘layer-in-class’ and ‘class-in-layer’. Each strategy has its own advantages over the other. For example, the advantages of ‘layer-in-class’ may include that layers can be well encapsulated through private fields in class and specific layers can be easily added to a new class. On the contrary, the main advantage of ‘class-in-layer’ may contain that a specific variation being implemented in a single module can largely improve the adaptability in an evolving application.

In ContXFS, layers are built on top of variations. In other words, layers can be referred to a set of variations. Depending on if a layer is activated, the activation is able to affect all the relevant components in an application. Specifically, layer activation is a synchronous operation, i.e., once a layer is activated, the related code block is executed immediately.

6.7.5 Layers Composition in ContXFS

Inspired by the implementation of ContextErlang, variations can be arranged in a layer. Variations are activated in a special order; they are kept in a stack conceptually. Layers refer to a set of variations that are activated. Figure 6.4 simply shows an example of the process of activation of multiple variations.

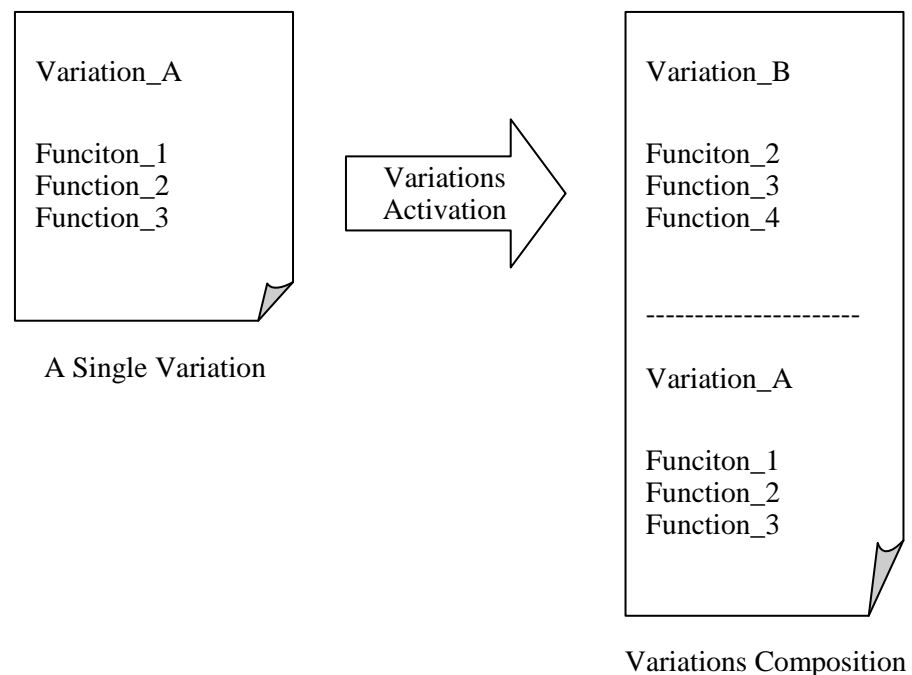


Figure 6.4 Variations Composition of Variation_A and Variation_B

In Figure 6.4, Variation_A contains three functions: function_1, function_2, and function_3. While Variation B contains other three functions: function_2, function_3, function_4. When Variation_A is activated only, all three functions, i.e., function_1, function_2, and function_3 are directly invoked. However, once context switching occurs, for example, Variation_B is activated. Variation_B will be added on top of Variation_A. In this case, when a message

is received to call function_1, it will behave as same as before Variation_B is activated. Nevertheless, when a message is arrived to call function_2 or function_3, those functions implemented in Variation_A will be overridden by the same name function_2 and function_3 in Variation_B, that is, the function_1 will fail, and the Variation_B version of function_2 and function_3 will be executed. Apparently, a call to function_4 occurs, then it will be executed as it is implemented in Variation_B which is current activated.

6.8 An Example of Reimplementation

The purpose of this case study is twofold: to validate if the CCO strategy is implemented and if the refined requirements fulfilled by the implementation using the chosen programming language models. On the programming language choice, the language F# is selected for our implementation benefiting from the refined requirements in Table 6.1 and programming models explanation in Section 6.6 and Section 6.7.

To walk through all the refined requirements generated in Table 6.1 is exhaustive. Thus, a few of them will be covered, i.e., discriminated unions; pattern matching; first-class events; asynchronous programming model; asynchronous agent-based programming model. Based on the architecture design proposed in Figure 6.1, an 'HTTPServiceAgent' type is defined to listen for incoming HTTP requests and handle them using the 'async' body. Given the specific url, the agent server starts and asynchronously waits for the HTTP requests.

```
type Agent<'T> = MailboxProcessor<'T>
```

```
//define an HTTPServiceAgent that listens for HTTP requests and handles them
```

```
type HTTPServiceAgent (url, f) as this =  
  
    let tokenSource = new CancellationTokencSource()  
  
    //(f this) used as an asynchronous workflow  
  
    let agent = Agent.Start(...)  
  
    let server = async {  
  
        use lis = new HttpListener()  
  
        ...  
  
        while true do  
  
            //create an asynchronously computation, when it is run, perform BeginAction, while the  
            //callback has been registered, when the callback is invoked, the EndAction to get the overall  
            results  
  
            let! context = Async.FromBeginEnd(lis. BeginGetContext, lis.EndGetContext())  
  
            agent.Post(context) }  
  
        //performs actions when the object is constructed  
  
        do Async.Start(server, cancellationToken = tokenSource.Token)
```

The type of context can be expressed using a discriminated type, and pattern matching gives it a concise way to match against the proper behaviours. The type ‘AsyncReplyChannel’ is to post a response to reply channel and continue.

```
type internal Message =  
  
    | GetContent of AsyncReplyChannel<string>  
  
    | SendMessage of string
```

First-class events facilitate the way of raising an event back to GUI thread [106]:

```
type SynchronizationContext with

    //A standard helper extension method to raise an event on the GUI thread

    member syncContext.RaiseEvent (event: Event<_>) args =

        syncContext.Post((fun _ > event.Trigger args),state=null)
```

Although our project is still ongoing, the intermediate results give us a very promising feedback that the basic concept of CCO can be fairly straightforward implemented in F# and the refined requirements constructed guide us to choose the right language candidate.

6.9 Summary

In this chapter, context-aware Web services reimplementation is described. Requirements for such services reimplementation are concluded by comparing to mainstream object-oriented programming languages. This chapter also points out the reimplementation concerns in the solution domain. The reimplementation strategies vary because of domain specific issues. On the other hands, this chapter briefly introduces programming language F#, and the concept of context-oriented programming (COP), and the development of ContXFS. ContXFS is an F# library for COP.

- ❑ Non-functional requirements for context-aware Web services reimplementation are more difficult to fulfil than the functional

requirements. Requirements mapping is used to associate the desired programming characteristics with the language features. A sample of functional and non-functional requirements is depicted in Table 6.1.

- ❑ Web services reimplementation concerns the following issues: performance issues (e.g., scalability and reliability etc), state sharing issues (e.g., avoiding race conditions), and long-running operation issues (e.g., asynchronous agent-based programming and performing parallel computing).
- ❑ A conclusion of F# general and specific features and its advantages is presented based on a brief comparison between object-oriented and functional language paradigm.
- ❑ The reimplementation strategies depend on the legacy system's requirements, e.g., overlapping communication and computation strategy and domains special language support.
- ❑ A small example of a HTTP service agent is showed. This server utilises asynchronous agent-based programming and other F# related features such as discriminated union and pattern matching.
- ❑ The work in this chapter describes the methods and steps of re-implementing the said systems. Detailed approaches are depicted in Section 6.2.2, 6.3 and 6.4 which contribute to the major quantitative methods of this thesis.
- ❑ F# is a succinct, expressive and efficient functional and object-oriented language targeting .NET Framework. It aims to adopt the 'best' programming concepts and attributes from functional and object-oriented programming paradigms.

- ❑ Typically, COP offer mechanisms to associating partial class and method definitions with layers as well as activate and deactivate the layers explicitly at run time.
- ❑ Server action module enables behaviour variations. A variation that activated contains all the relevant functions to be executed, that is, variation activation is bound to the application at run time.
- ❑ F# enables itself to specify behavioural variations by object-oriented programming encapsulation, union types, and pattern matching; group variations into layers by first-class functions with built-in data structures.
- ❑ ContXFS is a COP-inspired conceptual library in F# with an aim to facilitate the development of Web services-based context-aware systems with no shared memory feature.

Chapter 7 – Case Study

Objectives

- To demonstrate the way of applying the overall proposed reengineering approach to reconciling requirements and implementation gap for different types of legacy systems
- To illustrate the development toolkit for the propose approach

7.1 Overview

Software reengineering approach is a practical solution for the problems of evolving legacy systems. Reverse engineering and forward engineering are the two key methods that enable software evolution. As appropriate methods and techniques for the development of Web services-based context-aware systems are not yet mature, validation work in reengineering for such systems is difficult. Therefore, four case studies have been selected carefully and combined as the validation method.

In order to simplify the overall claim of this thesis, three further detailed claims are classified as follows:

- The first claim: the overall approach should be able to facilitate the services candidate discovery tasks. This claim is associated with the effectiveness criteria. For instance, the RRF approach can recover the code-related artifacts along with the requirements-related artifacts.
- The second claim: the overall approach should be able to manage the evolved requirements. This claim is associated with the generic usability criteria. For example, when the current requirements from source code level extracted from CASRM, reconstructing new context-aware services requirements is straightway and ease.
- The third claim: the overall approach should be able to address the redevelopment issues for the said systems. This claim is associated with the efficiency criteria. For instance, COP and appropriate programming languages can greatly mitigate the development burdens.

To validate the above claims, the proposed approach is applied to four further case studies. These case studies are chosen to investigate specific research questions and focus on corresponding claims discussed above, while some case studies may validate the same claims as others. Table 7.1 gives an overview of attributes of each case study. The details of the four case studies can be described as below:

- The first case study is performed on an open source platform for integrating mobile applications with Cloud services. This case study helps to illustrate detailed process of RRF approach. The claim of effectiveness can be evaluated through this case study which accommodates guidance for readers to adopt derived approaches in their own practice.
- The second case study is a location-aware based application that enables a Web application to obtain a user's geographical position. This case study

focuses on the claim of usability. CASRM and the requirements evolution model will be used to carry out this case study.

- The third case study is a framework aims to enable integration of location-awareness techniques in Linux platform applications. The claim of efficiency and effectiveness will be evaluated through this case study.
- The fourth case study is a chat prototype implemented with a context-oriented programming approach. The claims of effectiveness, efficiency and usability are evaluated by this case study. Context-oriented programming methods implemented in F# and the natively supported programming language features will be applied in the redevelopment of this case study.

The properties of the four case studies above are represented in the following Table 7.1. The table summarises the claim of each study that focuses on in the proposed approach, as well as the basic attributes of each study. The first column describes the name of the subject case study. The second columns depict the programming language(s) used to implement the original case study. The rest of columns represent the three core claims of this thesis.

Case Study	Language(s)	Usability	Effectiveness	Efficiency
Openmobster	Java	×	×	
Geolocation API	JavaScript	×		×
Geoclue	C		×	×
ContextChat	Erlang/Java	×	×	×

Table 7.1 Attributes of Each Case Study

7.2 Openmobster

Openmobster [88] is an open source platform for integrating mobile applications with Cloud services. It aims to facilitate the development of mobile applications that resides in the Cloud via an infrastructure. It has the following features:

- **Sync Platform:** Automatic bidirectional data sync between the devices and the Cloud.
- **Push Notifications:** A platform-agnostic Cloud-initiated push notification system.
- **Location Aware Applications:** A framework for creating end-to-end location aware applications.
- **Mobile RPC (Remote Procedure Call):** A simple name/value pair based method for invoking service components in the Cloud.
- **Management Console:** A GWT/SmartGWT based application to administrate the system.

7.2.1 Overview of Requirements Recovery Framework (RRF) Approach

The RRF contains Services Patterns Module (SPM), Concept Generator, and Event Concept. SPM contains Knowledge-Based Library (KBL), Source Code Information (SCI), and Requirements (REQ). SPM underpins the requirements elicitation and an initial services pattern module is created by domain experts and software engineers as a prerequisite. Concept Generator uses Hypothesis-Based Concept Assignment (HB-CA) method, which consists of further three stages: Hypothesis Generation, Segmentation and Concept Binding. A list of concepts associated with regions of source code will be generated. Event

Concepts is the phase that domain experts and software engineers fulfil the enhancement to further enhance the content of services pattern module where concepts are linked with associated events. KBL is a library that maintains lists of intermittently enhanced tuples: <Concept, Event>. SCI is composed of information directly reflected from the source code including identifiers, comments, and keywords. REQ comprises functional requirements and non-functional requirements.

7.2.2 RRF Approach on Openmobster

Location information can be accommodated by using the functions from Location Module of OpenMobster platform. In OpenMobster, the business components are encapsulated with this location information. The components then have easy access to the location data and can easily integrate it with the business data.

Based on our proposed CASSR approach, once legacy Openmobster passes the assessment, an initial SPM is created by domain experts and software engineer. This SPM should contain some initial information (e.g., historical records related to location-aware systems) and requirements associated with location-aware systems, e.g., Context, ContextTypes, ContextUsers, LocationContext, LocationService, ServiceHandler, Map, Location, Person, Methodxception, RequestData, SendMail, Retrieval, Widget, Condition, Callbacks, ValueChanged, Attributes, CommunicationServer, CommunicationClient, CommunicationHandler and their corresponding keywords, comments and requirements so on.

Table 7.2 presents an example of the content of SCI and REQ in the initial SPM for the location application example – LocationSampleApp in Openmobster:

SCI	<i>Identifier</i>	getAddress
	<i>Keywords</i>	public; Address
	<i>Comments</i>	Return the address associated with this context
REQ	<i>FR</i>	address-polling
	<i>NFR</i>	high responsiveness

Table 7.2 A Snapshot of Initial SPM for Openmobster

According to Table 7.2, a concept named – Get|CurrentAddress with its corresponding event – getAddress should be in an initial SPM. Therefore, <Get|CurrentAddress, iButton> as a tuple will be stored in the KBL for further matching and updates. To discover services candidates, firstly a SPM is created, and constructed a KBL accordingly.

In this case study, six instances are in a SPM and six corresponding tuples of <Concept, Event> are in the KBL. The list of tuples is: [<Address, getAddress>; <Longitude, getCurrentLongitude>; <Place, getPlaceDetails>; <NearbyPlaces, getNearbyPlaces>; <Position, getPosition>; <MapAttribute, getMapAttribute>]. Because of individual preference of concept naming, the final KBL might appear rather different. A more clarified concept naming mechanics could be introduced to address this problem.

When SPM and KBL are constructed, HB-CA will be applied on 5 source files (.java): HomeScreen, LoadAddressMapCommand, LoadMyMapCommand, LocationMapActivity, and MyItemizedOverlay. At this stage, strict matching criteria is not applied, in effect, flexible matching is allowed (i.e., sub-string matching or ambiguous matching). The results at this point are demonstrated in Table 7.3.

KBL Elements	Identifiers	Events in Source
<Address, getAddress>	LoadAddress	clickEvent.getAddress
<Longitude, getCurrentLongitude>	getAttribute	clickEvent.getAttribute
<Place, getPlaceDetails>	getPlaceDetails	clickEvent.getPlaceDetails
<NearbyPlaces, getNearbyPlaces>	getNearbyPlaces	clickEvent.getNearbyPlace
<Position, getPosition>	getPosition	clickEvent.getPosition
<MapAttribute, getMapAttribute>	getMapAttribute	clickEvent.getMapAttribute

Table 7.3 A Snapshot of Updated Content of KBL

The content in KBL indicates the location of concept segments. When KBL is available static program slicing techniques are used to further decompose the qualified source code reflected from the results of SPM. In fact, program slicing is particularly useful when the code segments are too big. This process generates code segments of interest. For instance, the following code could be of our interest:

```

public void postRender()
{
    //Get an instance of the currently active Activity
    ListActivity listApp =
(ListActivity)Services.getInstance().getCurrentActivity();

    //Populate the List with Actions to be performed
    String[] ui = new String[]{"Map by Address","Map by My Location"};

    listApp.setAdapter(new ArrayAdapter(listApp,
        android.R.layout.simple_list_item_1,
        ui));
}

```

```
        ListItemClickListener clickListener = new ClickListener();  
        NavigationContext.getInstance().addClickListener(clickListener);  
    }
```

Once the target code is extracted, the next step – services recode begins. It is this stage that some of the constraints may be fully fulfilled. Since this functional requirement is well addressed by the comments in this code (sometimes, it is not the case), software engineers can exercise their domain knowledge to play a key role for optimising the code. On the non-functional requirement side, the above code implies the need to perform asynchronous computing for better responsiveness. Nevertheless, the existing programming paradigm might not be able to express it straight forward. User experience will pose this demand sooner or later for other control buttons to achieve more responsiveness. It is this point when software developers reflect their approaches for maintaining the services evolution stage. Finally, in services integration stage, with the help of some wrappers and code gluing techniques, reengineered services and newly-built functional services are composed via connectors in order to construct the target system. Such steps will be evaluated in the following case studies.

In summary, from usability perspective, the availability of the recovered code-related artifacts and requirements-related artifacts enables reusability of components of the legacy system as well as a comparison of existing requirements and new requirements that navigates further strategies of redesign and reimplementation in the course of forward engineering. From effectiveness perspective, not only are code-related artifacts are extracted, but requirements-related artifacts are recovered for reimplementation in the downstream of reengineering activities. By comparing the recovered requirements and sought-after requirements, new redevelopment technologies can be discovered, e.g., there could be another programming language that translates the problem domains into the solution domains far more expressively.

7.3 The Geolocation API

The Geolocation API [50] enables a Web application to obtain a user's geographical position. Specifically,

- Obtain the user's current position, using the 'getCurrentPosition' method.
- Watch the user's position as it changes over time, using the 'watchPosition' method.
- Quickly and cheaply obtain the user's last known position, using the 'lastPosition' property.

The Geolocation API provides the best estimate of the user's position using location providers. These providers may be onboard (e.g., GPS) or server-based (e.g., a network location provider). The 'getCurrentPosition' and 'watchPosition' methods support an optional parameter of type 'PositionOptions' specifying which location providers to use.

7.3.1 Overview of CASRM and Requirements Evolution Model

Based on SPM and the derived viewpoints, CASRM is developed to build context-aware services requirements. The items included in the derived viewpoints are described in Table 5.3. The detailed contents can be found and drawn from KBL and REQ. Changes may be caused not only by users who keep changing their mind, but by availabilities of new programming techniques that developers would raise the demand to consider adopting alternative implementation strategies or methods. The two viewpoints must be in phase. Viewpoints, not only conventionally make changes consistent, but build a relation between both viewpoints and stress two types of constraints – design and implementation requirements in order to mitigate the pain of software evolution.

ARRE is a synthesis of conventional users' and developers' viewpoints, and context constrains and predicates that assert the requirements are satisfied. For example, during forward engineering phase, developers could face a decision to select proper programming languages to implement the overall requirements. ARRE, built by domain experts and seasonal software engineers, synchronises both derived viewpoints and provides suggestion of changes to functional requirements.

Interface requirements become less important when accessing desired services via protocol such as HTTP and SOAP without using an interface (e.g., a Web browser). User interface requirements and ARRE are composed of the ultimate desired requirements. For each time the context-aware services requirements are generated, they will be seen as the initial requirements for the proposed requirements evolution model that will be described in the following section.

The proposed requirements evolution model contains following states: initial requirements, defined requirements, and released requirements. The initial requirements of services and context are discovered via RRF approach. Based on the modification rules, services requirements can be decomposed into functional requirements, non-functional requirements and interface requirements. The modified requirements are subject to Quality of Services (QoS). Feedback will be sent back to each initial requirement for evaluation. When the final version of the desired context-aware services requirements is obtained in the following services reengineering activities, it reaches the third state. The combined requirements of the evolved services requirements and evolved context requirements will be seen as initial requirements upon the next requirements evolution.

7.3.2 Requirements Evolution Model for The Geolocation API Applications

As the previous case study has shown the steps of creating SPM, which contributes to the main components in CASRM, this case study focuses on the steps of managing context-aware services requirements evolution process. The case study is carried out based on the sample application – RunningMan [50] from one of The API Geolocation's applications, which is a JavaScript application that uses The Geolocation APIs on Android. RunningMan is a location-aware stopwatch that measures both the time and route taken for a journey, showing the journey on a map. It utilises the modules (i.e., Database, Desktop shortcuts, Geolocation, and LocalServer) from The Geolocation API.

When the corresponding SPM is available, services requirements of each code related artifact can be highlighted. Table 7.4 describes the services requirements for improving the summary of position information. The context-aware services requirements of function 'PositionInformation' are separated into services requirements and context requirements for different modifying rules.

PositionInformation	Description
Functional Requirements	Saving Historical Position Information
Non-Functional Requirements	Reliable Useful Information
Interface Requirements	Relative Environment Changed
Context Requirements	New Position Information Accepted

Table 7.4 Services Requirements of PositionInformation

Now that the description displayed in the journeys screen contains the number of positions saved, as well as the distance and the average speed travelled. In fact, function 'PositionInformation' in model.js depicted as below:

```

/* For a given row, returns distance travelled, average speed,
* and number of positions saved
* /

function positionInformation(rowID) {
    var distance = 0;
    var prevLat = null;
    var prevLon = null;
    var firstTime = 0;
    var lastTime = 0;
    var nbPositions = 0;

    var rs = global.db.execute('SELECT Latitude, Longitude, Date ' +
        'FROM Positions WHERE TimeID = (?) ' +

```

```
'ORDER BY Date', [rowID]);  
  
...  
  
var secTime = (lastTime - firstTime) / 1000;  
  
var averageSpeed = ((distance * 3600) / secTime);  
  
var roundedDistance = (((distance*1000)|0)/1000);  
  
var roundedSpeed = ((averageSpeed*1000)|0)/1000;  
  
var description = "(" + roundedDistance + " km);  
  
description += "<br>Average speed: " + roundedSpeed + " km/h";  
  
description += "<br>" + nbPositions + " positions saved";  
  
return description;  
  
}
```

In order to further improve the history information saved for future use, for instance, users may want to review the routes that they have taken to the previous destinations. In such case, the ‘add’ modifying rule is adopted, and this related functional requirements will be edited as “Including previous routes to destinations” and non-functional requirements will be added “Adding former detailed routes to destinations in order to improve the use experience”. Later, the modified requirements are subject to test based on the formula: Quality (Q, S) \models Constraint (C) in a specific context. Finally, feedback will be sent back to initial related requirements with corresponding actors, in his case, the users and developers for ultimate confirmation. As our model is evaluated with more cases, it is reported that they imply some promising results on context-aware services requirements analysis particularly during the early reengineering activities.

In conclusion, based on the initial results of RRF, raw requirements are generated. They are kept in KBL with SCI. CASRM is built in terms of user’ and developer’s viewpoints and the content of KBL can be described through these two viewpoints. In effect, the separation of context requirements and

services requirements provides a concise way of applying different modifying rules to maintain the quality of the evolved requirements. Each generation of context-aware services requirements evolves continuously. The proposed Requirements Evolution Model generates the latest evolved requirements which are available either for comparison against the new requirements or these are the requirements to be fulfilled in forward engineering. It is efficient in the way that requirements are always in place for reimplementation.

7.4 Geoclue

Geoclue [49] is a modular geoinformation service built on top of the D-Bus [36] messaging system. The goal of the Geoclue project is to create location-aware applications as simple as possible and to facilitate integration of location-awareness techniques in Linux desktop applications. It also provides a C library and exposes its functionality through D-Bus.

Geoclue provides three interfaces for querying current situation, i.e., Position, Address and Velocity. Each contains a method and a signal to acquire the information in question along with the time and accuracy of the measurement. For instance, ‘position-example.c’ is taken into account, which uses ‘Position client API’. ‘Position-example.c’ contains an asynchronous method call – ‘geoclue_position_get_position_async()’ with a callback – ‘position_callback()’, details are represented below [49]:

```
void geoclue_position_get_position_async (  
    GeocluePosition *position,  
    GeocluePositionCallback callback,  
    gpointer userdata);
```

Function returns immediately and calls ‘callback’ when current position is available or when D-Bus timeouts.

- position : A GeocluePosition object
- callback : A GeocluePositionCallback function that should be called when return values are available
- userdata : Pointer for user specified data

In F#, there is a more concise way in writing asynchronous call method via natively supported asynchronous programming model. For example, the above code can be rewritten in F# as follows:

```
module GeoclueCaseStudy
//define a type that contains position information
type GeocluePosition(fields: GeocluePositionFields, timestamp: int, latitude: float, longitude: float, altitude: float, accuracy: GeoclueAccuracy) =
    member p.Fields = fields
    member p.Timestamp = timestamp
    member p.Latitude = latitude
    member p.Longitude = longitude
    member p.Altitude = altitude
    member p.Accuracy = accuracy
    member p.AsyncGetPosition() : Async<seq<GeocluePositionFields, int, float, float, float, GeoclueAccuracy>> =
        //define an async operation
        (...)
//define an async operation that works on multiple positions
let asyncGetPositions(p: GeocluePosition) =
    let completed = ref false
    async {
        while not(!completed) do
            let! position = p.AsyncGetPosition()
            if p.Fields && p.Latitude && p.Longitude then
                //do something with this position value
                ...
            else
                completed := true }
```

The type – ‘GeocluePosition’ contains detailed position information as well as expose a callback method, i.e., ‘AsyncGetPosition’ where an asynchronous operation is defined. Finally, function – ‘asyncGetPositions’ is called to work on multiple positions asynchronously. Comparing to the original implementation of ‘position-example.c’, the F# implementation is able to magically express the uniformed abstraction (e.g., abstracting callback functions) through writing asynchronous workflows which enables developers to write normal control flows for asynchrony. Due to the computing needs of

Web services-based context-aware applications, performing asynchronous computation is evitable and essential.

To summarise, using F# asynchronous programming model can at large facilitate the implementation of this kind of applications, which in turn, software developers will benefit from the decision of making a correct choice of programming languages soon after the context-aware services requirements are available and before the implementation. For example, the lines of code are 69 excluding comments, whilst the lines of the translation code in F# are 53. The F# counterpart provides same asynchronous behaviour as the original code, though the performance of the F# code is in theory slower than the C code. However, once the program reaches much more lines of code, the performance is not the main issues. Instead, the maintenance cost of the giant code is crucial and essential.

7.5 ContextChat

ContextChat [29] is a chat prototype implemented in ContextErlang [51]. Users in the systems are represented as context-adaptable agents. User conditions (e.g. online/offline), logging, remote backup are represented as context and dynamically activated on each agent. In addition, ContextErlang is an Erlang extension for COP. It combines the COP concepts along with the effective Erlang concurrency model. Specifically, variations enable alteration of the behaviour of context-aware agents, that is, behavioural components that can be activated on the agent. Composing the active variations with basic behaviour leads to the actual behaviour of the agent.

7.5.1 Overview of COP and F# Agent-Based Programming Model

COP is a new programming paradigm that enables software entities to adapt their behaviour to the current execution context. This goal is achieved by providing abstractions that enable application to have context-awareness behaviour without excessively using local-level conditional statements in the source code. To support COP, programming languages entail the following properties [56]:

- Means to specify behavioural variations,
- Means to group variations into layers,
- Dynamic activation and deactivation of layers based on context, and
- Means to explicitly and dynamically control the scope of layers.

In other words, COP languages and environment extensions should be able to provide mechanisms for expressing, activating and composing layers at execution where contextual information is related. Therefore, at least five properties are addressed [56], i.e., Behavioural Variations, Layers, Activation, Context, Scoping.

While COP facilitates the development of context aware systems, the implementation complexity of Web services-based context aware systems can be greatly reduced by combining COP with native language support for asynchronous and parallel programming.

In F#, agent-based programming model is part of the language. It uses a type of ‘MailboxProcessor<’Msg>’ to represent agents. The body of the agent is written as an asynchronous workflow, in other words, agent-based programming is based on asynchronous programming and agent is lightweight.

More details about the type of ‘MailboxProcessor<’Msg>’ has been discussed in Chapter 6.

7.5.2 Reimplementation via ContXFS and F# Agent-Based Messaging Techniques

ContextChat is implemented using ContextErlang which is an Erlang extension to support COP. While ContXFS is F# library to allow COP, the F# native high-level features enable it to be a good candidate to embrace COP features concisely. When evolved requirements are available, F# library ContXFS can be used to fulfil those requirements.

The following piece of code shows an example of enabling message awareness via F# message passing and mailbox processing:

```
///define an internal union type of messages for the agent
type internal Message =
    | SendMessage of string
    | GetMessage of AsyncReplyChannel<string>

///Agent alias for MailboxProcessor
type Agent<'T> = MailboxProcessor<'T>

///define a type that enables message awareness with an asynchronous method of getting messages
type Chat() =
    let agent = Agent.Start(fun agent ->
        let rec loop messages =
            async {
                let! msg = agent.Receive()
                match msg with
                | SendMessage textMsg ->
                    return! loop (textMsg)
                | GetMessage replyChannel ->
                    do replyChannel.Reply messages
                    return! loop messages }
            loop " ")
        member c.SendMessage(msg) =
            agent.Post(SendMessage msg)
        member c.AsyncGetMessage(?timeout) =
            agent.PostAndAsyncReply(GetMessage, ?timeout=timeout)
```

Immutable data types are commonly used in functional programming paradigm. Values defined in F# are immutable by default, although F# supports mutability as well. Specifically, as F# is a .NET framework language, the CLR makes sure that an initialisation of a value is thread-safe. When the

initialisation completes, the value is defined as immutable and this property enables thread-safe operations. On the other hand, instead of using threads to meet the requirements for shared-memory concurrency, F# uses agents as an alternative implementation of message-passing concurrency. Using agents in turn can avoid race conditions and deadlocks that mutability causes. F# agent-based programming model is built based on asynchronous workflow. In other words, F# agents do not block threads while waiting. Furthermore, agents are lightweight, which can scale an application with hundreds of thousands of agents.

7.5.3 Quantitative Experiments

Based on the claims expounded in Section 7.1 and 7.5.2 and, an experiment is carried out. A list of questions that this experiment is expected to answer is drawn as follows:

- Do F# and ContXFS facilitate **usability**?

In F#, an agent can be encapsulated into a class type (i.e., a Class in OO programming paradigm) and it often has a loop that asynchronously waits for incoming messages and processes them. F# comes with a library implementation of in-memory agents – *MailboxProcessor*. This library accommodates many primitives for asynchronous programming and agent-based programming. The agent encapsulates a message queue that supports multiple-writers and a single reader agent. Moreover, delegation can be applied as a compositional technique for reusing fragments of implementations. For example, in asynchronous agent-based programming, delegating members in the defined class type to the underlying agent provides a replacement for OO implementation inheritance that often complicates the hierarchical relations between types. The following piece of code demonstrates the delegation technique:


```
type WorkerAgent<'T>() =
    let agent = Agent.Start(fun agent ->
        // Message processing
        (...))

    // Delegating AsyncOp1 member to agent
    member x.AsyncOp1(t:'T, ?timeout) =
        agent.PostAndAsyncReply((fun ch -> Op1(t, ch)), ?timeout=timeout)

    // Delegating Op2 member to agent
    member x.Op2(t:'T) =
        agent.Post(t)
    })
```

Agent can be reused for generic proposes. For example, in terms of Microsoft Developer Network Platforms, reusable agents such as *BlockingQueueAgent* (see below) and application-specific agents provide basic building blocks for agent-based concurrent applications. The agents often communicate by some common scheme. The code below was modified from [83] .

```
/// Agent that implements an asynchronous blocking queue
type BlockingQueueAgent<'T>(maxLength) =
    let agent = Agent.Start(fun agent ->

        let queue = new Queue<'T>()
        /// State machines
        let rec emptyQueue() = async {...}

        and fullQueue() = async {...}

        and runningQueue() = async {...}

        and enqueueAndContinue (value, reply) = async {...}

        and dequeueAndContinue (reply) = async {...}

        and chooseState() = async {...}
        // Enter the initial state - an empty queue
        emptyQueue() )
```

Furthermore, in ContXFS, the type of *MailboxProcessor* is extended with a static member *SpawnAgent*. It formalises continuation as well as timeout and error handlers in the parameters list for *SpawnAgent* and wraps underlying static *Start* method inside the extension type. Along with other static extended members, these methods empower the agent to perform more interesting computations.

- Do F# and ContXFS facilitate **effectiveness**?

Effectiveness can be reflected by the asynchronous workflow and agent-based programming model that supports asynchronous and parallel programming. The above example has already depicted the adequate ability of asynchronous computation. In effect, F# supports multiple active evaluations (e.g., CPU-bound computations) and waiting reactions (e.g., I/O bound computations) in parallel. For example, the code below describes the CPU-bound computations in parallel:

```
// Define a sequence of async blocks
let sequenceInput num = seq {for i in 0 .. num do yield async {return i * i}}

// Evaluate the sequence using Async.Parallel
let results = sequenceInput 100 |> Async.Parallel |> Async.RunSynchronously
```

The sample code presented in Section 6.5.2.4 has demonstrated the capability of fetching the content of multiple Web pages in parallel.

- Do F# and ContXFS facilitate **efficiency**?

Both posting and receiving messages are very efficient in F# because of the implementation of message-passing. Posting one million messages approximately takes 0.125 second, and receiving all the messages takes about 11.850 seconds on a machine with specifications of Intel Core duo 2.0 GHz, 2.0GB memory, and 32-bit Windows 7. The code below shows the results.

```
// Define a Agent<T> - an alias for the MailboxProcessor<T> type
type Agent<'T> = MailboxProcessor<'T>

// Number limit
let max = 1000000

// Array initialisation
let arr = [|1 .. max|]

// create a stopWatch object
let stopWatch = System.Diagnostics.Stopwatch()

let agent = Agent<int>.Start(fun inbox ->
    async {
        while true do
            // Get elapsed time of receiving all messages
```

```
// Watch starts
stopWatch.Start()
let! msg = inbox.Receive()
// Watch stops
let elapsedTime = stopWatch.ElapsedMilliseconds
if msg = max then printfn "Finished! with elapsedTime: %d" (elapsedTime
) else ()
})

// Get elapsed time of posting all messages
#time
arr |> Array.iter(fun i -> agent.Post(i))
```

The lines of code may also well reflect that the efficiency of combination of F# and the ContXFS. For example, F# code is always more concise and shorter comparing to C# code. Typically, the following comparison of F# and C# code demonstrate the difference.

```
// F#
type Currency = Sterling of float

// C#
public abstract class Currency { }

public abstract class BritishCurrency : Currency
{
    public Amount Amount {get; private set;}
    public BritishCurrency(Amount amount)
    {
        this.Amount = amount
    }
}
```

In summary, the overall reimplementations in F# and the ContXFS support the claims of usability, effectiveness, and efficiency.

7.6 Development Toolkit

The section focuses on the supporting toolkit for reimplementations of a subject Web services-based context-aware systems. The implementation environment is Microsoft Visual Studio 2010 Professional where F# is the primary programming language to implement the redevelopment projects. Specifically, WebSharper™ 2010 Platform is used to facilitate the Web development part of the subject system. Now WebSharper is open source. This versatile F#

HTML5/Mobile development tool is developed by IntelliFactory [63], a software and consulting company specialising in F# programming language. The aim of WebSharper is to enable developers to program only F# code to build Web services applications based on the latest Web development techniques without using extra programming languages for other specific Web development tasks. It potentially fills the blank where F# code cannot be generated for UI designer.

Technically, WebSharperTM compiles F# code to JavaScript, and it exposes extensions to JavaScript libraries. The main benefits of developing JavaScript/HTML5/mobile applications with F# as the development language is driven by the strengths of F#, e.g., along with the high-level abstraction of modern typed functional programming language, .NET interoperability, full intellisense, type inference, asynchrony all count for the advantages of developing in F#.

7.7 Summary

In this chapter, four case studies have been selected to evaluate that the fundamental gap between requirements and implementation can be reconciled via the proposed reengineering approach, which in turn, to validate the main claims in this thesis.

- ❑ Openmobster case study helps to illustrate the detailed phases of applying RRF approach.
- ❑ The Geolocation API case study demonstrates the efficiency of CASRM as well usability of the proposed requirements evolution model.
- ❑ Geoclue case study can be seen as one of typical examples of the importance of fulfilling constraints. For example, choosing better

implementation languages that mitigate the development burden is critical because implementation requirements must be satisfied for reducing maintenance issues.

- ❑ ContextChat case study describes how to apply COP and F# agent-based programming model to facilitate the redevelopment of the legacy system.
- ❑ The quantitative experiment demonstrates that F# is a very good candidate for the reimplementation task. And the ContXFS library presents a concise way of implementing F# domain specific library.
- ❑ Development toolkit mainly consists of Microsoft Visual Studio 2010 and WebSharper.

Chapter 8 – Conclusions and Future Work

Objectives

- To summarise the whole thesis
- To revisit and extend the original contributions
- To evaluate this work with answers to the key research questions; by reviewing the research propositions and the criteria of success
- To illustrate the limitations of this work
- To outline the future work

8.1 Summary of Thesis

The reconciliation between requirements and implementation is an important research topic in requirements engineering community. A set of software reengineering methods can be potentially adopted to address the problem during the conventional requirements engineering process. Based on this assumption, this work has provided an excellent combined approach to reconciling the underlying gap between requirements and implementation for Web services-based context-aware systems. The aim of this work is to strengthen the capabilities of traditional software reengineering methods with

relevant novel techniques in order that they are able to address the increasingly critical reconciliation issues in the evolution of Web services-based context-aware systems. The basic idea is to improve reverse engineering techniques to recover the underlying users' and constraints from legacy systems, and to develop context-oriented programming approaches to mitigate the burdens of reimplementation for the legacy systems. Typically, in the midst of this software reengineering process, constraints are always emphasised, and on the top priority.

Several general research methods are employed in this proposed research work. Modelling plays a central role in this work as it can guide requirements elicitation, provide a measure of completeness of the elicitation, and visualise the requirements. Classification guarantees that software development is consistent and systematic. Quantitative and qualitative methods, reasoning, and DSL design are also adopted. The primary research subjects in this work are requirements engineering, software reengineering, and domain specific language design (specifically, context-oriented programming language extension). The proposed framework approach consists of the following four core phases, namely, legacy system assessment, services candidate discovery, services reimplementation, and services integration. Legacy system assessment is an assessment of the subject legacy system from imperative and OO programming paradigms that is responsible for judging the applicability of Context-Aware Web Services Reengineering (CAWSRF) approach and deciding if other reengineering approaches should be performed. Services candidate discovery is carried out based on the proposed Requirements Recovery Framework (RRF). Hypothesis-Based Concept Assignment (HB-CA) and programme slicing are applied. Services reimplementation is the process to redevelop the legacy system in the light of synthesised requirements-related artifacts and code-related artifacts, this process contains requirements mapping and ContXFS development. Finally, services integration enables legacy services and newly-built functional services are composed via connectors in

order to construct the target system. This can be implemented via wrappers and code gluing techniques. In Appendix A, a prototype context-aware chatting application is implemented in F# with ContXFS support to evaluate the overall proposed framework approach.

8.2 Original Contributions Revisiting

This work aims to enhance the traditional software reengineering methods to reconcile the increasing gap between requirements and implementation for Web services-based context-aware systems. This section will revisit and extend the eight original contributions described in Chapter 1.

C1: In Chapter 3, a novel reengineering process is created to mitigate the underlying gap between requirements and implementation for the Web services-based context-aware systems. This proposed framework approach consists of legacy system assessment, services candidate discovery, services reimplementation, and services integration.

C2: In Chapter 4, a requirements recovery framework (RRF) has been described. Concept assignment and programming slicing techniques are applied within the framework.

C3: In Chapter 4, requirements elicitation approach has been depicted. Hypothesis-based concept assignment (HB-CA) is applied into the elicitation process.

C4: In Chapter 5, a context-aware services requirements model (CASRM) is proposed to recover requirements-related artifacts and code-related artifacts from source code.

C5: In Chapter 5, a combined users' and developers' customised derived viewpoint is described. The recovered requirements and new context-aware

services requirements can be easily synthesised to restructure the requirements for services reimplementation.

C6: In Chapter 5, a requirements evolution model is developed to manage the evolved requirements in order to facilitate context-aware services evolution.

C7: In Chapter 6, a requirements analysis prior to services reimplementation is carried out via requirements mapping. A table of desired characteristics and reflected terms in a programming language has been created.

C8: In Chapter 6, a novel of server/client architectural design model is proposed, and implementation issues and strategies for services reimplementation are discussed.

C9: In Chapter 6, F# features and advantages for services reimplementation are presented within a table.

C10: In Chapter 6, relevant language attributes and features in F# have been discussed and introduced.

C11: ContXFS as a library in F# that allows for context-oriented programming (COP) is developed. ContXFS adopts COP paradigm to functional implementation model in F# that natively supports concurrency and parallelism.

C12: An investigation of context-aware adaption has been carried out. ContXFS provides programmers with libraries that assist the development of Web services-based context-aware systems. Typically, ContXFS enables software developers to facilitate the implementation of context-awareness at run time while the programming models natively supported in F# allow for Web services development.

C13: In Chapter 7, four further case studies are carried out to evaluate the overall the overall framework approach.

C14: In Appendix A, a prototype implementation of ContXFS and the testing samples of applying this library are given.

8.3 Evaluation

8.3.1 Answering Research Questions

The principle research question in this work has been described in Chapter 1:

**How can a software reengineering approach
be developed in order to reconcile the gap
between requirements and implement for
Web services-based context-aware systems?**

The brief answer to this question has been addressed with recovering underlying requirements along with code segments from the source code in a legacy system and developing a context-oriented programming language extension/library for facilitating redevelopment tasks. In addition to the two fundamental approaches, constraints are always on the top priority in the course of the software reengineering process. Specifically, deep recovery (requirements-related artifacts discovery) is more appropriate for dramatic redevelopment when the gap between existing code-related artifacts and new requirements is too big, and furthermore, a new programming language (model) could be available for higher abstraction, which allows for more concise implementation. Typically, ContXFS enables developers to facilitate context-oriented programming (COP) for dynamic context-awareness while F# is a good fit for Web services-based system development in a late reengineering process.

A range of detailed research questions has been defined accordingly to refine this holistic question as follows:

REQ1: What does the context-aware Web services candidate discovery recover?

Two types of artifacts discovery can be recovered: requirements-related artifacts discovery and code-related artifacts discovery. (Section 3.2.1)

- *What is the common architectural design of context-aware systems?*

Traditionally, the client-server architectural model consists of a set of servers, a set of clients, and the network that underpins the communication between the servers and clients. (Section 6.3)

- *How may requirements be extracted from source code in legacy systems?*

Underlying requirements may be extracted by applying HB-CA method into the proposed Requirements Recovery Framework (RRF) to enable the requirements elicitation approach. (Section 4.3)

- *How may other reengineering tasks benefit from the recovered requirements-related and code-related artifacts?*

Benefiting from the two discovery artifacts via requirements elicitation approach, Context-Aware Services Requirements Model (CASRM) extracts current requirements from source code level and reconstructs new context-aware services requirements primarily based on users' and developers' customised derived viewpoints. (Section 5.2.3) These artifacts may also facilitate the services reimplementation. (Chapter 6)

REQ2: Why non-functional requirements (i.e., qualities and constraints) are so important?

Emphasis on constraints can potentially reduce the costs and risks of re-implementing a complete existing system. (Section 3.2.2)

- *How may software evolution be hindered by not fully evaluating implementation decisions during the reengineering process?*

Developers are always forced to give in their needs to compromise users' needs. Hence, inefficient implementation will make services evolution much more difficult in the future. (Section 5.4)

- *How may constraints be discovered during the reengineering process?*

The derived viewpoints-based Context-Aware Services Requirements Model (CASRM) is proposed to fully discover the importance of constraints. (Section 5.2.3)

- *How may software developer's time and effects be impeded by inappropriate language choice?*

Software developers always face language choice as it is one of the most critical implementation issues as programming language itself may deeply impede software developer's time and effects on tackling the development tasks. (Chapter 5 and Chapter 6)

REQ3: How is services reimplementaion carried out?

The services reimplementaion follows the traditional software development process in terms for the new and recovered requirements, architectural design model, as well as implementation issues and strategies. (Chapter 6)

- *What are the requirements for services reimplementaion?*

The redevelopment requirements are composed of the recovered requirements from the source code and the new requirements. Requirements analysis is to map the implementation requirements to the programming language features or properties by comparing the results with what the currently used languages can offer. (Section 6.2)

- *How may the architectural design model be developed?*

The proposed architecture design for context-aware Web services systems highlights the Web services application development by introducing another layer. (Section 6.3)

- *What are the reimplementaion concerns and strategies?*

The reimplementaion issues on the server side can be basically classified as: performance issues, state sharing issues, and long-running operation issues,

whilst the reimplementation strategies are domain-specific. It depends on the architectural design and the target computing environments after implementation. (Section 6.4)

REQ4: How may domain specific language help in the reimplementation process?

Domain specific language mitigates the developer's burdens by developing a custom language to express implementation problems and solve the problems. (Chapter 6)

- *Which language and language paradigm may be suitable for building a domain specific language?*

Functional programming languages have had a long-time connection with development of domain specific languages since functional features or properties are suitable for creating compilers. (Section 6.5 and Section 6.6)

- *How may context-oriented programming be able to address the need for context-aware adaption?*

COP addresses the fundamental need for Web services-based context-aware applications that they should behave accordingly subject to the changing context at run time. Instead of spreading excessive raw conditional statements over the source code, higher level abstractions embedded in the target programming language can greatly facilitate the functionality of context-awareness. (Section 6.6)

- *How may a context-oriented programming library be developed?*

ContXFS is developed in F# based on the properties that COP must own, i.e., behavioural variations, layers, activation, context, and scoping. (Section 6.6)

8.3.2 Research Proposition Revisiting

The underlying proposition of this work has depicted in Chapter 1 as:

Requirements elicitation during reverse engineering and domain specific language support during forward engineering can be combined in order to reconcile requirements and implementation for the said systems.

Requirements elicitation approach and ContXFS have been developed to show that this proposition is sounded. A subset of propositions can be further described as follows:

PRO1: *A combination of viewpoints-based requirements, as well as code-related artifacts can be recovered from legacy systems.*

A requirements elicitation approach based on the proposed requirements recovery framework have been developed, which shows that this proposition is sounded.

PRO2: *The language choice makes a profound impact on the structure of the development solutions as well as how software developers think of the implementation issues.*

The recovered code-related artifacts suggest the existence of convoluted development in the legacy system, which shows that this proposition is sounded.

PRO3: *Raising the importance of choosing language(s) for implementation.*

A comparison of various programming features and their corresponding advantages towards services reimplementations through requirements mapping has been presented; the architectural design model implies that the communication between Web services and Web applications has to be largely implemented in an asynchronous way; reimplementations concerns and strategies demand native programming language support, all of which show this proposition is sounded.

PRO4: *DSL allows software developers to quickly and efficiently develop a software system.*

ContXFS, as an F# library allowing for COP, provides libraries to facilitate context-awareness implementation and enable developers to embrace F# programming models and other features for Web services development, which shows the proposition is sounded.

8.3.3 Revisiting Criteria of Success

A set of measures of success has been defined in Chapter 1. These predefined criteria will be revisited as follows:

- *The proposed approach should be able to reconcile the underlying gap between requirements and implementation for the said systems.*

The proposed work is able to recover the underlying requirements from source code through a combination of techniques of reverse engineering and requirements modelling, and is able to mitigate the software developer's burdens by application of ContXFS for facilitating context-awareness reimplementation and Web services-based systems redevelopment via a range of F# high-level features and programming models.

- *The requirements recovery framework approach should be able to elicit users' requirements and constraints that reflect the original requirements.*

The proposed RRF and CASRM are derived users' and developers' viewpoints-based framework and model respectively. The CASRM found in RRF is able to recover users' underlying requirements and constraints.

- *The context-aware Web services requirements model should be able to reconstruct new requirements combining with existing requirements.*

The content of SPM in the proposed CASRM is well maintained in a table. New requirements along with their corresponding information can be easily to add to the table. The Associate Requirements Repository Engine (ARRE) in the proposed context-aware services requirements model (CASRM) contains a synthesis of traditional users' and developers' viewpoints, and context constrains and predicates that assert the requirements are satisfied, which are easily combined with new requirements that share the structure of corresponding viewpoints.

- *The requirements evolution model should be able to manage the services requirements and context in a way to support services evolution.*

The requirements evolution model is able to distill context-aware services requirements into services requirements and context requirements, and maintain such requirements. The changing services requirements and context requirements are two triggers of services evolution. As long as the evolved context-aware services requirements are available, software engineers are able to carry out a series of reengineering processes to fulfil those requirements.

- *The architectural design model should be able to uncover reimplementatation concerns and strategies.*

The proposed architectural design model presented in Section 6.3 is based on conventional server-client architectural model. It highlights the communication between Web server and Web applications. The actual action that handles different kinds of events is implemented in the Web applications, while the middleware - context server is implemented on Web services side to manage context-awareness.

- *The ContXFS should be able to address the reimplementatation issues and provide programmatic supporting for development.*

ContXFS is an library to F# which allows for context-oriented programming (COP). ContXFS is implemented in language F# and fulfils all the five core properties of COP, which itself enables software developers to facilitate implementation of context-awareness at run time. Moreover, F# is a good fit into Web services development. Particularly, F# provides various kinds of native programming models for addressing the non-functional implementation issues. In addition, high-level features from F# make source code more much concise than most of the current mainstream programming languages.

- *The implementation of a Web services-based context-aware system should be able to realise the architectural design and meet the combined requirements such as context-awareness, concurrency, reliability, and scalability.*

The prototype of the context-aware chat application is developed in the light of the proposed architecture design model with support of ContXFS and F# programming models. Specifically, it fulfils a range of implementation issues, e.g., context-awareness, concurrency, asynchrony, parallelism, and high scalability.

8.4 Limitations

Following the original contributions and measures of success, the limitations of this work can be described below:

- *Requirements extraction via RRF approach and requirements management via the requirements evolution model may demand manual work and become time consuming.*

It may involve fairly much manual work relying on domain or software engineering experts, in practice, some key tasks may be unlikely to be

automatic. Because of the complexity and depth of requirements-related recovery, it cannot be implemented (semi-)automatically. Managing requirements during services evolution entails a systematic maintenance of requirements evolution. Typically, context-aware systems have a potential application in parallel computing. For example, the rate of underlying context changes rising dramatically due to rapid multi-core development; the scalability to scale services up to even more users. In order to achieve correct and practical results, recovering and maintaining such frequently changing requirements may make manual work inevitable.

- *F# reimplementation may be not as efficient and effective as others.*

In effect, some F# implementation cannot as concise and expressive as other language implementation due to lack of certain higher abstractions from F#. Nevertheless, due to the natural implementation issues and strategies of the said system, F# is still a better candidate for the development. In addition to its interoperability with other .NET languages, F# lends itself to multi-paradigms where it enables appropriate ‘polyglot programming’ to solve the practical problems.

8.5 Future Work

In this thesis, a novel reengineering approach is proposed to reconcile the underlying gap between requirements and implementation for Web services-based context-aware systems. In terms of the discussions with respect to the research questions, the research propositions, the original contributions, the criteria of success, and the limitations, the following conclusions can be drawn.

The proposed context-aware Web services reengineering framework (CAWSRF) is an overall framework on which other proposed frameworks and models are found, i.e., Requirements Recovery Framework (RRF), Context-

Aware Services Requirements Model (CASRM), requirements evolution model. During the reimplementation process, context-oriented programming concept is adopted. ContXFS as a library in F# is developed to facilitate this task. Typically, the overall proposed framework approach consists of the following core phases: legacy system assessment, services candidate discovery, services reimplementation, and services integration.

The four case studies in Chapter 7 demonstrate that the overall reengineering process is able to achieve reconciliation to a great extent. Nevertheless, the research work in this thesis is not the terminus. Further work can be suggested to be enhanced based on the current work.

- Due to the individuality of naming approaches to binding concept discussed in Section 4.3.3 and Section 7.2.2, the initial and final content names in SPM might appear rather different. A more clarified concept naming mechanics (e.g., with ontology support) could be introduced to address this problem.
- The modifying rules for the requirements evolution model in Figure 5.2 and Section 5.3 can be added some more concrete rules to improve the efficiency of maintaining changing requirements.
- In addition to context-awareness and concurrency functional requirements, the implementation of a Web services-based context-aware system should be able to realise an architectural design and to meet more combined requirements such as, reliability, scalability, security, and portability.
- Further case studies are necessary to evaluate the present work as this novel reengineering approach to evolution of the said systems is still at a fairly early stage.

References

- [1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a Better Understanding of Context and Context-Awareness”, *Handheld and Ubiquitous Computing*, vol. 1707, pp. 304-307, 1999.
- [2] H. Ailisto, P. Alahuhta, V. Haataja, V. Kyllnen, and M. Lindholm, “Structuring Context Aware Applications: Five-Layer Model and Example Case”, *In Proceedings of The Ubicomp Workshop on Concepts and Models for Ubiquitous Computing*, Göteborg, Sweden, 2002.
- [3] R. Ali, F. Dalpiaz, and P. Giorgini, “A Goal-Based Framework for Contextual Requirements Modeling and Analysis”, *Requirements Engineering*, Vol. 15, Nr. 4, pp. 439–458, 2010.
- [4] R. Ali, F. Dalpiaz, P. Giorgini, and V. E. S. Souza, “Requirements Evolution: from Assumptions to Reality”, *In Exploring Modelling Methods for Systems Analysis and Design*, 2011.
- [5] D. R. de Almeida, C. de S. Baptista, E. R. da Silva, C. E. C. Campelo, H. F. de Figueirêdo and Y. A. Lacerda, “A Context-Aware System Based on Service-Oriented Architecture”, *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1*, pp. 205-210, IEEE Computer Society. 2006.
- [6] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid, “A Comparison of Context-Oriented Programming Languages”, *In International Workshop on Context Oriented Programming*, pp.1-6, 2009.

References

- [7] M. Appeltauer, R. Hirschfeld, M. Haupt and H. Masuhara, "ContextJ: Context-oriented Programming with Java", *Computer Software*, Vol.28, No.1, pp. 272-292, 2011.
- [8] D. Athanasopoulos, A. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis, "CoWSAMI: Interface-Aware Context Gathering in Ambient Intelligence Environments", *Pervasive Mob. Comput.*, 4 (3), pp. 360-389, 2008.
- [9] M. Baldauf, S. Dustdar, and F. Rosenberg, "A Survey on Context-Aware Systems", *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4), pp.263–277, 2007.
- [10] J. E. Bardram, "The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications," *Proceedings of the 3rd International Conference on Pervasive Computing*, Lecture Notes in Computer Science, Munich, Germany, Springer Verlag. May 2005.
- [11] F. Baude, D. Caromel, N. Furmento and D. Sagnol, "Overlapping Communication with Computation in Distributed Object Systems", *In Proceedings of the 7th International Conference on High Performance Computing and Networking*, pp. 744-754, Amsterdam, Netherland, April 1999.
- [12] K. H. Bennett, "Legacy Systems: Coping with Success", *IEEE Software*, 1995.
- [13] K. H. Bennett and V. Rajlich, "Software Maintenance and Evolution: A Roadmap", *ICSE - Future of SE Track*, pp.73-87, 2000.
- [14] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklase, A. Ranganathan and D. Riboni, "A Survey of Context Modelling and Reasoning Techniques", *Pervasive and Mobile Computing*, 6(2), pp. 161- 180, 2010.

References

- [15] T. J. Biggerstaff, B. G. Mitbender, and D. Webster, "The Concept Assignment Problem in Program Understanding", *In Proceedings International Conference on Software Engineering*, Baltimore, Maryland, pp. 482 – 498, 1993.
- [16] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues", *IEEE Software*, vol. 16, no. 5, pp. 103-111, September/October 1999.
- [17] J. Bloch, "Effective Java", *Prentice Hall*, First Edition, June 2001.
- [18] M. Brodie and M. Stonebraker, "Migrating Legacy Systems: Gateways, Interfaces, and the Incremental Approach", *Morgan Kauffman*, March 1995.
- [19] L. Capra, W. Emmerich, and C. Mascolo, "Reflective Middleware Solutions for Context-Aware Applications", *in Proc. of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Kyoto, Japan, vol. 2192 of LNCS, pp. 126–133, September 2001.
- [20] C. K. Chang, H. Jiang, H. Ming, and K. Oyama, "Situ: A Situation Theoretic Approach to Context-Aware Service Evolution", *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, pp. 261–275, 2009.
- [21] F. Chen, Z. Zhang, J. Li and H. Yang, "Service Identification via Ontology Mapping", *33rd Annual IEEE International Computer Software and Applications Conference*, Seattle, WA, pp. 486-491, 2009.
- [22] F. Chen, H. Zhou, H. Yang, M. Ward, and W. C. Chu, "Requirements Recovery by Matching Domain Ontology and Program Ontology", *IEEE 35th Annual Computer Software and Applications Conference*, pp.602–607, July 2011.

References

- [23] G. Chen, and D. Kotz, “A Survey of Context-Aware Mobile Computing Research”, *Hanover*, NH, USA, Dartmouth College, 2000.
- [24] H. Chen, “An Intelligent Broker Architecture for Pervasive Context-Aware Systems”, *PhD thesis*, University of Maryland, Baltimore County. 2004.
- [25] I. Y. L. Chen, S. J. Yang, and J. Zhang, “Ubiquitous Provision of Context Aware Web Services”, *Proceedings of the IEEE International Conference on Services Computing*, pp. 60-68, Washington: IEEE Computer Society. 2006.
- [26] B. H. C. Cheng and J. M. Atlee, “Research Directions in Requirements Engineering”, *In Future of Software Engineering*, IEEE Computer Society Washington, DC, USA, pp. 285-303, 2007.
- [27] E. Chikofsky and J. Cross, “Reverse Engineering and Design Recovery: A Taxonomy”, *IEEE Software*, vol. 7, no. 1, pp. 13-17, January 1990.
- [28] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, “Non-Functional Requirements in Software Engineering. Boston: *Kluwer Academic Publishers*. 2000.
- [29] ContextChat, “ContextChat”, <http://home.dei.polimi.it/salvaneschi/software/contexterlang/contexterlang.html> [Retrieved: Jan 2012].
- [30] COP Implementations, “COP Implementations of Language Extensions Such as ContextJ, ContextJS, ContextS, ContextL, and ContextPy”, <http://www.swa.hpi.uni-potsdam.de/cop/> [Retrieved: Oct 2011].
- [31] P. Costanza and R. Hirschfeld, “Language Constructs for Context-Oriented Programming: An Overview of ContextL”, *In Proceedings of the Dynamic Languages Symposium*, co-organised with OOPSLA'05, New York, NY, USA, ACM Press, 2005.

References

- [32] A. van Deursen and P. Klint, “Little Languages: Little Maintenance?” *ACM SIGPLAN Workshop on Domain-Specific Languages*. 1997.
- [33] A. van Deursen and P. Klint, “Domain-Specific Language Design Requires Feature Descriptions”, *Journal of Computing and Information Technology*, 10(1), pp. 1–17, 2002.
- [34] A. K. Dey, “Understanding and Using Context”, *Personal and Ubiquitous Computing*, vol. 5, pp. 4-7, 2001.
- [35] G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini, “Comprehending Web Applications by a Clustering Based Approach”, *In Proc. of the 10th International Workshop on Program Comprehension*, pp. 261–270, Paris, France, June 2002.
- [36] D-Bus, “D-Bus”, <http://www.freedesktop.org/wiki/Software/dbus> [Retrieved: Jan 2012].
- [37] M. El-Ramly, E. Stroulia, and P. Sorenson, “Recovering Software Requirements from System-User Interaction Traces”, *In Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 447–454, 2002.
- [38] N. A. Ernst, J. Mylopoulos, and Y. Wang, “Requirements Evolution and What (Research) to Do about It”, *In Design Requirements Engineering: A Ten-Year Perspective*, volume 14, chapter 3, pp. 186-214, Springer Berlin Heidelberg, 2009.
- [39] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, “The Many Faces of Publish/Subscribe”, *ACM Comp. Surveys*, vol.35, no.2, pp.114-131, 2003.
- [40] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, “Controlling Requirements Evolution: A Formal Concept Analysis-Based Approach”, *In International Conference on Software Engineering Advances*. 2007.

References

- [41] M. Felici, “Taxonomy of Evolution and Dependability”, *In Proceedings of the Second International Workshop on Unanticipated Software Evolution*, USE, Warsaw, Poland, pp. 95–104, 5-6 April 2003.
- [42] M. Felici, “Observational Models of Requirements Evolution”, *PhD thesis*, Laboratory for Foundations of Computer Science, School of Informatics, The University of Edinburgh, 2004.
- [43] A. Finkelstein and A. Savigni, “A Framework for Requirements Engineering for Context-Aware Services”, *First International Workshop from Software Requirements to Architecture*, 23d International Conference on Software Engineering, 2001.
- [44] M. Fowler, “Refactoring: Improving the Design of Existing Programs”, *Addison-Wesley*, 1999.
- [45] F#, “F# at Microsoft Research”, <http://research.microsoft.com/en-us/> [Retrieved: Oct 2011].
- [46] F# Language Specification, “The F# 2.0 Language Specification”, April 2010. <http://research.microsoft.com/en-us/> [Retrieved: Oct 2011].
- [47] M. Galster and E. Bucherer, “A Taxonomy for Identifying and Specifying Non-Functional Requirements in Service-Oriented Development”, *In Proceedings of the IEEE Congress on Services - Part I*, pp. 345–352, Washington, DC, USA, 2008.
- [48] M. L. Gassanenko, “Context-Oriented Programming”, *In EuroForth'98*, Schloss Dagstuhl, Germany, April 1998.
- [49] Geoclue, “Geoclue”, <http://www.freedesktop.org/wiki/Software/GeoClue> [Retrieved: Jan 2012].

References

- [50] Geolocation, “The Geolocation API”,
http://code.google.com/apis/gears/api_geolocation.html#overview
[Retrieved: Jan 2012].
- [51] C. Ghezzi, M. Pradella, and G. Salvaneschi, “Programming Language Support to Context-Aware Adaptation - A Case-Study with Erlang,” *Software Engineering for Adaptive and Self-Managing Systems*, International Workshop, ICSE 2010.
- [52] N. Gold and K. H. Bennett, “Hypothesis-Based Concept Assignment in Software Maintenance”, *IEEE Proceedings Software*, 149(4): pp. 103–110, 2002.
- [53] N. E. Gold, M. Harman, D. Binkley, and R. M. Hierons, “Unifying Program Slicing and Concept Assignment for Higher-Level Executable Source Code Extraction”, *Software Practice and Experience* 35 (10), pp. 977–1006, 2005.
- [54] B. Han, W. Jia, J. Shen, and M. C. Yuen, “Context-Awareness in Mobile Web Services”, *Parallel and Distributed Processing and Applications*, pp. 519-528, Springer-Verlag. 2008.
- [55] S. D. P. Harker, K. D. Eason, and J. E. Dobson, “The Change and Evolution of Requirements as A Challenge to The Practice of Software Engineering”, *In: IEEE International Symposium on Requirements Engineering*, pp. 266–272, 1993.
- [56] R. Hirschfeld, P. Costanza, O. Nierstrasz, “Context-Oriented Programming”, *Journal of Object Technology* 7(3), March/April 2008.
- [57] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, and J. Altmann, “Context-Awareness on Mobile Devices – The Hydrogen Approach”, *In Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 292–302, 2002.

References

- [58] J. Huang, H. Yang, and L. Liu, “Reconciling Requirements and Implementation via Reengineering for Context-Aware Service Evolution”, *IEEE 35th Annual Computer Software and Applications Conference Workshops*, pp.464–469, July 2011.
- [59] J. Huang, H. Yang, L. Xu, B. Xu, and H. Zhang, “Supporting Context-Aware Service Evolution with A Process Management Requirements Model”, *Knowledge and Service Technology for Life, Environment, and Sustainability*, International Workshop, SOCA 2011.
- [60] J. Huang and H. Yang, “A Functional Implementation Approach for Web Services-based Context-Aware Systems”, (Submitted to Compsac2012).
- [61] P. Hudak, “Conception, Evolution, and Application of Functional Programming Languages” *ACM Computing Surveys* 21(3): pp. 359–411, September 1989.
- [62] IEEE Standard Definition of Software Engineering, “IEEE Standard Collection: Software Engineering”, *IEEE Inc.*, New York, 1997.
- [63] IntelliFactory, “IntelliFactory”, <http://www.intellifactory.com/> [Retrieved: Oct 2011].
- [64] R. Keays and A. Rakotonirainy, “Context-Oriented Programming”, *In Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, pp. 9-16, ACM Press, New York, NY, USA, 2003.
- [65] M. Keidl, and A. Kemper, “Towards Context-Aware Adaptable Web Services”, *In: Proc. of 13th Int. World Wide Web Conference*, New York, pp. 55–65, 2004.

References

- [66] A. van Lamsweerde, R. Darimont, and E. Letier, “Managing Conflicts in Goal-Driven Requirements Engineering”, *IEEE Transactions on Software Engineering*, 24(11): pp. 908-926, 1998.
- [67] A. van Lamsweerde, “Requirements Engineering in the Year 00: A Research Perspective”, *In Proceedings of the 22nd International Conference on Software engineering*, pp. 5–19, Limerick, Ireland, 4-11 June 2000.
- [68] A. Van Lamsweerde and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering,” *IEEE Transactions on Software Engineering*, 26: pp. 978-1005, 2000.
- [69] A. van Lamsweerde, “Goal-Oriented Requirements Engineering: A Guided Tour”, *Proc. of the 5th International Symposium on Requirements Engineering*, pp. 249- 262, 2001.
- [70] F. Lanubile and G. Visaggio, “Extracting Reusable Functions by Flow Gragh-Based Program Slicing,” *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 246-259, April 1997.
- [71] P. A. Laplante, J. Zhang, and J. Voas, “What’s in a Name? Distinguishing between SaaS and SOA”, *IT Professional*, 10(3), pp. 46–50, May-June 2008.
- [72] M. M. Lehman, and J. F. Ramil, “Software Evolution – Background, Theory, Practice”, *Inf. Process. Lett.* 88 (1-2), 2003.
- [73] K. J. Lieberherr, and C. Xiao, “Object-Oriented Software Evolution”, *IEEE Trans. Software Eng.* 19 (4), 1993.
- [74] K. Liu, A. Alderson, and Z. Qureshi, “Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour”, *in the proceedings of International Conference on Software Maintenance*, pp. 2-12,1999.

References

- [75] K. Liu, “Requirements Reengineering from Legacy Information Systems Using Semiotic Techniques”, *Systems, Signs and Actions – The International Journal on Communication, Information Technology and Work*, vol. 1 (1), pp. 36–61, 2005.
- [76] M. Lormans, “Monitoring Requirements Evolution Using Views”, *In Proceedings of the European Conference on Software Maintenance and Reengineering*, pp. 349–352, 2007.
- [77] M. von Lowis, M. Denker, and O. Nierstrasz, “Context-Oriented Programming: Beyond Layers”, *In Proceedings of the International Conference on Dynamic Languages*, volume 286 of ACM International Conference Proceeding Series, pp. 143-156, ACM Press. 2007.
- [78] N. Maiden, “CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements”, *Automated Software Engineering*, 5(4): pp. 419-446, 1998.
- [79] H. Mei and X. Liu, “Internetware: An Emerging Software Paradigm for Internet Computing”, *Journal of Computer Science and Technology*, Volume 26, Number 4, pp. 588-599, 2011.
- [80] T. Mens, and T. Tourwé, “A Survey of Software Refactoring”, *IEEE Trans. Software Engineering*, 30(2) pp.126–162, February 2004.
- [81] A. Monden, S. Sato, K. Matsumoto, and K. Inoue, "Modeling and Analysis of Software Aging Process", *Lecture Notes in Computer Science*, Vol. 1840, pp. 140 - 153, 2000.
- [82] MPI, “The Message Passing Interface Standard”, <http://www-unix.mcs.anl.gov/mpi> [Retrieved: Oct 2011].
- [83] MSDN, “Microsoft Developer Network Platforms”, <http://msdn.microsoft.com/en-us/default.aspx> [Retrieved: Jan 2012]

References

- [84] B. Nuseibeh, “A Multi-Perspective Framework for Method Integration”, *PhD Thesis*, Department of Computing, Imperial College, London, October 1994.
- [85] B. Nuseibeh and S. Easterbrook, “Requirements Engineering: A Roadmap”, *In Proceedings of the Conference on The Future of Software Engineering*, pp. 35-46, New York, NY, USA, ACM, 2000.
- [86] B. Nuseibeh, J. Kramer, and A. Finkelstein, “ViewPoints: Meaningful Relationships are Difficult!” *In Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, USA. IEEE Computer Society, 2003.
- [87] W. F. Opdyke, “Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks”, *PhD thesis*, University of Illinois at Urbana-Champaign, 1992.
- [88] Openmobster, “Openmobster”, <http://code.google.com/p/openmobster/> [Retrieved: Dec 2011].
- [89] K. Oyama, C. Chang, H. Jaygarl, A. Takeuchi, J. Xia, and H. Fujimoto, “Requirements Analysis Using Feedback from Context Awareness Systems”, *In Proc. Computer Software and Applications Conference*, pp.625—630, 2008.
- [90] M. P. Papazoglou, “The Challenges of Service Evolution”, *In Proceedings of the 20th international conference on Advanced Information Systems Engineering* 2008.
- [91] Path of Go, “The Path of Go”, <http://research.microsoft.com/en-us/projects/pathofgo/> [Retrieved: Jan 2012].
- [92] A. G. de Prado and G. Ortiz, “Context-Aware Services: A Survey on Current Proposals”, *The Third International Conferences on Advanced Service Computing*, Service Computation 2011.

References

- [93] A. Rakotonirainy, "Context-Oriented Programming for Pervasive Systems", *Technical Report*, University of Queensland, September 2002.
- [94] C. Rolland, "Panel on Requirements Engineering for Services", *the 33rd Annual IEEE International Computer Software and Applications Conference*, 20-24 July, pp.19-24, 2009.
- [95] B. Schilit, and M. Theimer, "Disseminating Active Map Information to Mobile Hosts", *IEEE Network*, 8(5) pp. 22-32, 1994.
- [96] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications", *1st International Workshop on Mobile Computing Systems and Applications*. pp. 85-90, 1994.
- [97] N. F. Schneidewind and C. Ebert, "Preserve or Redesign Legacy Systems", *IEEE Software*, Vol. 15, No. 4, pp. 14 - 17, July/August 1998.
- [98] A. Shet, P. Sadayappan, D. Bernholdt, J. Nieplocha, and V. Tipparaju, "A Framework for Characterizing Overlap of Communication and Computation in Parallel Applications", *Cluster Computing*, vol. 11, no. 1, pp. 75–90, March 2008.
- [99] A. Sivasubramaniam, "Reducing the Communication Overhead of Dynamic Applications on Shared Memory Multiprocessors", *Technical Report CSE-96-047*, Dept. of Computer Science and Engineering, The Pennsylvania State University, July 1996.
- [100] H. Sneed, "Planning the Re-engineering of Legacy Systems", *IEEE Software*, vol. 12, no. 1, pp. 24 – 34, January 1995.
- [101] A. Sohn, J. Ku, Y. Kodama, M. Sato, H. Sakane, H. Yamana, S. Sakai, and Y. Yamaguchi, "Identifying the Capability of Overlapping Computation with Communication", *In Proceedings of ACM/IEEE*

References

- Conference Parallel Architectures and Compilation Techniques*, pp. 133-138, 1996.
- [102] I. Sommerville and P. Sawyer, “Requirements Engineering”, *Wiley*, 1997.
- [103] D. Syme, “Leveraging .NET Meta-Programming Components from F#: Integrated Queries and Interoperable Heterogeneous Execution”, *In Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, pp. 43-54, 2006.
- [104] D. Syme, A. Granicz, and A. Cisternino, “Expert F# 2.0”, *Apress*, 2010.
- [105] D. Syme, T. Petricek, D. Lomov, “The F# Asynchronous Programming Model”, *In Practical Aspects of Declarative Languages*, pp. 175-189, 2011.
- [106] D. Syme, “Don Syme's WebLog on F# and Related Topics”, <http://blogs.msdn.com/b/dsyme> [Retrieved: Oct 2011].
- [107] F. Tip, “A Survey of Program Slicing Techniques”, *Journal of Programming Languages* 3, pp. 121–189. 1995.
- [108] H. L. Truong, L. Juszczak, A. Manzoor, and S. Dustdar, “ESCAPE - An Adaptive Framework for Managing and Providing Context Information in Emergency Situations”, *Smart Sensing and Context, Second European Conference, EuroSSC*, pp. 207-222, Springer-Verlag, 2007.
- [109] H. L. Truong and S. Dustdar, “A Survey on Context-Aware Web Service Systems,” *International Journal of Web Information Systems*, 5(1): pp. 5–31, 2009.
- [110] E. Truyen, N. Cardozo, S. Walraven, J. Vallejos, E. Bainomugisha, S. Günther, T. D'Hondt, W. Joosen, “Context-Oriented Programming for Customizable SaaS Applications”, (accepted paper) *The 27th*

References

- Symposium On Applied Computing, Cloud Computing Track*, Italy, March 2012.
- [111] W. Tsai, Z. Jin, and X. Bai, “Internetwork Computing: Issues and Perspective”, *Int J Software Informatics*, Vol.3, No.4, pp. 415-438, December 2009.
- [112] T. T. Tun, Y. Yu, R. Laney, and B. Nuseibeh, “Recovering Problem Structures to Support the Evolution of Software Systems”, *Technical report*, The Open University, 2008.
- [113] UWCN, “University of Washington Campus Navigator”, <http://uwcampusnav.sourceforge.net> [Retrieved: Mar 2011].
- [114] S. Vinoski, “Welcome to the Functional Web”, *IEEE Internet Computing*, pp. 102–104, January 2009.
- [115] P. Wadler, “Why No One Uses Functional Languages”, *ACM SIGPLAN Notices*, pp. 23-27, 1988.
- [116] M. Waldburger, C. Morariu, P. Racz, J. Jähnert, S. Wesner, B. Stiller, “Grids in A Mobile World: Akogrimo’s Network and Business Views”, *IFI Technical Report No.05*, University of Zurich, 2006.
- [117] R. Want, A. Hopper, V. Falcao, and J. Gibbons, “The Active Badge Location System”, *ACM Transactions on Information Systems* 10(1) pp. 91-102, 1992.
- [118] WebSharper, “WebSharper”, <http://www.websharper.com> [Retrieved: Jan 2012].
- [119] M. Weiser, “Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method”, *PhD thesis*, University of Michigan, Ann Arbor, 1979.

References

- [120] M. Weiser. “Programmers Use Slices When Debugging”, *Communications of the ACM*, 25(7): pp. 446–452, 1982.
- [121] M. Weiser, “Program Slicing”, *IEEE Transactions on Software Engineering*, 10(4): pp. 352–357, 1984.
- [122] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, “A Brief Survey of Program Slicing”, *SIGSOFT Software Engineering Notes*, 30, 2, pp. 1-36, 2005.
- [123] F. Yang, J. Lv, and H. Mei, “Technical Framework for Internetware: An Architecture-Centric Approach”, *Science in China Series F: Information Sciences*. pp. 610-622, 2008.
- [124] H. Yang and M. Ward, “Successful Evolution of Software Systems”, *Artech House Publishers*, January 2003.
- [125] H. Yang, Z. Cui and P. O'Brien, “Extracting Ontologies from Legacy Systems for Understanding and Re-engineering”, *23rd Annual International Computer Software and Applications Conference*, Phoenix, AZ, October 1999.
- [126] Yaws, “Yaws”, <http://yaws.hyber.org/> [Retrieved: Jan 2012].
- [127] E. S. Yu, “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”, *3rd IEEE Int. Symp. On Requirements Engineering*, pp. 226-235, 1997.
- [128] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, and J. C. S. do Prado Leite, “Reverse Engineering Goal Models from Legacy Code”, *In 13th IEEE International Conference on Requirements Engineering*, pp. 363–372, 2005.
- [129] Y. Yu, J. Mylopoulos, Y. Wang, S. Liaskos, A. Lapouchnian, Y. Zou, M. Littou, and J. C. S. P. Leite, “RETR: Reverse Engineering to

- Requirements”, *In 12th Working Conference on Reverse Engineering*, 7-11 November 2005, Pittsburgh, PA, USA, pp. 234-234, 2005.
- [130] B. Zagajsek, K. Separovic, and Z. Car, “Requirements Management Process Model for Software Development Based on Legacy System Functionalities”, *9th International Conference on Telecommunications*, pp.115-122, 2007.
- [131] P. Zave, “Call for Papers and Associated Classification Scheme”, *IEEE International Symposium on Requirements Engineering*, 1995.
- [132] Z. Zhang, H. Yang, and W. Chu, “Extracting Reusable Object-Oriented Legacy Code Segments with Combined Formal Concept Analysis and Slicing Techniques for Service Integration”, *IEEE Proceedings of 6th International Conference of Software Quality (QSIC)*, pp. 385–392, 2006.
- [133] A. V. Zhdanova, J. Zoric, M. Marengo, H. van Kranenburg, N. Snoeck, M. Sutterer, C. Räck, O. Droegehorn, and S. Arbanowski, “Context Acquisition, Representation and Employment in Mobile Service Platforms”, *Proceedings of 15th IST Mobile & Wireless Communications Summit* 2006.
- [134] Y. Zou and K. Kontogiannis, “Migration to Object Oriented Platforms: A State Transformation Approach”, *In ICSM Proceedings of the International Conference on Software Maintenance (ICSM’02)*, pp. 530– 539, 2002.

Appendix A Prototype Implementation of ContXFS and Its Test Samples

This section presents a prototype implementation of ContXFS and the test samples of using this library. This agent-based ContXFS implementation is inspired by ‘A simpler F# MailboxProcessor¹’. Since it is a demonstration of ways in implementing a potential system, it does not cover all the components necessary to build a whole context-aware Web services application.

```
namespace MyPhDThesis
module ContXFS =

    //Messages for Control Purpose
    type internal ControlMessage<'T, 'State> =
        | Continue
        | Stop
        | Restart
        | GetControlState of 'State
        | SetControlState of AsyncReplyChannel<'State>
        | SetAgentHandler of ('T -> 'State -> 'State)

    //Messages
    type internal Message<'T, 'State> =
        | UserMsg of 'T
        | ControlMsg of ControlMessage<'T, 'State>
        | GetUserState of 'T
        | SetUserState of AsyncReplyChannel<'T>

    //Operations when errors occur
    type AfterError<'State> =
        | ContinueProcessing of 'State
        | StopProcessing
        | RestartProcessing
```

¹ <http://blogs.msdn.com/b/lucabol/>

```

//MailboxProcessor extension
type MailboxProcessor<T> with
  //Construct message-passing state agents
  static member SpawnAgent<'State>(messageHandler: 'T -> 'State -> 'State,
                                   initialState: 'State,
                                   ?timeout: 'State -> int,
                                   ?timeoutHandler: 'State -> AfterError<'State>,
                                   ?errorHandler: exn -> 'T option -> 'State -
    > AfterError<'State>
                                   ) : MailboxProcessor<'T> =
  //Initialise the optional arguments
  let timeout = defaultArg timeout (fun _ -> -1)
  let timeoutHandler = defaultArg timeoutHandler (fun state -
    > ContinueProcessing(state))
  let errorHandler = defaultArg errorHandler (fun _ _ state -
    > ContinueProcessing(state))

  //Wrap MailboxProcessor
  MailboxProcessor.Start(fun agent ->
    let rec loop(state) = async {
      let! controlMsg = agent.TryScan((fun msg -
    > if (msg.GetType().IsAssignableFrom(typeof<ControlMessage<_,_>>)) then Some (
      async.Return msg)
      else None), 0)

      match controlMsg with
      | Some m -> return! loop(state)
      | None -> return! loopAll(state)
    }
    and loopAll(state) = async {
      let! userMsg = agent.TryReceive(timeout(state))
      try
        match userMsg with
        //If timeout, timeoutHandler is called according to error types
        | None ->
          match timeoutHandler(state) with
          | ContinueProcessing(newState) -> return! loop(newState)
          | StopProcessing -> return ()
          | RestartProcessing -> return! loop(initialState)
        //If successful, handler the message
        | Some m -> return! loop(messageHandler m state)
      with
        //If exception is thrown, errorhandler is invoked
        | ex -> match errorHandler ex userMsg state with
          | ContinueProcessing(newState) -> return! loop(newState)
          | StopProcessing -> return ()
          | RestartProcessing -> return! loop(initialState)
    }
    loop(initialState)
  )

```

```

    )
    //Construct stateless agents, i.e.,workers
    static member SpawnWorker(messageHandler, ?timeout, ?timeoutHandler, ?error
Handler) =
        let timeout = defaultArg timeout (fun _ -> -1)
        let timeoutHandler = defaultArg timeoutHandler (fun _ -
> ContinueProcessing ())
        let errorHandler = defaultArg errorHandler (fun _ _ -> ContinueProcessing ())
        MailboxProcessor.SpawnAgent((fun msg _ -
> messageHandler msg; ()), (), timeout, timeoutHandler,
        (fun ex msg _ -> errorHandler ex msg))

    //Construct worker agents for parallel computing
    static member SpawnParallelWorker(messageHandler, workerNums, ?timeout, ?t
imeoutHandler, ?errorHandler) =
        let timeout = defaultArg timeout (fun _ -> -1)
        let timeoutHandler = defaultArg timeoutHandler (fun _ -
> ContinueProcessing ())
        let errorHandler = defaultArg errorHandler (fun _ _ -> ContinueProcessing ())
        MailboxProcessor.SpawnAgent((fun msg (agentWorkers: array<MailboxProce
ssor<_>>, index) ->
            agentWorkers.[index].Post msg
            (agentWorkers, (index+1) % workerNums)),
            (Array.init workerNums (fun _ -
> MailboxProcessor<_>.SpawnWorker(messageHandler, timeout, timeoutHandler, err
orHandler)), 0))

    //Facilitate agent Post method
    let public (<-->) (a:MailboxProcessor<_>) msg = a.Post msg

```

Test1 sample is to demonstrate the ways of building other agents by using the extended static method:

```

module Test1 =
    open ContXFS

    //An abbreviation for MailboxProcessor
    type Agent<T> = MailboxProcessor<T>

    //Messages for agent to process
    type internal Message = MultiplePlus of int * int | AsyncGetContent of AsyncReply
Channel<int> | Stop | Restart (*control messages and user's messages mix up*)
    //Define an F# exception type
    exception Exp

```

```
//Use extended static member SpawnAgent to create a new agent
type NumberAgent() =
  let counter = MailboxProcessor.SpawnAgent((fun msg n ->
    //process message accordingly
    match msg with
    | MultiplePlus (m, p) -> (m*p)+n
    | Stop -> raise (Exp)
    | Restart -> raise (Exp)
    | AsyncGetContent reply ->
      do reply.Reply n
      n
    ),
    0, (*initial state*)
    (fun s -> if s=8 then 1000 else -1), (*timeout condition*)
    (fun _ -> printfn "Restar"; RestartProcessing), (*handler timeout*)
    (fun _ _ -> printfn "Stop"; StopProcessing)) (*handler error*))

//Use agent counter to build NumberAgent object
member a.MultiplePlus n = counter.Post(MultiplePlus n)
member a.Stop() = counter.Post(Stop)
member a.Restart() = counter.Post(Restart)
member a.AsyncGetContent() = counter.PostAndReply(fun reply -
> AsyncGetContent reply)

//Create an NumberAgent()
let counter' = NumberAgent()
//(1*2)+0
counter'.MultiplePlus(1,2)
//2
counter'.AsyncGetContent()
//(2*3)+2
counter'.MultiplePlus(2,3)
//8, then Restar is printed as the timeout condition is fulfilled
counter'.AsyncGetContent()
//Stop
counter'.Stop()
//Restart
counter'.Restart()
```

Test2 sample takes the code example from the F# research website [45] and the code sample is rewritten based on ContXFS.

```

module Test2 =
    open System.Xml.Linq
    open ContXFS

    type Agent<'T> = MailboxProcessor<'T>

    exception Exp

    //ChatMessage for agent to process
    type internal ChatMessage =
        | SendMessage of string
        | GetMessage of AsyncReplyChannel<string>

    //Create a new agent
    type ChatRoom() =
        //Only messagehandler and initial state are given
        let agent = Agent.SpawnAgent((fun msgs lst ->
            match msgs with
            | SendMessage m ->
                let m = XElement(XName.Get("li"), msgs)
                m :: msgs
            | GetMessage reply ->
                let html = XElement(XName.Get("ul"), msgs)
                do reply.Reply(html.ToString())

            msgs), [ ] (*other handlers can be implemented here*))
        )

    //Build members via delegation
    member x.SendMessage(msg) = agent.Post(SendMessage msg)
    member x.AsyncGetMessage(?timeout) = agent.PostAndAsyncReply(GetMessage e, ?timeout=timeout)
    member x.GetMessage() = agent.PostAndReply(GetMessage)
    //Asynchronously get messages without cancellationToken
    member x.GetMessageAsync() =
        Async.StartAsTask(agent.PostAndAsyncReply(GetMessage))
    //Asynchronously get messages with cancellationToken
    member x.GetContentAsync(cancellationToken) =
        Async.StartAsTask(agent.PostAndAsyncReply(GetMessage), cancellationToken=cancellationToken)
    
```


Appendix B List of Publications

- [1] Jianchu Huang and Hongji Yang, “A Functional Implementation Approach for Web Services-based Context-Aware Systems”, *IEEE 36th Annual Computer Software and Applications Conference Workshops*, July 2012.
- [2] Jianchu Huang, Hongji Yang, Lei Xu, Baowen Xu, and He Zhang, “Supporting Context-Aware Service Evolution with A Process Management Requirements Model”, *Knowledge and Service Technology for Life, Environment, and Sustainability*, International Workshop, SOCA 2011.
- [3] Jianchu Huang, Hongji Yang, and Lei Liu, “Reconciling Requirements and Implementation via Reengineering for Context-Aware Service Evolution”, *IEEE 35th Annual Computer Software and Applications Conference Workshops*, pp.464–469, July 2011.
- [4] Jian Kang, Jianzhi Li, Jianchu Huang, Yingchun Tian, and Hongji Yang, “Automating Business Intelligence Recovery from a Web-based System”, *21st International Conference on Software Engineering and Knowledge Engineering*, pp. 262-267, July 2009.
- [5] Jian Kang, Jianjun Pu, Jianchu Huang, Zihou Zhou, and Hongji Yang, “Business Intelligence Recovery from Legacy Code”, *IEEE 32nd Annual Computer Software and Applications Conference*, pp. 765-770, July/August 2008.