A REUSABLE APPLICATION FRAMEWORK FOR CONTEXT-AWARE MOBILE PATIENT MONITORING SYSTEMS

MAHMOOD GHALEB MAHMOOD AL-BASHAYREH

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2014

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Pembangunan Sistem Pemantauan Konteks Sedar Pesakit Mudah Alih (CaMPaMS) menggunakan sensor tanpa wayar adalah sangat kompleks. Untuk mengatasi masalah ini, Rangka Kerja Pemantauan Konteks Sedar Pesakit Mudah Alih (CaMPaMF) telah diperkenalkan sebagai satu teknik yang sesuai untuk meningkatkan kualiti keseluruhan pembangunan dan mengatasi kerumitan pembangunan CaMPaMS. Walaupun terdapat beberapa kajian yang mereka bentuk CaMPaMF yang boleh digunakan semula, masih belum ada lagi kajian yang memfokus kepada bagaimana mereka bentuk dan menilai rangka kerja aplikasi berdasarkan aspek kebolehgunaan semula berganda dan menggunakan pendekatan penilaian kebolehgunaan semula berganda. Tambahan pula, tiada kajian yang mengintegrasikan kesemua keperluan domain CaMPaMS. Oleh itu, tujuan kajian ini adalah untuk mereka bentuk CaMPaMF yang boleh digunakan semula untuk CaMPaMS. Untuk mencapai matlamat ini, dua belas kaedah telah digunakan: kajian literatur, analisis kandungan, matriks konsep, pemodelan ciri, penggunaan pelbagai kes, kajian pakar domain, model yang berasaskan pendekatan senibina, analisis kod statik, pendekatan model kebolehgunaan semula dan prototaip, pengiraan jumlah nilai penggunaan semula, dan kajian pakar perisian. Hasil utama kajian ini adalah CaMPaMF boleh digunakan semula yang direka bentuk dan dinilai agar ia mengandungi pelbagai aspek keboleh gunaan semula. CaMPaMF terdiri daripada model domain yang disahkan oleh doktor pakar runding sebagai pakar domain, model seni bina, model platform bebas, model platform khusus yang disahkan oleh pakar perisian, dan tiga prototaip CaMPaMS untuk memantau pesakit tekanan darah tinggi, sawan, atau penyakit kencing manis, dan pelbagai pendekatan penilaian kebolehguna semula. Kajian ini menyumbang kepada badan pengetahuan kejuruteraan perisian, terutamanya dalam bidang mereka bentuk dan menilai rangka kerja aplikasi yang boleh digunakan semula. Penyelidik boleh menggunakan model domain untuk meningkatkan kefahaman tentang kehendak domain CaMPaMS, sekali gus diperluaskan dengan keperluan baharu. Pembangun juga boleh menggunakan semula dan memperluaskan CaMPaMF untuk membangunkan pelbagai CaMPaMS untuk penyakit yang berbeza. Industri perisian juga boleh menggunakan semula CaMPaMF untuk mengurangkan keperluan untuk berunding dengan pakar domain dan mengurangkan masa pembangunan CaMPaMS.

Kata kunci: Rangka kerja aplikasi guna semula, Penilaian kerangka kerja aplikasi kebolehgunaan semula berganda, Aspek kebolehgunaan semula berganda, Sistem pemantauan pesakit mudah alih

Abstract

The development of Context-aware Mobile Patient Monitoring Systems (CaMPaMS) using wireless sensors is very complex. To overcome this problem, the Context-aware Mobile Patient Monitoring Framework (CaMPaMF) was introduced as an ideal reuse technique to enhance the overall development quality and overcome the development complexity of CaMPaMS. While a few studies have designed reusable CaMPaMFs, there has not been enough study looking at how to design and evaluate application frameworks based on multiple reusability aspects and multiple reusability evaluation approaches. Furthermore, there also has not been enough study that integrates the identified domain requirements of CaMPaMS. Therefore, the aim of this research is to design a reusable CaMPaMF for CaMPaMS. To achieve this aim, twelve methods were used: literature search, content analysis, concept matrix, feature modelling, use case assortment, domain expert review, model-driven architecture approach, static code analysis, reusability model approach, prototyping, amount of reuse calculation, and software expert review. The primary outcome of this research is a reusable CaMPaMF designed and evaluated to capture reusability from different aspects. CaMPaMF includes a domain model validated by consultant physicians as domain experts, an architectural model, a platform-independent model, a platform-specific model validated by software expert review, and three CaMPaMS prototypes for monitoring patients with hypertension, epilepsy, or diabetes, and multiple reusability evaluation approaches. This research contributes to the body of software engineering knowledge, particularly in the area of design and evaluation of reusable application frameworks. Researchers can use the domain model to enhance the understanding of CaMPaMS domain requirements, thus extend it with new requirements. Developers can also reuse and extend CaMPaMF to develop various CaMPaMS for different diseases. Software industries can also reuse CaMPaMF to reduce the need to consult domain experts and the time required to build CaMPaMS from scratch, thus reducing the development cost and time.

Keywords: Reusable application framework, Multiple reusability evaluation approaches, Multiple reusability aspects, Mobile patient monitoring systems

Acknowledgement

All thanks and praises are due to Allah, who provides me with the substance, time, health, strength of mind, and patience to engage in this journey to acquire knowledge.

I would like to express my deepest appreciation and gratitude to my supervisor Dr Nor Laily Hashim for her continuous guidance and support from start to finish. I offer a special thank you for her motivation and support in helping me to publish my work. It has been a great pleasure to work under her supervision.

I would like to record special thanks to my father Galeb and my mother Sabah. Words fail to express my appreciation to them and without their unconditional prayers, support, and love I could not have completed, or even started, my study. My Lord, have mercy upon them as they have taught me a lot and to be whom I am today.

Praise be to Allah again, who blessed me with my wife Ola and my son Abdul Rahman. Thank you Ola for your patience, understanding, and for your real trust in my capabilities. Thank you for your continuous motivation, support, and love. Thank you Abdul Rahman for your smile that gave me the strength to keep going no matter what I faced during my study.

I would like also to thank my brothers Malek and Fares as well as my sisters Ala' and Eman. Thank you for your love, support, prayers, and for everything you did for me.

Finally, thank you for all of you for being the reasons to realize this dream. I am indebted to all of you more than you know.

Mahmood Ghaleb Al-Bashayreh
August 2013

Dedication

To my mother Sabah and my father Gheleb

To my dear wife Ola

To my dear son AbdulRahman

To my brothers, Malek & Fares and my sisters Ala' and Eman

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BP	Blood Pressure
BT	Body Temperature
CaMPaMF	Context-aware Mobile Patient Monitoring Framework
CCL	Context Characterization Layer
CIM	Computation Independent Model
CML	Context Monitoring Layer
CaMPaMS	Context-aware Mobile Patient Monitoring Systems
DIP	Dependency Inversion Principle
DRM	Design Research Methodology
ECG	Electrocardiogram
FODA	Feature-Oriented Domain Analysis
HR	Heart Rate
ISP	Interface-Segregation Principle
JMA	Jordan Medical Association
LSP	Liskov Substitution Principle
MDA	Model Driven Architecture
MDD	Model Driven Development
MDRE	Model Driven Requirement Engineering
MPMS	Mobile Patient Monitoring Systems
OCP	Open-Closed Principle
PIM	Platform Independent Model
PMS	Patient Monitoring Systems
PSM	Platform Specific Model
RR	Respiration Rate
SRP	Single Responsibility Principle
UML	Unified Modelling Language
WBS	Wireless Body Sensors
WHO	World Health Organisation

CHAPTER ONE
INTRODUCTION

1.1. Overview

This chapter introduces the research that is presented in this thesis. The research background and motivation is described, followed by a presentation of the research problem, the research questions, and the objectives, scope and framework of the research, along with its significance. Finally, this chapter presents an outline of the whole thesis.

1.2. Research Background and Motivation

Reuse-based software engineering is a development approach that increases the reuse of existing software [1]. Software reuse is one of the fundamental software engineering concepts [2] and one of the most commonly used principles to simplify application development and overcome development complexities. Reusing software reduces the number of software assets that need to be developed and reuses well-tested assets that have been used in many systems with minimal errors. Moreover, software reuse encapsulates the knowledge of specialists [3-5].

According to [6], identifying the aspects that affect software reusability can enhance the knowledge required to build a reusable software components and identify the potential of reusing existing software modules in new a software development. Therefore, it is important to identify the aspects that can affect software reusability.

The literature lists several reusability aspects that can affect software reusability [7, 8]. The following aspects need to be taken into account when reusing software: (1) design guidelines (e.g. naming guidelines, extensibility guidelines, using design patterns) [9]; (2) design rules (e.g. complexity, coupling, cohesiveness) [10]; (3) design principles (e.g. modularity, simplicity, abstraction) [10]; (4) reusability factors (e.g. flexibility and understandability) [10]; and (5) amount of reuse (e.g. reuse level, reuse frequency, reuse size) [11].

Reusability refers to the potential of an artefact for reuse [12]. According to [6], reusability can be evaluated in different ways. First, the applicability of design guidelines [9, 10] is evaluated as they provide a common language for communication between the artefact authors and the artefact users, thus determining the artefact's reusability [9, 10]. Second, reusability models [7, 13, 14] are used to represent the relations between design rules, design principles, and reusability factors that can affect software reusability [7, 8]. Third, prototyping is carried out as a proof of concept towards illustrating the artefact's reusability [15-17]. Fourth, the amount of artefact that can be reused is calculated [18]. Fifth, expert review is used to confirm the artefact's reusability in terms of the three key reusability aspects: design rules, design principles, and factors that can affect software reusability [19]. However, unlike using a single evaluation approach that assesses reusability based on a single reusability aspect, multiple reusability evaluation approaches can be used to complement each other by depicting different reusability aspects to provide a complete picture of the reusability [6]. Consequently, multiple reusability evaluation approaches can be used to evaluate whether a software design satisfies reusability aspects [6], thus providing a comprehensive evaluation results about reusability.

Many software reuse approaches have been developed in the literature [1]. However, an application framework is a core software engineering reuse approach [2, 20] and one of the most suitable solutions for simplifying application development and overcoming development complexities [2, 21, 22]. Compared to the other reuse techniques, an application framework is an ideal reuse technique because it is able to achieve maximum large-scale reuse [23]. Additionally, it provides a suitable solution to address business activities in a family of related applications in a specific domain [23].

An application framework is an approach for reusing both architecture and code [20, 23, 24]. Application frameworks benefit application development and enhance overall software development quality [25, 26]. For example, using application frameworks reduces development time [27], efforts [28], and cost [29]. Similarly, it decreases the lines of code [27, 30], increases developer productivity [31, 32], and reduces maintenance efforts [33].

Application frameworks can be used to develop a family of software systems [23, 31, 34] by capturing their domain requirements (reusable requirements) [35-37]. For example, frameworks can be used to develop clinical decision-support systems [38], electronic health record systems [39], and patient monitoring systems [40]. Moreover, they can be used to develop domain-specific applications [2, 28, 41] by identifying the domain concepts from domain sources such as developed applications in the literature and domain experts [35, 36, 42]. Frameworks can be used to develop applications in the business domain [2], manufacturing domain [43], learning domain [44], and biomedical informatics domain [45, 46]. Furthermore, they can adopt various

development approaches to support both architecture and code reuse. For example, frameworks can adopt Model Driven Architecture (MDA) [47], design pattern [48], or component-based approaches [29]. Additionally, they can be designed for either desktop platforms [46, 49, 50] or mobile platforms such as smartphone [48, 51, 52].

According to a 2009 World Health Organization (WHO) report, the increase in elderly populations worldwide correlates with an increase in chronic diseases such as hypertension, diabetes and epilepsy [53]. The elderly, especially those who suffer from chronic diseases, need continuous healthcare services. This need has increased the demand on healthcare services, which has in turn increased the cost of these services [54]. Similarly, this need has increased the load on healthcare professionals and has led to a reduction in the quality of services provided by healthcare organizations [40]. These challenges have led to healthcare professionals suggesting new methods to provide care for the elderly and to manage chronic diseases [55]. One of these methods involves the elderly and chronic disease patients monitoring themselves during their daily life [56, 57].

To enable elderly and chronic disease patients to monitor themselves, researchers have developed personal lifetime health monitoring systems [40, 58] known as Patient Monitoring Systems (PMS). Elderly and chronic disease patients often need to be monitored continuously, long term, anywhere, anytime as they go about their daily lives [40, 59, 60]. The emergence of mobile devices and wireless sensor technologies has inspired researchers to study the potential of adapting the existing systems to develop Mobile Patient Monitoring Systems (MPMS) using wireless sensors [61, 62]. Mobile devices, such as smartphones, provide a platform to develop MPMS [63, 64]

and can act as a base unit to collect biomedical data, such as vital signs, from wireless sensors [45, 65]. Wireless sensors such as Wireless Body Sensors (WBS) can be used to monitor patients' vital signs such as blood pressure (BP) and body temperature (BT), as well as monitor patient physical activities such as walking and running [45, 55]. Additionally, wireless environmental sensors can be used to monitor the surrounding environmental conditions that affect the patients such as room temperature, humidity, lighting level, and location [49, 66].

In fact, MPMS can play a key role in monitoring patients' responses to any medication [67], as well as the management and protection from the complications of chronic diseases [60]. These systems continuously perform repeatable tasks that are required to monitor patients and complement the role of healthcare professionals outside the boundary of healthcare organizations [60]. However, monitoring patients outside the boundary of healthcare organizations has introduced the need to identify the patient's context while they are being monitored, and this requirement poses a new challenge for MPMS [68, 69].

Identifying patient context based on patient context information enables effective characterization of patients' medical situation and allows MPMS to adapt to changes in that situation [70]. An example of such adaptability is raising an alarm or contacting healthcare professionals once a critical medical situation is detected [49, 68, 69]. However, despite the benefit of identifying the patient context, developing Context-aware Mobile Patient Monitoring Systems (CaMPaMS) using wireless sensors is very complex [46, 71-73].

Furthermore, the literature related to the domain requirements of PMS is fragmented yet. According to [70, 74, 75], there are five identified domain requirements that should be addressed in the design of PMS, which are: (1) support anywhere, anytime patient monitoring; (2) support real-time continuous monitoring; (3) support unlimited number of sensors at design time; (4) design to be hosted and executed completely on a mobile platform; (5) adopt context awareness computing by addressing measurable medical context, non-measurable medical context, risk factors medical context, prescribed medications medical context, environmental context, and physical activities context, using WBS, wireless environmental sensors, patient profile hosted on the patient's mobile device, mobile graphical user interface and rule-based reasoning approach. Nevertheless, an analysis of 20 previous studies that designed PMS reveals that none of these studies have addressed all these identified domain requirements [70, 74, 75].

Consequently, application frameworks have been developed as a suitable solution to encapsulate domain requirements, enhance the overall development quality and overcome the development complexity of CaMPaMS [74, 75]. However, the literature related to these emerging application frameworks remains fragmented. In addition, there is a recognized need to enhance the design of these application frameworks [74, 75], with more emphasis on reusability, as reusability is the most important quality goal for application frameworks [10, 23, 27] and domain requirements, as they encapsulate the business activities in a family of related applications in a specific domain [23].

According to [75], there is a gap in evaluating application frameworks reusability in the field of biomedical informatics. Only 30% of the designed frameworks were evaluated in terms of reusability while others are either evaluated in terms of functionality or other quality attributes such as performance and usability. In spite of the fact that the authors of these studies claim that their designed frameworks are reusable, they only used the prototyping approach to evaluate reusability. Using single evaluation approach provides reusability evaluation results from one perspective. In other words, there is no clear mapping between adopted reusability aspects and their corresponding reusability evaluation approaches, thus, it provides immature reusability evaluation results.

Other than that, Al-Bashayreh et al. [74, 75] revealed from their studies that there is a severe lack in considering the domain requirements of PMS in the designed application frameworks in the field of biomedical informatics. Among these application frameworks, 80% were designed to support anywhere, anytime patient monitoring [45, 61, 67, 76]; real-time continuous monitoring [45, 49, 61, 76]; and an unlimited number of sensors at design time [45, 49, 67, 76]. Additionally, 60% were designed to support an unlimited number of monitoring applications at design time [45, 67, 76] and 40% were designed by adopting context awareness computing [45, 49]. All of the frameworks were designed to address measurable medical context as a context information type; environmental context as a context information type; WBS as a data source of context information; wireless environmental sensors as a data source of context information; and rule-based reasoning as a context reasoning approach. Additionally, 50% of them [45] were designed to address a risk factors medical context as a context information type; a prescribed medications medical context as a context

information type; a physical activities context as a context information type; and a patient profile as a data source of context information. Moreover, none of the frameworks was designed to address a non-measurable medical context as a context information type; the mobile graphical user interface as a data source of context information; or hosting the patient profile on the patient's mobile device. Lastly, none of the five application frameworks designed to develop PMS were designed to be hosted and executed completely on a mobile platform. As a result, these designed application frameworks failed to achieve the primary goal of application frameworks that is to encapsulate business activities of PMS family in biomedical informatics domain. In other words, reusing these frameworks is not beneficial, because it will not substitute the need to consult domain experts and it will not reduce the time required to build PMS from scratch, thus it will not reduce the development cost and time.

1.3. Research Problem

Eleven studies have designed application frameworks in the field of biomedical informatics [45, 48, 49, 51, 52, 61, 67, 71, 76-78]. However, there are gaps in the design, domain requirements, and the evaluation of these application frameworks.

First, of these application frameworks only 45% were evaluated in terms of their reusability [48, 51, 67, 77, 78]. Although the authors of these studies claim that their frameworks are reusable, they only used the prototyping approach to evaluate reusability. Moreover, only two of them considered the design guidelines reusability aspect of their design patterns [9]. Neither of these frameworks was designed with multiple reusability aspects or evaluated using multiple reusability evaluation approaches. Using multiple reusability evaluation approaches can provide a clear

mapping between adopted reusability aspects and their corresponding reusability evaluation approaches. Thus, it provides a useful reusability evaluation results that can be used later to enhance the framework reusability by focusing on a particular reusability aspect based on its corresponding reusability evaluation approach.

Second, it was found that only 45% of these application frameworks were designed to develop PMS [45, 49, 61, 67, 76]. These application frameworks have failed to address all the following identified domain requirements [45, 70, 74, 75]: (1) support anywhere, anytime patient monitoring; (2) support real-time continuous monitoring; (3) support unlimited number of sensors at design time; (4) support unlimited number of monitoring applications at design time; (5) design to be hosted and executed completely on a mobile platform; (6) adopt context awareness computing by addressing measurable medical context, non-measurable medical context, risk factors medical context, prescribed medications medical context, environmental context, and physical activities context, using WBS, wireless environmental sensors, patient profile hosted on the patient's mobile device, mobile graphical user interface and rule-based reasoning approach. Addressing all these domain requirements will satisfy the primary goal of application frameworks that is to address business activities in a family of related applications in a specific domain [23]. Additionally, an application framework that address all these domain requirements can be reused as is or with minimal need for extensions to develop various CaMPaMS for different diseases. It also reduces the need to consult domain experts and the development time.

1.3.1. Statement of Problem

The above analysis has identified that there is no existing application framework in biomedical informatics that was designed using multiple reusability aspect and evaluated based on multiple reusability evaluation approaches. Furthermore, there is no existing application framework for CaMPaMS that integrates all of the identified domain requirements. This research therefore investigates reuse-based software engineering, application frameworks design, context awareness computing, and PMS to bridge the gap between: (1) application frameworks in biomedical informatics with reusability aspects and evaluation approaches; (2) application frameworks and the domain requirements of CaMPaMS. It explores how to design and evaluate a reusable application framework based on multiple reusability aspects and multiple evaluation approaches to ensure application framework reusability from different aspects and to help developers to develop various CaMPaMS for various diseases.

1.4. Research Questions

On the basis of the arguments presented thus far in this chapter, this research is designed to address the following main research question:

- How to design a reusable application framework for CaMPaMS?

To address this issue, the following research subquestions need to be answered:

1. What are the domain requirements of CaMPaMS that should be addressed by an application framework?
2. How can a reusable application framework for CaMPaMS be designed?
3. How can the reusability of the designed application framework be evaluated?

1.5. Research Objectives

The aim of this research is to design a reusable application framework for CaMPaMS. To achieve this, the following objectives need to be satisfied:

1. Develop the domain model of CaMPaMS that should be addressed by an application framework.
2. Design a reusable application framework for CaMPaMS by applying multiple reusability aspects.
3. Evaluate the reusability of the designed application framework using multiple reusability evaluation approaches.

1.6. Research Scope

The scope of this research is confined to the design of a reusable application framework. To achieve this, a literature review was performed to identify the domain requirements that need to be integrated into the design of an application framework. These domain requirements were then used to develop a domain model to design an application framework. Both the application framework and the CaMPaMS were hosted on a mobile phone. Furthermore, only three CaMPaMS prototypes for monitoring patients with hypertension, epilepsy and diabetes were used in this research to demonstrate the reusability of the application framework. This is because hypertension, epilepsy and diabetes are among the chronic diseases that are the leading causes of death around the world [53].

1.7. Research Framework

Figure 1.1 shows the overall research framework. The columns map the research questions with objectives, processes and steps, outcomes, and the validation. As the figure shows, the first outcome is the research agenda. The research agenda identifies the lacks and gaps in the literature in order to clarify the research problem and provide a foundation for researchers to extend the state of the art by bridging the gaps between application frameworks and the domain requirements of CaMPaMS. This research agenda resulted from reviewing, analysing, and synthesizing the relevant literature published in peer-reviewed academic journals [74, 75, 79].

The second research outcome is a domain model representing the requirements of a specific domain, that is, the area of knowledge targeted by the framework [23, 24, 80] in order to satisfy the first research objective. The domain model, discussed in Chapter 4, includes a feature model and an abstract use case model. The feature model is represented by common and variable features that are captured from a family of applications in a specific domain [36, 81, 82]. These common features are shared among all applications built using the framework, while the variable features represent the flexibility points of software frameworks that have to be extended to meet application-specific needs [83-85]. The abstract use case model presents the system boundary that embodies the system's abstract use cases. In addition, it captures the interactions between the application framework and its actors, which are the CaMPaMS that benefit from the use of this framework. The abstract use case model complements the feature model in terms of identifying the domain requirements [86]. The feature model and the abstract use case model were validated by documenting features, authoring scenarios, and expert review, and were also evaluated by experts

prior to the publication of the research in academic journals and conference proceedings [87, 88].

The third, fourth and fifth research outcomes, discussed in Chapter 5, are: (1) the architectural model; (2) Platform-Independent Model (PIM) and Platform-Specific Model (PSM); and (3) an application framework implementation. These outcomes aim to satisfy the second research objective. This is achieved by designing the application framework based on a set of reusability aspects, which are design guidelines, design rules, design principles, reusability factors, and amount of reuse. The architectural model describes the structural organization of the primary components of the proposed application framework and the relationships between them to satisfy quality attributes (non-functional requirements) [89] such as reusability. The PIM provides a valuable model that can be reused despite the rapid changes in mobile and wireless technologies and thus reduces development effort, time and cost [90, 91]. The PIM was transformed into a PSM using a C# model transformation. The PSM provides a physical model that is customized to depict the system implementation based on specific technology. C# was used to generate the PSM because it can be used to develop mobile applications that can be executed on various platforms such as Android, iOS, and Microsoft Windows Phone.

The application framework is the primary outcome of this research and is designed to enhance the overall development quality and overcome the development complexity of CaMPaMS by being designed and evaluated based on multiple reusability aspects and multiple reusability evaluation approaches, in addition to satisfying the identified domain requirements of CaMPaMS. The architectural model, PIM and PSM, and the

application framework implementation were validated by experts through publication of the research in academic journals and conference proceedings [79, 87].

The last research outcome, discussed in Chapter 6, is the testing and documenting of the application framework, which aims to ensure the reusability of the application framework to satisfy the third research objective. This is achieved via the activities of framework design guidelines application, framework reusability evaluation using reusability model, prototyping and documentation, amount of reuse calculation, and framework reusability evaluation using expert review of the software.

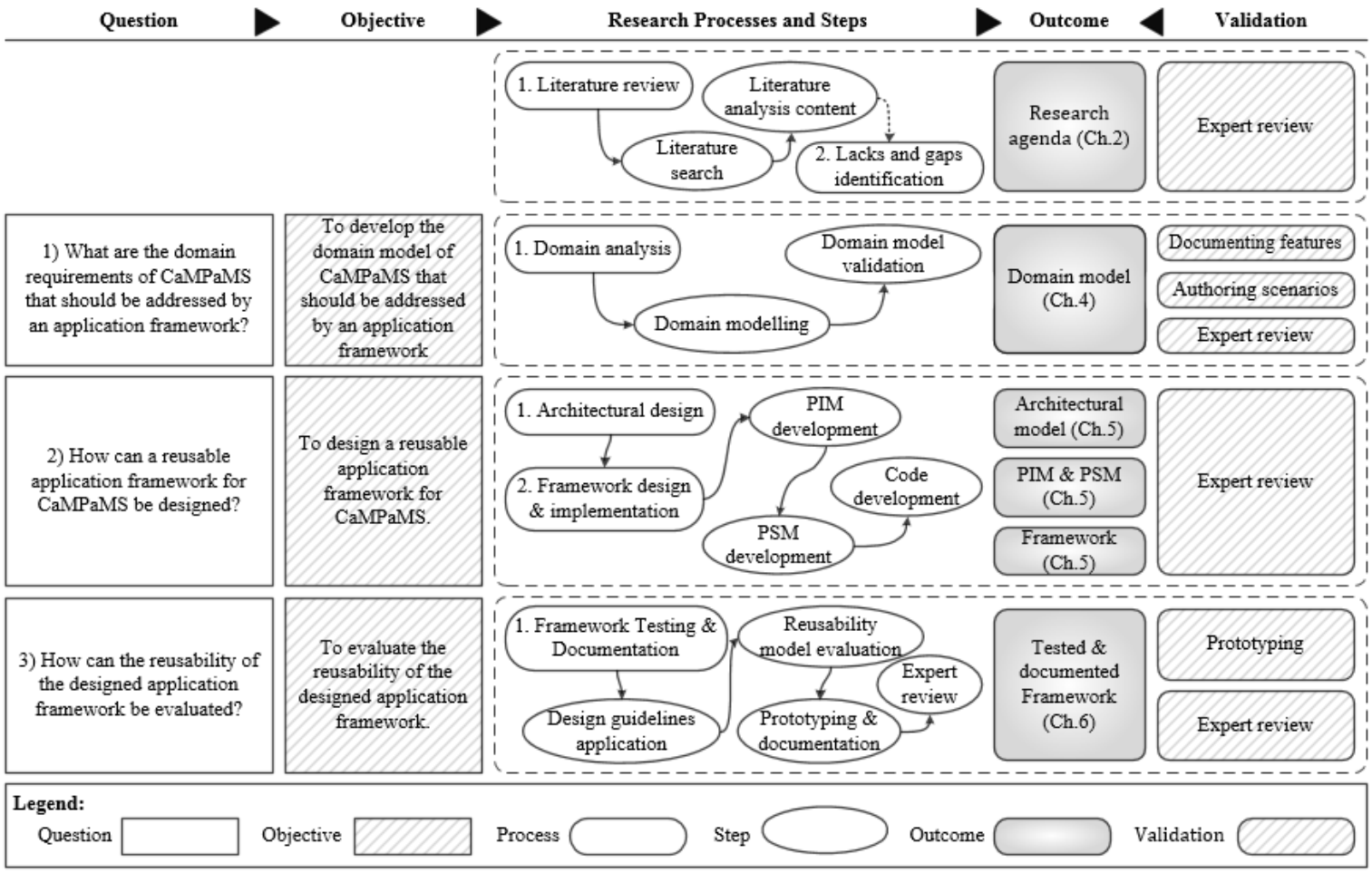

Figure 1.1. Research framework

1.8. Research Significance

This research presents a novel design for a reusable application framework for CaMPaMS. It contributes to the software engineering body of knowledge and software design, especially software structure and architecture in terms of designing reusable families of programs and frameworks [92]. The primary contribution of this research is a reusable application framework for CaMPaMS, in addition to other two contributions: an application framework reusability with multiple evaluation approaches and three CaMPaMS prototypes that were developed on top of the reusable application framework for monitoring patients with hypertension, epilepsy or diabetes.

Although many software reuse approaches have been developed in the literature [1], an application framework is a core software engineering reuse approach [2, 20] since it provides a suitable solution to address business activities for the family of CaMPaMS in the biomedical informatics domain. Consequently, there is a need to design an application framework with greater emphasis on (1) reusability, as this is the most important quality goal for application frameworks [10, 23, 27], and (2) domain requirements, which encapsulate the business activities in a family of related applications in a specific domain [23]. Furthermore, there is no existing application framework that integrates all of the identified domain requirements of CaMPaMS.

There are several aspects that can affect software reusability [7, 8]. The effects of these aspects were tested in several studies [7, 10, 13, 14, 93-95]. Therefore, it is important to consider multiple aspects when designing for reuse and in assessing software reuse [6]. Moreover, multiple reusability evaluation approaches should be used to complement each other by depicting different reusability aspects to provide a complete

picture of the reusability [6]. Currently, there is no existing application framework that was designed based on multiple reusability aspects and that was evaluated based on multiple reusability evaluation approaches.

The application framework developed in this research consists of a domain model, architectural model, PIM, PSM, and code development. Moreover, three CaMPaMS prototypes were developed on top of the application framework for monitoring patients with hypertension, epilepsy, and diabetes. As such, this research will be beneficial for a wide range of stakeholders. Researchers can use the constructed domain models, including the feature model and the abstract use case model, to enhance their understanding of designed application frameworks. In addition, they can use these models as a platform for the discovery of new requirements. Researchers can use the constructed architectural model as a solid foundation for efficient development of application framework and extend the architectural model with new emerging domain requirements. They can also use the constructed PIM to enhance the design and implementation of the application framework. Moreover, researchers can use the identified research agenda to extend the state of the art by bridging the gap between application frameworks and the domain requirements of CaMPaMS. They can also use the identified reusability aspects as a platform for the discovery of new aspects to enhance the design for reuse. Finally, researchers can use the reusability evaluation approaches to evaluate the reusability aspects. These approaches also hold significant potential for the discovery of new approaches.

Developers can use the PIM to generate one or more PSM to reflect the continuous technological changes, which reduces development effort, time and cost. Moreover,

developers can use the PSM to generate code and thus improve developers' productivity. The implementation of the application framework can also be reused to develop various CaMPaMS for different diseases, enhance the overall development quality, and overcome the development complexity of CaMPaMS.

Software industries can use the application framework to reduce the need for consulting domain experts and the time required to build CaMPaMS from scratch, thus reducing development cost and time. The contributions of this research are discussed in more detail in Section 7.3.

1.9. Thesis Outline

This thesis is divided into seven chapters. After this Introduction, Chapter 2 gives an overview of the literature on reuse-based software engineering with a focus on software frameworks, followed by an overview of the biomedical informatics domain, PMS, and the terminologies used to identify these systems. The role of mobile phone and wireless sensor technologies in these systems is elaborated. Additionally, the benefits of these systems are presented. This chapter also establishes the foundations of MPMS in the context awareness computing. Furthermore, it summarizes, synthesizes and critiques the literature that underlies this research. It also outlines a research agenda that shows the lacks and gaps in the existing application frameworks.

Chapter 3 presents the research methodology, starting by identifying this research as design research and then introducing the pragmatic research paradigm and Design Research Methodology (DRM), which is adopted in this research. The four stages of DRM are then described in detail.

The next three chapters report on the implementation of the six activities of application framework development. Chapter 4 is centred on the implementation of the first activity: domain analysis. Chapter 5 concentrates on the implementation of the second, third and fourth activities, which are architectural design, framework design, and framework implementation respectively. Chapter 6 is centred on the implementation of the fifth and sixth activities: framework testing and documentation.

Chapter 7 presents the conclusions of this research, starting with a research summary before detailing the contributions and limitations of the research. Finally, directions for future research are suggested.

CHAPTER TWO
SOFTWARE REUSE AND APPLICATION FRAMEWORKS FOR CaMPaMS

2.1. Overview

The aim of this chapter is to review the literature on reuse-based software engineering including the benefits, approaches, and evaluation. It focuses on software frameworks reuse approach, which is the primary research area of this study. Moreover, this chapter introduces the biomedical informatics domain as a body of knowledge and presents an overview of PMS that are hosted on mobile devices (e.g. a smartphone) and use wireless sensors (e.g. WBS and environmental sensors). It also establishes the foundations of MPMS in the context awareness computing. Furthermore, this chapter presents an analysis of 20 previous studies that designed software frameworks in the biomedical informatics domain or frameworks that can be applied in that domain. It also outlines a research agenda shows the lacks and gaps in the existing application frameworks, and provides a foundation to help researchers design enhanced application frameworks. Finally, a summary of the chapter is presented.

2.2. Reuse-Based Software Engineering

Software reuse has been one of the most important foci of software engineering for decades [2]. Reuse is defined in the software engineering vocabulary as "building a software system at least partly from existing pieces to perform a new application" [96]. Reuse-based software engineering is a development approach that increases the reuse of existing software [1].

There are three different sizes of software units that can be reused: (1) an application system reuse, where a whole system can be reused either by integrating it without change into other systems or by developing application families that can be customized according to specific need; (2) component reuse, where the size of components range from subsystems to single objects; (3) object and function reuse, where a software component that implements an object or single function can be reused [1].

Concept reuse is another form of reuse in software engineering and refers to the reuse of an idea rather than a software system or component [1]. It is represented with abstract notation such as a class model without any implementation details [1]. An example of concept reuse is a design pattern that can be configured and adapted to be reused in other cases [1].

Software reuse can be classified as either: (1) vertical reuse, where a reusable asset can be reused within the same area of application or domain, such as application frameworks; or (2) horizontal reuse, where a reusable asset can be reused across different areas of applications or domains, such as software libraries [97].

2.2.1. Benefits of Software Reuse

Reusing software assets brings obvious benefits to software development. Table 2.1 presents a summary of software reuse benefits.

2.2.2. Approaches of Software Reuse

To contextualize this study in the reuse landscape, it is necessary to survey the reuse approaches that have been developed over the past 20 years [1]. Table 2.2 provides a summary of each of these reuse approaches.

Table 2.1

Benefits of Software Reuse

Benefit	Explanation
Reduced development complexity	Reusing software assets reduces software development complexity by reducing the number of software assets that need to be developed with well-tested assets that have been used in many systems. Moreover, reusing software assets that encapsulate the knowledge of specialists reduces the software development complexity [3-5].
Reduced development and maintenance costs	Reusing software assets reduces software development and maintenance costs by reducing the number of components that need to be developed [1, 3].
Improved Productivity	Reusing software assets reduces the number of components that need to be developed, thus improves development productivity by reducing time and effort required for building software systems. This reduces time to market that leads to larger market share [3, 4].
Improved quality	Reusing software assets encourages investors to spend more on the software development quality, because the return of such large investment will be increased by multiple uses [4].
Increased reliability	Reusing software assets ensures well-tested software artefacts [3, 4], thus increase the reliability of the system by reducing the number of errors that can arise [3, 5, 98].
Increased dependability	Reusing software assets increases dependability by providing well-tested software assets that have been used in many systems [1].
Reduced process risk	Reusing software assets reduces process risk of error in cost estimation for new projects because the cost of existing reusable software assets is already known [1].
Effective use of specialists	Reusing software assets makes the most effective use of specialists by encapsulating their knowledge [1].
Standards compliance	Reusing software assets promotes standards compliance during software development. For example, if a menu control is reused in all systems as a user interface standard, then the system dependability will be improved because users will be more familiar with the interface [1].
Accelerated development	Reusing software assets may reduce development and validation time, which in turn accelerates system production [1].

Table 2.2

Software Reuse Approaches

Approach	Description
Architectural patterns	Standard software architectures that support common types of application systems are used as the basis of applications.
Design patterns	Generic abstractions that occur across applications are represented as design patterns showing abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards.
Application frameworks	Collections of abstract and concrete classes are adapted and extended to create application systems.
Legacy system wrapping	Legacy systems are 'wrapped' by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Service-oriented systems	Systems are developed by linking shared services, which may be externally provided.
Software product lines	An application type is generalized around a common architecture so that it can be adapted for different customers.
COTS product reuse	Systems are developed by configuring and integrating existing application systems.
ERP systems	Large-scale systems that encapsulate generic business functionality and rules are configured for an organization.
Configurable vertical applications	Generic systems are designed so that they can be configured to the needs of specific system customers.
Program libraries	Class and function libraries that implement commonly used abstractions are available for reuse.
Model-driven engineering	Software is represented as domain models and implementation independent models and code is generated from these models.
Program generators	A generator system embeds knowledge of a type of application and is used to generate systems in that domain from a user-supplied system model.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled.

Adopted from [1]

An application framework is the ideal reuse technique for this study because it captures the essence of software engineering reuse techniques [2, 20, 23] to achieve maximum [99] large-scale reuse [20, 23, 100]. For instance, a framework allows the reuse of

software design [100-102], including both architectural and non-architectural designs [103]. Architectural design (high-level design) covers all visible design decisions made by architects to meet the quality attributes and behavioural requirements of the system. Non-architectural design (detailed design), meanwhile, covers invisible design decisions made by developers, such as the selection of a specific algorithm [103]. Application framework provides a suitable solution to address business activities in a family of related applications in a specific domain [23]. In other words, the primary concepts related to identifying a patient's medical context can be abstracted using collections of interfaces and concrete classes that can be reused and extended each time a new CaMPaMS is required to be developed for a particular disease as a part of the CaMPaMS family within the biomedical informatics domain.

2.2.3. Application Framework versus Other Reuse Approaches

With reference to Table 2.2, there are three reuse approaches that have a strong relationship with application frameworks [104], which are: design patterns [99]; software components [105]; and software libraries [1]. For this reason, it is important to understand the main characteristics that distinguish application frameworks from other reuse techniques.

First, comparing application frameworks with design patterns reveals that application frameworks and patterns have different natures. A framework has a physical nature; it is instantiated in the programming language then reused as an executable artefact, while a pattern has a logical nature; it depicts design ideas and knowledge [20, 99, 106]. A pattern can only be instantiated in a programming language to give a demonstration example, and it must be instantiated each time it has to be used [99]. A

single framework, however, can realize or instantiate one or more patterns into an executable artefact [20, 99, 107]. As a result, a framework forms a larger unit of design than a pattern, and is more specialized than a pattern [99, 102, 107]. At the same time, patterns are more abstract than frameworks [99, 102] and express a proven design knowledge [99], hence they are more important for designers [102]. For this reason, it is recommended to use a large number of patterns to design and implement application frameworks [2, 23, 100]. Additionally, patterns can be used as a method to document frameworks because they provide a common vocabulary for depicting software design, helping developers to understand the framework [99, 108]. Table 2.3 shows the primary differences between application frameworks and design patterns.

Table 2.3

The Primary Differences between Application Frameworks and Design Patterns

Criteria	Application framework	Design pattern
Nature	It has a physical nature. It is reused as an executable artefact.	It has a logical nature. It depicts design ideas and knowledge.
Unit of reuse	It forms a large unit of reuse.	It forms a small unit of reuse.
Abstraction	It is more specialized.	It is more abstract.

Second, comparing application frameworks with components reveals that both have the same physical nature; unlike patterns, they can be reused as standalone executable software artefacts. However, frameworks are much more extensible than components [23, 104]. Frameworks correlate with components in a cooperation relation [23]. For example, a framework provides a context for reusing components [102, 104, 105] on a larger scale than what can be achieved by reusing individual components [100, 105, 109]. It also simplifies the development of new components by providing the specification of such components [23]. A framework is considered as a circuit board

that has empty slots into which components can be inserted according to their specifications [102]. Table 2.4 shows the primary differences between application frameworks and components.

Table 2.4

The Primary Differences between Application Frameworks and Components

Criteria	application framework	Software component
Extensibility	It is more extensible.	It is less extensible.
Unit of reuse	It forms a large unit of reuse.	It forms a small unit of reuse.

Third, comparing application frameworks with libraries reveals that inversion of control is their main distinguishing characteristic. Frameworks are active and take over control at run-time, while libraries are passive and are only executed once they are called [20, 23, 99]. An example of this concept, which is used in application frameworks, is the Hollywood principle: "Don't call us, we'll call you." Frameworks achieve the concept of inversion of control by encapsulating both control flows and object interfaces [29]. Moreover, frameworks are more specialized for a particular problem, while libraries are more general in nature [23, 41]. Table 2.5 shows the primary differences between application frameworks and libraries.

Table 2.5

The Primary Differences between Application Frameworks and Libraries

Criteria	Application framework	Software Library
Execution	It is active and takes over control at run-time.	It is passive and it is only executed once they are called
Generality	It is more specialized.	It is more general.

2.2.4. Evaluation of Software Reuse

While the term "software reuse" is used in the literature to refer to the practice of reuse itself, the term "software reusability" refers to evaluating the potential of an artefact for reuse [12]. Reusability is defined in [96] as "the degree to which an asset can be used in more than one software system, or in building other assets". Poulin [6] states that "knowing what makes software 'reusable' can help us learn how to build new reusable components and help us to identify potentially useful modules in existing programs." Therefore, it is important to identify the factors that can affect software reusability.

The literature lists several factors that can affect software reusability [7, 8]. The effects of these factors were tested using reusability models. Most reusability models are inspired by McCall's factor-criteria-metric model [110] which is a standard means of measuring reusability [111]. Based on this approach, the reusability model is constructed in a tree-like way, starting with the quality goal, i.e. reusability, which has to be quantified. This quality goal is composed of a number of factors related to the software artefact being evaluated. These factors are still abstract and have to be substantiated by a number of criteria. These criteria are easy to measure using a number of proposed metrics.

Several studies have proposed reusability models [7, 13, 14, 94, 95]. However, only two studies have addressed the special characteristics of an application framework in their proposed reusability models [10, 93]. In [93], a factors-criteria decomposition model was proposed for computing framework reusability. This model is a specialization of the REBOOT reusability model [94]. However, this framework

reusability model did not include any metrics to measure the proposed criteria, thus it was not tested.

In [10], Erni and Lewerentz extended McCall's factor-criteria-metric quality model [110] by focusing on a subset of the quality factors that are linked to reusability, as reusability is the most important quality goal for application frameworks. This model was evaluated by two case studies [10]. Furthermore, it was adopted in [112] to evaluate the framework's reusability. It replaces the criteria in the McCall's factor-criteria-metric quality model [110] with design principles at the same level of abstraction. Moreover, design rules were inserted as an additional level into the model between design principles and metrics. Furthermore, a "multi-metric" was proposed to model the design rules. A multi-metric is "a set of metrics that are all related to the same component (same granularity, e.g. a class)" [10]. By applying a multi-metric approach to one component (i.e. interface, abstract class, or class) of a framework, a more extensive idea of its quality can be obtained compared to applying a single metric as in the classical approach.

Despite the fact that the McCall's factor-criteria-metric approach [110] is widely cited in the software engineering literature, it has two main drawbacks that limit its usability [113]. First, the factor-criteria-metric approach hides the mapping of criteria onto metrics, which is based on a set of design principles and rules, behind the arrows that link the criteria to the metrics. Second, the factor-criteria-metric approach does not help to find the real causes (i.e. unsatisfied design principles and rules) that cause abnormal metric values, which are required to solve the design problems.

However, Erni and Lewerentz [10] handled these drawbacks in their proposed model by adding design principles and design rules as intermediate levels between the factor and metric levels. As shown in Figure 2.2, this model is divided into four levels: factor; design principle; design rule; and metric. On the first level, two factors are identified that affect reusability: flexibility (adaptability) and understandability. Flexibility (adaptability) is defined as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed" [96]. Understandability, meanwhile, is defined as "the ease with which a system can be comprehended at both the system-organizational and detailed-statement levels" [96]. According to [101], these are the most frequently used factors in reusability models, with flexibility (adaptability) being used in [10, 12, 93, 94, 114] and understandability in [8, 10, 93, 94, 114].

On the second level, three design principles are identified that affect flexibility and understandability: modularity; simplicity; and abstraction. Modularity is defined as "the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components" [96]. Simplicity is defined as "the degree to which a system or component has a design and implementation that is straightforward and easy to understand" [96]. Abstraction is defined as "a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information" [96].

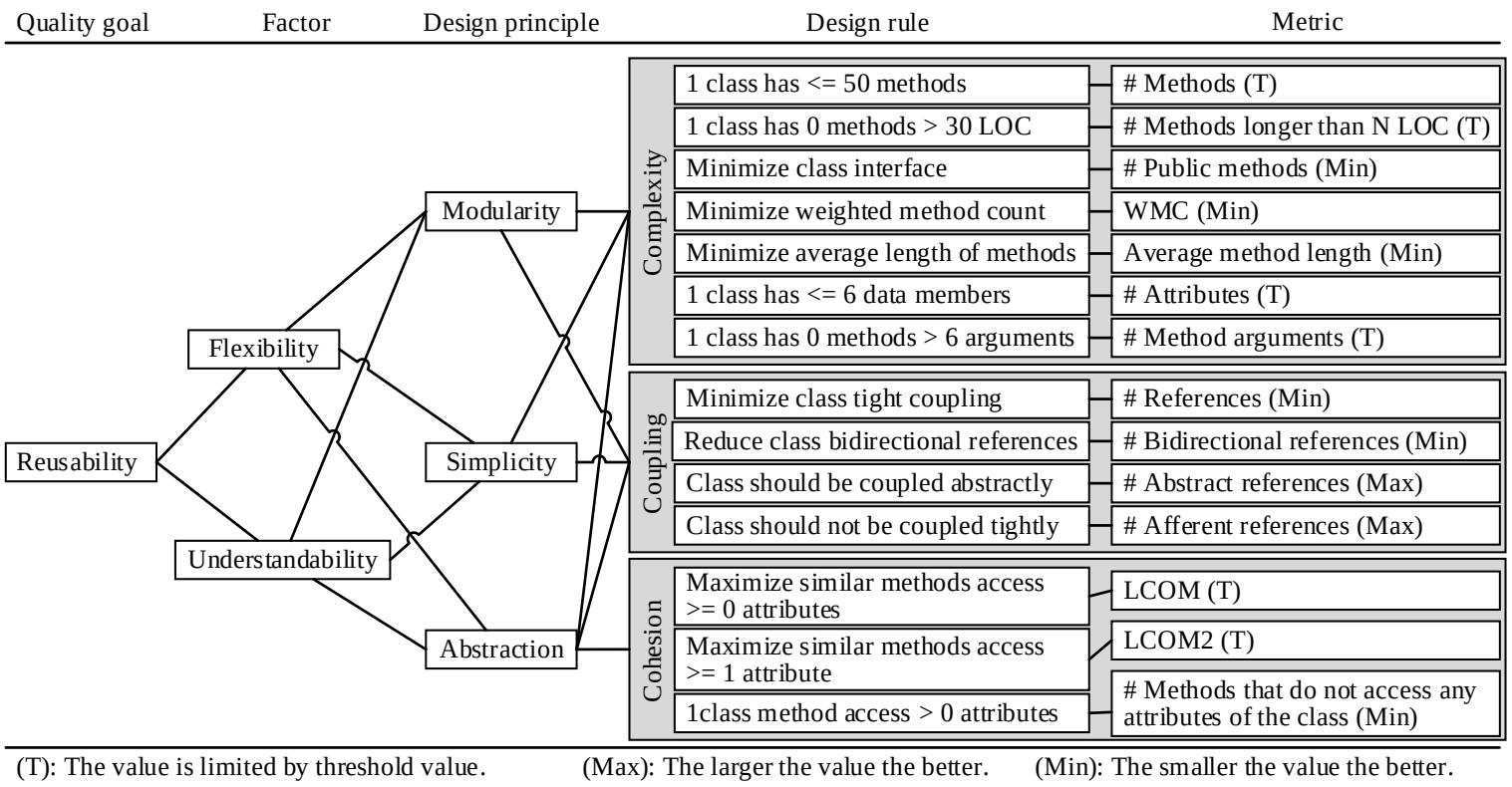

Figure 2.2. Application framework reusability model

Adopted from [71]

On the third level, three groups of design rules are identified: complexity; coupling; and cohesion. These groups include seven, four and three design rules respectively as shown in Figure 2.3. These design rules affect the design principles on the second level, as complexity and coupling affect all three design principles, while cohesion affects the abstraction design principle only.

On the fourth level, three groups of software metrics are identified: complexity; coupling; and cohesion. These groups include seven, four and three metrics respectively, which are mapped one to one onto the design rules in the second level. A metric is defined as "a quantitative indicator of an attribute of a thing" [18]. It is a numerical representation of an attribute of a measured software component (i.e. interface, abstract class, class, or method) [115]. Software metrics provide a quantitative measurement approach to verify the quality of the design of both architectural (high-level design) and non-architectural designs (detailed design) to improve the framework design and implementation [10, 115, 116]. Improving the framework design and implementation improves framework reusability [10, 115].

In the model of Erni and Lewerentz [10], metrics measurement values are interpreted based on thresholds as a set critical values (outliers) and a set of normal values: Set 1 = (Value | Value <= Threshold) and Set 2 = (Value | Value > Threshold). For each metric, a value must be assigned to a threshold which is used to identify which set contains the normal values and which set contains the outliers. In some cases the threshold can be obtained from proven design rules in the literature. However, if a threshold cannot be obtained for a particular metric in this way, it must be calculated using the average and standard deviation for the metric in the following way:

Minimum Threshold = average - standard deviation and Maximum Threshold = average + standard deviation. To determine whether the minimum or maximum threshold will be applied, the design rule related to the metric is referred to. If the design rule puts an upper limit on the metric values (e.g. “the value should not be too large”), then normal values should be small, thus the maximum threshold is the threshold. If the design rule puts a lower limit on the metric values (e.g. “the value should not be too small”), then normal values should be large and therefore the minimum threshold is the threshold.

2.3. What Is Software Frameworks?

There is a consensus among software engineering researchers that a software framework is defined as a reusable software design including both architectural and non-architectural designs as discussed in Section 2.2.3. In other words, a framework is simply an approach for reusing both architecture and code [20, 23, 24].

However, software frameworks vary based on their targeted family of applications, targeted domain, development approaches, and hosting platforms. Firstly, frameworks can be used to develop a family of software systems [23, 31, 34], which can be defined as “sets of programs that are related by sharing significant portions of requirements, design, and code” [96]. For example, frameworks can be used to develop clinical decision-support systems [38], electronic health record systems [39], and PMS [40]. Secondly, frameworks can be used to develop domain-specific applications [2, 28, 41]. Here the term “domain” describes “a bounded field of interest or knowledge” [117]. For example, frameworks can be used to develop applications in the business domain [2], manufacturing domain [43], learning domain [44], and biomedical informatics

domain [45, 46]. Thirdly, frameworks can adopt various development approaches to support both architecture and code reuse. For example, frameworks can adopt Model Driven Architecture (MDA) [47], design pattern [48], and component based approach [29]. Finally, frameworks can be designed for a specific platform. For example, frameworks can be designed for desktop platforms [46, 49, 50] or mobile platforms such as smartphone [48, 51, 52].

Therefore, software engineering researchers have introduced several framework definitions to emphasize various aspects that include targeted family of applications, targeted domain, development approaches, and hosting platforms. These definitions are not conflicting; in fact, they complement each other from different perspectives. To avoid defining framework from a specific perspective, this study adopted a standard definition from the software engineering vocabulary, which defines a framework as "a partially completed software subsystem that can be extended by appropriately instantiating some specific plug-ins" [96].

Nowadays, using software frameworks in application development is widely adopted, representing an essential part of software engineering [24, 41]. Among these well-known software frameworks are: Eclipse [118], Java Platform [119], and .NET [120].

2.4. Development of Software Frameworks

A software framework is a semi-complete application [23, 100, 121]. It provides a set of essential functionalities that application developers must tailor and extend to build complete applications [23, 34, 122]. The process of extending a framework is called

framework instantiation [23, 122] and each resulting complete application, which customizes the framework, is called a framework instance [83, 122, 123].

Frameworks consist of interfaces, concrete classes, abstract classes, or methods [24, 124] which are arranged into “frozen spots” and “hot spots” [125]. Frozen spots are concrete classes or methods that are shared among all applications that are built using the framework. The methods in frozen spots are called template methods [125, 126]. These spots do not change (they are frozen), even when the framework is instantiated by applications [124, 125]. Hot spots are interfaces, abstract classes, or methods that represent a software framework’s flexibility points that have to be instantiated by applications [24, 83, 127]. Hot spots are designed to be extended to meet application-specific needs [34, 126, 127]. The successful design of a framework depends on the adequacy of its provided hot spots [125]. The methods in hot spots are called hook methods [125, 126].

Framework instantiation is accomplished through hook methods. Hooks are the places in a framework where application developers can add their own code by extending the framework to meet an application-specific functionality. Framework developers define hooks as a means to enable application developers to use and extend software frameworks to build various applications for a specific domain [106, 123].

Framework extensibility, supported by hook methods within hot spots, is the dominant quality attribute that has to be satisfied when developing software frameworks [2, 99, 128]. A framework is considered to be useful if it is extensible [2]. Achieving extensibility ensures that a framework can be reused for developing domain-specific

applications [23, 127]. Framework extensibility techniques range from white-box to black-box techniques [23] and are based on the hook instantiation methods [125].

Framework development consists of six main activities: domain analysis, architectural design, framework design, framework implementation, framework testing, and documentation [15]. The following sub-sections describe each of these in turn.

2.4.1. Domain Analysis

Domain analysis was introduced in [129]. The term "domain" is defined as

> "an area of knowledge, scoped to maximize the satisfaction of the requirements of its stakeholders, including a set of concepts and terminology understood by practitioners in that area, and including knowledge of how to build software systems (or parts of software systems) in that area" [130].

While, "domain analysis" is defined as "a process by which information used in developing software systems in a specific domain is identified, captured, and organized with the purpose of making it reusable when creating new systems in that domain" [131].

Domain analysis aims to enhance the understanding of a domain [35, 80, 132]. In addition, it captures the domain requirements and identifies the domain concepts from domain sources such as developed applications in the literature and domain experts [35, 36, 42]. However, unlike requirement analysis that identifies the requirements of a single system, domain analysis identifies the reusable requirements for a family of systems, which are known as domain requirements [35-37].

Domain analysis is rooted in software reuse research [81, 133, 134]. It should be performed as the first activity in the software life cycle to achieve successful development and reuse [135]. Therefore, it is fundamental to support the development for reuse [36, 134, 136]. Domain analysis is essential for developing reusable frameworks, libraries, or product lines in a specific domain [133, 136]. The framework, which is a reuse technique, is considered an excellent candidate for domain analysis [42], which aims to explain the domain knowledge that is targeted by the framework [23, 24, 80]. It then identifies the domain requirements from literature, domain experts, or the existing standards for the domain [24, 80, 137]. This activity involves making improvements over a long period and therefore modelling domain knowledge is considered as an ideal approach to reduce the duration of this activity [138]. The main deliverable of this activity is a domain model [23, 80, 137], which includes the domain requirements and the relations among them [80, 137]. A domain model is defined as

> "a product of domain analysis that provides a representation of the requirements of the domain. The domain model identifies and describes the structure of data, flow of information, functions, constraints, and controls within the domain that are included in software systems in the domain. The domain model describes the commonalities and variabilities among requirements for software systems in the domain" [135].

Domain modelling can be conducted by applying the Model Driven Requirement Engineering (MDRE) approach, a flavour of Model Driven Development (MDD) methodology [139]. MDRE focuses on the visual modelling of domain requirements rather than textual description. This provides an easy way for various stakeholders to comprehend the domain requirements. In addition, it has been shown that using MDRE supports the discovery of new requirements [139]. In this approach, domain

requirements can be modelled using a number of models including a feature model and abstract use case model[139], which are described in the following sub-sections.

2.4.1.1. Feature Modelling

According to [130], the existing object-oriented analysis and design methods in the literature focus on modelling main concepts of the domain without considering feature models. The feature model provides an abstract model that captures the common and variable features as well as the interdependency of these features from a family of applications in a specific domain [36, 81, 82]. Common features and variable features can be mapped to frozen spots and hot spots in the framework design respectively [84, 85, 127].

A feature model is the primary outcome of domain modelling [82, 130, 133]. The feature model consists of a feature diagram and some additional information [140]. The feature diagram is a fundamental element of the feature model [133] that defines a set of features (domain requirements) that can be configured to meet the needs of a number of applications in a specific domain [81, 130]. The additional information can include a short semantic description about each feature and rationale for selecting each of them [140].

Feature modelling was introduced in Feature-Oriented Domain Analysis (FODA) method [141]. Since then, it has been extended with several concepts, such as feature and group cardinalities, attributes, and diagram references [140]. Later, a new feature modelling method was proposed in [130]. This modelling method is derived from FODA and is considered a powerful feature modelling method [117, 133, 142]. It has

also been used to model context-aware applications [143]. Recently, the cardinality-based notation for feature modelling was introduced in [140].

According to [140], the feature diagram organizes the identified common and variable features into a hierarchy and classifies them according to their type and cardinality. This feature diagram is represented as a tree that combines a number of nodes called features. Each feature may have at most one attribute with a specific data type and value. There are three types of feature: root feature, grouped feature, and solitary feature. A root feature is a root of the tree that represents a concept. A grouped feature appears within a feature group, while a solitary feature is not grouped within a feature group. Each grouped feature belongs to a feature group. The feature group represents a particular choice among the grouped features in the group. This choice is constrained by the group cardinality <n– n'>, which means that minimum n number of features can be selected and maximum n' of features can be selected from the grouped features within the group. If a feature group has no explicit cardinality, then its cardinality will be <1-1>.

A solitary feature is identified by its feature cardinality. The feature cardinality "is attached to the relationship between a solitary feature and its parent" [140]. Accordingly, there are three types of the solitary feature that can be identified based on their cardinality. First, the feature with cardinality [1...*] means that its parent can have one or more instances of this feature. Second, the feature with cardinality [0...1] means it is an optional feature, which is represented and marked with an empty circle. Third, a feature with cardinality [1...1] means it is a mandatory feature, which is represented and marked with a black-filled circle.

The feature modelling method supports four characteristics: (1) representing nested features using the cardinality-based approach; (2) representing common features (frozen spots); (3) representing variable features (hot spots); and (4) modelling context-aware applications. For this research, cardinality-based feature modelling is the most suitable domain analysis approach to construct a feature model to be used to design the application framework.

2.4.1.2. Abstract Use Case Modelling

An abstract use case model presents the system boundary that embodies the system's abstract use cases. In addition, it captures the internal interactions among these abstract use cases as well as the interactions between these abstract use cases and the external system actors [86]. According to [139], the abstract use case model is constructed based on the feature model. The resulting model will complement the feature model in terms of identifying the domain requirements of the application framework. Unlike the application use case model, which is specific to a particular application to support application developers, the abstract use case model includes general use cases that support many applications in a specific domain to support framework developers [86]. The focus of an abstract use case model is on capturing interactions between the application framework and its actors, which are the CaMPaMS that benefit from the use of this framework. An abstract use case model for a framework can be constructed using the use case assortment technique [86].

2.4.2. Architectural Design

Many definitions of software architecture exist and there is no consensus on a universal definition of software architecture [103]. Recently, software architecture has been defined in [103] as "the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both". In this definition, the elements are abstract or generic building blocks, while the properties are primarily a set of quality attributes such as performance, reusability, security, extensibility, and reliability. These quality attributes, which are also known as non-functional requirements [89] or quality requirements [144], are classified into two types: run-time quality attributes such as availability, performance, and security; and development-time quality attributes such as reusability and extensibility [145].

An architectural design represents a series of structural decisions that must satisfy a set of quality attributes [103]. These structural decisions can be supported by a number of structural organizations known as architectural styles [1, 146]. An architectural style

> "expresses a fundamental structural organisation schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them" [146].

Architectural styles focus on how to solve a particular problem in a specific context [103]. Catalogues of these architectural styles can be found in the pattern-oriented software architecture series [147-149] as well as in the book by Clements et al. titled Documenting Software Architectures: Views and Beyond [103].

It has been shown that using architecture styles can have both negative and positive effects on satisfying a number of quality attributes [146, 150]. In addition, it has been

found that using a particular architectural style that has a positive effect on satisfying a number of quality attributes can negatively affect other quality attributes [1, 146, 150]. For example, using a blackboard architectural style has a positive effect on satisfying maintainability, but a negative effect on satisfying testability [151]. To correct this, different architecture styles can be used to satisfy each of the quality attributes [1, 152]. For example, if both maintainability and testability are required, then the blackboard architectural style can be used for one part of the system to satisfy its maintainability, while a layered architectural style can be used for another part of the system to assure its testability [151].

The architectural design can be captured using a collection of components with a number of connectors that describe the interactions between the components [153]. According to [1], a block diagram is suitable for describing the architectural design, where each component in the architectural design can be represented as a box. A component that is deconstructed into sub-components can be represented as a box that contains a number of boxes. In addition, connectors in the architectural design can be represented as directed arrows, with the direction of the arrows representing the "allowed-to-use" relations between components [103].

Architectural design uses the constructed domain model as input to select the appropriate architectural style, which forms the foundation of the framework. The resulting architectural style is the main deliverable of this activity, the framework's architectural design [15].

2.4.3. Framework Design and Implementation

MDA is a standard approach adopted for the MDD methodology. The primary aim of the MDA approach is to support development for reuse [90, 154]. Therefore, the MDA approach is considered suitable to design and implement frameworks. MDA was introduced by Object Management Group (OMG) [155] as an industry standard to support and realize MDD [156]. MDA is defined as "an approach to information technology system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform" [157].

The MDA approach provides three viewpoints of the system: a computation-independent viewpoint; a platform-independent viewpoint; and a platform-specific viewpoint. The objective of the first viewpoint is to capture the requirements of the system using a Computation-Independent Model (CIM), which is also known as a domain model [157, 158]. The CIM plays a key role as a prerequisite of the MDA approach development activities [90]. In this research, the resulting domain model from the domain analysis activity is considered as the CIM. The CIM was used as input to the framework design and implementation activities.

The objective of the second viewpoint is to construct a high-level abstract model that is independent of any implementation technology and is therefore called a PIM. This long-lasting reusable PIM eliminates the need for redesigning the model when a particular underlying technology is changed and thus satisfies the portability of the MDA approach and reduces development efforts, time, and cost [91]. This makes the PIM the essence of the MDA approach. The PIM can be developed using the Unified

Modelling Language (UML) adopted in the MDA approach as a standard vendor-neutral modelling approach [90, 159]. In addition, the UML class diagram is a foundation of the PIM, which is a presentation of the system abstractions [90]. The class diagram should be refined by using four common techniques: hot spots, frozen spots, design patterns [23, 160], and design principles [161]. Both hot spots and frozen spots were introduced earlier in Section 2.4. Design patterns are defined in [99] as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context". Design patterns describe proven design ideas and knowledge and are thus very important for designers [102]. It is recommended to use as many patterns as possible in designing and applying application frameworks [23]. A framework can realize or instantiate one or more patterns into an executable artefact [20]. Additionally, patterns can be used as a method to document application frameworks because they provide a common vocabulary for depicting software design to help developers understand the framework [99]. Design principles help software developers build better designs. Design patterns are used as tools for applying the design principles. There are five primary design principles that support reusability [161] and are collectively referred to as SOLID design principles. The acronym is formed from: (1) Single Responsibility Principle (SRP); (2) Open-Closed Principle (OCP); (3) Liskov Substitution Principle (LSP); (4) Interface-Segregation Principle (ISP); (5) Dependency Inversion Principle (DIP). The SRP states that "a class should have only one reason to change" [161]. This means that a class should have only one responsibility so that there is only one reason to open and modify the class. In fact, the better responsibilities are defined, the more precise the names used to define interfaces and their methods can be, and these names can then

describe the details without the need to view the detail implementation. The OCP states that "software entities (e.g., classes, modules, functions) should be open for extension, but closed for modification" [161]. This means that the dependencies should be separated from their client, so that the dependencies can be changed without changing the client. The LSP states that "subtypes must be substitutable for their base types" [161]. In other words, it should be possible to substitute any concrete class for its interface to ensure that the interface truly represents the concrete implementation. This substitution should be performed without any run-time or compile-time errors and without creating unexpected results. The ISP states that "clients should not be forced to depend on methods that they do not use" [161]. In other words, interface methods should not be implemented that will not be used. The DIP states that "(1) high-level modules should not depend on low-level modules. Both should depend on abstractions. (2) Abstractions should not depend on details. Details should depend on abstractions" [161]. This means that the dependencies should be outside the client and the client should be able to control these dependencies through abstractions (interfaces) rather than concrete classes, so that if the concrete classes are changed the client will not have to change the dependencies.

The objective of the third viewpoint is to construct a specific model, which is platform-dependent and is therefore called a PSM [90]. A PIM should be transformed into one or more PSMs that are customized to depict the system implementation based on specific technology [90, 91] according to the needs of the enterprise [162]. For example, a particular PIM can be transformed to a PSM in J2EE and a PSM in Microsoft .NET technology [154]. The transformation process can be performed using an automated tool; thus, this process plays a key role in improving the developer

productivities [90, 91]. A PSM represents a physical model, while a PIM represents a conceptual model [91].

The MDA approach consists of three essential development activities: analysis, low-level design, and coding [90, 91]. The outcomes of the first and the second development activities represent the second and the third viewpoints of the MDA approach respectively. However, the outcomes of the third development activity is a code model, which defines the code used for implementation [90]. Similar to the transformation process from PIM to PSM, the PSM can be transformed into code model using an automated tool [90, 91]. The PIM development process is the only step that requires manual and innovative development. The next two development activities – the PSM development and the code development – are automated [90, 91, 163], with some manual development.

The suitability of the MDA approach for designing and implementing healthcare systems in the biomedical informatics domain has been established by [91]. The MDA approach has the capability to handle the challenges of healthcare systems development, such as platform dependency, portability, interoperability, and scalability [91]. In fact, meeting these challenges is necessary to design and implement the proposed application framework, where various mobile platforms and sensor vendors exists. Additionally, the application framework must be scalable and interoperable to support the communication with an unlimited number of monitoring applications and wireless sensors. Moreover, the resulting PIM provides a valuable artefact that can be reused despite the rapid changes in mobile and wireless technologies. Consequently, PIM will ensure this research remains relevant for the

long-term, due to the possibility of transforming the resulting PIM to various PSMs to reflect the continuous changes in the technology.

2.4.4. Framework Testing

Framework testing aims to identify if the framework satisfies the required functionality and to evaluate the framework's reusability [15]. Reusability is among the key characteristics that distinguish successful software frameworks [27]. Framework reusability, which are normally concerned with code and design [164], can be evaluated by: (1) ensuring that the framework design guidelines are applied to provide a common language for communication between framework authors and framework users [9, 10]; (2) evaluating framework reusability using an application framework reusability model [10, 93]; (3) instantiating the framework using prototypes to develop sample applications [15-17, 165, 166]; (4) calculating the amount of reuse based on the developed prototypes to measure how much reuse is achieved [18]; (5) evaluating framework reusability using software expert review to confirm the reusability of the framework [19]. In this research, all five of these evaluation methods were used to evaluate the reusability of the application framework from different aspects.

2.4.5. Framework Documentation

Framework documentation includes documents that describe the purpose of the framework [127], the use of framework [127, 165], the user manual [165], and the design of the framework [127, 165]. Moreover, good documentation contains various examples including sample code for customizing and extending the framework [24, 27, 127]. In addition, design patterns can be used as a documentation approach to

capture the framework design and help developers understand the framework [99, 108].

2.5. CaMPaMS in Biomedical Informatics Domain

Application frameworks can be used to develop applications directly as well as to address business activities in a family of related applications in a specific domain [23]. In the context of this research, application frameworks are used specifically in the biomedical informatics domain to develop a family of CaMPaMS, including monitoring patients with cardiovascular diseases [167], monitoring elderly people's vital signs [168], monitoring epileptic patients [63], or monitoring patients with diabetes [169]. The following sub-sections introduce the biomedical informatics domain and the family of CaMPaMS.

2.5.1. Biomedical Informatics Domain

Biomedical informatics originated when a doctor first started recording observations about a patient's sickness and used this information to treat other patients [170]. Doctors first began using computer applications for biomedical computation in the 1960s [171]. Since the emergence of biomedical informatics, various terminologies and definitions have evolved to shape the biomedical informatics body of knowledge.

Since the 1960s, several terms have been used in the literature to refer to biomedical informatics, such as "Medical Computer Science", "Biomedical Computing", "Biocomputation", "Medical Computing", "Medical Information Science", "Health Care Informatics" and "Healthcare Informatics". However, the term biomedical informatics is now widely recognized as a comprehensive term covering all areas of

application in health, clinical practice, and biomedical research. As a result, various academic groups have changed their names. For example, the medical informatics journal called Computers and Biomedical Research changed its name to The Journal of Biomedical Informatics.

Similarly, several definitions have been used in the literature to define biomedical informatics. Definitions in the 1970s focused on using the computer in all fields of medicine. In the 1980s, biomedical informatics studies often focused on theoretical, scientific, and practical approaches to develop analytical tools [172-174]. In the 1990s, biomedical informatics studies highlighted decision-making, information, and knowledge management [175-177]. More recently, definitions have concentrated on the analysing and processing of health medical data to support decision-making [171, 178].

This study adopts the definition of biomedical informatics presented by Shortliffe and Blois [171] as "the scientific field that deals with biomedical information, data, and knowledge—their storage, retrieval, and optimal use for problem solving and decision making" [171].

This definition is in harmony with the aim of this research, which is to design a reusable application framework for CaMPaMS, an application framework that uses received personal biomedical data to detect predefined health events that are of interest to PMS. PMS are considered an applied research area of biomedical informatics [171] and are among its earliest applications [179]; they are introduced in the next section.

2.5.2. Context-Aware Mobile Patient Monitoring Systems

There is no doubt that PMS have improved the quality of healthcare [45, 180, 181]. PMS do not replace the role of healthcare professionals; instead, they attempt to assist and complement their roles [45, 60, 182] and provide an alternative to monitoring patients solely within the boundaries of healthcare organizations [181]. Patient monitoring is defined as

> "repeated or continuous observations or measurements of the patient, his or her physiological function, and the function of life support equipment, for the purpose of guiding management decisions, including when to make therapeutic interventions, and assessment of those interventions" [183].

Therefore, PMS automate repeated or continuous tasks required for monitoring patients, focus on adherence to medical advice and detection of abnormal health events, carry out analysis, and inform healthcare professionals when abnormal health events are detected. Thus, PMS assist healthcare professionals to focus on providing experienced therapeutic intervention on time [60].

Wireless sensors and mobile devices, as the two primary technologies of PMS, have influenced the terminology used to describe PMS. However, all such systems have a common primary principle, which is to monitor patients. Terminology commonly used includes: (1) personal PMS [52, 184, 185]; (2) remote PMS [45, 181, 186] (the terms "Telecare", "Telemonitoring", and "Home Monitoring" are used to refer to remote PMS) [181]; (3) ambulatory PMS [187-189]; and (4) mobile PMS [190, 191]. However, for this research the term "MPMS" was used to refer to all PMS that are hosted on mobile devices (e.g. smartphones) and used wireless sensors (e.g. WBS or environmental sensors) for biomedical purposes.

Mobile devices, such as smartphones, have obviously contributed to the development of PMS [63]. For example, they provide a platform to develop MPMS [63, 64] and act as a base unit to collect biomedical data, such as vital signs, from wireless sensors [65]. Hosted on mobile devices, PMS can provide continuous real-time monitoring of a patient anytime, anywhere [51, 192]. Additionally, they enable healthcare professionals to monitor their patients remotely [45, 180, 190].

Using mobile phone technology in PMS simplifies the collection of data required for monitoring patients [65, 191]. The mobile device can process this data locally [64, 191, 193] and if required transmit the results to a dedicated backend server at a healthcare organization for further processing [180, 190, 191].

There is a consensus that PMS should be designed to support an unlimited number of wireless sensors [67, 76, 194]. Wireless sensors, including WBS and wireless environmental sensors, have contributed significantly to the development of PMS [52, 188, 195]. WBS are implantable or wearable sensors that are attached on a patient's skin or implanted in their tissues, each of which has its own microprocessor, battery, and provides wireless communication. Normally, each of these sensors measures a particular parameter, then performs low-level processing on the measured biomedical data (e.g. vital signs) and transmits this data using a wireless network to a local processing unit such as a mobile device (e.g. smartphone) for processing. If required, the local processing unit can wirelessly send biomedical data to a backend server for further processing [55].

ZigBee short-range wireless communication technology [196], which is based on the IEEE 802.15.4 standard [197, 198], is considered the most suitable technology to enable wireless communication between WBS [197-199] and a mobile device (e.g. smartphone) in PMS [200, 201]. ZigBee provides low-cost wireless communication with low-power consumptions (less processing and memory resources) within a short range [197-199].

Both WBS and environmental sensors have significantly contributed to the development of PMS [49, 188, 202]. WBS can be used to monitor patient biomedical data (vital signs) such as BP and BT, as well as monitor patient physical activities such as walking and running [45, 55, 203]. Environmental sensors can be used to monitor the surrounding environmental conditions that affect patients, such as air temperature, humidity, lighting level, and location [45, 49, 66]. These sensors can be placed in the environment as standalone dedicated sensors or it can be integrated sensors such as those integrated in a patient's mobile device [52, 204, 205].

MPMS have introduced numerous benefits to a wide range of stakeholders. A portion of these benefits has been associated with healthcare organizations, including hospitals and clinics in addition to healthcare professionals, including doctors and nurses. The greatest portion of these benefits however have been associated with the individuals who are the main users of these monitoring systems, including patients with chronic diseases, healthy people who are prone to chronic diseases by inheritance, those trying to change their unhealthy lifestyle, and athletes who need to keep track of their fitness and performance.

First, MPMS take over part of the routine tasks required for monitoring patients, reducing the pressure on healthcare organizations [60, 206]. Therefore, these systems improve healthcare organizations' efficiency [181] by allowing them to provide their services to a large number of patients suffering from critical conditions [206, 207] within a short time [186].

Second, MPMS decrease the load on healthcare professionals by allowing patients to participate in taking care of themselves [45, 60], meaning that healthcare professionals can focus on urgent conditions [181, 186]. Additionally, these systems provide real-time, continuous monitoring [45, 59, 60], anytime, anywhere during patients' day-to-day lives [40, 190, 191]. Accordingly, these systems can detect abnormal health events instantly and notify healthcare professionals to enable them to make suitable clinical decisions [45, 60, 180] as well as to provide proactive treatment to protect their patients from future complications [45, 55].

Third, individuals mainly benefit from MPMS through the reduction in cost of healthcare services [45, 60, 65], because MPMS allow individuals to stay in their homes [181, 206, 208] while being monitored long term [60]. Moreover, these systems improve individuals' lifestyle by making them more independent and allowing them flexibility and mobility while being monitored anytime, anywhere [45, 60, 181]. On top of that, using these systems increases patient adherence to treatment [60] and plays a key role in monitoring a patient's response to any medication [67].

Furthermore, individuals such as patients with chronic diseases and the elderly greatly benefit from using MPMS [45, 195]. This type of individuals needs continuous long-

term monitoring anytime, anywhere during their everyday lives, which can be successfully achieved by these systems [59, 60, 190]. MPMS enable such patients to participate in taking care of themselves [40, 45]. Therefore, MPMS play a key role in the management of and protection from chronic disease complications.

However, context awareness computing has introduced numerous benefits to MPMS. The term "context" is broad and unclear and thus must be defined for the purposes of this research. A review of the literature reveals a large number of definitions. Dey et al.'s [209] general definition of context is the most adopted and referenced in literature. They define "context" as

> "any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects" [209].

The term "situation" in this definition refers to "a description of the states of relevant entities" [209]. The term "context-aware computing" was coined in [210] and elaborated on in [209] to be more general in scope and reflect a system's capability to use "context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [96].

The main purpose of context-aware computing is to achieve application adaptability [211, 212]. An application is considered context-aware if it can adapt its behaviours to contextual changes without user intervention [46, 209, 213].

The emergence of wireless sensors and mobile technologies has played a key role in the advancement of context-aware computing [214, 215]. Wireless sensors have been

represented as a primary source of context data [213, 216, 217]. In fact, the greater the number of sensors, the more comprehensive the information gained [46, 213].

Similarly, mobile devices such as smartphones have been used widely in context-aware applications [218]. They are portable and have become part of users' lifestyles [219]. They obtain personalized context data from various sources [194, 205, 213] and process them locally [69, 220].

Part of context-aware computing research focuses on defining context awareness and part on building context-aware applications [63, 221]. These applications aim to make daily used appliances, devices and objects context-aware [217]. Biomedical informatics is considered one of the richest domains for context-aware applications [222]. Among the application families of biomedical informatics is CaMPaMS [167]. Examples in this family include applications that monitor patients with chronic diseases such as hypertension, diabetes, and epilepsy, in terms of vital signs, medication treatment, and disease symptoms.

Patient context can be defined as any information that can be used to characterize a patient's medical situation such as high BP. This definition is based on the general definition of context introduced in [223]. The context information in this definition can include patient vital signs (e.g. BT), medical symptoms (e.g. dizziness), risk factors (e.g. cholesterol level), prescribed medications (e.g. calcium-channel blocker), physical activities (e.g. sleeping), and surrounding environment (e.g. room temperature). However, it was found that characterizing patients' medical situations, such as high BP, depends on patient context information such as vital signs (e.g.

systolic and diastolic BP) and physical activities (e.g. running) [69]. For example, the normal BP during sleeping is less than during running [68, 224]. Therefore, identifying patient context based on context information enables effective characterization of the medical situation and allows MPMS to adapt to changes in a patient's medical situation. An example of such adaptation is the triggering of an alarm or the contacting of healthcare professionals once a critical medical situation is detected [49, 68, 69].

2.6. Software Framework for Biomedical Informatics Domain

This section presents an analysis of 20 previous studies that designed software frameworks in the biomedical informatics domain or software frameworks that can be applied in biomedical informatics.

These studies were characterized, in the following two sub-sections, based on two categories, which are the framework reusability, as reusability is the most important quality goal for application frameworks [10, 23, 27], and domain requirements, as domain requirements encapsulate the business activities of the family of CaMPaMS in the biomedical informatics domain [23].

2.6.1. Reusability of Application Frameworks

In spite of the advancement in software, application design, implementation, and maintenance are complex [225]. However, frameworks are among the most suitable solutions to simplify application development and overcome complexity [22]. This is because frameworks represent the fundamental reuse techniques of software engineering [2]. Framework reusability supports the domain knowledge and previous development of experts to avoid rebuilding applications from scratch. Therefore, it is

required to ensure framework reusability for developing new applications [23], which is defined as "the degree to which an asset can be used in more than one software system, or in building other assets" [96].

Accordingly, it is important to ensure the key characteristics of successful frameworks such as reusability [48, 51, 67, 77, 78, 226], rather than other framework characteristics [23]. Based on the literature analysis, it was found that eleven studies have designed application frameworks in the biomedical informatics or application frameworks that can be applied in the biomedical informatics [45, 48, 49, 51, 52, 61, 67, 71, 76-78]. However, of these application frameworks only five were evaluated in terms of their reusability by developing application prototypes on top of their frameworks [48, 51, 67, 77, 78]. Although the authors of these studies claimed that their frameworks were reusable, they used only prototyping approach to evaluate their frameworks reusability. Moreover, only two of them considered design guidelines reusability aspect in their design that is using design patterns [9]. Neither of the frameworks were designed, nor evaluated based on multiple reusability aspects and multiple reusability evaluation approaches respectively.

2.6.2. Domain Requirements for CaMPaMS

Domain requirements are the reusable requirements for a family of systems [35-37, 130]. Based on the literature analysis, it was found that there are six reusable requirements that should be addressed in the design of application frameworks for CaMPaMS. The justification for selecting each of these domain requirements is presented in the following sub-sections.

2.6.2.1. Support Anywhere, Anytime Monitoring

Monitoring patients anywhere, anytime allows detecting their abnormal health events instantly, which in turn allows PMS to react immediately. For example, these systems can call healthcare professionals to enable them to make suitable clinical decisions [45, 60]. In addition, monitoring patients anywhere, anytime can improve their lifestyles by allowing them to be more independent, more flexible, and mobile while being monitored [45, 60, 181]. Therefore, anywhere, anytime monitoring is required for monitoring patients in the biomedical informatics domain [45, 50, 61, 67, 76, 194, 226, 227]. Consequently, this research selected this domain requirement as one of the domain requirements of CaMPaMS.

2.6.2.2. Support Real-Time Continuous Monitoring

Real-time continuous patient monitoring allows instant detection of patients' abnormal health events [45, 60]. Similar to anywhere, anytime patient monitoring, real-time continuous monitoring allows PMS to react immediately. For example, the system can call healthcare professionals to enable them to make suitable clinical decisions [45, 60]. Accordingly, this provides proactive medical care to protect patients from future complications [45, 55], especially those who suffer from chronic diseases [54]. Therefore, real-time continuous patient monitoring is required for monitoring patients in biomedical informatics [45-47, 49, 51, 76, 192, 226, 228, 229]. Consequently, this research selected this domain requirement as one of the domain requirements of CaMPaMS.

2.6.2.3. Support Unlimited Sensors at Design Time

Sensors play a primary role in supporting PMS [55]. In fact, the greater the number of sensors, the more comprehensive the information gained is. This enhances the detection efficiency of a patient's medical situation [46]. Therefore, supporting an unlimited number of sensors at design time is required for monitoring patients in biomedical informatics [45, 46, 48-52, 71, 78, 192, 194, 227-229]. Consequently, this research selected this domain requirement as one of the domain requirements of CaMPaMS.

2.6.2.4. Support Unlimited Monitoring Applications at Design Time

The elderly, especially those who suffer from chronic diseases, need to be monitored by different dedicated applications such as those monitoring hypertension and diabetes [54]. Therefore, supporting an unlimited number of applications to be developed at design time is required for monitoring patients in biomedical informatics [45, 46, 48, 50, 51, 67, 78, 192, 227, 228, 230]. Consequently, this research selected this domain requirement as one of the domain requirements of CaMPaMS.

2.6.2.5. Support Mobile Platform

Frameworks can be designed for a specific platform, such as desktop (e.g. backend server) [46, 49, 50] and mobile platforms (e.g. smartphone) [48, 51, 52]. Undoubtedly, the mobile platform supports portability and mobility in general [45, 229]. Compared with the desktop platform (backend server), the mobile platform (smartphone) has obvious benefits for running a framework to develop MPMS. In fact, the technological advancements of mobile devices in terms of hardware and software provide the

required computations to monitor a patient without being connected to a backend server. Among these advancements are processing and wireless capabilities, operating systems, multithreading ability, and storage capacity [52, 67, 231]. Aside from this, they provide the required computations to extract patients' contextual information from context sources with sufficient accuracy [194].

The mobile platform also supports real-time patient monitoring [52, 229, 231]. In this case, it can support context awareness and adaptation through direct detection of context changes [45, 51]. It also supports privacy protection of the patient's contextual data [51] and can support patient monitoring anywhere, anytime [205, 232]. It supports active (always turned on) continuous monitoring [67, 205, 232]. This provides proactive monitoring in the form of early detection of abnormal health situations [233]. It also enables patients to monitor themselves during their daily activities without interruption [205]. Finally, the mobile platform avoids the continuous network communication costs required to transmit the data to a backend server [51, 194, 229].

Developing frameworks in biomedical informatics hosted on a mobile platform is therefore the ideal solution to achieve the above-mentioned benefits [48, 51, 52, 71, 192, 234]. Consequently, this research selected the mobile platform (e.g. smartphone) as one of the domain requirements of CaMPaMS.

2.6.2.6. Support Context-Aware Monitoring

Context awareness in MPMS allows effective detection of patient medical situations (e.g. high BP) based on patient contextual information (e.g. systolic BP, diastolic BP, and dizziness). Accordingly, these systems can change behaviour by adapting to the

changes of a patient's medical situation, for example by triggering an alarm [49, 68, 69]. Therefore, adopting context awareness to develop MPMS is required for monitoring patients in the biomedical informatics domain [45, 194]. Consequently, this research selected this domain requirement as one of the domain requirements of CaMPaMS.

According to [227], the more context information obtained, the higher the context-reasoning accuracy achieved. Therefore, this study aims to combine multiple types of context information to support the design of context-aware PMS using a mobile device and wireless sensors. To achieve this goal, context information types have to be identified within the biomedical informatics domain, which are related to context-aware PMS. Analysis of the existing literature revealed that there is no consensus on the types of context information adopted in biomedical informatics studies. However, there are three types of context information that are centred on the patient and can contribute to the design of MPMS. These types are classified as medical, physical activities, and environmental contexts. They are elaborated on in the following sub-sections.

2.6.2.6.1. Medical Context Type

The medical context includes biomedical information that is required for monitoring patients. This type of context is classified into four sub-types of context information as shown in Table 2.6. First, the measurable medical context that mainly represents patients' vital signs and is widely adopted in the literature to provide continually measured medical personal information [49, 50, 68, 69, 192, 205, 218, 220, 224, 227]. In fact, vital signs represent the signs of life [235], defined in [236] as the "body's

physiological status and provide information critical to evaluating homeostatic balance". Five standard vital signs must be measured and continually monitored: BT, respiration rate (RR), Heart Rate (HR), BP, and Electrocardiogram (ECG) [40].

Second, the non-measurable medical context that describes medical symptoms that are difficult to measure by wireless sensors (e.g. dizziness, vomiting, sleepiness, or headache) and is thus rarely adopted in biomedical informatics studies. It also provides dynamic medical personal information that is difficult to measure by sensors [67]. However, this sub-type is able to complement the information obtained from the measurable medical context. For example, monitoring hypertension requires monitoring non-measurable medical contexts such as headache and constipation. These non-measurable medical symptoms complement measurable medical contexts such as BP and HR vital signs [67].

Table 2.6

Summary of Previous Studies that Support Context-Aware Monitoring

Context information		Previous studies
Types	Measurable medical context	[49, 50, 68, 69, 192, 205, 218, 220, 224, 227]
	Non-measurable medical context	[67]
	Risk factors medical context	[50, 192, 218, 227]
	Prescribed medications medical context	[67, 69]
	Physical activities context	[68, 69, 194, 224]
	Environmental context	[49, 50, 68, 69, 237, 238]
Sources	Wireless body sensors	[49, 50, 68, 69, 192, 205, 218, 220, 224, 227]
	Wireless environmental sensors	[68, 69, 194, 224]
	Mobile graphical user interface	[67]
	Patient profile	[192, 205, 220, 227]
Reasoning	Rule-based reasoning	[49, 192]

Third, the risk factors context (also known as a health risk) that is defined by WHO as "a factor that raises the probability of adverse health outcomes" [53]. These factors were adopted in a number of biomedical informatics studies to represent personal health information that changes infrequently [50, 192, 218, 227]. In fact, risk factors are countless, and each disease has a number of associated risk factors [53]. For instance, there are eight risk factors associated with hypertension: family history, aging, gender, lack of physical activity, obesity, alcohol consumption, smoking, and cholesterol level [239]. The obesity risk factor is calculated based on the body mass index (body mass index = weight in kilograms / [height in meters × height in meters]) [239]. These eight risk factors are jointly responsible for more than 75% of the deaths of hypertensive patients [53]. It has been shown that risk factors affect the normal readings of vital signs [53, 69, 240]. For instance, the blood-pressure reading is affected negatively by either alcohol consumption or obesity. Similarly, the normal cholesterol level is affected by either smoking or fat intake [53].

Fourth, the prescribed medications context that describes the current prescribed medications for a patient [67, 69] but is rarely adopted in biomedical informatics studies. In fact, these prescribed medications have effects on the patient's normal vital signs [67, 69, 240]. Therefore, healthcare professionals use this context information to assess the effects of the prescribed medications on patients and to evaluate the patient's response to treatment. For example, a healthcare professional can manage hypertension by prescribing a medication such as Amlodipine, a calcium-channel blocker, with suitable frequency and dosage (such as 5 mg every morning). Then, the professional can monitor the effect of such medication on a patient's BP to assess the

patient's response to the prescribed medication, and then make an appropriate follow-up decision or action [67].

2.6.2.6.2. Physical Activities Context Type

This type of context information describes the patient's physical activities such as walking, running, eating or sleeping, and has been adopted in several previous studies [68, 69, 194, 224]. Patients' physical activities have direct effects on their vital signs. For example, the normal HR while running or climbing up stairs is higher than the normal HR while walking or lying down [194, 240]. Similarly, the normal BP during sitting or sleeping is less than the normal BP during eating or doing physical exercise, such as running [68, 224, 240].

2.6.2.6.3. Environmental Context Type

This context type provides information about the surrounding environment that can affect a patient's medical state, such as temperature, light, humidity and noise. It has been adopted widely in previous biomedical informatics studies [49, 50, 68, 69, 237, 238]. It also contributes to the monitoring of diseases. For instance, patients with Amyotrophic Lateral Sclerosis, which is "a disease of the nerve cells in the brain and spinal cord that control voluntary muscle movement" [241], can benefit from monitoring floor humidity to prevent them from falling [68]. In addition, environmental context information also affects vital signs, for example room temperature affects the normal heartbeat and consequently change in heartbeat affects BP [68].

2.6.2.6.4. Context Information Sources

This study aims to obtain the context information of the three context information types from four different context data sources: a mobile patient profile, WBS, wireless environmental sensors, and a mobile graphical user interface as shown in Table 2.6. These context data sources have been adopted based on an analysis of previous studies related to context-aware PMS within the biomedical informatics domain and a consideration of the identified context information types.

First, the mobile patient profile that hosted on the patient's mobile device is widely adopted in biomedical informatics studies as a main data source for obtaining the risk factors and the prescribed medications context [192, 205, 220, 227]. It contributes to the accuracy of CaMPaMS [242] and also plays a primary role in personalizing the patient monitoring process [50, 192]. For example, this source can provide information about alcohol consumption. Alcohol consumption is one of the risk factors associated with hypertension and it negatively affects BP. Therefore it has to be considered when monitoring a patient with hypertension [53]. However, if a patient does not consume alcohol, then the patient monitoring process has to be personalized by ignoring the effect of such a factor, thereby optimizing the patient monitoring process. Moreover, using a patient profile hosted on the patient's mobile device can contribute significantly to CaMPaMS. One advantage is that it supports the privacy protection of the patient's contextual data [51]. Furthermore, it avoids the continuous network communication costs required to transmit and receive data to and from a backend server [52, 194, 229]. Aside from this, it avoids the problems associated with wireless network interruptions. Moreover, a mobile patient profile can support context

awareness and adaptation through direct detection of context changes [51]. Finally, it supports real-time continuous patient monitoring [52] anywhere, anytime [205].

Second, the WBS that is adopted as a primary data source for measurable medical context information and has been used in most previous studies that have adopted this type of context information. Additionally, WBS have been also used as a main data source for the physical activities context in many previous studies [49, 50, 68, 69, 192, 205, 218, 220, 224, 227].

Third, the wireless environmental sensors that is also used as a primary data source for environmental context in most studies that have adopted this type of context [68, 69, 194, 224]. Indeed, it was used as an essential data source for environmental context in most studies that adopted this type of context [49, 50, 68, 69]. It also plays a primary role in supporting CaMPaMS by providing context information that can be measured continuously during patients' daily lives [55].

The fourth context data source is the mobile graphical user interface that supports obtaining data directly from patients through manual answering of Yes/No questions. It has been only rarely adopted in the literature [67], but is considered as a main data source for obtaining a non-measurable medical context. Moreover, it plays a primary role in supporting context-aware PMS that require dynamic context information that cannot be measured by wireless sensors or retrieved from the mobile patient profile [67].

2.6.2.6.5. Context Reasoning

The context situations of a patient that are of interest for CaMPaMS cannot be directly obtained. In fact, identifying a patient's context situation based on a single type of context information is insufficient; other types of patient context information such as medical context, physical activities context, and environmental context need to be incorporated.

For example, various patients' context information types can be used to identify a change in patient situation, such as the change from normal BP to high BP. In other words, deciding that a patient has high BP situation can be inferred by integrating at least three types of context information. First, the medical context types, which include the measurable context, such as BP, non-measurable context, such as headache [67], risk factors context, such as overweight [53], and prescribed medications context, such as Amlodipine; a calcium channel blocker [67]. Second, the physical activity context type such as doing some physical exercises including running [68, 224, 240]. Third, the environmental context type such as room temperature [68].

The inference process used to identify a patient's context situation, as in the previous example, is the core of context-aware reasoning. The new derived context is called high-level context, which is also known as context situation. All the other context information used to derive the high-level context is called low-level context, which is obtained directly from context sources [72, 192].

Context reasoning aims to detect the change in high-level context information based on low-level context information [72, 192]. In the example above, high BP is the high-

level patient context information or patient context situation. The other context information is the low-level patient context information.

Rule-based reasoning is one of the used approaches for context reasoning of PMS in the biomedical informatics domain [49, 192], as shown in Table 2.6, and is defined as "a natural knowledge representation, in the form of IF–THEN rule statements. Rules are simply patterns and an inference engine searches for patterns in the rules that match patterns in the data" [243].

In the context of this research, one or more context-aware monitoring queries are required to detect the change in the patient's medical situation (e.g. a change from normal to high BP as high-level context information). Each query consists of one or more query elements that represent low-level context information (e.g. non-measurable context such as headache, risk factors context such as obesity, and physical activity context such as running). According to the rule-based reasoning approach, each context monitoring query represents a rule that consists of a set of sub-rules that are the query elements. The context monitoring queries (rules) are obtained from healthcare professionals as domain experts. Figure 2.3 illustrates the general form and an example of a rule.

Base rule:	**IF**	**{**	**(***sub-rule (1)* **)** *AND* **(***sub-rule (2)* **)** *AND* **...** **(***sub-rule (n)* **)**
		}	
	THEN	**{**	*Patient context situation*
		}	

Example:	**IF**	**{**	**(***blood-glucose level >= 200 mg/dL) AND* **(***physical activity = watching TV) AND* **(***aging = false) AND* **(***smoking = false) AND* **(***obesity = false) AND* **(***chronic disease = false) AND* **(***question: (Did you take your breakfast during the last 2 hours?) = true)*
		}	
	THEN	**{**	*Patient has overt diabetes mellitus*
		}	

Figure 2.3. Diabetes context monitoring queries

2.7. Lacks and Gaps Identification Based on Previous Studies

This section presents the identification of the lacks and gaps in previous studies based on the analysis in Section 2.6. This information has been mentioned earlier in Section 1.3. The detailed of the percentage and proportion values of the gaps and lacks of the studied application frameworks are shown in Table 2.7 and Table 2.8.

Table 2.7

Percentages and Proportions of Domain Requirements in Previous Studies that Designed Application Frameworks for PMS

Previous studies	Anywhere, anytime monitoring	Real-time continuous monitoring	Unlimited number of sensors	Unlimited number of applications	Context awareness computing	Hosted and executed on a mobile platform	Total
Villarreal et al. [61]	✓	✓					**2/6**
Paganelli and Giuli [49]		✓	✓		✓		**3/6**
Koutkias et al. [67]	✓		✓	✓			**3/6**
Ahmad et al. [76]	✓	✓	✓	✓			**4/6**
Broens et al. [45]	✓	✓	✓	✓	✓		**5/6**
Percentages	**80%**	**80%**	**80%**	**60%**	**40%**	**0%**	
Proportions	**4/5**	**4/5**	**4/5**	**3/5**	**2/5**	**0/5**	
This Study	✓	✓	✓	✓	✓	✓	

Table 2.8

Percentages and Proportions of Sub-Domain Requirements Related to Context Awareness Computing Domain Requirement in Previous Studies that Designed Application Frameworks for PMS

Previous studies	Types of context information						Data sources of context information				Reasoning approach	Total
	Measurable medical context	Non-measurable medical context	Risk factors medical context	Prescribed medications medical context	Physical activities context	Environmental context	Wireless body sensors	Wireless environmental sensors	Mobile graphical user interface	Patient profile	Rule-based reasoning	
Paganelli and Giuli [49]	✓					✓	✓	✓			✓	5/11
Broens et al. [45]	✓		✓	✓	✓	✓	✓	✓		✓	✓	9/11
Percentages	**100%**	**0%**	**50%**	**50%**	**50%**	**100%**	**100%**	**100%**	**0%**	**50%**	**100%**	
Proportions	**2/2**	**0/2**	**1/2**	**1/2**	**1/2**	**2/2**	**2/2**	**2/2**	**0/2**	**1/2**	**2/2**	
This Study	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

2.8. Summary

In conclusion, it can be clearly seen that software reuse is one of the most used principles to simplify application development and overcome development complexities. Additionally, it has been shown that in comparison with other reuse techniques, the application framework is an ideal reuse technique because it captures the essence of software engineering reuse techniques to achieve maximum large-scale reuse. It has also been shown that there are six framework development activities: domain analysis; architectural design; framework design; framework implementation; framework testing; and documentation.

PMS have been shown to be a well-established applied research area in the biomedical informatics domain. These systems continuously perform repeatable tasks that are required for monitoring patients to complement the role of healthcare professionals beyond the boundaries of healthcare organizations. Different terminology has been used in the literature to describe PMS and there are different purposes for using these systems. For example, they could be used for monitoring a wide variety of individuals, especially the elderly and patients with chronic diseases such as hypertension or diabetes. This research uses the term "MPMS" to refer to all PMS that are hosted on mobile devices (e.g. smartphones) and that use wireless sensors (e.g. WBS or environmental sensors) for biomedical purposes. Biomedical informatics is considered one of the richest domains for context-aware applications and among the application families of biomedical informatics is CaMPaMS.

It has also been shown that identifying patient context based on context information enables effective characterization of the medical situation, which in turn allows MPMS

to adapt to changes in a patient's medical situation. In addition, it was found that there are three types of context information that are centred on the patient and that can contribute to the design of MPMS: the medical context; the physical activities context; and the environmental context. The medical context information type is further classified into four subtypes of context information: measurable medical context; non-measurable medical context; risk factors context; and prescribed medications context. Furthermore, it was found that there are four context data sources to obtain context information: mobile patient profile; WBS; wireless environmental sensors; and mobile graphical user interface. It was also established that rule-based reasoning is one of the most used approaches for PMS context reasoning in the biomedical informatics domain.

The study of Broens et al. [45] was found to satisfy the highest number of the domain requirements among existing application frameworks, but there is no framework that integrates all of the identified domain requirements. Furthermore, there is no existing application framework that was designed or evaluated based on an application framework reusability model. Therefore, there is a need to bridge the gap between: (1) application frameworks in biomedical informatics and reusability models; and (2) application frameworks and the domain requirements of CaMPaMS.

CHAPTER THREE
METHODOLOGY

3.1. Overview

This chapter explains the research processes conducted to achieve the objectives of this research as presented in Chapter 1. After an introduction to design research science and justification for considering this research as a design research, the pragmatic research paradigm is discussed briefly. The adopted DRM is then introduced, followed by an elaboration of each of its four stages: research clarification, descriptive study 1, prescriptive study, and descriptive study 2. Finally, a summary of the chapter is presented.

3.2. Design Research

This research is considered as design research because its objectives are in line with the primary objectives of design research. According to [244], design research focuses on achieving two primary aims: (1) to develop an understanding of the current designs based on the related studies; (2) to develop and validate what is required to improve the current designs to be more effective and efficient for the purpose of developing more successful products. In this research, the first three objectives match the first aim of design research, while the fourth and fifth objectives match the second aim of design research.

Design research is multidisciplinary and hence complex [245]. However, design research is worthwhile in practice and industry as well as in the scientific and academic fields [244, 246, 247]. Design research aims to explore artificial phenomena rather than natural ones [248, 249] based on human needs [250, 251]. It also develops solutions [252] through iterating in a cyclic process, which may start in the lab and end in the field in the form of pilot project [253]. These solutions have to solve problems in unique and creative ways with generalization for a specific domain [254, 255].

Furthermore, design research can use various research methods such as the quantitative research approach, qualitative research approach, or mixed research approach that combines quantitative and qualitative research methods. Choosing the most suitable research approach and methods to be applied is dependent on the fact that each school of thought has fundamental research paradigms. Therefore, it can be concluded that research paradigm is the primary factor to be considered when choosing the research approach and methods [244].

3.3. Pragmatic Research Paradigm

Design process in design research generally follows an engineering paradigm [249]. In the field of software engineering, it was found that the pragmatism philosophy was among the fundamental engineering paradigms [256]. Pragmatists do not consider one thought to be better than another; the practical applications of different thoughts are the only tangible measurement to differentiate between them [257]. Consequently, pragmatists believe in the mixed research approach that combines quantitative and qualitative research methods so that they complement each other [258].

Accordingly, this research adopted the pragmatism philosophy as an engineering research paradigm to justify using the mixed research approach that combines quantitative and qualitative research methods. This research approach provides rigorous guidelines for both the construction and evaluation of designed artefacts [250, 252]. Additionally, design research in general supports the mixed research approach [244, 245, 254].

3.4. Design Research Methodology

In relation to the above research paradigm, a scientific approach is considered essential for this type of design research [255], where quantitative and qualitative research approaches have to be practically woven and applied in the research processes [259]. To fit these requirements, this research adopted and customized the DRM that is proposed in [244]. Consequently, this research consists of four stages: research clarification, descriptive study 1, prescriptive study, and descriptive study 2.

Figure 3.4 shows this study's research methodology divided into the four research stages. In the same way, each stage is depicted in terms of its research processes, steps, and outcomes. The figure also illustrates the mapping between each stage, the research objectives, and resulting publications. The relevant research methods used in each research process within each research stage are discussed in detail in the following four sections.

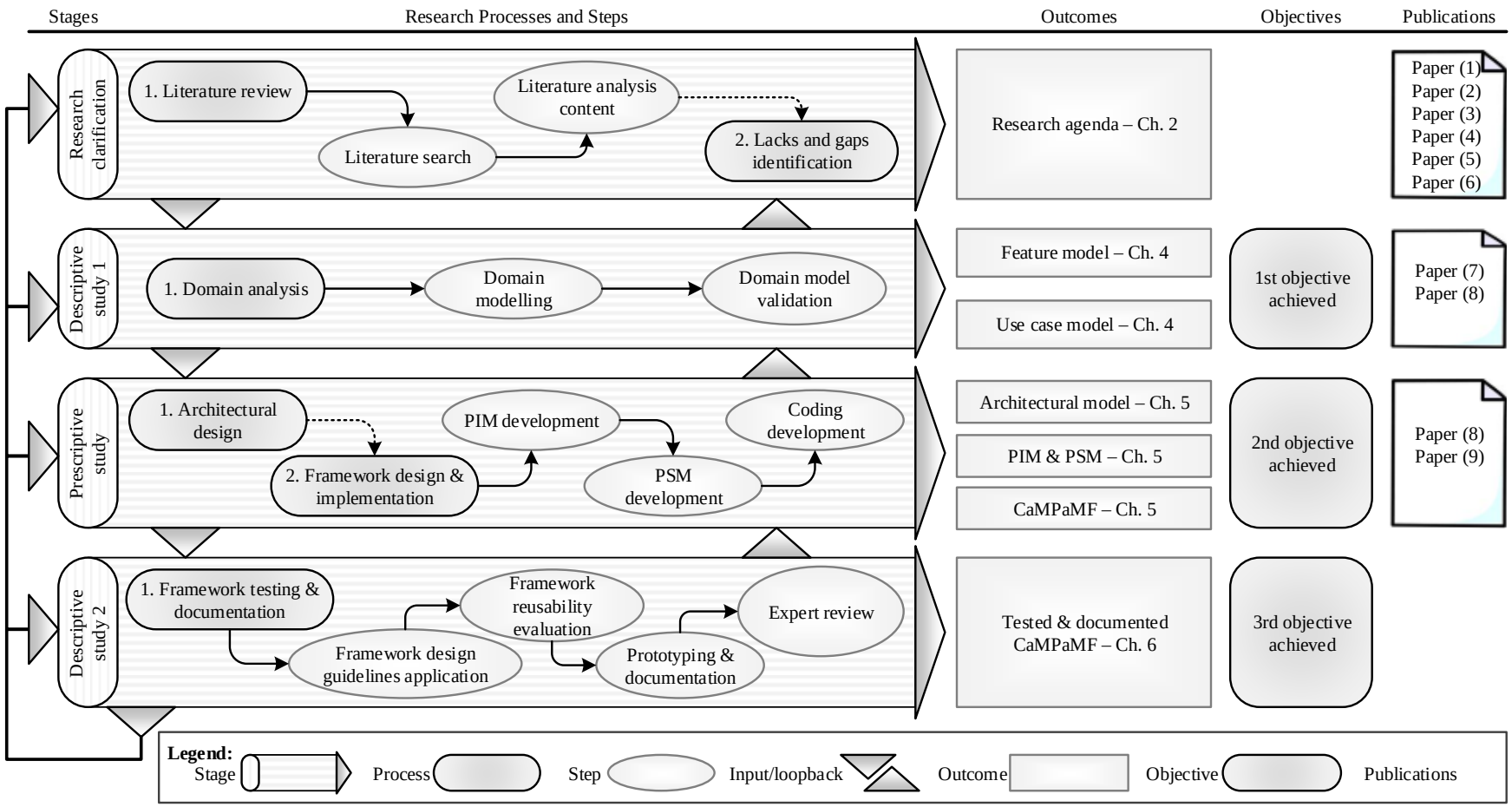

Figure 3.4. Research methodology

3.5. Stage 1: Research Clarification

The primary objectives of this stage are to clarify the research problem by investigating the current designed frameworks in the biomedical informatics domain and to justify the need for further research to enhance the design of a reusable Context-aware Mobile Patient Monitoring Framework (CaMPaMF). To achieve the objectives of this stage, two research processes were conducted, as shown in Figure 3.4: (1) literature review process; (2) lacks and gaps identification process. The outcome of this stage is the research agenda. To ensure the validity of the outcome and to be able to use it in this research with greater confidence, the outcome was published in three conference proceedings [260-262], two journal articles [74, 75], and one book chapter [70]. The following sub-sections discuss the steps and the methods used in these processes, in addition to their outcomes.

3.5.1. Literature Review Process

This process reviewed the literature using two steps. First, a literature search was conducted to collect scholarly articles related to this research and document them in a bibliography. A set of related studies was the primary outcome of this step. Second, the relevant literature was analysed to clarify the research problem. The following sub-sections discuss the methods used in these two steps.

3.5.1.1. Literature Search Step

In this step, the literature-searching method introduced in [263] was adapted for the purposes of this study. The method includes three phases that focus on searching and documenting the literature to provide a comprehensible and credible literature review

process. This process is intended to increase researcher confidence in using the current study's outcomes in further research.

In the first phase, the literature scope was defined by identifying the following five characteristics. First, the focus of the literature search involved all scholarly articles related to design research. Second, the goal of the literature search was to identify the various schools of thought in academic research on designing frameworks in biomedical informatics. Third, the perspective of the literature search was neutral, which means it does not reflect any opinions that support a specific idea or principle. Fourth, the audience of the literature search results was specialized scholars designing frameworks in biomedical informatics. Fifth, the coverage of the literature search was a representative sample, which is selected based on specific criteria (i.e. year of publication and leading article source) to represent all research articles designing frameworks in biomedical informatics.

In the second phase, a set of key terms were identified, including design, reusability, context-aware, application framework, mobile, patient, monitoring, system, and sensors. In the third phase, the literature search process was conducted based on the identified key terms by focusing on scholarly articles from leading journals, conference proceedings, and scholarly databases. However, it was difficult to focus on a specialized range of journals because designing reusable CaMPaMF in the biomedical informatics domain is an interdisciplinary field of study that requires considering a wide range of articles. In fact, the range of journals dealing with designing such frameworks spans biomedical informatics journals, mobile computing journals, information systems journals, communication journals, systems and software

journals, software engineering journals, computer science journals, ubiquitous computing journals, and even network journals. Therefore, interdisciplinary online databases were chosen to begin searching.

3.5.1.2. Literature Analysis Content Step

In this research, the content analysis method based on the inductive approach proposed in [264], was used to clarify the research problem. This approach consists of three main phases: preparation, organizing and reporting.

In the preparation phase, two activities were performed. First, the units of analysis are selected, which include: software reuse and reusability, software framework, framework development, context-aware, PMS, wireless sensors, and mobile technology, domain requirements. Second, the content data were read several times to make sense of the data in terms of the identified unit of analysis, to gain a comprehensive understanding, and to obtain a working knowledge of them.

In the organizing phase, five activities were conducted. First, open coding was performed by writing down notes to describe all aspects of the content. Second, the open coding data were collected and stored in a spreadsheet file. Third, the related data of the spreadsheet file initially were grouped based on observing similarities among them. These groups were called sub-categories. Fourth, these sub-categories were organized under high-level categories. These high-level categories were called generic categories. Fifth, the generic categories were abstracted further based on their similarities or relations with other generic categories to provide a new abstract main category.

In the reporting phase, the reported analysis results were validated. The validation was performed by experts peer-reviewing the analysis results in three conference proceedings [260-262] and two journal articles [74, 75].

3.5.2. Lacks and Gaps Identification Process

The objective of this research process is to identify the lacks and gaps in the literature using a single step. The objective of this step is to synthesize previous studies that designed frameworks in the biomedical informatics domain. To meet this objective, this process uses the concept matrix technique that was introduced in [265]. This matrix provides a method to organize, analyse and synthesize previous studies in order to develop a research agenda. The research agenda identifies the lacks and gaps in the literature to provide a foundation for the researchers to extend the state of the art by filling the gaps [263].

In this research, the concept matrix technique was customized as shown in Table 3.9. The first column in the table lists (*s*) number of the previous studies that designed frameworks in the biomedical informatics domain. The next columns represent (*c*) number of the identified criteria related to the design of reusable CaMPaMF. Each tick (✓) indicates that a specific study has satisfied a particular criterion. The last column represents the total number of satisfied factors in each study out of the total number of all criteria (*n/c*). The percentage row represents the percentage of studies that satisfied a particular criterion (*g%*), while the proportion row represents the number of studies that satisfied a particular criterion out of the total number of studies (*n/s*).

Table 3.9

Concept Matrix

Previous studies	**Criteria**				**Total**
	Criterion (1)	**Criterion (2)**	**...**	**Criterion (c)**	
Study (1)	✓	✓			2 / *c*
Study (2)		✓			1 / *c*
Study (3)					0 / *c*
Study (*s*)					*n* / *c*
Percentage (%)	**25%**	**50%**	**...**	***g*%**	
Proportion	**1 / *s***	**2 / *s***	**...**	***n* / *s***	

Adopted from Webster and Watson [265]

3.6. Stage 2: Descriptive Study 1

This is a transition stage between the research clarification stage (Stage 1) and the perspective study stage (Stage 3) [244]. In this research, the research clarification stage, as discussed in Section 3.5, clarifies the research problem and justifies the need for this research. The perspective study stage, discussed in Section 3.7, focuses on developing an enhanced design of a reusable CaMPaMF.

The objective of this stage is to gain a better understanding of the current situation. The outcomes can be used as a foundation for efficient development in Stage 3 [244]. A comprehensive study was conducted including a literature review and a domain analysis to develop domain models. To achieve the objectives of this stage, one research process was conducted, the domain analysis process (see Figure 3.4). The outcomes of this stage include feature model, use case model, and domain expert review. To ensure the validity of the outcomes and to be able to use them in this research with greater confidence, the outcomes were published first in a conference

proceedings [88] and later in a journal [87]. The following sub-sections describe the steps and the methods used in these processes, in addition to their outcomes.

3.6.1. Domain Analysis Process

The domain analysis process includes two steps: domain modelling and domain model validation. The first step aims to capture the domain requirements of CaMPaMS. The second step aims to validate the domain model resulting from the first step. The outcomes from the domain analysis are domain models [15, 23, 80]. These models form primary inputs to support the activities of framework development [15, 266]. The following sub-sections discuss these two steps in detail.

3.6.1.1. Domain Modelling Step

In this research, domain modelling was conducted by applying MDRE approach [139] as discussed in detail in Section 2.4.1. In this approach, domain requirements can be modelled using a number of models including a feature model and abstract use case model. These models complement each other to extract the domain requirements. In other words, these models depict different views of the requirements from different perspectives. The primary outcomes of the domain modelling step include a feature model and an abstract use case model. The following sub-sections describe the methods used to construct these models.

3.6.1.1.1. Feature Modelling

The feature modelling method introduced in [130] was adopted and customized to construct a feature model. According to [130], the feature modelling method includes four main steps. First, identify sources of features. These sources can be domain

experts, domain literature, and existing systems. Second, identify common and variable features by applying the following five strategies: (1) Look for important domain terminologies that imply variability. (2) Examine domain concepts for diverse sources of variability. For example, various stakeholders have different requirements. (3) Use feature starter sets to begin the analysis. These sets give an initial set of elements that are useful in modelling a specific domain. (4) Look for features at any point in the development. This requires maintaining and updating feature models during the entire development cycle. (5) Identify more features than those were initially intended to implement. This strategy supports future improvements for potential features. Third, construct a feature diagram by applying four general iterative steps. These steps are: (1) record the common features (similarity) between instances; (2) record variable features (differences) between instances; (3) organize features in feature diagrams into hierarchies and classify them according to their types and cardinality; (4) analyse feature combinations and interactions. For example, some features cannot be combined simultaneously. Alternatively, some features depend on the existence of other features. Analysis of the relationships among features may discover new features. Fourth, record all the additional information about features such as a short description about each feature and the rationale for selecting each feature.

The FeaturePlugin tool, developed by the Waterloo University, was used to model the feature diagram because it fully supports the adopted cardinality-based feature modelling approach [267].

3.6.1.1.2. Abstract Use Case Modelling

The use case assortment technique proposed in [86] was adapted for this research to construct an abstract use case model that can be used to develop CaMPaMF. In this technique, an abstract use case model is constructed to capture the requirements of application frameworks. Three steps are involved. First, identify the abstract actors by associating the roles that are played by various actors to achieve a common abstract use case. Second, identify the abstract use cases by grouping all use cases that have the same behaviours into a set of use cases that are related to an abstract use case under an abstract actor. Third, describe the abstract use cases by writing a use case narrative or specification for each abstract use case to explain its behaviour.

3.6.1.2. Domain Model Validation Step

The objective of this step is to validate the domain model, including the feature model and the abstract use case model, based on the following three activities. First, document features by writing short descriptions as well as the rationale for selecting each of them. According to [81], this is achieved by applying the last step of the feature modelling method. Second, author scenarios to explain real situations or concrete behaviours [268]. Authoring scenarios requires detailed knowledge to enforce understanding and learning of the targeted domain. These scenarios must be validated by domain experts [269]. The objective of executing each scenario is to demonstrate the identified domain requirements that result from the domain modelling step. The scenarios are used to facilitate the domain model validation because they describe real situations that are easily understood by domain experts, unlike abstract models [268, 269]. In this research, the authoring of scenarios was based on published medical

guidelines as well as scenarios in the literature that are related to CaMPaMS [45, 68, 228, 270-276].

Third, conduct an expert review to validate the domain model by a number of domain experts [36, 277, 278]. This is achieved using the scenarios authored in the previous activity. A domain expert is defined as "an individual who is intimately familiar with the domain and can provide detailed information to the domain engineers" [135]. To validate the proposed domain model, it has to be easily comprehended by the domain experts and use natural language [277] such as narrative scenarios [269]. A set of simple Yes/No questions is generated that covers each domain requirement discussed in each scenario. These questions can be then answered by domain experts in interviews [277].

In this research, the Jordan Medical Association (JMA), as the official representative of healthcare professionals in Jordan, was asked to nominate suitable healthcare professionals as experts based on specific criteria. Accordingly, the experts were consultant physicians who have a minimum of 15 years' experience and who regularly monitor patients who suffer from diabetes, epilepsy or hypertension. Based on these criteria, the JMA nominated 15 experts who were willing to participate in the domain expert review. The expert review was conducted through structured interviews, as the data collection instrument, by the JMA.

The 15 experts were interviewed individually by the JMA based on one of the three expert review forms. However, one of the experts was interviewed twice because he was specialized in monitoring both hypertension and diabetes. Therefore, 16

interviews were conducted as follows: (1) five interviews with experts in monitoring hypertensive patients; (2) five interviews with experts in monitoring epileptic patients; (3) six interviews with experts in monitoring diabetic patients.

Three expert review forms were developed as shown in Appendix A. Each form consists of seven sections. The first section presents a scenario of daily activities for monitoring a patient who suffers from diabetes, epilepsy or hypertension and equipped with a CaMPaMS. The second and the third sections consist of Yes/No questions adopted and customized from [279] seeking information about the completeness and the correctness of the scenario respectively, which are among the characteristics of excellent requirements [279]. The fourth section consists of Yes/No questions seeking information about all the domain requirements that must be satisfied in CaMPaMF. The fifth section consists of Yes/No questions seeking information about other issues related to monitoring patients who suffer from more than one chronic disease. The questions in the fifth and sixth sections are based on the identified feature model and the abstract use case model. The experts were required to answer these questions by saying “Yes” if they agreed, “No” if they disagreed, and “Do not know” if they neither agreed nor disagreed. Additionally, the experts were encouraged to write any further comments they felt were relevant in the sixth section. They were expected to give general comments based on their understanding of the scenario. Experts were also asked to write down their demographic information such as specialization, age, experience, and gender in the seventh section.

3.7. Stage 3: Prescriptive Study

This stage is designed to answer the second research question and satisfy the second research objective, which is considered the core of this study. The objectives of this stage are to use the understanding obtained from the previous stages to describe the proposed application framework using an architectural design and then design and implement the proposed application framework based on the constructed architectural design. The outcomes of this stage include: (1) an architectural model that identifies the key components of the proposed application framework; and (2) a detail design and implementation of the proposed application framework. The resulting framework artefact was used as a starting point for the framework evaluation in the next stage [244].

As discussed in Section 2.4, framework development activities include: domain analysis, architectural design, framework design, framework implementation, framework testing, and documentation [15]. The first activity was discussed in the descriptive study 1 (Stage 2), while the second, third, and fourth activities were addressed in this stage. A comprehensive study was therefore conducted to satisfy these three development activities. To achieve the objectives of this stage, an architectural design process was conducted to address the second activity of framework development followed by a framework design and implementation process to address both the third and the fourth activities of framework development. To ensure the validity of the outcomes and to be able to use them in this research with greater confidence, the outcomes were published in two journal articles [79, 87]. The following sub-sections discuss the two processes that were used to accomplish this stage.

3.7.1. Architectural Design Process

In this research, the CaMPaMF architectural design was created based on multiple reusability aspects: (1) design guidelines; (2) design rules; (3) design principles; (4) reusability factors; and (5) amount of reuse. The architectural design is a transitional process between a domain analysis process (requirements engineering) and a framework design process to describe a structural organization of the primary components of the proposed CaMPaMF and the relationships between them. The resulting domain model of the domain analysis process was used as an input for the architectural design research process. The outcome of this process is an architectural model that provides a primary input for the framework design process.

There are three common steps followed in the literature to construct an architectural design [1, 144, 151] : (1) identify quality attributes that should be satisfied; (2) identify suitable architecture styles that satisfy the identified quality attributes; and (3) construct an architectural diagram by structuring components and relationships between them based on the identified architecture styles.

3.7.2. Framework Design and Implementation Process

The MDA approach consists of three essential development activities: analysis, low-level design, and coding [90, 91]. These essential processes were adopted and customized in this research as three steps, which are PIM development, PSM development, and code development respectively. To facilitate applying the MDA approach, the Enterprise Architect tool [280] was used, because it supports an automated transformation of the PIM to PSM and from PSM to code using built-in transformation rules. The Enterprise Architect tool also supports UML 2.3 based

modelling and can generate code in C# [281]. The following sub-sections elaborate on these three essential development steps.

3.7.2.1. PIM Development Step

The objective of this step is to develop a PIM that cannot be affected by rapid changes in technology. To construct the PIM, there are three common steps applied in the literature [90, 91, 282]. First, a UML class diagram should be constructed based on the application framework reusability model and the identified domain requirements. Second, the class diagram should be refined by using four common techniques: hot spots, frozen spots, design patterns [23, 160], and design principles [161]. These techniques are required to meet the identified domain requirements [70, 74]. Third, a UML sequence diagram was used to show the interaction between the framework components and to provide scenarios for application developers to help them to better reuse the CaMPaMF. These scenarios can be used later as a part of the framework documentation.

3.7.2.2. PSM Development Step

The objective of this step is to transform the resulting PIM to PSM using a C# model transformation. This transformation process is performed using an automated tool. This transformation takes the resulting PIM as an input and generates a PSM using special C# stereotypes. The C# platform technology was selected to support the following identified domain requirements. First, it supports soft real-time systems [283], required to support continuous monitoring. Second, it supports asynchronous method calls [284], required to support a non-blocking communication with an

unlimited number of sensors and unlimited number of mobile monitoring applications. Last, it supports cross-platform mobile development [285, 286]. In fact, with the emergence of the Mono project, C# can be used to develop mobile applications that can be executed on various platforms, including iOS, Android, and Microsoft Windows Phone [285, 286]. Mono "is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C# and the Common Language Runtime" [287]. Table 3.10 shows that C# is the best for writing native applications across various mobile platforms in comparison with other programming languages.

3.7.2.3. Code Development Step

The objective of this step is to transform the PSM into a code model that is required to implement a system [90, 91]. In this research, the resulting PSM should be transformed to C# code using an automated tool. Then some manual implementation should be developed. However, the PSM development step typically is automated [163], while the code development step is partially automated in this research.

Table 3.10

Native Mobile Platform Languages

Programming languages	iOS	Android	Windows Phone
C/C++	✓	✓	
Objective-C	✓		
Java		✓	
Visual Basic.Net			✓
C#	✓	✓	✓

Adopted from [286]

The smartphone was used as the mobile phone technology to provide a platform to host the designed CaMPaMF. C# programming language was used for development

on top of the Android operating system, using the Android virtual device manager with Mono as the mobile development platform. Wireless sensors were simulated to provide sensed data. It is also important to mention that this research is neither covering the communication process of connecting to sensors, nor sending any data to a backend server. Furthermore, this research is neither covering the process of analysing the biomedical signals, nor analysing the physical activity signals.

The CaMPaMF was implemented in three projects: (1) CaMPaMF.Core that contains all the framework interfaces, discussed in Section 5.5.1, provided in a single component that are shared among the other two projects; (2) CaMPaMF.Core.CCL, which is dependent on the CaMPaMF.Core and contains the framework default implementation of the Context Characterization Layer (CCL); (3) CaMPaMF.Core.CML, which is dependent on the CaMPaMF.Core and contains the framework default implementation of the Context Monitoring Layer (CML). In addition to these three projects, the SimpleInjector open source project was adopted and used as a container component to simplify framework initialization [288].

The SimpleInjector is used to initialize CaMPaMF by wiring up the default implementation in both the CaMPaMF.Core.CCL and CaMPaMF.Core.CML projects with their corresponding interfaces in the CaMPaMF.Core project. In other words, framework initialization aims to wire all the framework interfaces with their suitable concrete implementations in order for the framework to work properly, which is accomplished by a bootstrapper. A bootstrapper is "the little program that gets the big program going" [289]. In this research, the bootstrapper uses the SimpleInjector component that provides a container to simplify mapping each interface with its

suitable implementation. Additionally, the container is used to instantiate the framework components based on the mapping between the framework interfaces and their concrete classes. The primary reason for using the SimpleInjector component is to support dependency injection. Dependency injection is defined as "set of software design principles and patterns that enable us to develop loosely coupled code" [290]. The purpose of developing loosely coupled code is to push framework extensibility to its ultimate [99].

3.8. Stage 4: Descriptive Study 2

This stage is designed to answer the third research question and satisfy the third research objective. The objectives of this stage are to evaluate the resulted design and implementation of CaMPaMF as well as to derive the conclusions of this research project based on the results. The outcomes of this stage prove the framework reusability and provide documentation to be used by developers to reuse the framework. To achieve the objectives of this stage, the framework was tested and the documentation process was conducted. The following sub-section describes the process used to accomplish this stage.

3.8.1. Framework Testing and Documentation Process

The objective of this process is to evaluate the reusability of the CaMPaMF. To achieve this objective, five steps were followed: (1) framework design guidelines application to evaluate the applicability of design guidelines aspect; (2) framework reusability evaluation using reusability model to evaluate the applicability of four reusability aspects, which are: design rules (complexity, coupling, cohesiveness),

design principles (modularity, simplicity, abstraction), and reusability factors (flexibility and understandability); (3) prototyping and documentation; (4) amount of reuse calculation, and (5) framework reusability evaluation using software expert review. Together, these five steps capture the reusability of CaMPaMF from different aspects. The following sub-sections discuss the five steps used to accomplish this process.

3.8.1.1. Framework Design Guidelines Application Step

Framework design guidelines provide a common language for communication between framework authors and framework users. Appling framework design guidelines confirms framework reusability [9, 10], which are among the primary characteristics that distinguish successful frameworks [27, 51, 67].

In this research, a Microsoft static code analysis tool called FxCop (version 10.0) was used to analyse the compiled code based on a number of design guidelines (rules) described in [9]. These design guidelines are: (1) naming guidelines that are used for naming assemblies, namespaces, types, and members in classes; (2) type guidelines that are used for using static and abstract classes, interfaces, enumerations, and structures; (3) member guidelines that are used for designing and using properties, methods, constructors, fields, events, operators, and parameters; (4) extensibility guidelines that are used for extensibility mechanisms such as sub-classing, using events, virtual members, and callbacks; (5) exceptions guidelines for designing, throwing, and catching exceptions; (6) usage guidelines that are used for using common types such as arrays, attributes, and collections, supporting serialization, and

overloading equality operators; and (7) common design patterns that are used for choosing and implementing dependency properties and the dispose pattern.

The FxCop tool inspects compiled code for 211 different possible code violations of the design guidelines and provides recommendations for implementing a well-designed framework that is reusable [9, 23]. When applying the framework design guidelines using the FxCop tool, a number of problems were discovered and so an iterative back-and-forth process was conducted between the framework design and implementation process and the framework verification process to optimize the framework design and implementation.

3.8.1.2. Framework Reusability Evaluation Using Reusability Model Step

In this step, the framework reusability model introduced in [10] was adopted for evaluating the reusability of CaMPaMF. This model is discussed in detail in Section 2.2.4. The reusability of CaMPaMF was evaluated based on four activities, which are: (1) calculate the values of the metrics; (2) identify the threshold for each metric; (3) identify outliers; (4) design review. The values of the metrics were calculated by the support of Microsoft Visual Studio 2013 Code Map [291] and Code Metrics [292].

3.8.1.3. Prototyping and Documentation Step

A successful application framework must provide a base for developing various CaMPaMS within the biomedical informatics domain and therefore must be reusable. Consequently, the objective of this step is to evaluate the key characteristic of a successful framework, which is framework reusability [27]. In order to achieve this, the prototype approach [15-17, 165, 166] was adopted and used to implement three

examples of CaMPaMS on top of CaMPaMF as a proof of concept towards illustrating the framework reusability. This approach was used in several related studies to evaluate frameworks [48, 51, 52, 67, 77, 226, 293]. The implementation of the three CaMPaMS provides guideline documentation for application developers to help them to better reuse the existing framework's frozen spots and to extend the existing hot spots of the CaMPaMF.

In this research, the three CaMPaMS prototypes are: a diabetes CaMPaMS, an epilepsy CaMPaMS, and a hypertension CaMPaMS. These CaMPaMS demonstrate how the framework can be reused and show how the framework can be extended to satisfy the specific requirements of each CaMPaMS. These CaMPaMS were implemented based on the requirements of CaMPaMS derived from the three scenarios which are described in Chapter 4 and which were validated by domain experts.

3.8.1.4. Amount of Reuse Calculation Step

The amount of reuse is calculated to measure how much reuse is achieved [18]. In fact, there is no single metric can be used to capture the effect of reuse [11]. Therefore, the amount of reuse should be measured by multiple metrics, each of which represent different points of view that complement each other to provide a complete picture of the effects of reuse [11].

In this research, the amount of reuse of each prototype was calculated by three metrics that use different types of data. These metrics are: (1) reuse level metric that captures the effect of reuse in terms of the number of reused items [18]; (2) reuse frequency metric that captures the effect of reuse in terms of the number of references to reused

items [18]; (3) reuse size and frequency metric that captures the effect of reuse in terms of the number of references to reused items by taking into account the size of the items (lines of code) [294]. The values of the metrics were calculated by the support of Microsoft Visual Studio 2013 Code Map [291] and Code Metrics [292].

3.8.1.5. Framework Reusability Evaluation Using Expert Review Step

The objective of this step is to evaluate the reusability of the resulting CaMPaMF by software experts using a set of simple Yes/No questions. In this research, four software experts were selected from software industry and contacted via email based on specific criteria. Accordingly, the experts are people who have certified knowledge in the area of software design and a minimum of 10 years' experience [19].

The software expert review form was developed as shown in Appendix L. The form consists of five sections. The first three sections consist of Yes/No questions based on the adopted application framework reusability model [10], which are seeking information about the design rules, principles, and factors that affect the CaMPaMF reusability. The experts were required to answer these questions by saying "Yes, without modification" if they agreed without any suggestion for improvement, "Yes, with modification" if they agreed with a suggestion for improvement, "No" if they disagreed. Additionally, the experts were encouraged to write any further comments they felt were relevant in the fourth section. The experts were also asked to write down their demographic information such as specialization, age, experience, and gender in the fifth section.

Moreover, the software expert review form consists of five appendixes that contain detail information related to the CaMPaMF to help the expert to answer the questions, which are the CaMPaMF class diagram, the CaMPaMF interfaces description, the CaMPaMF default implementation, the adopted application framework reusability model, and the application of multi-metric approach to CaMPaMF.

3.9. Summary

In conclusion, the objectives of this research were achieved using four research stages: research clarification, descriptive study 1, prescriptive study, and descriptive study 2. The research clarification stage consisted of two research processes, which are the literature review and the lacks and gaps identification. Together, these research processes were used to clarify the research problem. The descriptive study 1 stage consisted of the domain analysis process. This research process was used to achieve the first research objective. The prescriptive study stage consisted of two research processes, which are the architectural design and the framework design and implementation. Together, these research processes were used to achieve the second research objective. The descriptive study 2 stage consisted of the framework testing and documentation process. This research process was used to achieve the third research objective. All the research stages were used together to achieve the aim of this research.

CHAPTER FOUR
DOMAIN ANALYSIS

4.1. Overview

This chapter presents the two main outcomes of the domain analysis. The step-by-step implementation of the domain modelling activities to develop a domain model, including the feature modelling and the abstract use case modelling as visual representations of the domain requirements, is described. The domain model validation activities are also outlined. Finally, a summary of the chapter is presented.

4.2. Feature Modelling

In this research, the source of features was the domain literature. The domain literature includes 20 software frameworks that were either designed in the biomedical informatics domain or can be applied in that domain. From analysis of the identified source of features, a set of common features was derived. Table 4.11 summarizes these five features that have to be addressed when designing CaMPaMF.

Table 4.11

Common Features of CaMPaMF

ID	Common features	Literature
1	Anywhere and anytime monitoring	[45, 50, 51, 61, 67, 71, 76, 192, 194, 226, 227]
2	Real-time continuous monitoring	[45, 46, 49, 51, 61, 76, 192, 226, 228, 229]
3	Unlimited number of sensors	[46, 48-52, 67, 71, 76-78, 192, 194, 226-229]
4	Unlimited number of monitoring applications	[45, 46, 48, 50, 51, 67, 78, 192, 227, 228, 230]
5	Context-aware monitoring query	[45, 46, 48-52, 69, 71, 78, 192, 194, 227-229]

In addition, the context-aware monitoring query feature (ID 5 in Table 4.11), has a set of three common sub-features. Table 4.12 summarizes these sub-features, which have to be addressed when designing any context-aware monitoring query. The query notification feature (ID 5.1 in Table 4.12) has a set of two alternative variable grouped features.

Table 4.12

Common Features of Context-Aware Monitoring Query Feature

ID	Common features	Literature
5.1	Query notification	[50, 51, 192]
5.2	Query evaluation approach using rule-based reasoning	[45, 49, 50, 192, 194, 228]
5.3	Query element	[49, 51, 67, 192, 227]

Table 4.13 summarizes these grouped features that have to be addressed when designing the query notification. The variable duration notification feature (ID 5.1.2 in Table 4.13) has a feature attribute, called minutes, with an integer data type, which represents the duration in minutes. The query element feature (ID 5.3 in Table 4.12) has two common dimensions, each of which has a set of alternative variable grouped features. The first dimension, which is the context information type of query element, should have exactly one value selected among its alternative variable grouped features, and the second common dimension, which is the context data source of query element, should also have exactly one value selected among its alternative variable grouped features.

Table 4.13

Variable Features of Query Alarm Feature

ID	Variable Features	Literature
5.1.1	Instant notification	[50, 51, 192]
5.1.2	Duration notification	[51, 67]

Table 4.14 summarizes these two common dimensions and their alternative variable grouped features that have to be addressed when designing the query element.

Table 4.14

Two Common Dimensions of Alternative Variable Features of the Query Element Feature

Dimensions	ID	Variable Features	Literature
5.3.1. Context information type	5.3.1.1	Measurable medical context	[45, 46, 49, 50, 52, 192, 194, 227-229]
	5.3.1.2	Non-measurable medical context	[67]
	5.3.1.3	Prescribed medication medical context	[45, 67, 194]
	5.3.1.4	Risk factors medical context	[45, 50, 192, 227]
	5.3.1.5	Physical activity context	[45, 52, 68, 194, 229]
	5.3.1.6	Environmental context	[45, 46, 49, 50, 52, 192, 227]
5.3.2. Context data source	5.3.2.1	Wireless body sensors	[45, 46, 49, 50, 52, 192, 194, 227-229]
	5.3.2.2	Wireless environmental sensors	[45, 46, 49, 50, 52, 192, 227]
	5.3.2.3	Patient profile	[45, 50, 192, 194, 227]
	5.3.2.4	Mobile graphical user interface	[67]
	5.3.2.5	Patient profile hosted on patient mobile device	[192]

Figure 4.5 illustrates the resulting feature diagram based on the identified set of common and variable features. As shown in Figure 4.5, the feature diagram organizes the identified common and variable features into a hierarchy and classifies them according to their types and cardinality. The following sub-sections present an

explanation of these features including a semantic description about each feature and a rationale for selecting each of them.

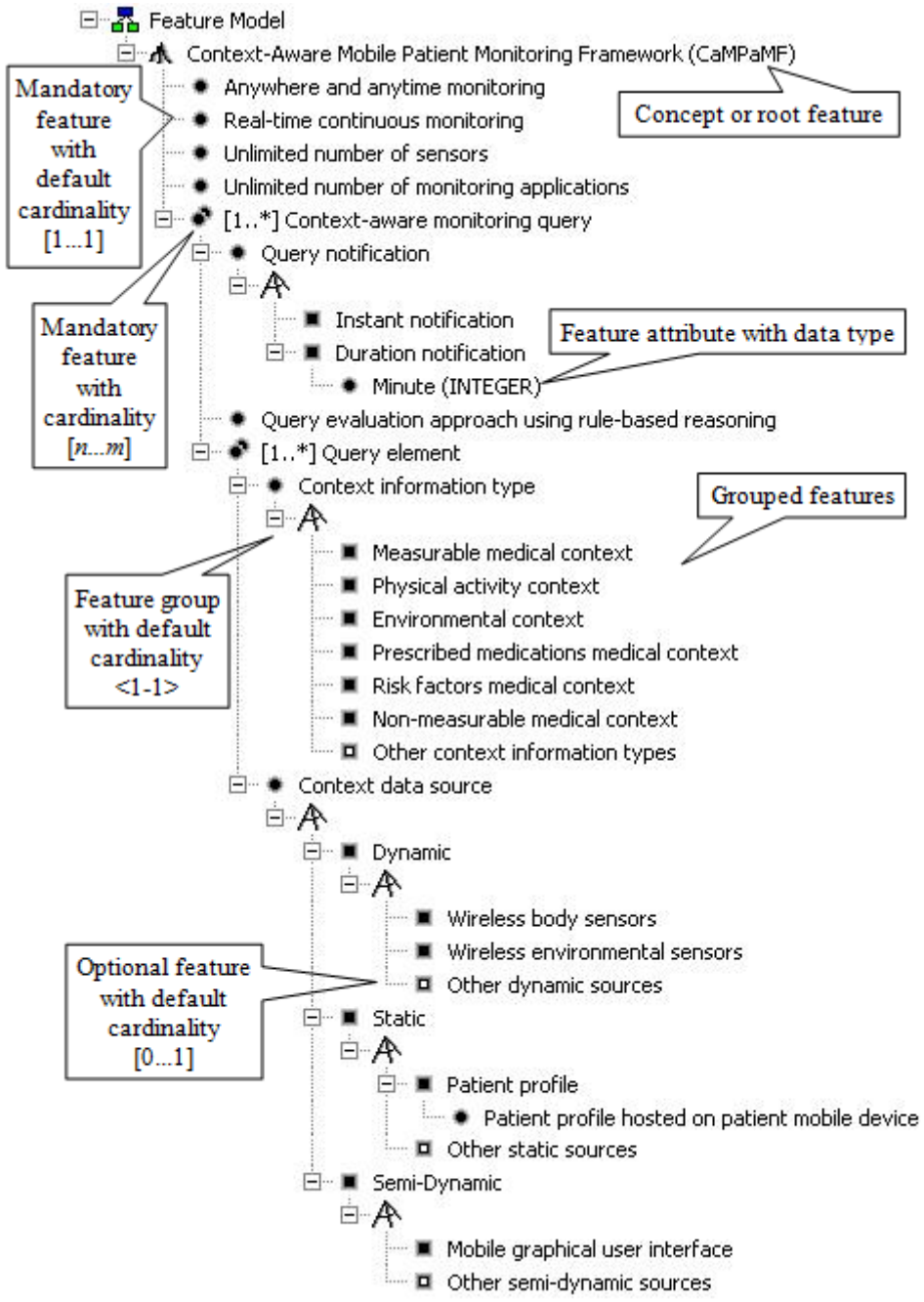

Figure 4.5. A feature model to design CaMPaMF

4.2.1. Anywhere, Anytime Monitoring

This common feature allows the monitoring of patients anywhere, anytime to allow instant detection of abnormal health events. This enables MPMS to react immediately by, for example, calling healthcare professionals who can make the suitable clinical decisions [45, 60]. Additionally, monitoring patients anywhere, anytime can improve patient life styles by allowing them to be more independent, more flexible, and mobile while being monitored [45, 60, 181].

4.2.2. Real-Time Continuous Monitoring

This common feature allows instant detection of patients' abnormal health events [45, 60]. Real-time continuous monitoring allows MPMS to react immediately [45, 60]. For example, the system can call healthcare professionals who can make suitable decisions [45, 55, 60]. Therefore, this feature provides proactive healthcare to protect patients from future complications [55], especially those who suffer from chronic diseases [54].

4.2.3. Unlimited Number of Sensors

This common feature requires enabling the framework design to add an unlimited number of sensors at design time. However, sensors play a primary role in supporting CaMPaMS [55]. In fact, the greater the number of sensors, the more comprehensive the information gained. This enhances the detection efficiency of a patient's medical situation [46].

4.2.4. Unlimited Number of Monitoring Applications

This common feature requires enabling the framework design to support an unlimited number of CaMPaMS at design time. In fact, the elderly who suffer from chronic diseases need to be monitored by different dedicated applications [54]. For example, they might require an application for monitoring hypertension and another one for monitoring diabetes.

4.2.5. Context-Aware Monitoring Query

This common feature is required in CaMPaMS to allow effective detection of a patient's medical situation such as high BP based on their contextual information such as physical activities and vital signs. In fact, the framework design should allow each CaMPaMS to register one or more context-aware monitoring queries which are used to detect a change in a patient's medical situation such as a change from normal to high BP. Accordingly, these systems can change their behaviours by adapting to the changes of a patient's medical situation, for example by triggering an alarm [49, 68, 69]. Each query consists of three common sub-features: (1) query notification; (2) query evaluation; and (3) query elements, which are described in the following sub-sections.

4.2.5.1. Query Notification

The query notification common feature is responsible for notifying the MPMS, based on their registered queries, once the patient's medical situation has changed, whether the change was from normal to abnormal or vice versa. In this research, the concept of query notification feature is adapted from the SeeMon framework [51]. Unlike

SeeMon, the query notification feature in this research has two alternative sub-features: instant notification feature and duration notification feature. One of these sub-features should be selected to identify the notification technique of the query notification feature. The instant notification variable feature provides an immediate notification to the MPMS once its registered context-aware monitoring query evaluation is changed from true to false or vice versa. The duration notification variable feature has an attribute, called minutes, with an integer data type. If this feature is selected, the query notification will not notify the MPMS unless this duration elapsed without any change to the patient's medical situation. For example, there is a registered query that monitors a patient's medical situation such as the high BP. This query depends on the patient's vital signs, including systolic and diastolic BP, and their physical activity as shown in Figure 4.6.

High Blood Pressure = (Systolic Blood Pressure > 120 mm Hg)
(Diastolic Blood Pressure > 80 mm Hg)
(Physical Activity = "Sitting")

Figure 4.6. High BP monitoring query

If the patient was running, the normal systolic and diastolic BP will be higher than the normal systolic and diastolic BP during sitting [68, 224]. In fact, if the patient suddenly stopped running and sits down, then the query will be evaluated as true and notify the application that the patient's medical situation has changed from normal BP to high BP, which is considered a false alarm [194]. This problem appears because both systolic and diastolic BP need some time to recover to their norm when sitting after running. Therefore, there is a need for the duration sub-feature in such cases to avoid false alarms. For example, if the query shown in Figure 4.6 has a duration of 5 minutes,

which is enough to allow systolic and diastolic BP to recover to their norm, then the framework will not raise a false alarm.

4.2.5.2. Query Evaluation

In the query evaluation common feature, the context evaluation approach aims to detect the change in high-level context based on low-level context information [192]. As elaborated in Section 2.6.2.6, rule-based reasoning is one of the used solutions for context reasoning of PMS in biomedical informatics domain [49, 50, 192].

4.2.5.3. Query Elements

The query elements common feature represents one or more elements that form the building blocks of each context-aware monitoring query, which allows effective detection of patient medical situations such as high BP. In fact, detecting high BP is based on a number of patient context information types, which are retrieved from a number of context data sources. For example, detecting high BP is based on the following patient context information. First, identify vital signs such as systolic and diastolic BP [53] which are retrieved from WBS. Second, identify symptoms such as headache [67] by asking Yes/No questions using mobile device graphical user interface. Third, identify physical activity such as running [68] which is retrieved from WBS. Fourth, identify room temperature [68] which is retrieved from wireless environmental sensors. Each query element consists of only one context information type that is retrieved from one context information source. Each context-aware monitoring query should have one or more query elements which can be selected based on the patient's medical situation. Query elements have two sub-features: context

information types and context data sources, which are described in the following sub-sections.

4.2.5.3.1. Context Information Types

The context information types common dimension feature represents a property of the query elements feature, which has an exact single value. This value should be selected among a set of alternative variable grouped features. These variable grouped features represent three types of patient context information, which are classified in this research as: (1) medical context information type; (2) physical activity context information type; and (3) environmental context information type. In addition, the medical context information type is classified further as: (1) measurable medical context information type; (2) non-measurable medical context information type; (3) prescribed medication context information type; and (4) risk factors medical context information type.

The measurable medical context information type feature mainly includes a patient's vital signs. There are five standard vital signs that must be measured and continually monitored. These are: BT, RR, HR, BP, and ECG [40]. In fact, the interpretation of their values, whether they are normal or not, depends on other types of medical context information such as risk factors and prescribed medications.

The non-measurable medical context information type feature involves medical symptoms that are difficult to measure with sensors [67] such as dizziness or vomiting. While rarely adopted in the literature, this context information type complements the measurable medical context. For example, monitoring high BP requires monitoring

non-measurable medical contexts such as headache and constipation [67]. Monitoring these non-measurable medical symptoms complements monitoring measurable medical context such as BP and HR vital signs [67].

The prescribed medications context information type feature provides information about the current prescribed medications for a patient [67, 69]. Also rarely adopted in the literature, it affects the patient's vital signs. Therefore, healthcare professionals assess the effects of prescribed medications for a patient to evaluate the patient's response to treatment [67]. For example, a healthcare professional can manage high BP by prescribing a medication. Then the professional can monitor the effects of such medications on a patient's BP to assess the patient's response to treatment, and take the appropriate medical decisions [67].

The risk factors context information type represents risk factors or health risks. These risk factors are adopted in the literature to represent the personal health information that changes infrequently [192, 218, 227]. These factors are countless, and each disease has a number of associated risk factors. For instance, there are a number of risk factors associated with hypertension such as alcohol and obesity. These risk factors jointly are responsible for more than 75% of the deaths of hypertensive people [53]. Furthermore, they affect the normal readings of vital signs [69]. For example, smoking affects the normal cholesterol level [53].

The physical activity context information type feature represents the patient's physical activities such as walking, running or sleeping. These physical activities have direct effects on the patient's normal vital signs. For example, normal HR while running or

climbing up stairs is higher than while walking or lying down [194]. Similarly, normal BP during sitting or sleeping is less than during eating or doing physical exercise such as running [68, 224].

The environmental context information type feature is commonly used in the literature to represent patient location [46, 51, 52, 71, 227] as a fundamental requirement to provide rescue services. It is also used to provide information about the surrounding environment affecting a patient's medical state such as humidity and room temperature. Aside from this, this feature contributes to disease monitoring. For example, patients with voluntary muscle movement disorders can benefit from monitoring floor humidity to prevent them from falling [68]. In addition, this feature affects vital signs. For instance, room temperature can affect heartbeat, which in turns affects BP [68].

4.2.5.3.2. Context Data Sources

The context data sources common dimension feature represents a property of the query elements feature, which has an exact single value. This value should be selected from a set of alternative variable grouped features. These variable grouped features represent three sources of patient context information, which are classified in this research as: (1) sensors context information source; (2) patient profile context information source; and (3) mobile graphical user interface context information source. In addition the sensors context information source is classified further as: (1) WBS; and (2) wireless environmental sensors.

The WBS context source feature is used as a common data source in the literature for the measurable medical context information. It is also used as a primary data source for the physical activity context in most previous studies that have adopted this type of context [68, 69, 194].

The wireless environmental sensors context source feature is used as an essential data source for the environmental context in most previous studies that have adopted this type of context [49, 68, 69, 237]. It also plays a primary role in supporting CaMPaMS by providing context information that can be measured continuously during patients' normal daily lives [55].

The patient profile context source feature is commonly used in the literature as a main data source for obtaining risk factors and the prescribed medication context. Using this data source contributes to the accuracy of CaMPaMS [242]. Moreover, it plays a key role in personalizing and optimizing the patient monitoring process [192]. For example, alcohol consumption is one of the risk factors associated with hypertension [53], and this can be obtained from this data source. In fact, alcohol consumption affects BP and thus has to be considered when monitoring a patient with hypertension [53]. However, if a patient does not consume alcohol, then the patient monitoring process has to be personalized by ignoring this factor to optimize the monitoring process.

Using a mobile patient profile is required as a main data source for obtaining risk factors and prescribed medication context. Hosting the profile on the patient's mobile device can contribute significantly to the design of CaMPaMF. For example, it

supports the privacy protection of the patient's contextual data [51]. Furthermore, it avoids the continuous network communication costs required to transmit and receive data to and from a backend server [51, 194, 229]. Aside from this, it avoids wireless network interruptions [194]. Moreover, a mobile patient profile can support context awareness and adaptation through direct detection of context changes [45, 51]. Finally, it supports real-time continuous patient monitoring [52, 229] anywhere, anytime [205].

The mobile graphical user interface context source feature supports obtaining data directly from patients through manual answering of Yes/No questions. While rarely adopted in the literature [67], it is the main data source for obtaining the non-measurable medical context. Moreover, it plays a primary role in supporting CaMPaMS with dynamic context information that can neither be measured by wireless sensors nor retrieved from the mobile patient profile [67].

4.3. Abstract Use Case Modelling

Figure 4.7 shows the resulting abstract use case model. The primary actors of application frameworks are various systems that can be developed on top of the frameworks. In this research, there are various CaMPaMS that can be developed on top of the proposed CaMPaMF, including a hypertension CaMPaMS, an epilepsy CaMPaMS, and a diabetes CaMPaMS.

As shown in Figure 4.7, these actors were abstracted in this research as CaMPaMS. The abstract use cases shown in the figure are then identified by grouping all concrete use cases that have the same behaviours under an abstract actor. The specifications of the abstract use cases are given in Appendix C.

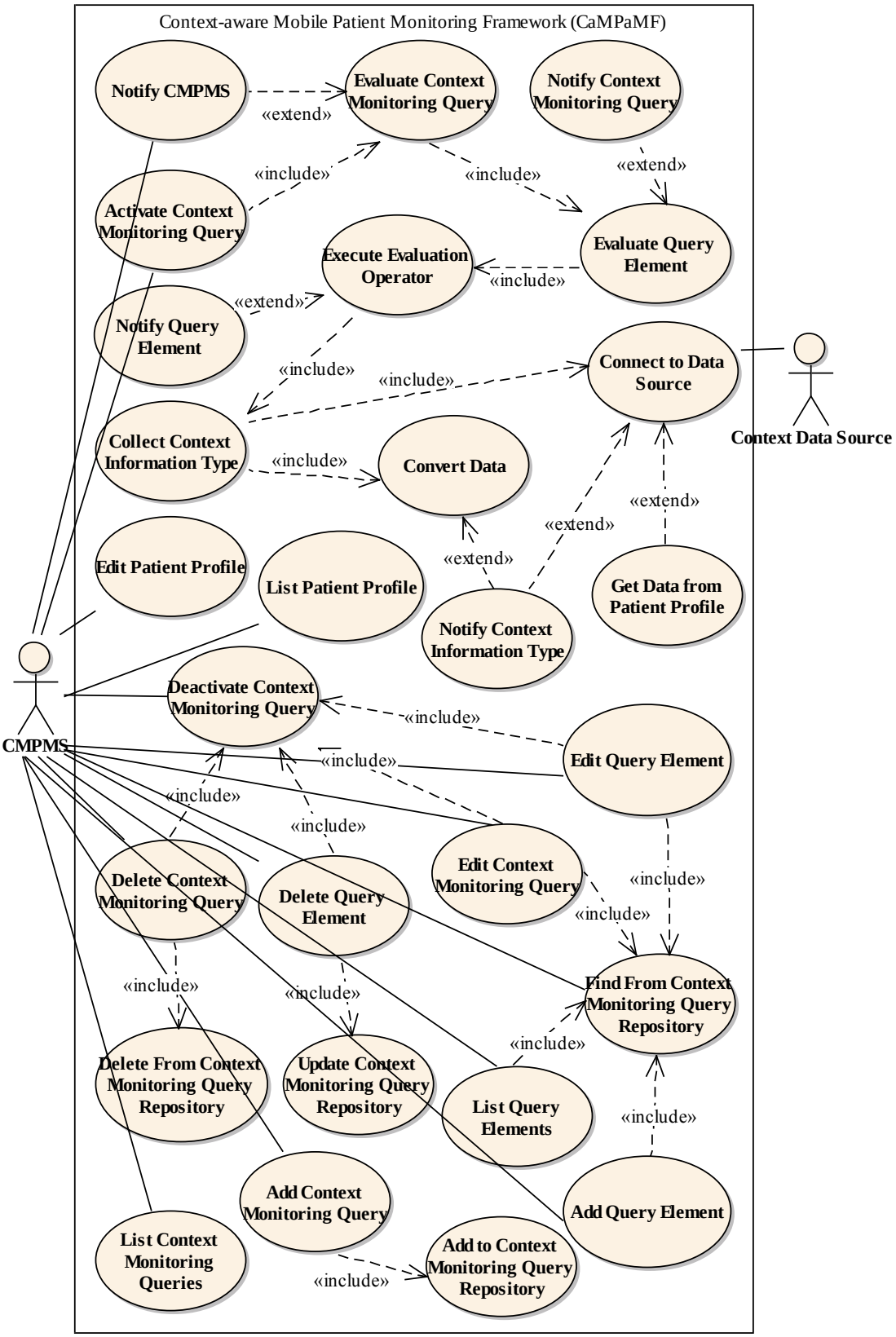

Figure 4.7. Abstract use case model

4.4. Domain Model Validation

This section is a presentation of domain model validation outcomes, which include document features, authoring scenarios, and expert review. Document features were covered earlier in Section 4.2. The following sub-sections present the outcomes of the second and third activities of domain model validation.

4.4.1. Authoring Scenarios

This sub-section presents three idealized scenarios authored based on published medical guidelines and scenarios from the literature related to CaMPaMS.

4.4.1.1. Scenario of Monitoring a Hypertensive Patient

This section presents a scenario of the daily activities involved in monitoring a patient who suffers from hypertension and who is equipped with intelligent MPMS. This scenario is adopted from [68] and has been customized based on medical guidelines for monitoring BP [270].

Mohammad is a patient who suffers from hypertension. He lives alone in a house, thus his healthcare professional has equipped him with a hypertension intelligent MPMS. The intelligent MPMS aims to provide real-time management and protection from the complications of chronic diseases anywhere, anytime. This is achieved through continuously performing repeatable tasks that are required for monitoring patients to complement the role of healthcare professionals outside the boundaries of healthcare organizations. The intelligent MPMS is installed and run on Mohammad's mobile device to provide him with 24-hour ambulatory BP monitoring while he performs his usual activities of daily life, including waking and sleeping hours.

Mohammad's healthcare professional enters any number of monitoring queries (rules), which are required to monitor a hypertensive patient, into the intelligent MPMS. Based on these queries the intelligent MPMS raises an alarm when Mohammad's BP becomes high. For example, a monitoring query could check that normal ambulatory systolic BP is equal to or lower than 130 mmHg and that diastolic BP is equal to or lower than 80 mmHg [270]. When Mohammad's BP becomes high, his vital signs, current physical activity, prescribed medication, and risk factors are also checked before an alarm is raised. Mohammad's vital signs and his current physical activity are retrieved using WBS, while his prescribed medication and risk factors are retrieved using a medical profile installed on his mobile device.

One day, Mohammad wakes up and leaves his home to go to the gym. While he is walking on the treadmill, the intelligent MPMS continuously receives measurements from his WBS. At a specific time, the following measurements were received: systolic BP (142 mmHg), diastolic BP (85.5 mmHg), physical activity (walking), as well as prescribed medication (calcium-channel blocker) and risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease) that are retrieved from Mohammad's medical profile. Based on the predefined queries, when the patient's physical activity is walking, the systolic BP is increased by +12.0 mmHg, while the diastolic BP is increased by +5.5 mmHg [270]. Therefore, the system considers Mohammad's BP to be normal.

After 1 hour of walking, Mohammad finishes his exercise and sits down to rest. The following measurements are received: systolic BP (142 mmHg), diastolic BP (85.5 mmHg), physical activity (resting), prescribed medication (calcium-channel blocker),

risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, when the patient's physical activity is resting, the systolic and the diastolic BP is increased by +0 mmHg [270]. However, when the patient's physical activity changes from any physical activity to rest, the system will not raise an alarm until a predefined time period has elapsed, which is the time required for a patient to rest, such as 30 minutes. After 30 minutes, the following measurements were received: systolic BP (130 mmHg), diastolic BP (80 mmHg), physical activity (resting), prescribed medication (calcium-channel blocker), risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Therefore, the system considers Mohammad's BP to be normal.

At night, Mohammad goes to sleep. While he is sleeping, the following measurements were received: systolic BP (120 mmHg), diastolic BP (72.4 mmHg), physical activity (sleeping), prescribed medication (calcium-channel blocker), risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, when the patient's physical activity is sleeping, the systolic BP is decreased by -10.0 mmHg, while the diastolic BP is decreased by -7.6 mmHg [270]. Therefore, the system considers Mohammad's BP to be normal.

On the next day, while Mohammad is watching TV, the following measurements were received: systolic BP (136 mmHg), diastolic BP (86 mmHg), physical activity (watching TV), prescribed medication (calcium-channel blocker), risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, when the patient's physical activity is watching TV, the systolic BP and diastolic BP are increased by +0.3 mmHg and +1.1 mmHg

respectively [270]. Therefore, the system considers Mohammad's BP to be high. Consequently, the system raises an alarm to warn Mohammad and informs his healthcare professional. Accordingly, the healthcare professional can make the suitable decisions and provide Mohammad with emergency assistance if required.

4.4.1.2. Scenario of Monitoring a Diabetic Patient

This section presents a scenario of the daily activities for monitoring a patient who suffers from diabetes and who is equipped with intelligent MPMS. This scenario was adopted from [228] and is customized based on medical guidelines for monitoring diabetes mellitus [271, 273].

Ahmad is a patient who suffers from type 2 diabetes. Because abnormal blood-glucose level can cause serious problems [272], his healthcare professional has equipped him with a diabetes intelligent MPMS. The intelligent MPMS aims to provide real-time management and protection from the complications of chronic diseases anywhere, anytime. This is achieved through continuously performing repeatable tasks that are required for monitoring patients to complement the role of healthcare professionals outside the boundaries of healthcare organizations. The intelligent MPMS is installed and run on Ahmad's mobile device to provide him with 24-hour ambulatory blood-glucose level monitoring, while he performs his usual activities of daily life, including waking and sleeping hours.

Ahmad's healthcare professional enters any number of monitoring queries (rules), which are required to monitor a diabetic patient, into the intelligent MPMS. Based on these queries the MPMS raises an alarm when Ahmad's blood-glucose level becomes

abnormal. For example, a monitoring query could check that the normal fasting blood-glucose level is less than 100 mg/dL and that the normal blood-glucose level 2 hours after a meal is less than 140 mg/dL [271]. When Ahmad's blood-glucose level becomes abnormal, his vital signs, current physical activity, prescribed medication, and risk factors, and environmental information are also checked, and Yes/No questions asked before an alarm is raised. Ahmad's vital signs and his current physical activity are retrieved using WBS. Similarly, environmental information such as his location is retrieved using wireless environmental sensors. His risk factors are retrieved using a medical profile installed on his mobile device. The Yes/No questions are displayed on Ahmad's mobile device, which he uses to answer them.

One morning, Ahmad leaves his house without taking his breakfast and drives his car to the family farm to spend a pleasant holiday. While he is driving, the intelligent MPMS continuously receives measurements from his wireless sensors. At a specific time, the following measurements were received: blood-glucose level (72 mg/dL) and physical activity (driving), as well as risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease) that are retrieved from Ahmad's medical profile. Based on the predefined queries, when the blood-glucose level approaches the minimum normal fasting blood-glucose level (70 mg/dL), the patient is considered under threat of hypoglycaemia [273]. According to the predefined queries, the intelligent MPMS advises Ahmad to consume something that has about 15 grams of carbohydrates such as drinking ½ cup of fruit juice [273]. Consequently, Ahmad stops his car safely and drinks ½ cup of fruit juice, then continues on his journey.

After 15 minutes, the following measurements were received: blood-glucose level (69 mg/dL), an increase of pulse, an increase of level of perspiration, physical activity (driving), and risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, when the blood-glucose level is below 70 mg/dL, the patient may lose consciousness and need immediate assistance [273]. Therefore, the intelligent MPMS considers Ahmad's medical situation is hypoglycaemia. Accordingly, the intelligent MPMS raises an alarm to warn Ahmad about his medical situation. In addition, the MPMS informs Ahmad's healthcare professional of his current medical situation and location. This enables the healthcare professional to make the necessary decisions based on Ahmad's medical situation and provide him, if required, with emergency assistance to his exact location.

Next day, Ahmad wakes up and takes his breakfast at 7:00 am. At 9:00 am, while Ahmad is watching TV, the following measurements were received: blood-glucose level (126 mg/dL), physical activity (watching TV), and risk factors (30 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, if the patient's fasting blood-glucose level is more than or equal 126 mg/dL or if the patient's blood-glucose level 2 hours after a meal is more than or equal 200 mg/dL, then the patient has overt diabetes mellitus [271]. Accordingly, the intelligent MPMS asks Ahmad: "Did you take your breakfast during the last 2 hours?" Ahmad answers with yes. Therefore, the intelligent MPMS considers Ahmad's blood-glucose level to be normal.

4.4.1.3. Scenario of Monitoring an Epileptic Patient

This section presents a scenario of the daily activities for monitoring a patient who suffers from epilepsy and is equipped with intelligent MPMS. This scenario combines and customizes two scenarios for monitoring a patient with epilepsy [45, 274] based on an epilepsy monitoring algorithm [275].

Ali is a 45-year-old man who suffers from epilepsy. Because the epileptic seizures often happen suddenly and unexpectedly, Ali feels limited in his everyday life and unsafe when he is alone [45]. Therefore, his healthcare professional has equipped him with an epilepsy intelligent MPMS. The MPMS aims to provide real-time management and protection from the complications of chronic diseases anywhere anytime. This is achieved through continuously performing repeatable tasks that are required for monitoring patients to complement the role of healthcare professionals outside the boundaries of healthcare organizations. The intelligent MPMS is installed and run on Ali's mobile device to provide him with 24-hour ambulatory epileptic seizure monitoring while he performs his usual activities of daily life, including waking and sleeping hours.

Ali's healthcare professional enters any number of monitoring queries (rules), which are required to monitor an epileptic patient, into the intelligent MPMS. Based on these queries the intelligent MPMS detects that seizure is about to happen and raises an alarm to warn Ali several seconds before. Two examples of such queries are: (1) if a patient's physical activity is sleeping or resting, that is, the heart is beating steadily between 70 and 90 times a minute, and a sudden acceleration of HR of more than 10 beats per minute within the timeframe of 10 seconds is detected, then a seizure is

expected within several seconds and the alarm is raised [275] ; (2) if a patient's physical activity is not resting, such as walking, running or jogging, and the HR acceleration is proportional to the patient's activity level, then the patient's medical situation is considered normal [275] ; otherwise, if the HR acceleration is disproportional to the patient's activity level, then a seizure is expected within several seconds and the alarm is raised [275].

Ali's HR (monitored using an ECG), his current physical activity, environmental information, and risk factors are all taken into account before the alarm is raised. Ali's vital signs and his current physical activity are retrieved using WBS. Similarly, environmental information such as his location is retrieved using wireless environmental sensors. Ali's risk factors are retrieved using a medical profile installed on his mobile device.

One day, Ali leaves his house at 7:30 am and drives his car to work. The intelligent MPMS continuously receives measurements from Ali's WBS and at 7:45 am the following measurements were received: HR acceleration (18 bpm), physical activity (driving), risk factors (45 years old, non-smoker, with ideal body weight, and without any other chronic diseases) that are retrieved from Ali's medical profile. Based on the predefined queries, when the patient's physical activity causes mental stress such as driving a car, the proportional HR acceleration should be about 6 to 11 beats per minute within the timeframe of 10 seconds [275]. Therefore, Ali's HR acceleration is disproportional to his activity level. Accordingly, the intelligent MPMS detects a seizure within several seconds, and instantly raises an alarm to warn Ali that without the intervention of a healthcare professional a seizure will occur. Consequently, Ali

can stop the car safely before the seizure occurs. This prevents him and others from danger. Based on the seizure's severity, the intelligent MPMS informs Ali's healthcare professional about his medical situation and location. Accordingly, the healthcare professional can make the necessary decisions and provide Ali, if required, with emergency assistance to his exact location.

In the weekend, Ali wakes up after a good night's sleep and walks to the nearby park to meet his friends. He walks on a flat, smooth street around 7 km/h, which is classified as heavy work [276]. While he is walking, the following measurements are received: HR (120 bpm), physical activity (walking), risk factors (45 years old, non-smoker, with ideal body weight, and without any other chronic disease). Based on the predefined queries, when the patient's physical activity is walking and the patient's HR acceleration is proportional to the patient's activity level then the patient's medical situation is considered normal [275]. Therefore, the intelligent MPMS considers Ali's HR acceleration to be normal.

4.4.2. Domain Expert Review

This sub-section presents the third activity of domain model validation, which is expert review. The findings of the expert review are elaborated in the following sub-sections.

4.4.2.1. Demographic Profiles of Experts

As shown in Table 4.15, the demographic data collected in this research were the experts' specialization, disease monitoring, age, experience and gender. Four of the 16 (25%) of the experts are neurologists, 6 (37.5%) internists, and 2 (12.5%) endocrinologists. There was 1 nephrologist, 1 internist/endocrinologist, 1

internist/neurologist, and 1 endocrinologist/diabetes specialist (6.25% respectively). These experts were specialized in monitoring patients who suffer from three diseases: diabetes, epilepsy, and hypertension.

Table 4.15

Demographic Profiles of Experts

No.	Specialization	Monitoring disease	Age	Experience (years)	Gender
1	Neurologist	Epilepsy	58	33	Male
2	Internist	Hypertension	64	30	Male
3	Internist + Endocrinologist	Diabetes	47	25	Male
4	Neurologist	Epilepsy	56	30	Male
5	Internist + Neurologist	Epilepsy	65	30	Male
6	Endocrinologist + Diabetes	Diabetes	61	35	Male
7	Neurologist	Epilepsy	41	16	Male
8	Internist	Hypertension	62	20	Male
9	Internist	Diabetes	62	20	Male
10	Internist	Hypertension	61	35	Male
11	Neurologist	Epilepsy	49	20	Male
12	Internist	Diabetes	58	30	Male
13	Nephrologist	Hypertension	58	34	Male
14	Internist	Hypertension	62	20	Female
15	Endocrinologist	Diabetes	64	41	Male
16	Endocrinologist	Diabetes	60	30	Male

Six of the 16 (37.5%) monitored diabetes, 5 (31.25%) epilepsy, and 5 (31.25%) hypertension. The age of the experts ranged from 41 to 65, reflecting the level of maturity in their opinions and assessments. The average and the median age of the experts were 56.3 and 59 respectively. The experts' experience in their respective specializations varied from 16 to 41 years, which fulfils the requirements of expert in

this research. The average and the median years of experience were 29.25 and 31.5 respectively. Fifteen of the 16 experts (93.75%) were males and 1 (6.25%) was female.

4.4.2.1.1. Domain Experts' Specialization

As shown in Figure 4.8, the experts represent different specializations and were classified as one of the following: Neurologist; Internist; Endocrinologist; Nephrologist; Internist + Endocrinologist; Internist + Neurologist; Endocrinologist + Diabetes. Figure 4.8 shows that 4 out of 16 or 25% were Neurologists, 6 or 37.5% were Internists, 2 or 12.5% were Endocrinologists, 1 or 6.25% was a Nephrologist, 1 an Internist + Endocrinologist, 1 an Internist + Neurologist, and 1 an Endocrinologist + Diabetes.

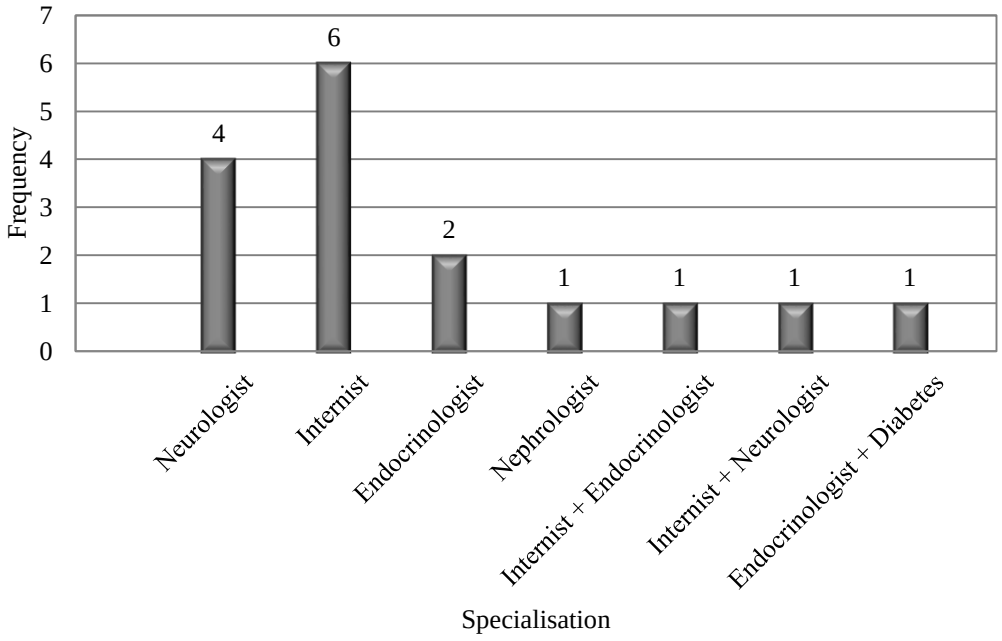

Figure 4.8. Experts' specialisation

4.4.2.1.2. Diseases Monitored by Domain Experts

The experts are specialized in monitoring patients who suffer from three diseases: diabetes, epilepsy, and hypertension. Figure 4.9 shows that 6 out of the 16 or 37.5% monitor diabetes, 5 or 31.25% monitor epilepsy, and 5 monitor hypertension.

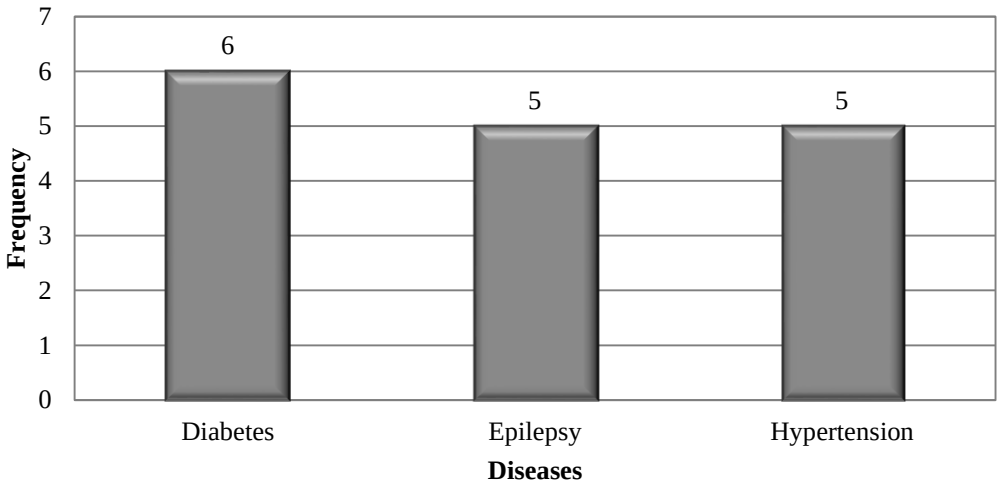

Figure 4.9. Diseases monitored by experts

4.4.2.1.3. Domain Experts' Ages

The age of experts varied from 41 to 65, which shows their level of maturity in giving opinions and assessments and suitability for the expert review activity. Figure 4.10 shows that one out of the 16 or 6.25% was 41 years old, one 47 years old, one 49 years old, one 56 years old, 3 or 18.75% were 58 years old, one 60 years old, 2 or 12.5% were 61 years old, 3 were 62 years old, 2 were 64 years old, and 1 was 65 years old.

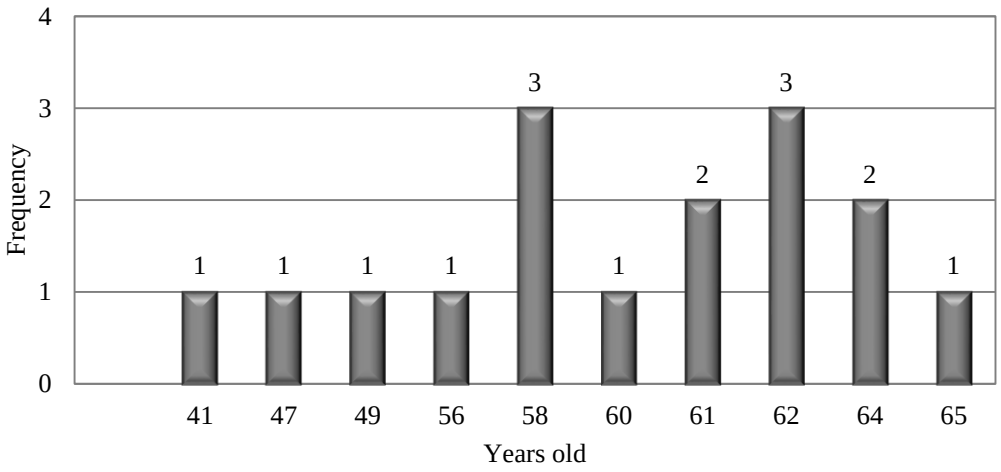

Figure 4.10. Experts' ages

4.4.2.1.4. Domain Experts' Experience

The experience of experts in their respective specializations varied from 16 to 41 years, which fulfils the requirements of expert in this research. Figure 4.11 shows that 1 out of the 16 or 6.25% had 16 years of experience, 4 or 25% had 20 years of experience, 1 had 25 years of experience, 5 or 31.25% had 30 years of experience, 1 had 33 years of experience, 1 had 34 years of experience, 2 or 12.5% had 35 years of experience, and 1 had 41 years of experience.

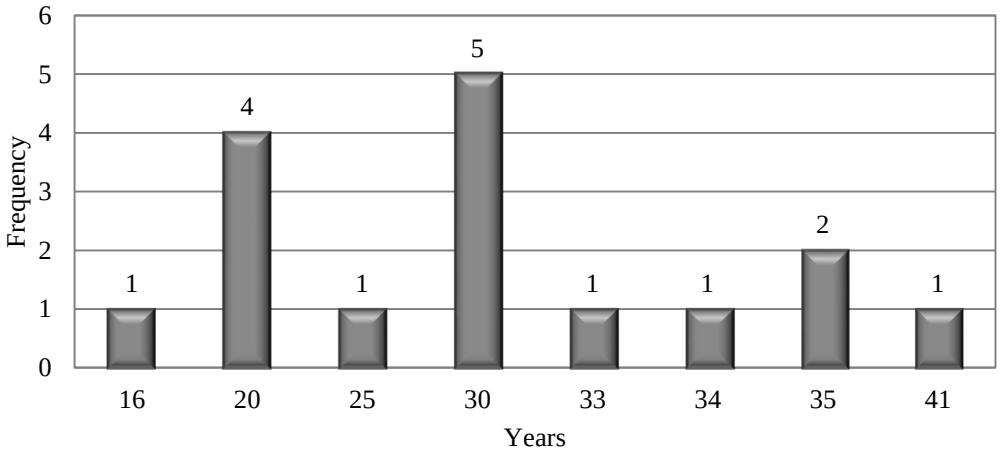

Figure 4.11. Experts' experience

4.4.2.1.5. Domain Experts' Genders

Figure 4.12 shows that 15 or 93.75% of the 16 experts were male and one was female.

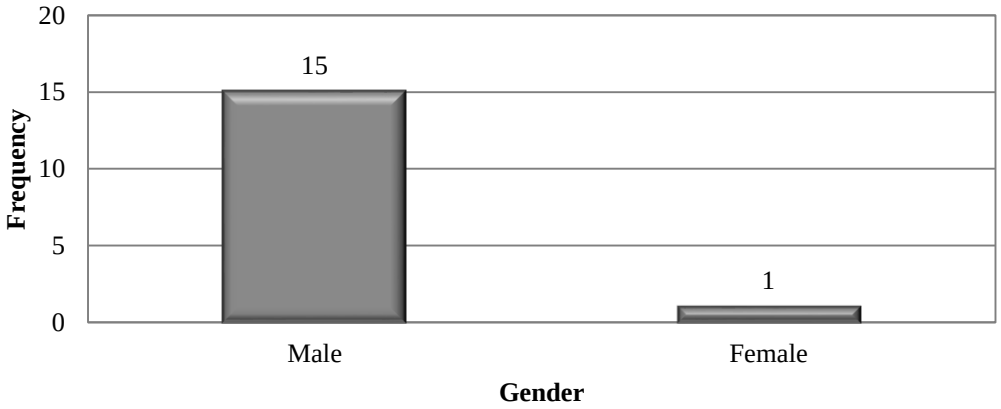

Figure 4.12. Experts' genders

4.4.2.2. Frequency of Responses from Domain Expert Review Instrument

The data was collected from three expert review instruments: a diabetes instrument, an epilepsy instrument, and a hypertension instrument. Based on the collected data from these three instruments, the frequency of responses for each question of each instrument is illustrated in Appendix B.

As shown in Appendix B, the majority of the experts agreed that the proposed domain model was complete, correct, and representative of the requirements. First, it was found that 100% of the experts from the three expert review instruments agreed that the proposed domain model was complete.

Second, it was found that 100% of the experts from the diabetes and epilepsy expert review instruments agreed that the proposed domain model was correct. While, 93.3% of the experts from the hypertension expert review instrument agreed that the proposed

domain model was correct. The remaining 6.7% of the experts from the hypertension expert review instrument did not know if the proposed domain model was correct.

Third, it was found that 98.8% of the experts from the diabetes expert review instrument agreed that the proposed domain model was representative of the requirements. The remaining 1.2% of the experts from the diabetes expert review instruments did not know if the proposed domain model was representative of the requirements. Meanwhile, it was found that 96.9% of the experts from the epilepsy and hypertension expert review instruments agreed that the proposed domain model was representative of the requirements. The remaining 3.1% of the experts from the epilepsy and hypertension expert review instruments did not know if the proposed domain model was representative of the requirements.

Fourth, it was found that 88.8% of the experts from the diabetes expert review instrument agreed about other issues related to monitoring more than one chronic disease. The remaining 11.2% of the experts from the diabetes expert review instrument disagreed about other issues related to monitoring more than one chronic disease. Moreover, it was found that 93.3% of the experts from the epilepsy expert review instrument agreed about other issues related to monitoring more than one chronic disease. The remaining 6.7% of the experts from the epilepsy expert review instrument disagreed about other issues related to monitoring more than one chronic disease. However, it was found that 100% of the experts from the hypertension expert review instrument agreed about other issues related to monitoring more than one chronic disease.

In addition, further comments from the experts were recorded and these are shown in Table 4.16. From the comments, it can be concluded that the majority of the experts agreed that the proposed CaMPaMS is both practical and suitable for the effective monitoring of chronic diseases such as diabetes, epilepsy, and hypertension.

Table 4.16

Further Comments from the Experts

Scenario	Comments
Diabetes	It is a creative project for future medicine.
Hypertension	If this scenario succeeds, we can apply this system to monitor vital signs of patient (BP, Pulse, and Temperature) and other diseases such as diabetes mellitus. I'm sure that this project is going to establish a great revolution in the medical and computer sciences, it is really a smart start to help the mankind. Thanks for this promising trial. I think that system is needed for BP monitoring especially in patient with so called white coat hypertension and in non-dipper i.e. patient with failure to dip BP at night and sleep.
Epilepsy	Very practical way to monitor epileptic fits.

4.5. Summary

In conclusion, this chapter has demonstrated that the constructed domain model, including the feature model and the abstract use case model, captures the domain's requirements and identifies its concepts. In addition, it was found that the constructed domain model is complete, correct, and representative of the domain requirements. It provides a practical and effective means of monitoring chronic diseases such as hypertension, diabetes and epilepsy. Therefore, it provides a solid foundation for efficient framework development.

CHAPTER FIVE
FRAMEWORK DESIGN AND IMPLEMENTATION

5.1. Overview

In this chapter, three processes of the framework development are described: the architectural design, framework design, and framework implementation. The implementation of the three steps to create the architecture of CaMPaMF – identifying quality attributes, selecting architectural styles based on the adopted reusability model, and constructing the architectural diagram – is presented. Then, the implementation of the three steps to design and implement the CaMPaMF based on the MDA approach – PIM development, PSM development, and code development – is covered. Finally, a summary of the chapter is presented.

5.2. Identify Quality Attributes

As elaborated in Section 2.2.4, reusability is the primary focus of this research. Therefore, reusability was identified as the primary quality attribute that must be satisfied in the architectural design.

5.3. Select Architectural Styles

Based on the reusability aspects, minimizing complexity and coupling are among the design rules that satisfy the three design principles, which are modularity, simplicity, and abstraction [10]. These design principles positively affect the flexibility and understandability factors, which in turn improve the reusability [10]. Accordingly, the

layers architectural style [146] was used because it minimizes complexity and coupling [103, 146], and therefore satisfy the identified quality attribute of the CaMPaMF that is the reusability. In principle, the layers architectural style satisfies the separation of concerns principle where the system responsibilities are decoupled and distributed over a number of logical layers. By using this style, the framework is divided into layers; each layer represents a set of components that provide a number of functionalities. The layers are connected by unidirectional relationships to satisfy the principle of information hiding, meaning a change in a lower layer is hidden by the interface of that layer and, thus, will not affect the next upper layer [103]. In the proposed CaMPaMF, there are rapid changes in both mobile and wireless sensor technologies, but the upper layers will work successfully and independently of the changes that may occur in the lower layers.

5.4. Construct the Architectural Diagram

Figure 5.13 shows the proposed architectural design of the CaMPaMS developed over the proposed CaMPaMF. This architecture consists of three primary layers: the context-aware mobile patient monitoring application layer; the CaMPaMF layer; and the context data source layer. The focus of this step is on the internal architecture of the CaMPaMF layer; its internal architecture can be captured using the "layered design with segmented layers" notation [103] to reveal the internal layered architectural design. This consists of two layers: the CML and the CCL, with arrows representing the "allowed-to-use" relations among layers themselves and among segments (i.e. components) within each layer [103]. The CML is responsible for monitoring the

context data sources – in the context data source layer – and sensing any changes in their contextual data that are of interest to the CCL.

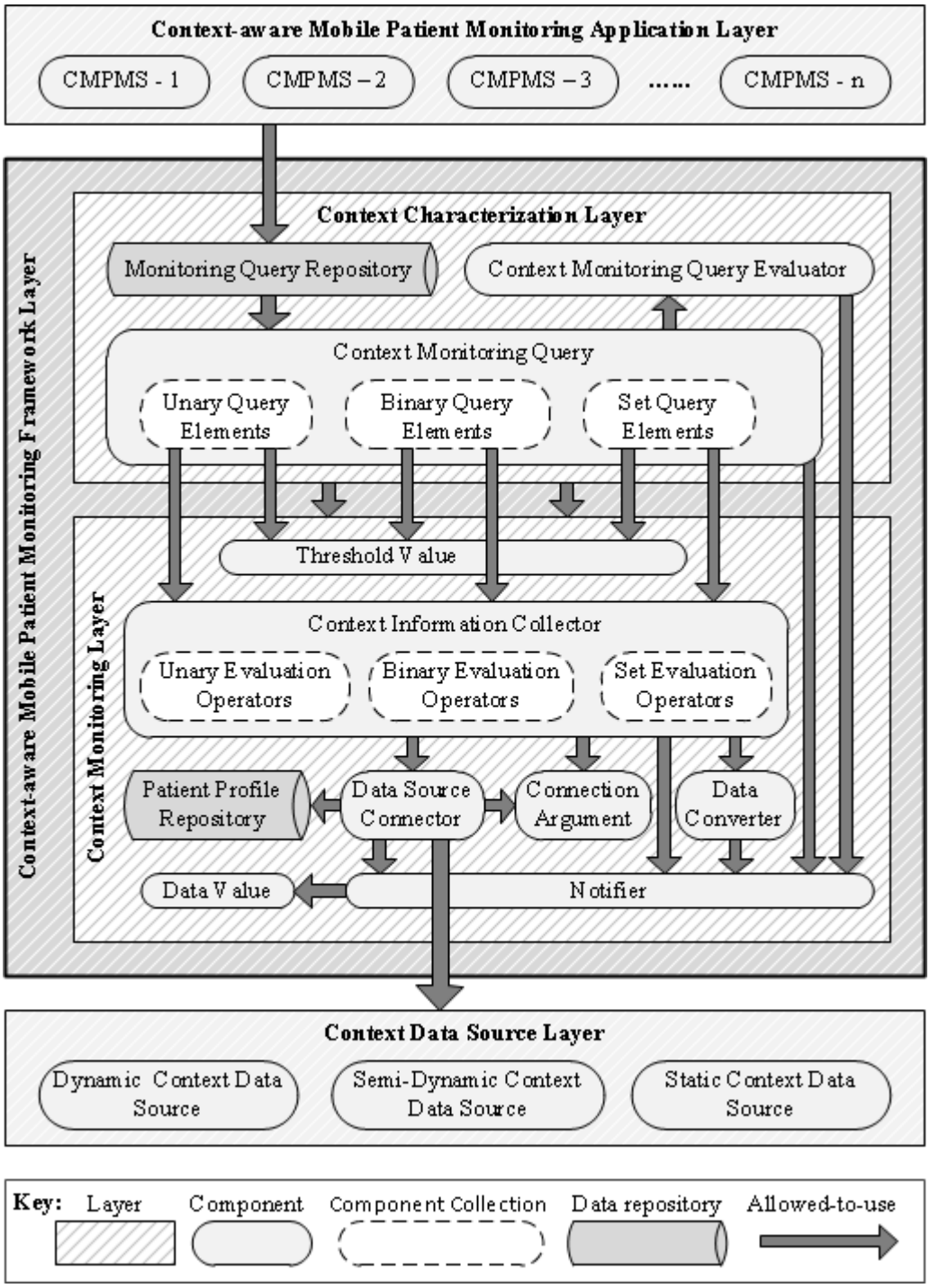

Figure 5.13. The proposed architecture of the CaMPaMF

The CCL is responsible for characterizing a particular medical situation that is of interest to CMPMA – in the CMPMA layer – by evaluating a number of logical expressions. The following sub-sections elaborate on each of these layers and their components.

5.4.1. Context Monitoring Layer

This layer consists of three components and one data repository: the context information collector, the data source connector, the data converter, and the patient profile repository. A patient context can be defined as any contextual information that can be used to characterize a patient's medical situation [70]. A patient's medical situation can involve, for example, abnormal BP, an imminent seizure, or an abnormal blood-glucose level.

Contextual information can be categorized into the following types: (1) measurable medical context including a patient's vital signs (e.g. BT); (2) non-measurable medical context including medical symptoms (e.g. dizziness); (3) risk factors (e.g. cholesterol level); (4) prescribed medications; (5) physical activities (e.g. sleeping); and (6) environment context (e.g. room temperature) [70].

These types of contextual information can be retrieved from three context data sources [70] : (1) a mobile patient profile that can be used to retrieve static context information (e.g. risk factors and prescribed medications); (2) WBS that can be used to retrieve dynamic context information types (e.g. measurable medical context and physical activities) and wireless environmental sensors that can be used to retrieve dynamic context information types (e.g. environment context); and (3) a mobile graphical user

interface that can be used to retrieve semi-dynamic context information types (e.g. non-measurable medical context).

For each context data source, the CaMPaMS define a context information collector. Context information collectors provide a level of abstraction, separating the CaMPaMS from the underlying context data sources. Context information collectors are software components that continuously monitor data changes in the context data sources. They can be marked as one of the six contextual information types. For example, BT context information, which is retrieved from a dynamic context data source such as a WBS, would be marked as a measurable medical context information type. The context information collector includes three collections of evaluation operators: (1) the unary evaluation operators, such as the IsEqual(input) operator, which takes a single operand, i.e. only one input; (2) the binary evaluation operators, such as the IsBetween(input-1, input-2) operator, which takes two operands, i.e. two inputs; (3) the set evaluation operators, such as the IsIn(input-1, input-2,..., input-n) operator, which takes a set of operands, i.e. a set of inputs. Context information collectors subscribe to context data sources to be notified whenever there are changes in the data of the context data sources. Context information collectors may use various connection techniques to communicate with the context data sources.

Data source connectors implement the connection technique that is appropriate for data communication between the context data sources and the CML. They can be marked as one of the three context data sources. For example, a static data source connector can be implemented using an asynchronous technique with suitable connection arguments to communicate with context data source that are located on a mobile

device, such as retrieving data from a static (e.g. mobile patient profile) context data source. The patient profile repository provides a static data source to store patient profile on the mobile device as part of the framework.

A semi-dynamic data source connector can be implemented using an asynchronous technique with suitable connection arguments to communicate with context data sources that are located on a mobile device, such as retrieving data from a semi-dynamic (e.g. mobile graphical user interface) context data source.

A dynamic data source connector can be implemented using an asynchronous technique with suitable connection arguments to communicate with external sensors, such as retrieving data from a dynamic context data source (e.g. WBS or environmental sensors). This highlights the need to define a data source connector for each relationship between a context data source and a context information collector.

The data collected from the context data sources may need to be converted into an appropriate format that can be understood by the CaMPaMS. A data converter is used to accomplish the conversion process. When the context information collector receives data from a context data source, an appropriate data converter is selected. For example, if the context information collector receives BT in Fahrenheit and the CaMPaMS requires receiving BT in Celsius, then the data converter can be used to convert the data from Fahrenheit to Celsius.

Once the data are converted, the context information collector provides a single access point to enable the CCL to subscribe to its context information collector to request data from context data sources. Each context information collector component can notify

the CCL whenever new data that are of interest are received from context data sources and after their conversion. The context information collector is the interface component of the CML.

5.4.2. Context Characterization Layer

This layer consists of two components and one data repository: the context monitoring query, the context monitoring query evaluator, and the monitoring query repository. Characterizing a patient's medical situation requires evaluating a number of logical expressions based on different collected context information types. Logical expressions are grouped under a number of context monitoring queries, each of which is responsible for characterizing a particular patient medical situation. These logical expressions are encapsulated in query elements. Each query element encloses one logical expression. Each context monitoring query includes three collections of query elements: (1) the unary query elements, which consist of a context information collector, a unary evaluation operator and an input (threshold); (2) the binary query elements, which consist of a context information collector, a binary evaluation operator and two thresholds; and (3) the set query elements, which consist of a context information collector, a set evaluation operator and a set of thresholds.

Each query element must be evaluated based on the data of a specific collected context information type by subscribing to its context information collector component. Once the context monitoring query is activated, it checks its evaluation period value. If the evaluation period is zero, the context monitoring query requests to evaluate its state using the context monitoring query evaluator. Otherwise, if the evaluation period is greater than zero, the context monitoring query uses its evaluation period starting time

to be evaluated based on a periodical schedule starting from the evaluation period starting time and repeated frequently every evaluation period. The context information collector component will then notify each subscribed query element whenever new data are received and converted. When the query element receives new data from the context information collector component, the query element must evaluate its logical expression. The evaluation result identifies the state of the query element that is of interest to a specific context monitoring query.

The context monitoring query is a software component that characterizes a patient's medical situation based on the state of evaluated query elements by subscribing to a number of query elements to be notified whenever the state of a query element is changed. The context monitoring query uses the context monitoring query evaluator to ascertain its state once a notification is received from any of its query elements. Once the query is evaluated, the context monitoring query checks its state. If the state is changed, the context monitoring query checks its alarm duration value. If the alarm duration is zero, the context monitoring query uses the evaluation result to notify the CMPMA layer instantly when its state is changed. Otherwise, the context monitoring query uses the evaluation result to notify the CMPMA layer after the value of the alarm duration is elapsed without any new change in the state of the context monitoring query.

The context monitoring query evaluator implements an optimized evaluation strategy using rule-based reasoning. For example, the context monitoring query evaluator will retrieve the state of all query elements from static context data sources. If all of the states are true, then it will retrieve the state of all query elements from dynamic context

data sources. If all of the states are true, then it will retrieve the state of all query elements from semi-dynamic context data sources. If all of the states are true, then it will evaluate the state of the context monitoring query as true. If any of the previous conditions is false then it will discontinue the evaluation procedure and evaluate the state of the context monitoring query as false. Otherwise, if the state of any of the query elements is unspecified, then it will evaluate the state of the context monitoring query as unspecified. An unspecified state means that the data of a particular collected context information type cannot be retrieved from their context data source.

The context monitoring query repository is used to allow the CMPMA layer to register new context monitoring queries in the framework. Additionally, it enables the CMPMA layer to find, add, edit and delete a specific context monitoring query. Therefore, the context monitoring query repository is the interface component of the CCL.

5.5. PIM Development

Figure 5.14 shows the constructed PIM resulting from applying the PIM development step. The four refinement techniques, mentioned in sub-section 3.7.2.1, are discussed in the following sub-sections. Additionally, the UML sequence diagram was used to show the interaction between the framework components.

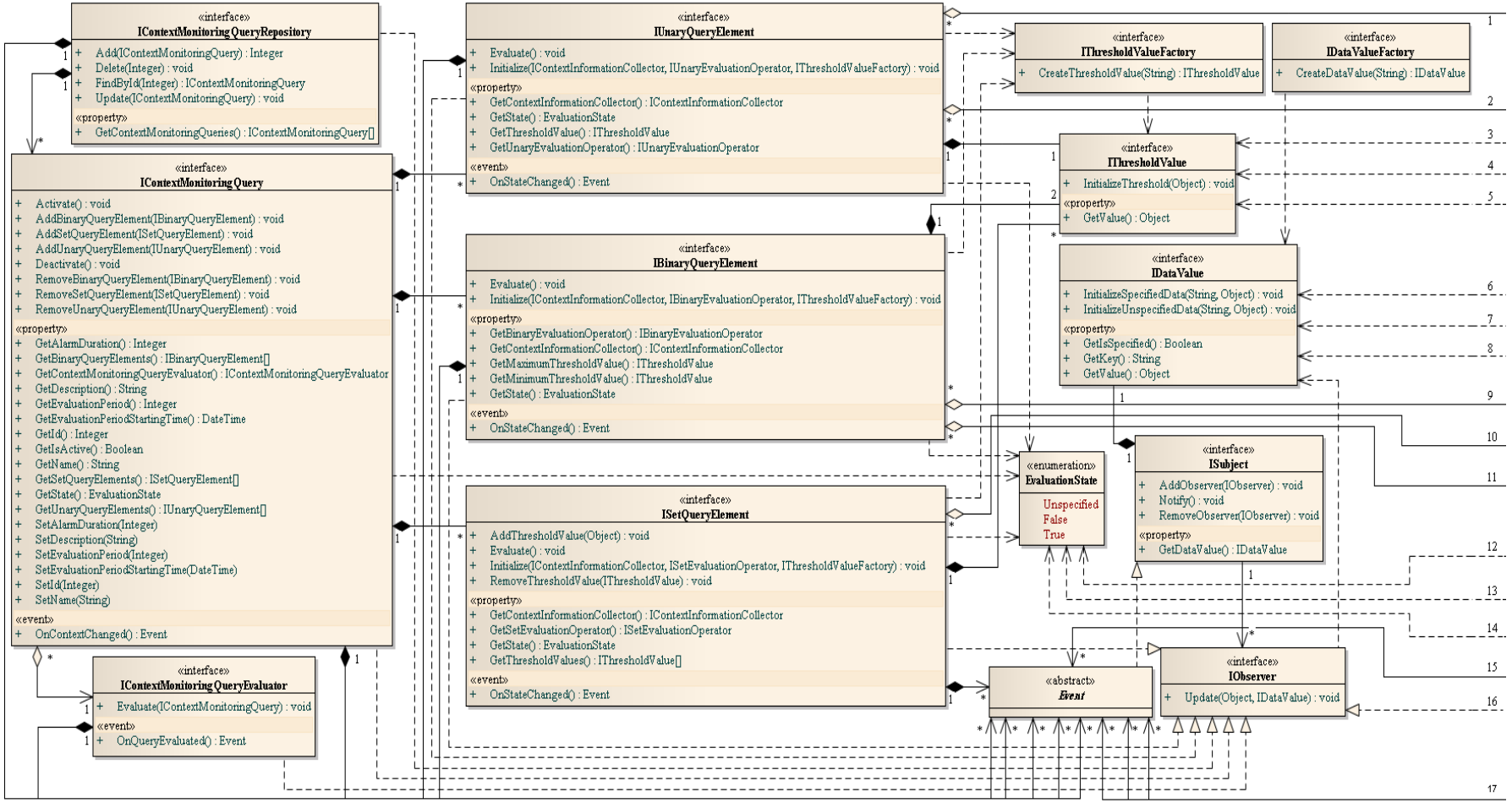

Figure 5.14. Platform independent model

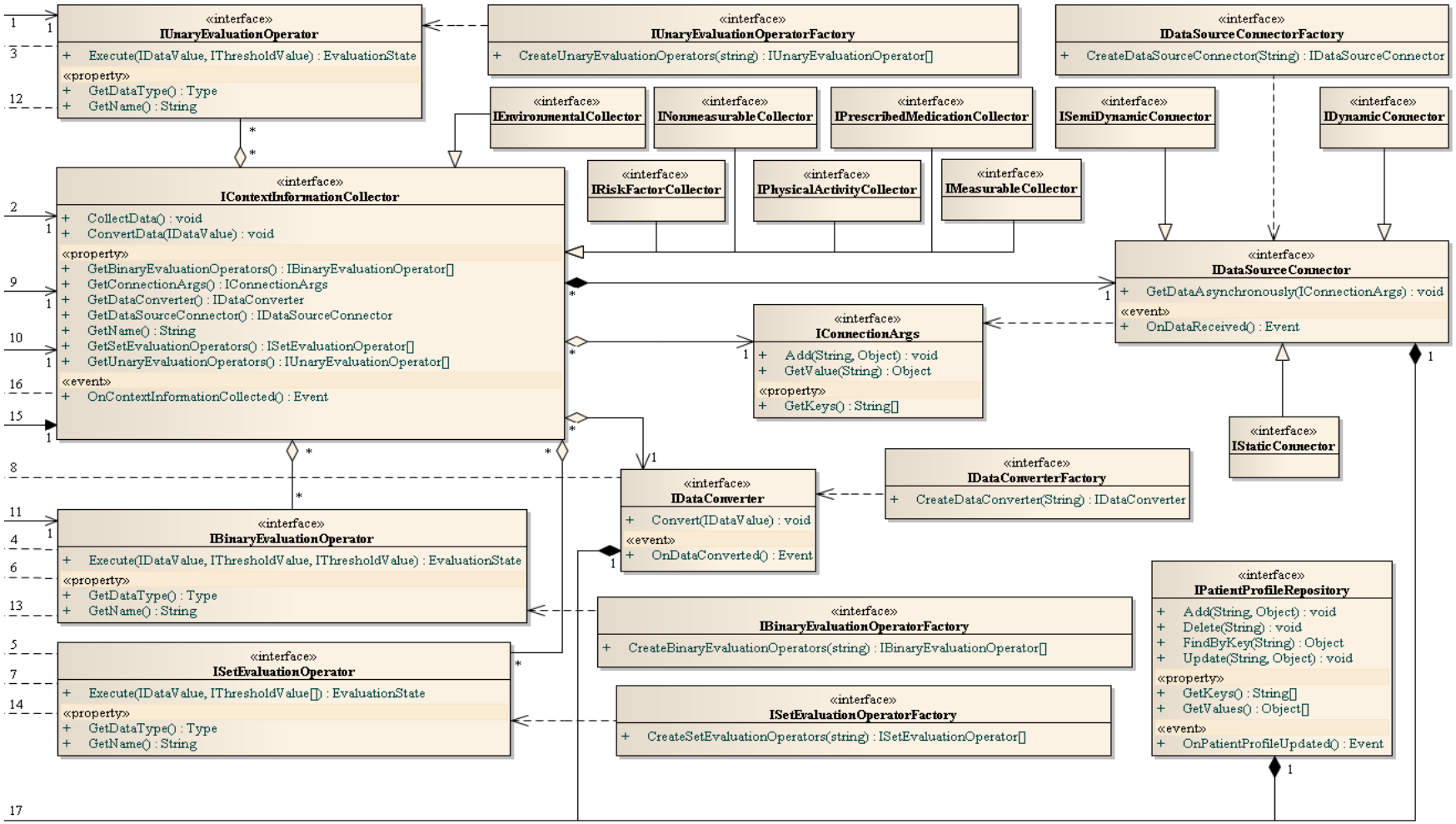

Figure 5.14 continued

5.5.1. Hot Spots and Frozen Spots

First, hot spots represent the variable domain requirements that were identified in the domain analysis process. These variable domain requirements may be mapped to interfaces in the framework design. In fact, an interface-based design improve reusability by minimizing complexity and coupling [99, 161], which are among the design rules that should be satisfied as a reusability aspect [10]. Second, frozen spots represent the common domain requirements. These common domain requirements may be mapped to concrete classes in the framework design [23]. The identified interfaces, concrete classes, and enumerations of the CaMPaMF are introduced in the following sub-sections.

5.5.1.1. EvaluationState Enumeration

This provides an enumeration for the three values of the evaluation state of the IContextMonitoringQuery and its three types of query elements, which are: (1) IUnaryQueryElement; (2) IBinaryQueryElement; (3) ISetQueryElement. This enumeration is set to True if the logical expression of the context monitoring query or the query element is true. The enumeration is set to False if the logical expression of the context monitoring query or the query element is false. The enumeration is set to Unspecified if the data of a particular context information type cannot be retrieved from their context data source.

5.5.1.2. IDataValue Interface

This provides three properties and two methods required to provide a general data type to store the data transmitted among the framework components and is a member of

ISubject. It provides an extensibility point to represent various data types. Table 1 in Appendix D shows a description of the properties and methods of this interface.

5.5.1.3. IDataValueFactory Interface

This is an abstract factory that provides one method required to create a suitable object of the IDataValue based on the caller name. Table 2 in Appendix D shows a description of the method of this interface.

5.5.1.4. ISubject Interface

This provides one property and three methods required to provide a standard mechanism for communication between the framework components. It allows a particular component (e.g. component A) to request IDataValue from another component (e.g. component B) asynchronously. It also allows component B to notify component A once its requested IDataValue is ready. It provides an extensibility point to represent various events that transmit various data values. The ISubject is realized by an Event abstract class. Using the Event abstract class enables the observers to explicitly register and thus be notified about specific events that are of interest. According to Microsoft Corporation [295], using the ISubject, the Event abstract class, and an IObserver interface (introduced below) enhances the original observer pattern [99]. In the original observer pattern, the subject and the observer were abstract classes, thus the ability of the observers' concrete classes to explicitly register a specific event of a subject concrete class was limited. The CaMPaMF provides five concrete classes of the Event abstract class, which are: (1) IntegerEvent to transmit integer data value; (2) DecimalEvent to transmit decimal data value; (3) BooleanEvent to transmit

boolean data value; (4) EvaluationStateEvent to transmit EvaluationState data value; (5) ObjectEvent to transmit object data value. The ISubject uses the IDataValueFactory to create data values based on the ISubject name using the DataValueFactory concrete class as a default implementation of the IDataValueFactory. Table 3 in Appendix D shows a description of the property and methods of this interface.

5.5.1.5. IObserver Interface

This provides one method required to provide an asynchronous callback method used to handle the event notifications of the ISubject. Table 4 in Appendix D shows a description of the method of this interface.

5.5.1.6. Event Abstract Class

This is an abstract class that implements the ISubject. In the CaMPaMF, each server component must contain at least one Event exposed as public property. This allows client components to access the specific Event exposed by a server component and subscribe to it to be notified about the state of the server component. Moreover, in this implementation of the observer pattern, the server component can have more than one Event, for example, one per exposed activity [295].

5.5.1.7. IPatientProfileRepository Interface

This provides two properties, four methods, and one event required to store the patient profile on the mobile device as part of the framework. It provides an extensibility point to represent various patient profile repositories. Once the data of the patient profile repository are updated, the IPatientProfileRepository initializes the IDataValue of the

ISubject event and raises the PatientProfileUpdated event notification. This is achieved by using the IEventFactory to create a suitable event based on the IPatientProfileRepository name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 5 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.8. IConnectionArgs Interface

This provides one property and two methods required to encapsulate connection arguments to connect to a specific context data source. It provides an extensibility point to represent various connection arguments of various data types. Table 6 in Appendix D shows a description of the property and methods of this interface.

5.5.1.9. IDataSourceConnector Interface

This provides one method and one event required to connect asynchronously to the context data sources. It provides an extensibility point to represent various connection techniques that must be implemented by application developers. It is realized with three marker interfaces to represent the three context information sources. These marker interfaces are: (1) ISemiDynamicConnector; (2) IDynamicConnector; and (3) IStaticConnector. The IDataSourceConnector connects to a specific context data source to request data asynchronously and observes the context data source. Once the data are received, the IDataSourceConnector initializes the IDataValue of the ISubject event and raises the DataReceived event notification. This is achieved by using the IEventFactory to create a suitable event based on the IDataSourceConnector name using the SubjectFactory concrete class as a default implementation of the

IEventFactory. Table 7 in Appendix D shows a description of the method and event of this interface.

5.5.1.10. IDataSourceConnectorFactory Interface

This is an abstract factory that provides one method required to create a suitable object of the IDataSourceConnector based on the caller name. Table 8 in Appendix D shows a description of the method of this interface.

5.5.1.11. IDataConverter Interface

This provides one method and one event required to convert raw data to the suitable data required by the CaMPaMS. It provides an extensibility point to represent various conversion algorithms. Once the data are converted, the IDataConverter initializes the IDataValue of the ISubject event and raises the DataConverted event notification. This is achieved by using the IEventFactory to create a suitable event based on the IDataConverter name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 9 in Appendix D shows a description of the method and event of this interface.

5.5.1.12. IDataConverterFactory Interface

This is an abstract factory that provides one method required to create a suitable object of the IDataConverter based on the caller name. Table 10 in Appendix D shows a description of the method of this interface.

5.5.1.13. IThresholdValue Interface

This provides one property and one method required to store a threshold value that must be used to evaluate the collected data values. It provides an extensibility point to represent various threshold value types. Table 11 in Appendix D shows a description of the property and method of this interface.

5.5.1.14. IThresholdValueFactory Interface

This is an abstract factory that provides one method required to create a suitable object of the IThresholdValue based on the caller name. Table 12 in Appendix D shows a description of the method of this interface.

5.5.1.15. IUnaryEvaluationOperator Interface

This provides two properties and one method required to execute a particular comparison operator to compare between the data value and only one threshold value, which is used to evaluate a logical expression of a query element. The IUnaryEvaluationOperator provides an extensibility point to represent various comparison operators. Table 13 in Appendix D shows a description of the properties and method of this interface.

5.5.1.16. IUnaryEvaluationOperatorFactory Interface

This is an abstract factory that provides one method required to create a collection of suitable objects of the IUnaryEvaluationOperator based on the caller name. Table 14 in Appendix D shows a description of the method of this interface.

5.5.1.17. IBinaryEvaluationOperator Interface

This provides two properties and one method required to execute a particular comparison operator to compare between the data value and two threshold values, which is used to evaluate a logical expression of a query element. The IBinaryEvaluationOperator provides an extensibility point to represent various comparison operators. Table 15 in Appendix D shows a description of the properties and method of this interface.

5.5.1.18. IBinaryEvaluationOperatorFactory Interface

This is an abstract factory that provides one method required to create a collection of suitable objects of the IBinaryEvaluationOperator based on the caller name. Table 16 in Appendix D shows a description of the method of this interface.

5.5.1.19. ISetEvaluationOperator Interface

This provides two properties and one method required to execute a particular comparison operator to compare between the data value and a set of threshold values, which is used to evaluate a logical expression of a query element. The ISetEvaluationOperator provides an extensibility point to represent various comparison operators. Table 17 in Appendix D shows a description of the properties and method of this interface.

5.5.1.20. ISetEvaluationOperatorFactory Interface

This is an abstract factory that provides one method required to create a collection of suitable objects of the ISetEvaluationOperator based on the caller name. Table 18 in Appendix D shows a description of the method of this interface.

5.5.1.21. IContextInformationCollector Interface

This provides seven properties, two methods, and one event required to collect data asynchronously from context data sources and convert the collected raw data to the suitable data required by the CaMPaMS. The IContextInformationCollector delegates these two responsibilities to the IDataSourceConnector and the IDataConverter respectively. The IContextInformationCollector provides an extensibility point to represent various context data sources that must be implemented by application developers. It also represents the primary interface of the CML by providing a single access point to enable the CCL to register to context information collectors.

The IContextInformationCollector is realized with six marker interfaces to represent the six context information types. These marker interfaces are: (1) IRiskFactorCollector to represent the risk factor context information type (e.g. cholesterol level); (2) IMeasurableCollector to represent the measurable context information type such as a patient's vital signs (e.g. BT); (3) INonmeasurableCollector to represent the non-measurable context information type such as medical symptoms (e.g. dizziness); (4) IPhysicalActivityCollector to represent the physical activity context information type (e.g. sleeping); (5) IEnvironmentalCollector to represent the environmental context information type (e.g. room temperature); (6)

IPrescribedMedicationCollector to represent the prescribed medication context information type (e.g. calcium-channel blocker).

The IContextInformationCollector requests to collect specific data asynchronously from the IDataSourceConnector and observes the IDataSourceConnector. Once new data are received from the IDataSourceConnector, the IContextInformationCollector requests to asynchronously convert the received data from the IDataConverter. Once the data are converted by the IDataConverter, the IContextInformationCollector initializes the IDataValue of the ISubject event and raises the ContextInformationCollected event notification. This is achieved by using the IEventFactory to create a suitable event based on the IContextInformationCollector name using the SubjectFactory concrete class as a default implementation of the IEventFactory. The IContextInformationCollector contains three collections of evaluation operators of three types, which are: (1) IUnaryEvaluationOperator; (2) IBinaryEvaluationOperator; and (3) ISetEvaluationOperator.

The IContextInformationCollector uses five abstract factories, which are: (1) the IDataSourceConnectorFactory to create data source connectors based on the IContextInformationCollector name using the DataSourceConnectorFactory concrete class as a default implementation of the IDataSourceConnectorFactory; (2) the IDataConverterFactory to create data convertors based on the IContextInformationCollector name using the DataConverterFactory concrete class as a default implementation of the IDataConverterFactory; (3) the IUnaryEvaluationOperatorFactory to fill the IUnaryEvaluationOperator collection based on the IContextInformationCollector name using the

UnaryEvaluationOperatorFactory concrete class as a default implementation of the IUnaryEvaluationOperatorFactory; (4) the IBinaryEvaluationOperatorFactory to fill the IBinaryEvaluationOperator collection based on the IContextInformationCollector name using the BinaryEvaluationOperatorFactory concrete class as a default implementation of the IBinaryEvaluationOperatorFactory; and (5) the ISetEvaluationOperatorFactory to fill the ISetEvaluationOperator collection based on the IContextInformationCollector name using the SetEvaluationOperatorFactory concrete class as a default implementation of the ISetEvaluationOperatorFactory. Table 19 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.22. IUnaryQueryElement Interface

This provides four properties, two methods, and one event required to create and evaluate a logical expression, which includes an IContextInformationCollector, an IUnaryEvaluationOperator, and an IThresholdValue. The IUnaryQueryElement delegates the responsibility of evaluating the logical expression to the IUnaryEvaluationOperator. It also provides an extensibility point to represent various unary query elements. The IUnaryQueryElement requests to collect context information asynchronously from IContextInformationCollector and observes the IContextInformationCollector. Once the context information is collected from the IContextInformationCollector, the IUnaryQueryElement passes both the IDataValue collected from the IContextInformationCollector and the IThresholdValue to the IUnaryEvaluationOperator to evaluate its logical expression. Once the logical expression is evaluated, the IUnaryQueryElement checks its state. If the state is

changed, the IUnaryQueryElement initializes the IDataValue of the ISubject event and raises the StateChanged event notification. This is achieved by using the IEventFactory to create a suitable event based on the IUnaryQueryElement name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 20 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.23. IBinaryQueryElement Interface

This provides five properties, two methods, and one event required to create and evaluate a logical expression, which includes an IContextInformationCollector, an IBinaryEvaluationOperator, and two threshold values of type IThresholdValue. The IBinaryQueryElement delegates the responsibility of evaluating the logical expression to the IBinaryEvaluationOperator. It also provides an extensibility point to represent various binary query elements. The IBinaryQueryElement requests to collect context information asynchronously from IContextInformationCollector and observes the IContextInformationCollector. Once the context information is collected from the IContextInformationCollector, the IBinaryQueryElement passes both the IDataValue collected from the IContextInformationCollector and the two threshold values of type IThresholdValue to the IBinaryEvaluationOperator to evaluate its logical expression. Once the logical expression is evaluated, the IBinaryQueryElement checks its state. If the state is changed, the IBinaryQueryElement initializes the IDataValue of the ISubject event and raises the StateChanged event notification. This is achieved by using the IEventFactory to create a suitable event based on the IBinaryQueryElement name using the SubjectFactory concrete class as a default implementation of the

IEventFactory. Table 21 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.24. ISetQueryElement Interface

This provides four properties, four methods, and one event required to create and evaluate a logical expression, which includes an IContextInformationCollector, an ISetEvaluationOperator, and a set of IThresholdValue. The ISetQueryElement delegates the responsibility of evaluating the logical expression to the ISetEvaluationOperator. It also provides an extensibility point to represent various set query elements. The ISetQueryElement requests to collect context information asynchronously from the IContextInformationCollector and observes the IContextInformationCollector. Once the context information is collected from the IContextInformationCollector, the ISetQueryElement passes both the IDataValue collected from the IContextInformationCollector and the set of IThresholdValue to the ISetEvaluationOperator to evaluate its logical expression. Once the logical expression is evaluated, the ISetQueryElement checks its state. If the state is changed, the ISetQueryElement initializes the IDataValue of the ISubject event and raises the StateChanged event notification. This is achieved by using the IEventFactory to create a suitable event based on the ISetQueryElement name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 22 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.25. IContextMonitoringQueryEvaluator Interface

This provides one method and one event required to evaluate a context monitoring query to characterize a patient medical situation based on the state of the three types of query element, which are: (1) IUnaryQueryElement; (2) IBinaryQueryElement; and (3) ISetQueryElement. It provides an extensibility point to represent various context monitoring query evaluation strategies. The IContextMonitoringQueryEvaluator evaluates an IContextMonitoringQuery by implementing an optimized evaluation strategy using rule-based reasoning.

Once the IContextMonitoringQuery is evaluated, the IContextMonitoringQuery-Evaluator initializes the IDataValue of the ISubject event and raises the QueryEvaluated event notification. This is achieved by using the IEventFactory to create a suitable event based on the IContextMonitoringQueryEvaluator name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 23 in Appendix D shows a description of the method and event of this interface.

5.5.1.26. IContextMonitoringQuery Interface

This provides 18 properties, eight methods, and one event required to create and evaluate a context monitoring query to characterize a patient medical situation based on the state of its three types of query element. It represents the primary interface of the CCL by providing an access point to enable the CaMPaMS to register context monitoring queries. The IContextMonitoringQuery delegates the responsibility of characterizing a particular patient medical situation to the IContextMonitoring-QueryEvaluator. It also provides an extensibility point to represent various context monitoring queries. Once the IContextMonitoringQuery is activated, it checks its

EvaluationPeriod value. If the EvaluationPeriod is zero, the IContextMonitoring-Query requests to asynchronously evaluate its state using the IContextMonitoring-QueryEvaluator and observes the IContextMonitoringQuery-Evaluator. Otherwise, if the EvaluationPeriod is greater than zero, the IContextMonitoringQuery uses its EvaluationPeriodStartingTime to be evaluated based on a periodical schedule starting from the EvaluationPeriodStartingTime and repeated frequently every EvaluationPeriod.

Once the query is evaluated, the IContextMonitoringQuery checks its state. If the state is changed, the IContextMonitoringQuery checks its AlarmDuration value. If the AlarmDuration is zero, the IContextMonitoringQuery initializes the IDataValue of the ISubject event and raises the ContextChanged event notification. Otherwise, if the AlarmDuration is greater than zero, the IContextMonitoringQuery initializes the IDataValue of the ISubject event and raises the ContextChanged event notification after the value of the AlarmDuration is elapsed without any new change in the state of the IContextMonitoringQuery. This is achieved by using the IEventFactory to create a suitable event based on the IContextMonitoringQuery name using the SubjectFactory concrete class as a default implementation of the IEventFactory. Table 24 in Appendix D shows a description of the properties, methods, and event of this interface.

5.5.1.27. IMonitoringQueryRepository Interface

This provides one property and four methods required to store context monitoring queries on the mobile device as part of the framework. It provides an extensibility point to represent various monitoring query repositories. Table 25 in Appendix D shows a description of the properties and methods of this interface.

5.5.2. Design Patterns and Design Principles

In this research, the design patterns singleton, observer, strategy, and abstract factory were used as strategies for applying the SOLID design principles, which are SRP, OCP, LSP, ISP, and DIP. The design patterns and design principles were applied to refine the PIM as described in the following sub-sections. However, observer, strategy, and abstract factory design patterns improve reusability by minimizing complexity and coupling and maximizing cohesion [99], which are the design rules that should be satisfied as a reusability aspect [10].

5.5.2.1. Singleton Design Pattern

Considering the limited resources of mobile devices, in the proposed design of the CaMPaMF only one instance of some concrete implementation must be instantiated. This is applied on the concrete implementation of the following interfaces: (1) IMonitoringQueryRepository; (2) IPatientProfileRepository; (3) IDataValueFactory; (4) IDataSourceConnectorFactory; (5) IDataConverterFactory; (6) IUnary-EvaluationOperatorFactory; (7) IBinaryEvaluationOperatorFactory; (8) ISet-EvaluationOperatorFactory; (9) IThresholdValueFactory. To meet this need, the singleton design pattern [99] was used.

5.5.2.2. Observer Design Pattern

In the proposed design of the CaMPaMF, an asynchronous communication between the framework components is used as a standard mechanism for communication. This allows a particular component (e.g. component A) to request data from another component (e.g. component B) asynchronously. It also allows component B to notify

component A by raising an event once its requested data are ready. Such a mechanism is required to support the communication between the following seven pairs of interfaces: (1) IDataSourceConnector and IContextInformationCollector; (2) IDataConverter and IContextInformationCollector; (3) IUnaryQueryElement and IContextMonitoringQuery; (4) IBinaryQueryElement and IContextMonitoringQuery; (5) ISetQueryElement and IContextMonitoringQuery; (6) IContextMonitoring-QueryEvaluator and IContextMonitoringQuery; (7) IPatientProfileRepository and IDataSourceConnector. To meet this need, the observer design pattern was used [99, 295, 296].

This pattern ensures non-blocking communication and supports connecting to an unlimited number of context data sources and notifying an unlimited number of CaMPaMS. Moreover, this pattern conforms to the OCP that allows registering new observers (e.g. CaMPaMS) without changing the subject (e.g. IContextMonitoringQuery). Looking back at Figure 5.14, it can be seen that the IContextInformationCollector, IUnaryQueryElement, IBinaryQueryElement, ISetQueryElement, IContextMonitoringQueryEvaluator, and IContextMonitoring-Query are substitutable for the IObserver and the Event concrete class is substitutable for the ISubject. Therefore, the LSP is applied. Furthermore, the Event concrete class depends on the IObserver interface and the concrete methods of the ISubject also depend on the IObserver interface. Thus, the DIP is applied.

5.5.2.3. Strategy Design Pattern

In the proposed design of the CaMPaMF, there is a need to enable, for example, the IContextInformationCollector to collect data from various context data sources,

including dynamic context data sources (e.g. WBS), semi-dynamic context data sources (e.g. a mobile graphical user interface), and a static context data sources (e.g. a mobile patient profile). Each of these context data sources requires different connection techniques and each connection technique may require different connection arguments. Such a need requires defining different communication strategies or algorithms and encapsulating them, so that any of these strategies or algorithms can interchange with each other to support the extensibility of the proposed CaMPaMF. This situation appears with the following interfaces: (1) IContextInformationCollector uses the GetDataAsynchronously strategy of the IDataSourceConnector and the Convert strategy of the IDataConverter; (2) IUnaryQueryElement uses the CollectData strategy of the IContextInformationCollector and the Execute strategy of the IUnaryEvaluationOperator; (3) IBinaryQueryElement uses the CollectData strategy of the IContextInformationCollector and the Execute strategy of the IBinaryEvaluationOperator; (4) ISetQueryElement uses the CollectData strategy of the IContextInformationCollector and the Execute strategy of the ISetEvaluationOperator; (5) IContextMonitoringQuery uses the Evaluate strategy of the IContextMonitoringQueryEvaluator; (6) IContextMonitoringQueryEvaluator uses the Evaluate strategy of the IUnaryQueryElement, IBinaryQueryElement, and ISetQueryElement. To meet this need, the strategy design pattern was used [99, 296]. This pattern conforms to the DIP which allows each concrete class to be manipulated by various algorithms (strategies) [161]. It also fully conforms to the OCP, which supports the component-based development and black-box extensibility approach using the composition approach.

5.5.2.4. Abstract Factory Design Pattern

In the proposed design of the CaMPaMF, the SRP was applied by decoupling responsibilities across different components to support CaMPaMF extensibility. In addition, the DIP was applied by depending on interfaces rather than concrete classes. For example: (1) the IContextInformationCollector delegates the responsibility of collecting data asynchronously from various context data sources to IDataSourceConnector and delegates the responsibility of converting raw data to the suitable data required by the CaMPaMS to the IDataConverter; (2) IUnaryQueryElement delegates the responsibility of collecting data asynchronously to the IContextInformationCollector and delegates the responsibility of evaluating its logical expression to the IUnaryEvaluationOperator; (3) IBinaryQueryElement delegates the responsibility of collecting data asynchronously to the IContextInformationCollector and delegates the responsibility of evaluating its logical expression to the IBinaryEvaluationOperator; (4) ISetQueryElement delegates the responsibility of collecting data asynchronously to the IContextInformationCollector and delegates the responsibility of evaluating its logical expression to the ISetEvaluationOperator; (5) IContextMonitoringQuery delegates the responsibility of characterizing a particular patient medical situation to the IContextMonitoringQueryEvaluator. However, to connect the concrete classes of these interfaces, there is a need to provide a mechanism to prepare an object from a server component (e.g. the concrete class of the IDataSourceConnector) to serve a client component (e.g. the concrete class of the IContextInformationCollector). This object preparation responsibility must be delegated to an intermediary component (e.g. the concrete class of the IDataSourceConnectorFactory). To meet this need, the

abstract factory design pattern was used [99, 296]. This pattern supports component-based development and the black-box extensibility approach using the composition approach. It also fully conforms to the OCP.

5.5.3. Sequence Diagram

This section presents a number of scenarios that were constructed based on the identified abstract use cases to illustrate the dynamic behaviour of the proposed framework and to show the interactions between the components. Additionally, it illustrates the scenarios of how the CaMPaMS interacts with the framework and how the framework reacts to CaMPaMS calls. The following sub-sections present these scenarios.

5.5.3.1. Scenario of Listing Context Monitoring Queries

Figure 1 in Appendix E illustrates the interactions between the CaMPaMS and the framework to list context monitoring queries. First, the CaMPaMS invokes the GetContextMonitoringQueries method on the IContextMonitoringQueryRepository, and the IContextMonitoringQueryRepository returns a collection of context monitoring queries.

5.5.3.2. Scenario of Adding a Context Monitoring Query

Figure 2 in Appendix E illustrates the interactions between the CaMPaMS and the framework to add a context monitoring query. First, the CaMPaMS invokes the Add(IContextMonitoringQuery) method on the IContextMonitoringQueryRepository, passing the new context monitoring query as a parameter, and the IContextMonitoringQueryRepository returns a unique identity of the added context

monitoring query. Then, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing the unique identity as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query.

Next, the CaMPaMS invokes the following methods on the IContextMonitoringQuery: (1) the SetName(String) method, passing the query name as a string parameter; (2) the SetDescription(String) method, passing the query description as a string parameter; (3) the SetAlarmDuration(Integer) method, passing the query alarm duration as an integer parameter; (4) the SetEvaluationPeriod(Integer) method, passing the query evaluation period as an integer parameter; (5) the SetEvaluationPeriodStartingTime(DateTime) method, passing the query evaluation period starting time as a date time parameter. Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.3. Scenario of Finding a Context Monitoring Query

Figure 3 in Appendix E illustrates the interactions between the CaMPaMS and the framework to find a context monitoring query. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query.

5.5.3.4. Scenario of Editing a Context Monitoring Query

Figure 4 in Appendix E illustrates the interactions between the CaMPaMS and the framework to edit a context monitoring query. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Then, the CaMPaMS invokes the following methods on the IContextMonitoringQuery: (1) the SetName(String) method, passing the query name as a string parameter; (2) the SetDescription(String) method, passing the query description as a string parameter; (3) the SetAlarmDuration(Integer) method, passing the query alarm duration as an integer parameter; (4) the SetEvaluationPeriod(Integer) method, passing the query evaluation period as an integer parameter; (5) the SetEvaluationPeriodStartingTime(DateTime) method, passing the query evaluation period starting time as a date time parameter. Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter. Accordingly, the IContextMonitoringQueryRepository invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active.

5.5.3.5. Scenario of Deleting a Context Monitoring Query

Figure 5 in Appendix E illustrates the interactions between the CaMPaMS and the framework to delete a context monitoring query. First, the CaMPaMS invokes the Delete(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter. Then, the

IContextMonitoringQueryRepository invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active.

5.5.3.6. Scenario of Listing Query Elements

Figure 6 in Appendix E illustrates the interactions between the CaMPaMS and the framework to list query elements. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Then, the CaMPaMS invokes the following methods on the IContextMonitoringQuery: (1) the GetUnaryQueryElements() method, and the IContextMonitoringQuery returns a collection of unary query elements; (2) the GetBinaryQueryElements() method, and the IContextMonitoringQuery returns a collection of binary query elements; (3) the GetSetQueryElements() method, and the IContextMonitoringQuery returns a collection of set query elements.

5.5.3.7. Scenario of Adding a Unary Query Element

Figure 7 in Appendix E illustrates the interactions between the CaMPaMS and the framework to add a unary query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRe-pository returns the context monitoring query. Then, the CaMPaMS invokes GetUnaryEvaluationOperators() method on the

IContextInformationCollector, and the IContextInformationCollector returns a collection of unary evaluation operators.

Next, the CaMPaMS invokes the AddUnaryQueryElement(IUnaryQueryElement) method on the IContextMonitoringQuery, passing a unary query element as a parameter. Consequently, the IContextMonitoringQuery invokes Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active, and then creates IUnaryQueryElement.

Later, the CaMPaMS invokes the Initialize(IContextInformationCollector, IUnaryEvaluationOperator, IThresholdValueFactory) method on the IUnaryQueryElement, passing a context information collector, unary evaluation operator, and a threshold value factory as parameters. Consequently, the IUnaryQueryElement invokes the CreateThresholdValue(String) method on the IThresholdValueFactory, passing the unary query element type name as a string parameter. Then, the IThresholdValueFactory creates the IThresholdValue, and returns the created threshold value.

Finally, the CaMPaMS invokes the InitializeThreshold(Object) method on the IThresholdValue, passing threshold value as an object parameter, and then it invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.8. Scenario of Adding a Binary Query Element

Figure 8 in Appendix E illustrates the interactions between the CaMPaMS and the framework to add a binary query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a binary identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Then, the CaMPaMS invokes the GetBinaryEvaluationOperators() method on the IContextInformationCollector, and the IContextInformationCollector returns a collection of binary evaluation operators. Next, the CaMPaMS invokes the AddBinaryQueryElement(IBinaryQueryElement) method on the IContextMonitoringQuery, passing a binary query element as a parameter. Consequently, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active, and then creates a IBinaryQueryElement.

Later, the CaMPaMS invokes the Initialize(IContextInformationCollector, IBinaryEvaluationOperator, IThresholdValueFactory) method on the IBinaryQueryElement, passing a context information collector, binary evaluation operator, and a threshold value factory as parameters. Consequently, the IBinaryQueryElement invokes the CreateThresholdValue(String) method on the IThresholdValueFactory to create a minimum threshold value, passing the binary query element type name as a string parameter. Then, the IThresholdValueFactory creates a IThresholdValue, and returns the minimum threshold value. Next, the IBinaryQueryElement invokes the CreateThresholdValue(String) method on the IThresholdValueFactory to create a maximum threshold value, passing the binary

query element type name as a string parameter. Then, the IThresholdValueFactory creates a IThresholdValue, and returns the maximum threshold value.

Finally, the CaMPaMS invokes the InitializeThreshold(Object) method on the IThresholdValue two times to pass the minimum threshold value and the maximum threshold value as an object parameters, and then the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.9. Scenario of Adding a Set Query Element

Figure 9 in Appendix E illustrates the interactions between the CaMPaMS and the framework to add a set query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Next, the CaMPaMS invokes the GetSetEvaluationOperators() method on the IContextInformationCollector, and then the IContextInformationCollector returns a collection of set evaluation operators.

Later, the CaMPaMS invokes the AddSetQueryElement(ISetQueryElement) method on the IContextMonitoringQuery, passing a set query element as a parameter. Consequently, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active, and then creates a ISetQueryElement. Next, the CaMPaMS invokes the Initialize(IContextInformationCollector, ISetEvaluationOperator,

IThresholdValueFactory) method on the ISetQueryElement, passing a context information collector, set evaluation operator, and a threshold value factory as parameters.

When the CaMPaMS have to add a new threshold value, it invokes the AddThresholdValue(Object) method on the ISetQueryElement, passing threshold value as an object parameter. Consequently, the ISetQueryElement invokes the CreateThresholdValue(String) method on the IThresholdValueFactory, passing the set query element type name as a string parameter. Then, the IThresholdValueFactory creates the IThresholdValue and returns the created threshold value. Next, the ISetQueryElement invokes the InitializeThreshold(Object) method on the IThresholdValue, passing threshold value as an object parameter.

Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.10. Scenario of Editing a Unary Query Element

Figure 10 in Appendix E illustrates the interactions between the CaMPaMS and the framework to edit a unary query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Next, the CaMPaMS invokes the following methods on the original IUnaryQueryElement: (1) the GetContextInformationCollector() method, and the original IUnaryQueryElement

returns the context information collector; (2) the GetUnaryEvaluationOperator() method, and the original IUnaryQueryElement returns the unary evaluation operator; (3) the GetThresholdValue() method, and the original IUnaryQueryElement returns the threshold value.

Next, the CaMPaMS invokes the RemoveUnaryQueryElement(IUnaryQueryElement) method on the IContextMonitoringQuery, passing a unary query element as a parameter. Accordingly, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active, and then destroys the original IUnaryQueryElement.

Later, the CaMPaMS invokes the GetUnaryEvaluationOperators() method on the IContextInformationCollector, and the IContextInformationCollector returns a collection of unary evaluation operators. Then, the CaMPaMS invokes the AddUnaryQueryElement(IUnaryQueryElement) method on the IContextMonitoringQuery, passing a unary query element as a parameter. Consequently, the IContextMonitoringQuery creates a new unary query element.

Next, the CaMPaMS invokes the Initialize(IContextInformationCollector, IUnaryEvaluationOperator, IThresholdValueFactory) method on the IUnaryQueryElement, passing the context information collector, unary evaluation operator, and threshold value factory. Accordingly, the IUnaryQueryElement invokes the CreateThresholdValue(String) method on the IThresholdValueFactory, passing the unary query element type name as a string parameter. The IThresholdValueFactory creates a threshold value, and returns the created threshold value. Then, the CaMPaMS

invokes the InitializeThreshold(Object) method on the IThresholdValue, passing a threshold value as an object parameter.

Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.11. Scenario of Editing a Binary Query Element

Figure 11 in Appendix E illustrates the interactions between the CaMPaMS and the framework to edit a binary query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Then, the CaMPaMS invokes the following methods on the edited IBinaryQueryElement: (1) the GetContextInformationCollector() method, and the IBinaryQueryElement returns the context information collector; (2) the GetBinaryEvaluationOperator() method, and the IBinaryQueryElement returns the binary evaluation operator; (3) the GetMinimumThresholdValue() method, and the IBinaryQueryElement returns the minimum threshold value; (4) the GetMaximumThresholdValue() method, and the IBinaryQueryElement returns the maximum threshold value.

Later, the CaMPaMS invokes the RemoveBinaryQueryElement(IBinaryQueryElement) method on the IContextMonitoringQuery, passing a unary query element as a parameter. Accordingly, the IContextMonitoringQuery invokes the Deactivate() method on the

IContextMonitoringQuery if the context monitoring query is active, and then destroys the edited IBinaryQueryElement.

Next, the CaMPaMS invokes the GetBinaryEvaluationOperators() method on the IContextInformationCollector, and the IContextInformationCollector returns a collection of binary evaluation operators. Then, the CaMPaMS invokes the AddBinaryQueryElement(IBinaryQueryElement) method on the IContextMonitoringQuery, passing a binary query element as a parameter. Consequently, the IContextMonitoringQuery creates a new binary query element. Then, the CaMPaMS invokes the Initialize(IContextInformationCollector, IBinaryEvaluationOperator, IThresholdValueFactory) method on the IBinaryQueryElement, passing the context information collector, binary evaluation operator, and threshold value factory. The IBinaryQueryElement invokes the following methods on the IThresholdValueFactory: (1) the CreateThresholdValue(String) method, passing the binary query element type name as a string parameter to create a minimum threshold value, and the IThresholdValueFactory creates a minimum threshold value, and then returns the created minimum threshold value; (2) the CreateThresholdValue(String) method, passing the binary query element type name as a string parameter to create a maximum threshold value, and the IThresholdValueFactory creates a maximum threshold value, and then returns the created maximum threshold value.

After that, the CaMPaMS invokes the InitializeThreshold(Object) method on the IThresholdValue, passing a minimum threshold value as an object parameter, and invokes the InitializeThreshold(Object) method on the IThresholdValue, passing a

maximum threshold value as an object parameter. Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.12. Scenario of Editing a Set Query Element

Figure 12 in Appendix E illustrates the interactions between the CaMPaMS and the framework to edit a set query element. First, the CaMPaMS invokes the FindById(Integer) method on the IContextMonitoringQueryRepository, passing a unique identity of a context monitoring query as an integer parameter, and the IContextMonitoringQueryRepository returns the context monitoring query. Next, the CaMPaMS invokes the following methods on the edited ISetQueryElement: (1) the GetContextInformationCollector() method, and the ISetQueryElement returns the context information collector; (2) the GetSetEvaluationOperator() method, and ISetQueryElement returns the set evaluation operator; (3) the GetThresholdValues() method, and the ISetQueryElement returns a set of threshold values.

Later, the CaMPaMS invokes the RemoveSetQueryElement(ISetQueryElement) method on the IContextMonitoringQuery, passing a set query element as a parameter. Accordingly, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active. Then, the IContextMonitoringQuery destroys the original ISetQueryElement.

After that, the CaMPaMS invokes the GetSetEvaluationOperators() method on the IContextInformationCollector, and the IContextInformationCollector returns a collection of set evaluation operators. Then, the CaMPaMS invokes the

AddSetQueryElement(ISetQueryElement) method on the IContextMonitoringQuery, passing set query element as a parameter. Consequently, the IContextMonitoringQuery creates a new set query element. Next, the CaMPaMS invokes the Initialize(IContextInformationCollector, ISetEvaluationOperator, IThresholdValueFactory) method on the ISetQueryElement, passing the context information collector, set evaluation operator, and threshold value factory.

When a CaMPaMS have to add new threshold values, it invokes the AddThresholdValue(Object) method on the ISetQueryElement, passing the threshold value as an object parameter. Consequently, the ISetQueryElement invokes the following methods on the IThresholdValueFactory: (1) the CreateThresholdValue(String) method, passing the set query element type name as a string parameter, and the IThresholdValueFactory creates IThresholdValue, and then returns the created threshold value; (2) the InitializeThreshold(Object) method, passing the threshold value as an object parameter.

Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.13. Scenario of Deleting a Query Element

Figure 13 in Appendix E illustrates the interactions between the CaMPaMS and the framework to delete a query element. First, if a unary query element must be deleted, the CaMPaMS invokes the RemoveUnaryQueryElement(IUnaryQueryElement) method on the IContextMonitoringQuery, passing a unary query element as a

parameter. Consequently, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active. Then, the IContextMonitoringQuery destroys the IUnaryQueryElement.

If a binary query element must be deleted, the CaMPaMS invokes the RemoveBinaryQueryElement(IBinaryQueryElement) method on the IContextMonitoringQuery, passing a binary query element as a parameter. Consequently, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active. Then, the IContextMonitoringQuery destroys the IBinaryQueryElement.

If a set query element must be deleted, the CaMPaMS invokes the RemoveSetQueryElement(ISetQueryElement) method on the IContextMonitoringQuery, passing a set query element as a parameter. Consequently, the IContextMonitoringQuery invokes the Deactivate() method on the IContextMonitoringQuery if the context monitoring query is active. Then, the IContextMonitoringQuery destroys the ISetQueryElement.

Finally, the CaMPaMS invokes the Update(IContextMonitoringQuery) on the IContextMonitoringQueryRepository, passing the updated context monitoring query as a parameter.

5.5.3.14. Scenario of Listing a Patient Profile

Figure 14 in Appendix E illustrates the interactions between the CaMPaMS and the framework to list a patient profile. First, the CaMPaMS invokes the GetKeys() method

on the IPatientProfileRepository, and the IPatientProfileRepository returns a collection of patient profile fields. Then the CaMPaMS invokes the GetValues() method on the IPatientProfileRepository, and the IPatientProfileRepository returns a collection of patient profile values.

5.5.3.15. Scenario of Editing a Patient Profile

Figure 15 in Appendix E illustrates the interactions between the CaMPaMS and the framework to edit a patient profile. First, the CaMPaMS invokes the FindByKey(String) method on the IPatientProfileRepository, passing a field name in the patient profile repository to obtain its value, and then the IPatientProfileRepository returns the value of a specific field in the patient profile repository. Finally, the CaMPaMS invokes the Update(String, Object) method on the IPatientProfileRepository, passing two parameters: the field name as a string unique identity and its updated value as an object.

5.5.3.16. Scenario of Deactivating a Context Monitoring Query

Figure 16 in Appendix E illustrates the interactions between the CaMPaMS and the framework to deactivate a context monitoring query. First, the CaMPaMS invokes the Deactivate() method on the IContextMonitoringQuery. Accordingly, the IContextMonitoringQuery invokes the RemoveObserver(IObserver) method on the OnQueryEvaluated Event located in the IContextMonitoringQueryEvaluator. Consequently, the OnQueryEvaluated Event located in the IContextMonitoringQueryEvaluator invokes the RemoveObserver(IObserver) method

on the OnStateChanged Event located in the IUnaryQueryElement, IBinaryQueryElement, and ISetQueryElement.

Accordingly, the OnStateChanged Event located in the IUnaryQueryElement, IBinaryQueryElement, and ISetQueryElement invokes the RemoveObserver(IObserver) method on the OnContextInformationCollected Event located in the IContextInformationCollector. Finally, the OnContextInformationCollected Event located in the IContextInformationCollector invokes the RemoveObserver(IObserver) method on the OnDataReceived Event located in the IDataSourceConnector and the OnDataConverted Event located in the IDataConverter.

5.5.3.17. Scenario of Activating a Continuous Instant Context Monitoring Query with a Unary Query Element

Figure 17 in Appendix E illustrates the interactions between the CaMPaMS and the framework to activate a continuous instant context monitoring query with a unary query element. Due to limitations of space, this scenario only covers activating a continuous instant context monitoring query with a unary query element. A continuous context monitoring query means that the evaluation period equals zero. An instant context monitoring query means that the alarm duration equals zero. The unary query element represents a query element with single threshold value. First, the CaMPaMS invokes the GetContextMonitoringQueries() method on the IContextMonitoringQueryRepository, and the IContextMonitoringQueryRepository returns a collection of context monitoring queries. Then, the CaMPaMS invokes the OnContextChanged() method on the IContextMonitoringQuery. Accordingly, the

IContextMonitoringQuery invokes the AddObserver(IObserver) method on the OnContextChanged Event, passing the CaMPaMS as an IObserver parameter.

Later, the CaMPaMS invokes the Activate() method asynchronously on the IContextMonitoringQuery. Consequently, the IContextMonitoringQuery invokes the GetEvaluationPeriod() method on the IContextMonitoringQuery. If the evaluation period is zero, the IContextMonitoringQuery invokes the OnQueryEvaluated() method on the IContextMonitoringQueryEvaluator. Then, the IContextMonitoringQueryEvaluator invokes the AddObserver(IObserver) method on the OnQueryEvaluated Event, passing the IContextMonitoringQuery as an IObserver parameter. Accordingly, the IContextMonitoringQuery invokes the Evaluate(IContextMonitoringQuery) method asynchronously on the IContextMonitoringQueryEvaluator. Then, the IContextMonitoringQueryEvaluator invokes the OnStateChanged() method on the IUnaryQueryElement. Next, the IUnaryQueryElement invokes the AddObserver(IObserver) method on the OnStateChanged Event, passing the IContextMonitoringQueryEvaluator as an IObserver parameter. The IContextMonitoringQueryEvaluator invokes the Evaluate() method asynchronously on the IUnaryQueryElement.

Next, the IUnaryQueryElement invokes the OnContextInformationCollected() method on the IContextInformationCollector. Then, the IContextInformationCollector invokes the AddObserver(IObserver) method on the OnContextInformationCollected Event, passing the IUnaryQueryElement as an IObserver parameter. After that, the IUnaryQueryElement invokes the CollectData() method asynchronously on the IContextInformationCollector. The IContextInformationCollector invokes the

Add(String, Object) method on the IConnectionArgs, passing the key as a string parameter and the value as an object parameter. Then, the IContextInformationCollector invokes the OnDataReceived() method on the IDataSourceConnector. Next, the IDataSourceConnector invokes the AddObserver(IObserver) method on the OnDataReceived Event, passing the IContextInformationCollector as an IObserver parameter. The IContextInformationCollector also invokes the GetDataAsynchronously(IConnectionArgs) method asynchronously on the IDataSourceConnector, passing the connection arguments as a parameter. Next, the IDataSourceConnector invokes the GetValue(String) method on the IConnectionArgs, passing an argument name as a string parameter.

If the IDataSourceConnector receives new data from a data source, it invokes the CreateDataValue(String) method on the IDataValueFactory, passing the data source connector name as a string parameter. Accordingly, the IDataValueFactory creates an IDataValue, and returns a data value. Next, the IDataSourceConnector invokes the InitializeUnspecifiedData(String, Object) method on the IDataValue, passing the key as a string parameter and value as an object parameter. Then, the IDataSourceConnector invokes the Notify() method on the OnDataReceived Event. Accordingly, the OnDataReceived Event invokes the Update(Object, IDataValue) callback method on the IContextInformationCollector, passing the IDataSourceConnector as an object parameter and the raw data value as a parameter.

Later, the IContextInformationCollector invokes the OnDataConverted() method on the IDataConverter. Accordingly, the IDataConverter invokes the

AddObserver(IObserver) method on the OnDataConverted Event, passing the IContextInformationCollector as an IObserver parameter. Next, the IContextInformationCollector invokes the Convert(IDataValue) method asynchronously on the IDataConverter, passing the received raw data value from the IDataSourceConnector as a parameter.

If the IDataConverter accomplishes converting the data, it invokes the CreateDataValue(String) method on the IDataValueFactory, passing the data converter name as a string parameter. Consequently, the IDataValueFactory creates an IDataValue, and returns a data value. Next, the IDataConverter invokes the InitializeSpecifiedData(String, Object) method on the IDataValue, passing the key as a string parameter and converted value as an object parameter. Then, the IDataConverter invokes the Notify() method on the OnDataConverted Event. Accordingly, the OnDataConverted Event invokes the Update(Object, IDataValue) callback method on the IContextInformationCollector, passing the IDataConverter as an object parameter and the converted data value as a parameter.

If the IContextInformationCollector receives data, it invokes the CreateDataValue(String) method on the IDataValueFactory, passing the context information collector name as a string parameter. Then, the IDataValueFactory creates an IDataValue and returns a data value. Next, the IContextInformationCollector invokes the InitializeSpecifiedData(String, Object) method on the IDataValue, passing the key as a string parameter and the collected data value as an object parameter. Then, the IContextInformationCollector invokes the Notify() method on the OnContextInformationCollected Event. The OnContextInformationCollected Event

invokes the Update(Object, IDataValue) callback method on the IUnaryQueryElement, passing the IContextInformationCollector as an object parameter and the collected context information data value as a parameter. After that, the IUnaryQueryElement invokes the Execute(IDataValue, IThresholdValue) method on the IUnaryEvaluationOperator passing the collected context information data value as a parameter and the threshold value as a parameter, and the IUnaryEvaluationOperator returns the evaluation state.

If the IUnaryQueryElement evaluation state changed, then it invokes the CreateDataValue(String) method on the IDataValueFactory, passing the context information unary query element name as a string parameter. Next, the IDataValueFactory creates an IDataValue, and the IDataValueFactory returns a data value. Then, the IUnaryQueryElement invokes the InitializeSpecifiedData(String, Object) method on the IDataValue, passing the key as a string parameter and the evaluation state value as an object parameter. After that, the IUnaryQueryElement invokes the Notify() method on the OnStateChanged Event. Then, the OnStateChanged Event invokes the Update(Object, IDataValue) callback method on the IContextMonitoringQueryEvaluator, passing the IUnaryQueryElement as an object parameter and the evaluation state as a data value parameter.

If the IContextMonitoringQueryEvaluator completes evaluating a context monitoring query, then it invokes the CreateDataValue(String) method on the IDataValueFactory, passing the context monitoring query evaluator name as a string parameter. Then, the IDataValueFactory creates an IDataValue and returns a data value. Next, the IContextMonitoringQueryEvaluator invokes the InitializeSpecifiedData(String,

Object) method on the IDataValue, passing the key as a string parameter and the evaluation state value as an object parameter. Then, the IContextMonitoringQueryEvaluator invokes the Notify() method on the OnQueryEvaluated Event. After that, the OnQueryEvaluated Event invokes the Update(Object, IDataValue) callback method on the IContextMonitoringQuery, passing the context monitoring query evaluator as an object parameter and the evaluation state as a data value parameter.

If the IContextMonitoringQuery evaluation state is changed, it invokes the GetAlarmDuration()method on the IContextMonitoringQuery. If the alarm duration of the context monitoring query equals zero, the IContextMonitoringQuery invokes the CreateDataValue(String) method on the IDataValueFactory, passing the context monitoring query evaluator name as a string parameter. Then, the IDataValueFactory creates an IDataValue and returns a data value. Next, the IContextMonitoringQuery invokes the InitializeSpecifiedData(String, Object) method on the IDataValue, passing the key as a string parameter and the evaluation state value as an object parameter. After that, the IContextMonitoringQuery invokes the Notify() method on the OnContextChanged Event. Finally, the OnContextChanged Event invokes the Update(Object, IDataValue) callback method on the CaMPaMS, passing the context monitoring query as an object parameter and the evaluation state as a data value parameter.

5.6. PSM Development

In this research, as elaborated in Section 3.7.2.2, the PIM was transformed using a C# model transformation into a PSM by using an automated tool, in addition to some

manual transformation. First, the getter and setter methods were replaced by C# property methods. Second, by following the implementation of the observer pattern proposed by Microsoft [295] and framework design guidelines [9], the ISubject, Event, and IObserver types were replaced by the generic event handler EventHandlet<T> provided by Mono. The generic type T is represented as an AbstractNotificationEventArgs that inherits the EventArgs type implemented by the Mono platform. The resulting PSM is shown in Figure 5.15.

5.7. Code Development

The following sub-sections present the default implementation of the CaMPaMF. Appendix F includes the class diagrams that illustrate the default implementation of the CaMPaMF.

5.7.1. IDataValue Default Implementation

The CaMPaMF provides five concrete classes as the default implementation of the IDataValue, which are: (1) IntegerDataValue to represent integer data values; (2) DecimalDataValue to represent decimal data values; (3) BooleanDataValue to represent boolean data values; (4) EvaluationStateDataValue to represent EvaluationState data values; and (5) ObjectDataValue to represent object data values.

5.7.2. IDataValueFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IDataValueFactory, which is DataValueFactory.

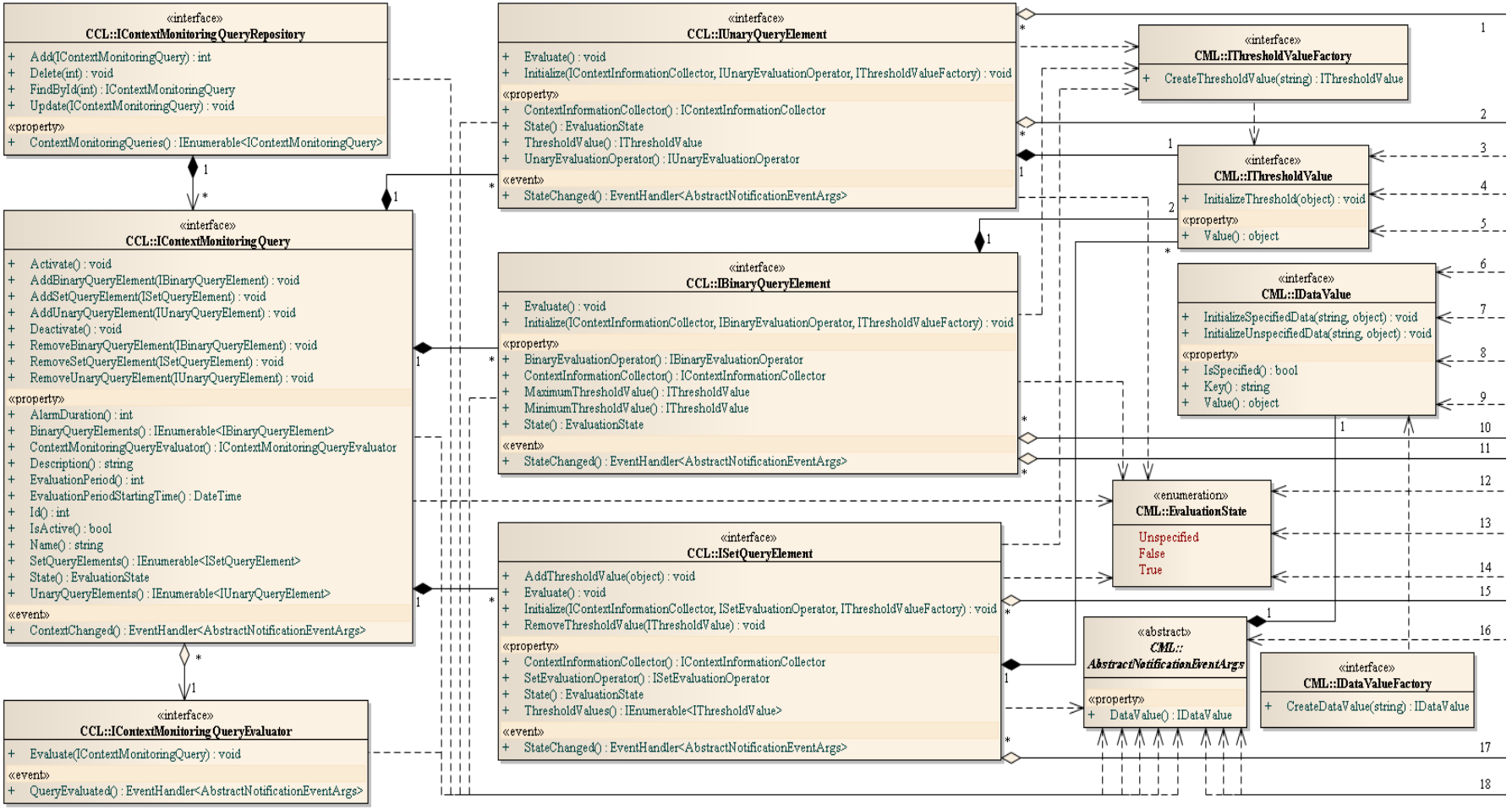

Figure 5.15. Platform specific model

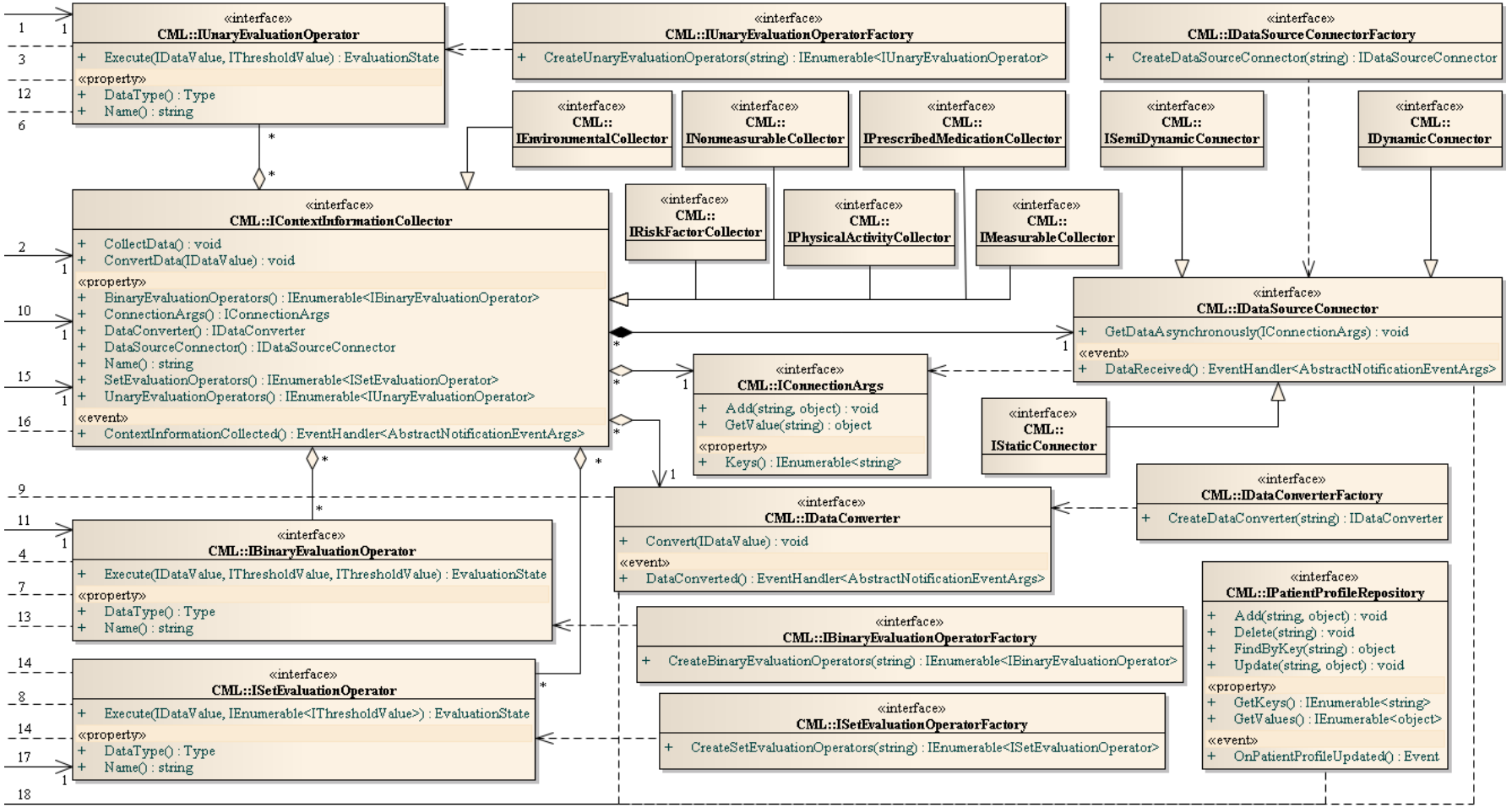

Figure 5.15 continued

5.7.3. AbstractNotificationEventArgs Default Implementation

The CaMPaMF provides five concrete derived classes as the default implementation of the AbstractNotificationEventArgs, which are: (1) IntegerNotificationEventArgs to represent integer event arguments; (2) DecimalNotificationEventArgs to represent decimal event arguments; (3) BooleanNotificationEventArgs to represent boolean event arguments; (4) EvaluationStateNotificationEventArgs to represent EvaluationState event arguments; and (5) ObjectNotificationEventArgs to represent object event arguments.

5.7.4. IPatientProfileRepository Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IPatientProfileRepository, which is PatientProfileRepository.

5.7.5. IConnectionArgs Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IConnectionArgs, which is ConnectionArgs.

5.7.6. IDataSourceConnectorFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IDataSourceConnectorFactory, which is DataSourceConnectorFactory.

5.7.7. IDataConverter Default Implementation

The CaMPaMF provides three concrete classes as the default implementation of the IDataConverter, which are: (1) IntegerDataConverter to convert to integer data; (2)

DecimalDataConverter to convert to decimal data; and (3) BooleanDataConverter to convert to boolean data.

5.7.8. IDataConverterFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IDataConverterFactory, which is DataConverterFactory.

5.7.9. IThresholdValue Default Implementation

The CaMPaMF provides three concrete classes as the default implementation of the IThresholdValue, which are: (1) IntegerThresholdValue to represent integer threshold values; (2) DecimalThresholdValue to represent decimal threshold values; and (3) BooleanThresholdValue to represent boolean threshold values.

5.7.10. IThresholdValueFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IThresholdValueFactory, which is ThresholdValueFactory.

5.7.11. IUnaryEvaluationOperator Default Implementation

The CaMPaMF provides 14 concrete classes as the default implementation of the IUnaryEvaluationOperator, which are: (1) IsEqualBoolean-EvaluationOperator; (2) IsEqualDecimalEvaluationOperator; (3) IsEqualInteger-EvaluationOperator; (4) IsGreaterThanDecimalEvaluationOperator; (5) IsGreaterThanIntegerEvaluationOperator; (6) IsGreaterThanOrEqualDecimalEvaluationOperator; (7) IsGreaterThanOrEqualIntegerEvaluationOperator; (8) IsLessThanDecimalEvaluationOperator;

(9) IsLessThanIntegerEvaluationOperator; (10) IsLess-ThanOrEqualDecimalEvaluationOperator; (11) IsLessThanOrEqualIntegerEvaluationOperator; (12) IsNotEqualBooleanEvaluationOperator; (13) IsNotEqualDecimalEvaluationOperator; and (14) IsNotEqualIntegerEvaluationOperator.

5.7.12. IUnaryEvaluationOperatorFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IUnaryEvaluationOperatorFactory, which is UnaryEvaluationOperatorFactory.

5.7.13. IBinaryEvaluationOperator Default Implementation

The CaMPaMF provides four concrete classes as the default implementation of the IBinaryEvaluationOperator, which are: (1) IsBetween-DecimalEvaluationOperator; (2) IsBetweenIntegerEvaluationOperator; (3) IsNot-BetweenDecimalEvaluationOperator; and (4) IsNotBetweenIntegerEvaluationOperator.

5.7.14. IBinaryEvaluationOperatorFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IBinaryEvaluationOperatorFactory, which is BinaryEvaluationOperatorFactory.

5.7.15. ISetEvaluationOperator Default Implementation

The CaMPaMF provides four concrete classes as the default implementation of the ISetEvaluationOperator, which are: (1) IsInSetDecimalEvaluationOperator; (2) IsInSetIntegerEvaluationOperator; (3) IsNotInSetDecimalEvaluationOperator; and (4) IsNotInSetIntegerEvaluationOperator.

5.7.16. ISetEvaluationOperatorFactory Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the ISetEvaluationOperatorFactory, which is SetEvaluationOperatorFactory.

5.7.17. IUnaryQueryElement Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IUnaryQueryElement, which is UnaryQueryElement.

5.7.18. IBinaryQueryElement Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IBinaryQueryElement, which is BinaryQueryElement.

5.7.19. ISetQueryElement Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the ISetQueryElement, which is SetQueryElement.

5.7.20. IContextMonitoringQueryEvaluator Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IContextMonitoringQueryEvaluator, which is ContextMonitoringQueryEvaluator.

5.7.21. IContextMonitoringQuery Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IContextMonitoringQuery, which is ContextMonitoringQuery.

5.7.22. IMonitoringQueryRepository Default Implementation

The CaMPaMF provides one concrete class as the default implementation of the IMonitoringQueryRepository, which is MonitoringQueryRepository.

5.8. Summary

In conclusion, this chapter has shown that using the layers architectural style best satisfies the reusability aspects and the CaMPaMF domain requirements in terms of connecting an unlimited number of sensors and an unlimited number of CaMPaMS. Additionally, using an asynchronous notification mechanism to transmit the data from layer to layer and among the components within the same layer supports real-time continuous CaMPaMS. Designing the architectural components to be hosted on mobile devices provides anywhere, anytime CaMPaMS. Furthermore, it was found that the data source collector, using the data source connector, is used to satisfy the requirements of collecting context information types from various context data sources. It was also shown that using the context monitoring query evaluator component satisfies the requirement of context reasoning, which is one of the primary elements of context-awareness computing. Finally, the chosen architectural design provides a solid foundation for efficient framework development.

Based on the framework design and implementation, it was found that the PIM was designed based on the reusability aspects and refined by identifying the hot spots and frozen spots and applying the SOLID design principles. Four design patterns were then adopted: the singleton, observer, strategy, and abstract factory design patterns. Additionally, 17 scenarios were presented to illustrate how the CaMPaMS interacts with the framework and how the framework reacts to the CaMPaMS calls. Finally, it

was shown that, in comparison to PSM and code, PIM is the most creative process and that it takes longer because it is not automatically generated.

CHAPTER SIX
FRAMEWORK TESTING AND DOCUMENTATION

6.1. Overview

In this chapter, the framework testing and documentation processes are described. Based on the framework design and implementation process outlined in Chapter 5, the resulting framework was used as an input to this process. The implementation of the four steps of the testing and documentation of the CaMPaMF is presented, starting with framework design guidelines application, followed by framework reusability evaluation, prototyping and documentation, and framework reusability expert review. Finally, a summary of the chapter is presented.

6.2. Framework Design Guidelines Application

The Microsoft static code analysis tool FxCop was used to analyse the CaMPaMF compiled code based on a number of design guidelines, which include 61 design rules, 10 globalization rules, 16 interoperability rules, 2 mobility rules, 23 naming rules, 16 performance rules, 3 portability rules, 21 security rules, 20 security transparency rules, and 39 usage rules [9]. The results show that the CaMPaMF satisfies 98.1% or 207 out of 211 of these guidelines, which confirms the CaMPaMF's reusability.

6.3. Framework Reusability Evaluation Using Reusability Model

As discussed in Section 3.8.1.2, the evaluation of the CaMPaMF using the adopted reusability model was conducted by applying four activities. The following subsections illustrate the results of applying the four activities.

6.3.1. Calculate Values of Metrics

The multi-metric approach was applied to the CaMPaMF to get an extensive idea of its complexity, coupling, and cohesion. Accordingly, fourteen metric values were calculated for each CaMPaMF component (i.e. interface, abstract class, and class), providing 1232 measurement values (88 CaMPaMF components, 14 metrics each). These metrics were grouped into complexity, coupling, or cohesion which included seven, four and three metrics respectively as shown in Table 6.17. The results of calculating the metrics values are shown in Appendix G.

Table 6.17

Multi-Metric Approach Applied to CaMPaMF

Group	Name	Description	Interpretation model / Design rule
Complexity	nr-methods	Number of methods of a CaMPaMF component.	CaMPaMF component should not have more than 50 methods [297].
	nr-long-methods	Number of methods longer than X LOC, (X=30).	CaMPaMF component should have no method longer than 30 lines of code [297].
	nr-public-methods	Number of public methods of a CaMPaMF component.	Keep class interface narrow [298].
	WMC	Weighted method count[299]. The weight=1 and complexity = nr-non-comment-LOC.	The larger WMC the more application-specific, the more effort needed for maintenance [299].
	avg-method-length	Average length of methods = WMC / nr-methods.	Redundant metric that summarizes the effects of WMC and number of methods [299].
	nr-attributes	Number of attributes of a CaMPaMF component.	Should not have more than 6 data members [300].
	nr-long-arguments	Number of methods with more than X arguments, (X = 6).	Reduce number of arguments of all methods to <= 6 [297].

Adopted from [10]

Table 6.17 *continued*

Group	Name	Description	Interpretation model / Design rule
Coupling	nr-references	Number of other classes components used by a CaMPaMF component.	It should not be coupled too tightly [301].
	nr-bidirect-refs	Number of bidirectional references (usages) of a CaMPaMF component with other classes.	It should have as few bidirectional references as possible [302].
	nr-abstract-refs	Number of usages of abstract classes or interfaces of a CaMPaMF component.	It should be coupled abstractly [99].
	nr-afferent-refs	Number of classes using a CaMPaMF component.	It should not be coupled to tightly [10].
Cohesion	LCOM	Lack of cohesion in methods [299]. Consider a Class C1 with n methods M1, M2… Mn. Let {Ij} = set of instance variables used by method Mi. There are n such sets {I1}, {I2}… {In}. Let P = {(Ii, Ij) | Ii ∩ Ij = 0} and Q = {(Ii, Ij) | Ii ∩ Ij ≠ 0}. If all n sets {I1}, {I2}… {In} are 0 then let P = 0. LCOM = |P|-|Q|, if |P|>|Q| = 0 otherwise.	The larger the number of similar methods, the more cohesive the class [299].
	LCOM 2	LCOM, which does not take methods without access to any attributes into account	The larger the number of similar methods that access at least one attribute, the more cohesive the class [299].
	nr-meth-w/o-attr	Number of methods of a CaMPaMF component that do not access any attributes of itself	Every method of a class should access an attribute of that class [301].

Adopted from [10]

6.3.2. Identify Thresholds of Metrics

In this activity, the thresholds of metrics were identified as shown in Table 6.18.

Table 6.18

Thresholds of Metrics

Group	Metric	Threshold
Complexity	nr-methods	Maximum threshold = 50
	nr-long-methods	Maximum threshold = 30
	nr-public-methods	Maximum threshold = average + standard deviation Maximum threshold = 4.23 + 4.65 = 8.87
	WMC	Maximum threshold = average + standard deviation Maximum threshold = 15.24 + 32.57 = 47.81
	avg-method-length	Maximum threshold = average + standard deviation Maximum threshold = 1.73 + 1.84 = 3.56
	nr-attributes	Maximum threshold = 6
	nr-long-arguments	Maximum threshold = 0
Coupling	nr-references	Maximum threshold = average + standard deviation Maximum threshold = 0 + 0 = 0
	nr-bidirect-refs	Maximum threshold = average + standard deviation Maximum threshold = 0 + 0 = 0
	nr-abstract-refs	Minimum threshold = average - standard deviation Maximum threshold = 2.60 – 2.11 = 0.49
	nr-afferent-refs	Maximum threshold = average + standard deviation Maximum threshold = 0 + 0 = 0
Cohesion	LCOM	Maximum threshold = 0
	LCOM 2	Maximum threshold = 0
	nr-meth-w/o-attr	Maximum threshold = average + standard deviation Maximum threshold = 0.25 + 0.44 = 0.69

6.3.3. Identify Outliers

In this activity, the outlier values of metrics were identified as shown in Table 6.19.

6.3.4. Design Review

Fifty-three out of the 1232 measurement values were outlier values, or 4.3%. Table 6.20 shows the outlier value percentage for the metrics that have outlier values. According to [10], if the outlier value percentage is less than 30% then there is no need

for refactoring. As shown in Table 6.20, all the outlier value percentages are less than 30%. Therefore, the CaMPaMF is reusable and there is no need for refactoring.

Table 6.19

Outlier Values of Metrics

Group	Metric	Outlier value
Complexity	nr-methods	Value > 50
	nr-long-methods	Value > 30
	nr-public-methods	Value > 8.87
	WMC	Value > 47.81
	avg-method-length	Value > 3.56
	nr-attributes	Value > 6
	nr-long-arguments	Value > 0
Coupling	nr-references	Value > 0
	nr-bidirect-refs	Value > 0
	nr-abstract-refs	Value < 0.49
	nr-afferent-refs	Value > 0
Cohesion	LCOM	Value > 0
	LCOM 2	Value > 0
	nr-meth-w/o-attr	Value > 0.69

Table 6.20

Outlier Value Percentage

Group	Metric	Outlier value percentage
Complexity	nr-public-methods	10.23%
	WMC	5.68%
	avg-method-length	6.82%
	nr-attributes	4.55%
Coupling	nr-abstract-refs	3.41%
Cohesion	LCOM 2	4.55%
	nr-meth-w/o-attr	25.00%

6.4. Prototyping and Documentation

This section starts by describing the CaMPaMF initialization procedure, followed by the implementation of three CaMPaMS as prototypes to demonstrate the reusability and extensibility of the CaMPaMF, which are hypertension a CaMPaMS, a diabetes

CaMPaMS, and an epilepsy CaMPaMS. However, due to space limitations, only the hypertension CaMPaMS will be included here; the diabetes and epilepsy CaMPaMS are included in Appendix H and Appendix I respectively. This section forms the framework documentation together with Chapter 5.

6.4.1. Framework Initialization

As shown in Figure 6.16, the framework initialization process starts when the CaMPaMS initialize the framework by calling the bootstrapper. Then the bootstrapper registers all the concrete classes mapped to their corresponding interfaces in the container. Once the CaMPaMS request to resolve the IMonitoringQueryRepository interface – the interface components of the CCL – the container will resolve the IMonitoringQueryRepository and all its dependences at once, acting as a domino effect. The CaMPaMF dependency graph is shown in Figure 6.17.

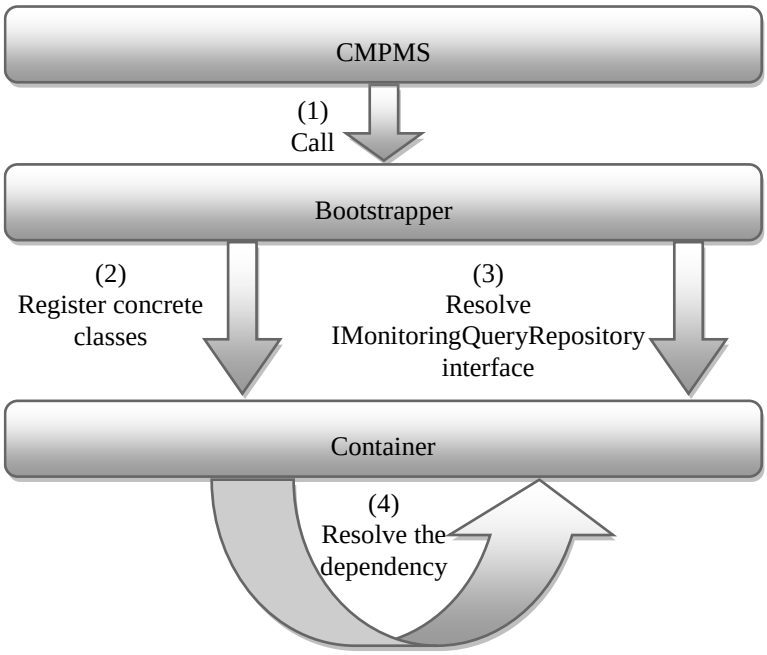

Figure 6.16. CaMPaMF initialization process

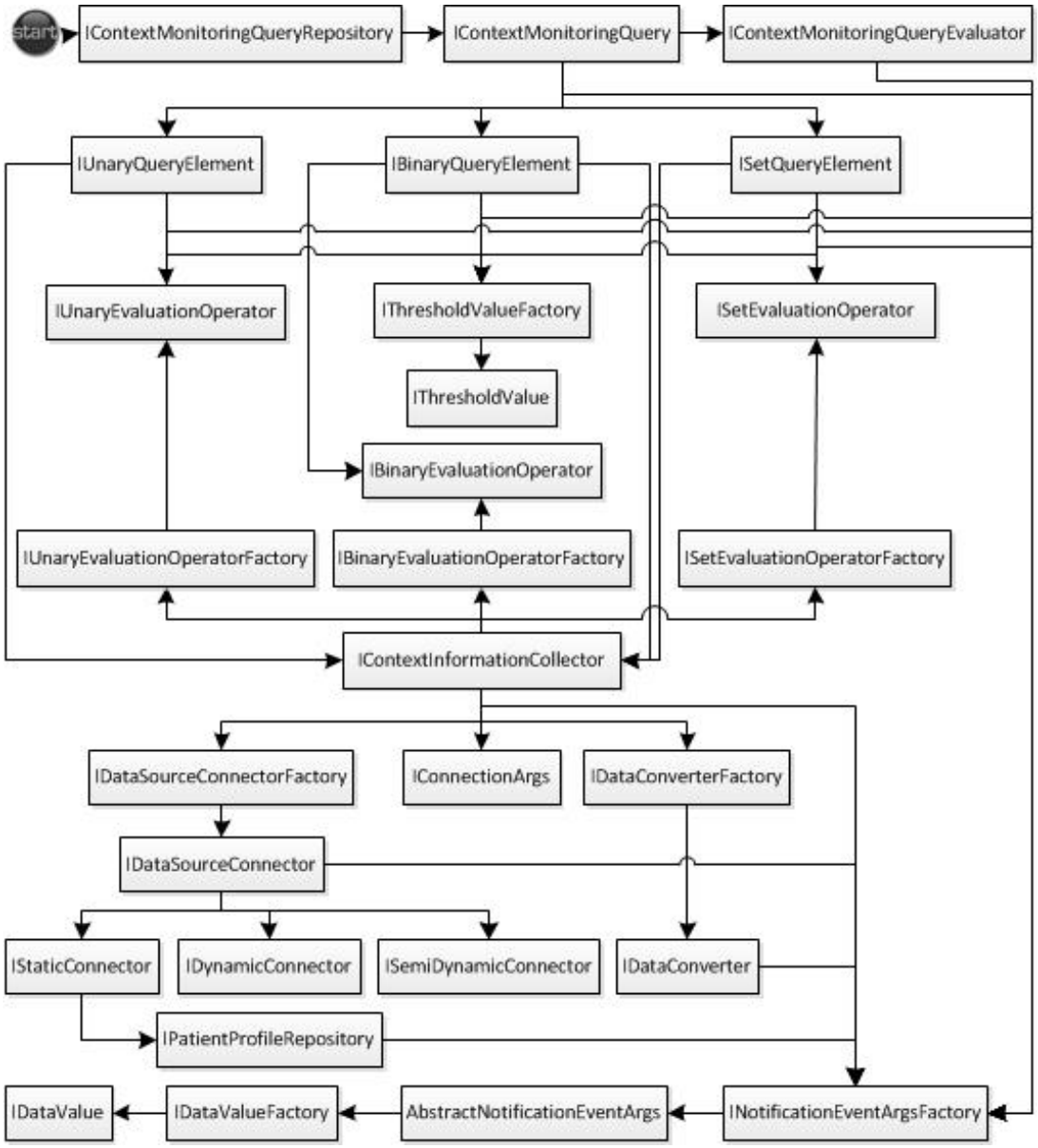

Figure 6.17. CaMPaMF dependency graph

Using the SimpleInjector component enables application developers to extend the framework and add, for example, an unlimited number of sensors by adding new a concrete class to implement the IDataSourceConnector interface. Additionally, the SimpleInjector component enables application developers to replace a particular sensor with a new one by replacing the map between the IDataSourceConnector

interface and the old sensor concrete class with the new data source connector concrete class. In other words, the SimpleInjector component allows for late bindings.

6.4.2. Hypertension CaMPaMS

Following the scenario of monitoring a hypertensive patient in Section 4.4.1.1, Figure 6.18 illustrates four context monitoring queries required to monitor a hypertensive patient. As shown in Figure 6.18, each query has eight query elements and one property, which is the alarm duration. If the alarm duration is equal to zero, an instant notification from the CaMPaMF will be raised once the query evaluation state is changed. Otherwise, if the alarm duration is greater than zero, the CaMPaMF will not raise any notification until the alarm duration has elapsed without any change in the query evaluation state.

The first two query elements are based on systolic BP and diastolic BP, which are classified in the CaMPaMF as measurable context information types. The physical activity in the third query element is classified in the CaMPaMF as a physical activity context information type. The prescribed medication in the fourth query element is classified in the CaMPaMF as a prescribed medication context information type. The last four query elements (aging, smoking, obesity, and chronic disease) are classified in the CaMPaMF as risk factors context information types. The following sub-section demonstrates how a developer can reuse and extend the CaMPaMF to develop the hypertension CaMPaMS based on these monitoring queries.

6.4.2.1. Hypertension CaMPaMS Implementation Based on the CaMPaMF

In order to simplify demonstrating the system implementation, this section is divided and arranged into further sub-sections according to the usage of the CaMPaMF architectural components. Moreover, Appendix J includes screen snapshots of Hypertension CaMPaMS implementation.

High BP 1 = { **(** *systolic BP* > 142 mmHg **) AND**
(*diastolic BP* > 85.5 mmHg **) AND**
(*physical activity* = walking **) AND**
(*alarm duration* = 0 minute **) AND**
(*prescribed medication* = calcium-channel blocker **) AND**
(*aging* = false **) AND**
(*smoking* = false **) AND**
(*obesity* = false **) AND**
(*chronic disease* = false **)**
}

High BP 2 = { **(** *systolic BP* > 130 mmHg **) AND**
(*diastolic BP* > 80 mmHg **) AND**
(*physical activity* = resting **) AND**
(*alarm duration* = 30 minute **) AND**
(*prescribed medication* = calcium-channel blocker **) AND**
(*aging* = false **) AND**
(*smoking* = false **) AND**
(*obesity* = false **) AND**
(*chronic disease* = false **)**
}

High BP 3 = { **(** *systolic BP* > 120 mmHg **) AND**
(*diastolic BP* > 72.4 mmHg **) AND**
(*physical activity* = sleeping **) AND**
(*alarm duration* = 0 minute **) AND**
(*prescribed medication* = calcium-channel blocker **) AND**
(*aging* = false **) AND**
(*smoking* = false **) AND**
(*obesity* = false **) AND**
(*chronic disease* = false **)**
}

High BP 4 = { **(** *systolic BP* > 130.3 mmHg **) AND**
(*diastolic BP* > 81.1 mmHg **) AND**
(*physical activity* = watching TV **) AND**
(*alarm duration* = 0 minute **) AND**
(*prescribed medication* = calcium-channel blocker **) AND**
(*aging* = false **) AND**
(*smoking* = false **) AND**
(*obesity* = false **) AND**
(*chronic disease* = false **)**
}

Figure 6.18. Hypertension context monitoring queries

6.4.2.1.1. Context Information Collector Component

With reference to Figure 1 in Appendix K, for each context information type, a context information collector must be created by implementing the suitable marker interface of the IContextInformationCollector. Accordingly, the first two measurable context information types must be created by extending the CaMPaMF by implementing the IMeasurableCollector as SystolicBloodPressureCollector and DiastolicBloodPressureCollector respectively. Furthermore, the physical activity context information type must be created by extending the CaMPaMF by implementing the IPhysicalActivityCollector as PhysicalActivityCollector. Additionally, the prescribed medication context information type must be created by extending the CaMPaMF by implementing the IPrescribedMedicationCollector as CalciumChannelBlockerCollector. The last four risk factors context information types must be created by extending the CaMPaMF by implementing the IRiskFactorCollector as AgingCollector, SmokingCollector, ObesityCollector, and ChronicDiseaseCollector.

For each context information collector, a connection argument must be created by implementing the IConnectionArgs interface. In the CaMPaMF, the default implementation of the IConnectionArgs interface, which is the ConnectionArgs class, can be reused.

For each context information collector, one or more evaluation operator must be created. Each evaluation operator must be created by implementing one of the following interfaces: IUnaryEvaluationOperator, IBinaryEvaluationOperator, or ISetEvaluationOperator.

In the CaMPaMF, there are 14 default implementations of the IUnaryEvaluationOperator interface that cover evaluating data of three types: integer, decimal, and boolean. Accordingly, for the SystolicBloodPressureCollector and DiastolicBloodPressureCollector, six classes among the 14 default implementations of the IUnaryEvaluationOperator interface are applicable to be reused, such as the IsLessThanDecimalEvaluationOperator. For the PhysicalActivityCollector two evaluation operator classes must be created by extending the CaMPaMF by implementing the IUnaryEvaluationOperator interface, which are the IsEqualPhysicalActivityEvaluationOperator and the IsNotEqualPhysicalActivityEvaluationOperator. Additionally, for the CalciumChannelBlockerCollector, AgingCollector, SmokingCollector, ObesityCollector, and ChronicDiseaseCollector, two classes among the 14 default implementations of the IUnaryEvaluationOperator interface are applicable to be reused, such as the IsNotEqualBooleanEvaluationOperator.

In the CaMPaMF, there are four default implementations of the IBinaryEvaluationOperator interface that cover evaluating data of two types: integer and decimal. Accordingly, for the SystolicBloodPressureCollector and DiastolicBloodPressureCollector, two classes among the four default implementations of the IBinaryEvaluationOperator interface are applicable to be reused, which are the IsBetweenDecimalEvaluationOperator and the IsNotBetweenDecimalEvaluationOperator. For the other collectors, implementing the IBinaryEvaluationOperator interface is not applicable.

In the CaMPaMF, there are four default implementations of the ISetEvaluationOperator interface that cover evaluating data of two types: integer and decimal. Accordingly, for the SystolicBloodPressureCollector and DiastolicBloodPressureCollector, two classes among the four default implementations of the ISetEvaluationOperator interface are applicable to be reused, which are the IsInSetDecimalEvaluationOperator and the IsNotInSetDecimalEvaluationOperator. Additionally, for the PhysicalActivityCollector, two evaluation operator classes must be created by extending CaMPaMF by implementing the ISetEvaluationOperator interface, which are the IsInSetPhysicalActivityEvaluationOperator and IsNotInSetPhysicalActivityEvaluationOperator. For the other collectors, implementing the ISetEvaluationOperator interface is not applicable.

A unary evaluation operator factory must be created by implementing the IUnaryEvaluationOperatorFactory interface. In the CaMPaMF, the default implementation of the IUnaryEvaluationOperatorFactory, which is the UnaryEvaluationOperatorFactory class, can be reused. This unary evaluation operator factory must be used in each context information collector to create a collection of suitable unary evaluation operators. A binary evaluation operator factory must be created by implementing the IBinaryEvaluationOperatorFactory interface. In the CaMPaMF, the default implementation of the IBinaryEvaluationOperatorFactory, which is the BinaryEvaluationOperatorFactory class, can be reused. This binary evaluation operator factory must be used in each context information collector to create a collection of suitable binary evaluation operators. A set evaluation operator factory must be created by implementing the ISetEvaluationOperatorFactory interface. In the CaMPaMF, the default implementation of the ISetEvaluationOperatorFactory, which

is the SetEvaluationOperatorFactory class, can be reused. This set evaluation operator factory must be used in each context information collector to create a collection of suitable set evaluation operators.

6.4.2.1.2. Data Source Connector Component

With reference to Figure 2 in Appendix K, for each context information collector a data source connector must be created by implementing the suitable marker interface of the IDataSourceConnector. In the CaMPaMF, the IMeasurableCollector and the IPhysicalActivityCollector use a dynamic data source connector while the IPrescribedMedicationCollector and the IRiskFactorCollector use a static data source connector. Accordingly, for the SystolicBloodPressureCollector, DiastolicBlood-PressureCollector, and PhysicalActivityCollector, a SystolicBloodPressure-Connector, DiastolicBloodPressureConnector, and PhysicalActivityConnector must be created respectively by extending the CaMPaMF by implementing the IDynamicConnector. Additionally, for the CalciumChannelBlockerCollector, AgingCollector, SmokingCollector, ObesityCollector, and ChronicDiseaseCollector, a CalciumChannelBlockerConnector, AgingConnector, SmokingConnector, ObesityConnector, and ChronicDiseaseConnector must be created respectively by extending the CaMPaMF by implementing the IStaticConnector.

A data source connector factory must be created by implementing the IDataSourceConnectorFactory interface. In the CaMPaMF, the default implementation of the IDataSourceConnectorFactory, which is the DataSourceConnectorFactory class, can be reused. This data source connector factory

must be used in each context information collector to create its suitable data source connector, as shown in Figure 1 in Appendix K.

6.4.2.1.3. Data Converter Component

With reference to Figure 3 in Appendix K, for each context information collector a data converter must be created by implementing the IDataConverter interface. In the CaMPaMF, there are three default implementations of the IDataConverter interface that can be reused, which are the IntegerDataConverter class, the DecimalDataConverter class, and the BooleanDataConverter class. Accordingly, for the SystolicBloodPressureCollector and DiastolicBloodPressureCollector, the DecimalDataConverter class can be reused. While for the PhysicalActivityCollector a PhysicalActivityConverter must be created by extending the CaMPaMF by implementing the IDataConverter interface. Additionally, for the CalciumChannelBlockerCollector, AgingCollector, SmokingCollector, ObesityCollector, and ChronicDiseaseCollector the BooleanDataConverter class can be reused.

A data converter factory must be created by implementing the IDataConverterFactory interface. In the CaMPaMF, the default implementation of the IDataConverterFactory, which is the DataConverterFactory class, can be reused. This data converter factory must be used in each context information collector to create its suitable data converter, as shown in Figure 1 in Appendix K.

6.4.2.1.4. Context Monitoring Query Component

After preparing all the required implementation of the IContextInformationCollector and its dependency interfaces, which are the IDataSourceConnector, IDataConverter, IConnectionArgs, IUnaryEvaluationOperator, IBinaryEvaluationOperator, and ISetEvaluationOperator, each context monitoring query must be created by implementing the IContextMonitoringQuery interface. In the CaMPaMF, there is a default implementation of the IContextMonitoringQuery interface that can be reused, which is the ContextMonitoringQuery class, as shown in Figure 4 in Appendix K.

For each context monitoring query, one or more query element must be created. Each query element must be created by implementing one of the following interfaces: IUnaryQueryElement; IBinaryQueryElement; or ISetQueryElement. In the CaMPaMF, there are three default implementations of these interfaces, which are UnaryQueryElement class, BinaryQueryElement class, and SetQueryElement class respectively. Accordingly, for all the query elements illustrated in Figure 6.18, the UnaryQueryElement class can be reused.

For each unary query element, the suitable context information collector, the unary evaluation operator, and the threshold value must be initialized. However, the threshold value must be created by implementing the IThresholdValue interface. In the CaMPaMF, there are three default implementations of the IThresholdValue interface that can be reused, which are the IntegerThresholdValue class, the DecimalThresholdValue class, and the BooleanThresholdValue class. Accordingly, the first query element for example in the first context monitoring query is initialized with the SystolicBloodPressureCollector as context information collector, the

IsGreaterThanDecimalEvaluationOperator as unary evaluation operator, and the DecimalThresholdValue as threshold value. However, for the PhysicalActivityCollector a PhysicalThresholdValue must be created by extending the CaMPaMF by implementing the IThresholdValue interface.

A threshold value factory must be created by implementing the IThresholdValueFactory interface. In the CaMPaMF, the default implementation of the IThresholdValueFactory, which is the ThresholdValueFactory class, can be reused. This threshold value factory must be used in each query element to create its suitable threshold value.

6.4.2.1.5. Context Monitoring Query Evaluator Component

With reference to Figure 5 in Appendix K, for each context monitoring query a context monitoring query evaluator must be created by implementing the IContextMonitoringQueryEvaluator interface. In the CaMPaMF, the default implementation of the IContextMonitoringQueryEvaluator, which is the ContextMonitoringQueryEvaluator class, can be reused.

6.4.2.1.6. Notification Event Argument

With reference to Figure 6 in Appendix K, for each data source connector the suitable notification event argument must be created by inheriting the AbstractNotificationEventArgs abstract class. The AbstractNotificationEventArgs encapsulates the data value that must be passed among CaMPaMF components. The data value must be created by implementing the IDataValue interface. In the CaMPaMF, there are five default implementations of the IDataValue interface that can

be reused, which are the IntegerDataValue class, the DecimalDataValue class, the BooleanDataValue class, the ObjectDataValue class, and the EvaluationStateDataValue class. Accordingly, there are five default derived implementations of the AbstractNotificationEventArgs abstract class that can be reused, which are the IntegerNotificationEventArgs, DecimalNotificationEventArgs, BooleanNotificationEventArgs, EvaluationStateNotificationEventArgs, and ObjectNotificationEventArgs. However, the ObjectNotificationEventArgs can be reused by all the data source connectors to pass their collected raw data to the registered context information collectors.

For each data converter, the IntegerNotificationEventArgs, DecimalNotification-EventArgs, or BooleanNotificationEventArgs can be reused to pass its converted data to the registered context information collectors. Accordingly, for the DecimalDataConverter the DecimalNotificationEventArgs class can be reused. While for the PhysicalActivityConverter the PhysicalActivityNotificationEventArgs must be created by extending the CaMPaMF by inheriting from the AbstractNotificationEventArgs abstract class. A PhysicalActivityDataValue must also be created by extending the CaMPaMF by implementing the IDataValue interface to be encapsulated in the PhysicalActivityNotificationEventArgs. Additionally, for the BooleanDataConverter, the BooleanNotificationEventArgs class can be reused.

Similarly, for each context information collector, the IntegerNotificationEventArgs, DecimalNotificationEventArgs, or BooleanNotificationEventArgs can be reused to pass its collected data to the registered query elements. Accordingly, for the SystolicBloodPressureCollector and DiastolicBloodPressureCollector, the

DecimalNotificationEventArgs class can be reused. While for the extended PhysicalActivityCollector, the extended PhysicalActivityNotificationEventArgs, with its extended PhysicalActivityDataValue, can be reused. Additionally, for the CalciumChannelBlockerCollector, AgingCollector, SmokingCollector, ObesityCollector, and ChronicDiseaseCollector, the BooleanNotificationEventArgs class can be reused.

Furthermore, for each query element, the EvaluationStateNotificationEventArgs can be reused to pass its evaluation state data to the registered context monitoring query evaluator. Additionally, the context monitoring query evaluator can reuse the EvaluationStateNotificationEventArgs to pass the evaluation result to the registered context monitoring query. For each context monitoring query, the EvaluationStateNotificationEventArgs can be reused to pass its evaluation state data to the registered CaMPaMS.

A notification event argument factory must be created by implementing the INotificationEventArgsFactory interface. In the CaMPaMF, the default implementation of the INotificationEventArgsFactory, which is the NotificationEventArgsFactory class, can be reused. This notification event argument factory must be used in all CaMPaMF components that raise events to create its suitable notification event argument.

A data value factory must be created by implementing the IDataValueFactory interface. In the CaMPaMF, the default implementation of the IDataValueFactory,

which is DataValueFactory class, can be reused. This data value factory must be used in each notification event argument to create its suitable data value.

6.4.2.1.7. Context Monitoring Query Repository Component

Context monitoring queries must be created within the context monitoring query repository. The context monitoring query repository must be created by implementing the IContextMonitoringQueryRepository interface. In the CaMPaMF, the default implementation of the IContextMonitoringQueryRepository, which is the ContextMonitoringQueryRepository class, can be reused, as shown in Figure 7 in Appendix K.

6.4.2.1.8. Patient Profile Repository Component

The patient profile must be stored in a patient profile repository. The patient profile repository must be created by implementing the IPatientProfileRepository interface. In the CaMPaMF, the default implementation of the IPatientProfileRepository, which is the PatientProfileRepository class, can be reused, as shown in Figure 8 in Appendix K.

6.5. Amount of Reuse Calculation

As discussed in Section 3.8.1.3, the calculation of how much of each CaMPaMS prototype is reused involves three metrics: the reuse level metric; the reuse frequency metric; and the reuse size and frequency metric. The following subsections present the results of applying the three metrics.

6.5.1. Reuse Level (RL)

The RL is calculated as the ratio of the Number of Reused Items (NRI) to the Total Number of Items (TNI) [18], as shown in Equation 6.1.

$$\mathrm{RL} = \frac{\mathrm{NRI}}{\mathrm{TNI}} \quad (6.1)$$

Where the NRI is the number of CaMPaMF reused components and the TNI is the total number of both CaMPaMF reused components and CaMPaMS components. The calculated value of RL is between 0 and 1. The results of this calculation are shown in Table 6.21. With reference to Table 6.21, the average value of RL was 0.88, which demonstrates a high level of reuse.

Table 6.21

Reuse Level of CaMPaMS Prototypes

CaMPaMS prototypes	NRI	TNI	RL
Hypertension CaMPaMS	198	230	0.86
Diabetes CaMPaMS	198	223	0.89
Epilepsy CaMPaMS	198	224	0.88
Average			**0.88**

6.5.2. Reuse Frequency (RF)

The RF is calculated as the ratio of the Number of References to the Reused Items (NRRI) to the Total Number of References (TNR) [18], as shown in Equation 6.2.

$$\mathrm{RF} = \frac{\mathrm{NRRI}}{\mathrm{TNR}} \quad (6.2)$$

Where the NRRI is the number of references to CaMPaMF reused components and the TNR is the total number of both the references to CaMPaMF reused components and

the references to CaMPaMS components. The calculated value of RF is between 0 and 1. The results of this calculation are shown in Table 6.22. With reference to Table 6.22, the average value of RF was 0.87, which demonstrates a high frequency of reuse.

Table 6.22

Reuse Frequency of CaMPaMS Prototypes

CaMPaMS prototypes	NRRI	TNR	RF
Hypertension CaMPaMS	1194	1331	0.86
Diabetes CaMPaMS	1005	1152	0.87
Epilepsy CaMPaMS	1007	1161	0.87
Average			**0.87**

6.5.3. Reuse Size and Frequency (RSF)

The RSF is the ratio of the number of references to reused items to the size of the items (lines of code) [294]. The RSF is calculated based on the Expanded Size (ES) and the Total Lines of Code (TLOC), as shown in Equation 6.3.

$$\mathrm{RSF} = \frac{\mathrm{ES} - \mathrm{TLOC}}{\mathrm{ES}} \tag{6.3}$$

Where the ES is calculated based on the Line of Code (LOC) for each item and its Number of References (NR), as shown in Equation 6.4.

$$\mathrm{ES} = \sum_{i=0}^{n} \mathrm{LOC}(\mathrm{Item_i}) \times \mathrm{NR}(\mathrm{Item_i}) \tag{6.4}$$

The calculated value of RSF is between 0 and 1. The results of this calculation are shown in Table 6.23. With reference to Table 6.23, the average value of RSF was 0.72, which demonstrates a high reuse size and frequency.

Table 6.23

Reuse Size and Frequency of CaMPaMS Prototypes

CaMPaMS prototypes	ES	TLOC	RSF
Hypertension CaMPaMS	11830	3196	0.73
Diabetes CaMPaMS	10033	2925	0.71
Epilepsy CaMPaMS	10075	2935	0.71
Average			**0.72**

6.6. Framework Reusability Evaluation Using Software Expert Review

This section presents the fourth activity of framework reusability evaluation, which is expert review. The findings of the expert review are elaborated in the following subsections.

6.6.1. Demographic Profiles of Software Experts

As shown in Table 6.24, the demographic data collected in this research were the experts' specialization, age, experience and gender. The following subsections discuss each of these in turn.

Table 6.24

Demographic Profiles of Experts

No.	Specialization	Age	Experience (years)	Gender
1	Solutions designer	33	10	Male
2	Senior software engineer	35	14	Male
3	Software analyst and database architect	43	20	Male
4	System consultant	35	13	Male

6.6.1.1. Software Experts' Specialization

The four experts represented different specializations and were classified as one of the following: solution designer; senior software engineer; software analyst and database architect; system consultant. Figure 6.19 shows that 1 or 25% was a solutions designer,

1 was senior software engineer, 1 was software analyst and database architect, and 1 was system consultant.

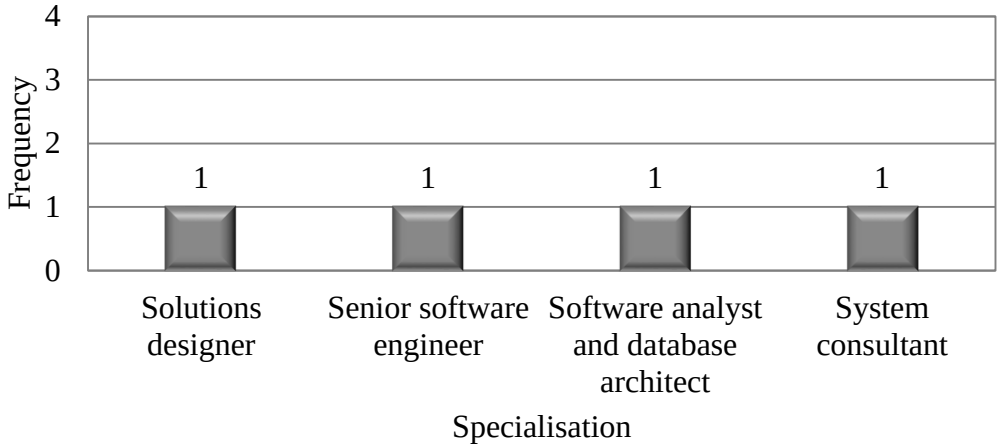

Figure 6.19. Software experts' specialisation

6.6.1.2. Software Experts' Ages

The age of experts varied from 33 to 43, which shows their level of maturity for giving opinions and assessments and suitability for the expert review activity. Figure 6.20 shows that 1 or 25% was 33 years old, 2 or 50% were 35 years old, and 1 was 43 years old.

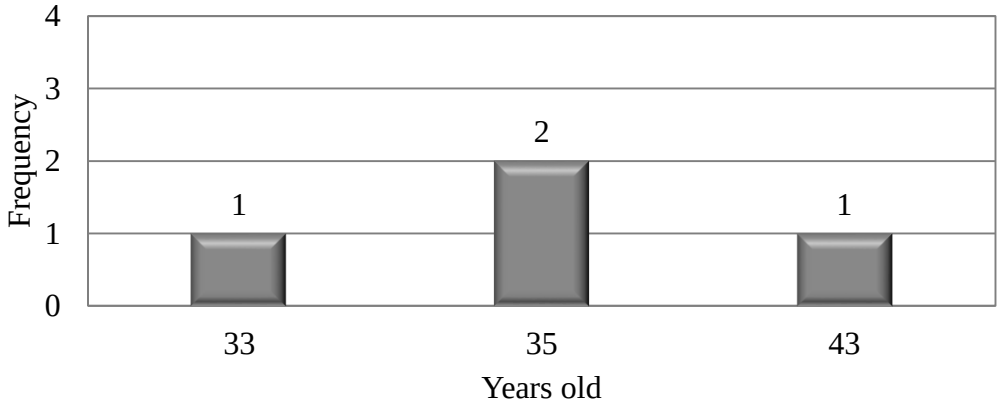

Figure 6.20. Software experts' ages

6.6.1.3. Software Experts' Experience

The experience of the experts in their respective specializations varied from 10 to 20 years, which fulfils the requirements of "expert" in this research. Figure 6.21 shows that 1 out of the 4 or 25% had 10 years' experience, 1 had 14 years' experience, 1 had 20 years' experience, and 1 had 13 years' experience.

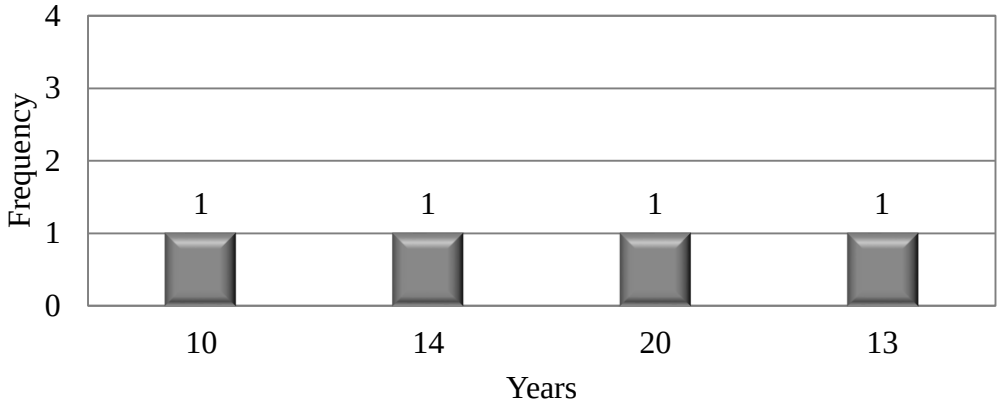

Figure 6.21. Software experts' experience

6.6.1.4. Software Experts' Genders

Figure 6.22 shows that all (100%) of the experts were male.

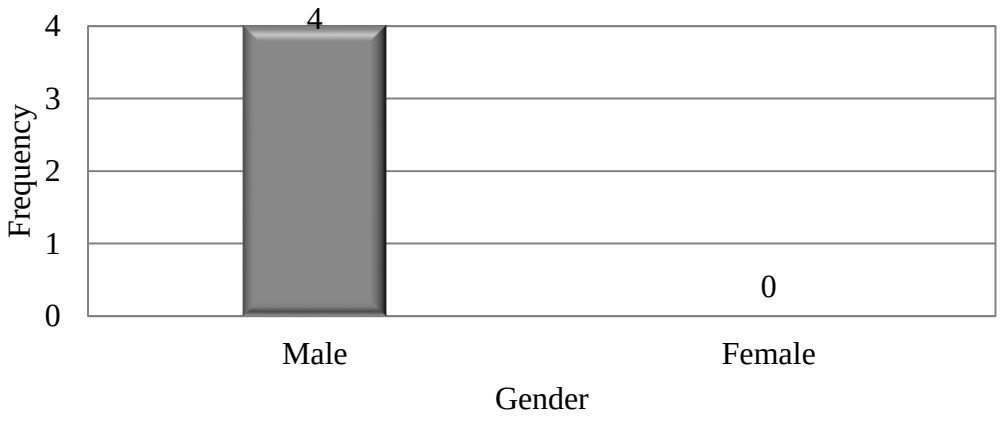

Figure 6.22. Software experts' genders

6.6.2. Frequency of Responses from Software Expert Review Instrument

The data from the software expert review instrument was collected and the frequency of responses for each question is illustrated in Appendix M. The majority of the experts agreed that the proposed CaMPaMF has acceptable complexity, coupling, cohesiveness, modularity, simplicity, abstraction, flexibility, understandability, and reusability.

First, all of the four experts agreed that 78 of the 88 (88.6%) of the proposed design of the CaMPaMF interfaces or classes had acceptable complexity without modification. Additionally, 75% of the experts agreed that 4 of the 88 (4.5%) of the CaMPaMF interfaces or classes had acceptable complexity without modification. Moreover, 50% of the experts agreed that 6 of the 88 (6.8%) of the CaMPaMF interfaces or classes had acceptable complexity without modification. Furthermore, 50% of the experts agreed that 6 of the 88 (6.8%) of the CaMPaMF interfaces or classes had acceptable complexity with modification. Additionally, 25% of the experts agreed that 4 of the 88 (4.5%) of the CaMPaMF interfaces or classes had acceptable complexity with modification. Accordingly, only 10 of the 88 (11.4%) of the CaMPaMF interfaces or classes had acceptable complexity with suggested modifications. Based on [10], if the percentage of the interfaces or classes that need modification is less than 30% then there is no need for refactoring. Therefore, the complexity of the CaMPaMF is acceptable and there is no need for refactoring.

Second, all of the four experts agreed that 85 of the 88 (96.6%) of the proposed design of the CaMPaMF interfaces or classes had acceptable coupling without modification. Additionally, 50% of the experts agreed that 3 of the 88 (3.4%) of the CaMPaMF

interfaces or classes had acceptable coupling without modification. Furthermore, 50% of the experts agreed that 3 of the 88 (3.4%) of the CaMPaMF interfaces or classes had acceptable coupling with modification. Accordingly, only 3 of the 88 (3.4%) of the CaMPaMF interfaces or classes had acceptable coupling with suggested modifications. Based on [10], if the percentage of the interfaces or classes that need modification is less than 30% then there is no need for refactoring. Therefore, the coupling of the CaMPaMF is acceptable and there is no need for refactoring.

Third, all of the four experts agreed that 62 of the 88 (70.5%) of the proposed design of the CaMPaMF interfaces or classes had acceptable cohesiveness without modification. Additionally, 75% of the experts agreed that 22 of the 88 (25%) of the CaMPaMF interfaces or classes had acceptable cohesiveness without modification. Moreover, 50% of the experts agreed that 4 of the 88 (4.5%) of the CaMPaMF interfaces or classes had acceptable cohesiveness without modification. Additionally, 50% of the experts agreed that 4 of the 88 (4.5%) of the CaMPaMF interfaces or classes had acceptable cohesiveness with modification. Furthermore, 25% of the experts agreed that 22 of the 88 (25%) of the CaMPaMF interfaces or classes had acceptable cohesiveness with modification. Accordingly, only 26 of the 88 (29,5%) of the CaMPaMF interfaces or classes had acceptable cohesiveness with suggested modifications. Based on [10], if the percentage of the interfaces or classes that need modification is less than 30% then there is no need for refactoring. Therefore, the cohesiveness of the CaMPaMF is acceptable and there is no need for refactoring.

Fourth, based on the results of the design rules section, it was found that all of the four experts agreed that 100% of the CaMPaMF interfaces or classes had acceptable modularity, simplicity, and abstraction without modification.

Fifth, based on the results of the design principles section, it was found that all of the four experts agreed that 100% of the CaMPaMF interfaces or classes had acceptable flexibility and understandability without modification.

Sixth, based on the results of the reusability factors section, it was found that all of the four experts agreed that 100% of the CaMPaMF interfaces or classes had acceptable reusability without modification.

In addition, further comments from one expert were collected and these are shown in Table 6.25. The expert suggested using class inheritance, however the CaMPaMF was designed according to the object-oriented design principle proposed in [99], which is to "favour object composition over class inheritance". Accordingly, the CaMPaMF's design favours composition over class inheritance. Unlike class inheritance, composition can be used by application developers to reuse frameworks by plugging in components at run-time with no programming. Hence, application developers, especially beginners, find reusing application frameworks by composition easier to learn and use because they do not have to learn the implementation of these frameworks [105]. Reusing application frameworks by class inheritance requires application developers to know the internal structure of these frameworks and thus reduces the framework's understandability [23]. Moreover, applications that are built on top of a framework using class inheritance are strongly dependent on the inherited

classes of the framework, thus the framework's flexibility (adaptability) is also reduced [23].

Table 6.25

Further Comments from the Software Experts

Comments
The researcher might evaluate creating a CCL:IBaseQueryElement that CCL:IUnaryQueryElement, CCL:IBinaryQueryElement and CCL:ISetQueryElement inherit from.
The researcher might evaluate creating a CML:IBaseEvaluationOperator that CML:IUnaryEvaluationOperator, CML:IBinaryEvaluationOperator and CML:ISetEvaluationOperator inherit from. This could lead to a change in the xyzEvaluationOperators() properties in CML:IContextInformationCollector.

6.7. Summary

In conclusion, this chapter has shown that the CaMPaMF satisfies all the framework design guidelines. Additionally, the CaMPaMF reusability evaluation based on the adopted reusability model shows that 53 out of 1232 measurement values were outlier values, or 4.3%. This value is much lower than 30% and thus verifies that there is no need for refactoring and that the CaMPaMF is reusable. Furthermore, the CaMPaMF was reused and extended successfully to develop three CaMPaMS, which are a hypertension CaMPaMS, a diabetes CaMPaMS, and an epilepsy CaMPaMS. Additionally, the amount of reuse was calculated for these prototypes by three metrics: the reuse level metric; the reuse frequency metric; and the reuse size and frequency metric. This yielded the average values of 0.88, 0.87, and 0.72 respectively. These values together reflect a high amount of reuse. Finally, the expert review of the framework reusability resulted in 100% of the experts agreeing that the proposed CaMPaMF is reusable.

CHAPTER SEVEN
CONCLUSION AND FUTURE WORK

7.1. Overview

This chapter presents the conclusions of this research. It starts by summarizing the research, and then outlines its contributions and limitations. Finally, directions for future work are suggested.

7.2. Research Summary

This research explored how to design a reusable CaMPaMF to enhance the overall development quality and overcome the development complexity of CaMPaMS, thus assisting developers to develop various CaMPaMS for different diseases to enable the elderly and chronic disease patients to monitor themselves using their mobile devices and wireless sensor technologies. The results of previous studies show that there is a recognized need to enhance the design of these application frameworks, with more emphasis on (1) reusability, which is the most important quality goal for application frameworks, and (2) domain requirements, which encapsulate the business activities in the CaMPaMS family of the biomedical informatics domain. An analysis of previous studies identified that there is no existing CaMPaMF that was both designed on multiple reusability aspects and evaluated using multiple reusability evaluation approaches. Furthermore, there is no existing CaMPaMF that integrates all of the identified domain requirements of CaMPaMS. In light of this, the following research objectives were arrived at:

1. Develop the domain model of CaMPaMS that should be addressed by an application framework.

2. Design a reusable application framework for CaMPaMS.

3. Evaluate the reusability of the designed application framework.

The following subsections are a summary of the findings of this study in relation to these objectives.

7.2.1. Domain Model of CaMPaMS

The first objective was to develop the domain model of CaMPaMS that should be addressed by an application framework. This objective was satisfied by the two main domain analysis research procedures: domain modelling and domain model validation. A description of the step-by-step implementation of the domain modelling activities to develop a domain model was followed by an outline of the domain model validation activities. The outcome of this procedure was the constructed domain model that captures the domain's requirements and identifies its concepts. In addition, it was found that the constructed domain model is complete, correct, and representative of the domain requirements. It provides a practical and effective means of monitoring chronic diseases such as hypertension, diabetes and epilepsy. Therefore, it provides a solid foundation for efficient framework development.

7.2.2. Design of Reusable Application Framework for CaMPaMS

The second objective was to design a reusable application framework for CaMPaMS. This objective was satisfied by a research procedure that involved the three processes

of the framework development: architectural design, framework design, and framework implementation. The implementation of the three steps to create the architecture of the CaMPaMF was conducted based on the reusability aspects, starting by identifying quality attributes, followed by selecting architectural styles, then constructing the architectural diagram. Then, the implementation of the three steps to design and implement the CaMPaMF based on the MDA approach was conducted, the steps being PIM development, PSM development, and code development.

In the architectural design process, the layers architectural style was selected to satisfy reusability aspects by minimizing complexity and coupling [103, 146], which are among the design rules that satisfy the three design principles (modularity, simplicity, and abstraction) [10]. These design principles positively affect the flexibility and understandability factors, which in turn improve the reusability [10]. The outcome of this process shows that using the layers architectural style satisfies the identified quality attribute of the CaMPaMF, i.e. reusability. It also satisfies the identified domain requirements in terms of connecting an unlimited number of sensors and an unlimited number of CaMPaMS. Additionally, using an asynchronous notification mechanism to transmit the data from layer to layer and among the components within the same layer supports real-time continuous CaMPaMS. Designing the architectural components to be hosted on mobile devices provides anywhere, anytime CaMPaMS. It was found that the data source collector, using the data source connector, satisfies the requirements of collecting context information types from various context data sources. Similarly, it was found that using the context monitoring query evaluator component satisfies the requirement of context reasoning, which is one of the primary

elements of context awareness computing. Finally, the architectural design provides a solid foundation for efficient framework development.

In the framework design and implementation processes, it was found that the PIM was designed and refined by using four techniques, starting by identifying the hot spots and frozen spots, then applying the SOLID design principles, and finally applying four design patterns (singleton, observer, strategy, and abstract factory). The hot spots were mapped to interfaces in the framework design. An interface-based design improves reusability by minimizing complexity and coupling [99, 161], which are among the design rules that should be satisfied as a reusability aspect [10]. Additionally, applying the observer, strategy, and abstract factory design patterns improves reusability by minimizing complexity and coupling and maximizing cohesion [99], which are the design rules that should be satisfied as a reusability aspect [10]. Moreover, 17 scenarios were presented to illustrate how the CaMPaMS interacts with the framework and how the framework reacts to CaMPaMS calls. It was found that, in comparison to PSM and code, PIM was the most creative process and that it took longer because it is not automatically generated.

7.2.3. Application Framework Reusability Evaluation

The last objective was to evaluate the reusability of the designed application framework. This objective was satisfied by a research procedure that involved framework testing and documentation processes. Based on the framework design and implementation process, the resulting framework was used as an input to this process. The implementation of the five steps of testing and documentation of the CaMPaMF was conducted, starting with framework design guidelines application, then

framework reusability evaluation, then prototyping and documentation, followed by calculating the amount of reuse, and finally expert review of the framework reusability. The outcome of this procedure shows that the CaMPaMF satisfies all the framework design guidelines. Additionally, the CaMPaMF reusability evaluation based on the adopted reusability model [10] shows that 53 out of 1232 measurement values were outlier values, or 4.3%. According to [10], if the outlier value percentage less than 30% then there is no need for refactoring. This verifies that the CaMPaMF is reusable. Furthermore, the CaMPaMF was reused and extended successfully to develop three CaMPaMS, which are a hypertension CaMPaMS, a diabetes CaMPaMS, and an epilepsy CaMPaMS. Additionally, the amount of reuse was calculated for these prototypes by three metrics: the reuse level metric; the reuse frequency metric; and the reuse size and frequency metric. This yielded the average values of 0.88, 0.87, and 0.72 respectively. All these resulted values together reflect high amount of reuse. These values together reflect a high amount of reuse. Finally, the expert review of the framework reusability resulted in 100% of the experts agreeing that the proposed CaMPaMF is reusable.

7.3. Research Contributions

This research contributes to the software engineering body of knowledge, particularly to software design, explicitly software structure and architecture in terms of designing reusable families of programs and frameworks, as shown in Figure 7.23 [92]. The contributions of this research are discussed in the following subsections, starting with those relating to the CaMPaMF as the primary contribution of this research, followed

by those relating to the application framework reusability evaluation approach developed in this study.

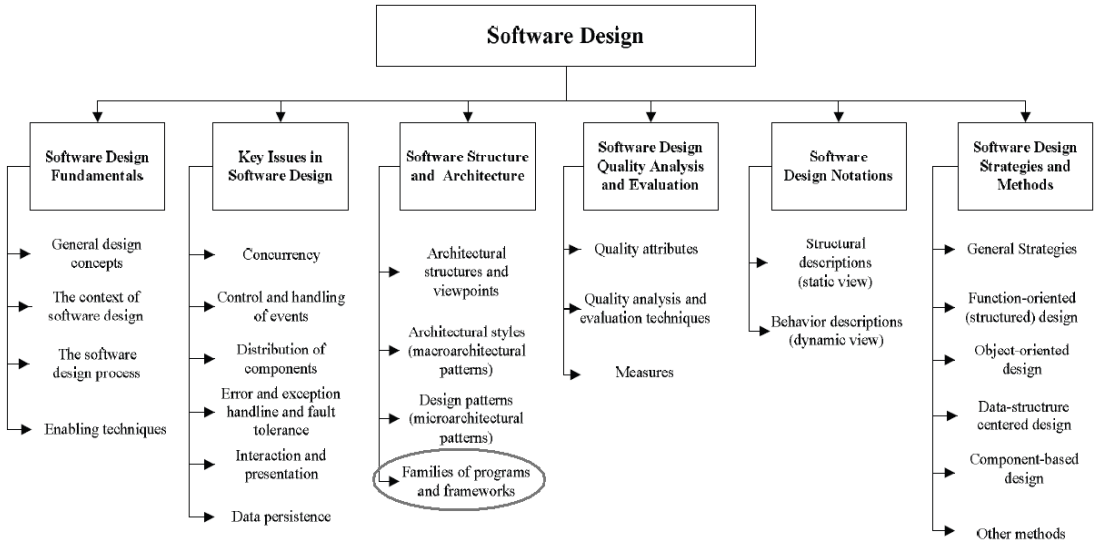

Figure 7.23. Contributions to the software engineering body of knowledge related to software design

Adopted from [92]

7.3.1. CaMPaMF

To the best of the researcher's knowledge, this work is the first attempt to design and evaluate a CaMPaMF based on both multiple reusability aspects and multiple reusability evaluation approaches. Moreover, this work is the first attempt to design an application framework that fully addresses the identified domain requirements of CaMPaMS. The CaMPaMF developed in this research enhances the overall development quality and overcomes the development complexity of CaMPaMS.

Software industries can use the CaMPaMF to reduce the need for consulting domain experts and improve software development productivity by reducing the time and

effort required for building and maintaining CaMPaMS, which will result in the reduction of CaMPaMS development costs. Thus, using the CaMPaMF can reduce the development cost and time required to build CaMPaMS from scratch and hence reduce time to market which is one of the factors affecting the success of software systems.

Developers can use the CaMPaMF to improve the reliability of CaMPaMS by using a well-tested CaMPaMF, thus reducing the number of errors that could arise. Furthermore, developers can reuse and extend the CaMPaMF to develop various CaMPaMS for different diseases. For example, developers can reuse the built-in components of the CaMPaMF such as the context monitoring query evaluator component to reuse the default evaluation strategy of the context monitoring queries. Developers can extend the CaMPaMF, for example, by adding a new class that implements the IDataConverter interface by providing a new data converter component to convert the BT data from Fahrenheit to Celsius. In this research, the CaMPaMF was reused and extended to develop three CaMPaMS, including a hypertension CaMPaMS, a diabetes CaMPaMS, and an epilepsy CaMPaMS.

The CaMPaMF consists of a domain model, architectural model, PIM, PSM, and code development. The following subsections elaborate on these research contributions.

7.3.1.1. Domain Model

The domain model consists of the feature model and the abstract use case model. The feature model is represented by common and variable features as well as interdependency between these features that are captured from a family of applications in a specific domain. These common features are shared among all applications that

are built using the framework, while the variable features represent framework flexibility points that have to be extended to meet application-specific needs. This model consists of a diagram and some additional information. The diagram is also a fundamental element of the feature model that defines a set of features, which are reusable requirements that can be configured to meet the need of a number of applications in a specific domain. The additional information can include a short semantic description about each feature and rationale for selecting each feature.

The abstract use case model presents the system boundary that embodies the system's abstract use cases. In addition, it captures the interactions between the CaMPaMF and its actors, which are the CaMPaMS that benefit from the use of this framework. The resulting model complements the feature model in terms of identifying the domain requirements.

Researchers can use the constructed domain models – the feature model and the abstract use case model – to enhance their understanding of the designed CaMPaMF. This is achieved by understanding the domain requirements, the domain concepts, the rationale behind selecting the domain requirements and concepts, and the interactions between the CaMPaMF and CaMPaMS that are captured in the domain models. In addition, they can use these models as a foundation from which to discover new requirements to extend the proposed domain models. Moreover, developers can use the constructed domain models to identify the frozen spots (common features) and hot spots (variable features) that are required to understand how to reuse and extend the CaMPaMF. In this research, the constructed domain models were used as an input to

the three framework development phases: architectural design, framework design, and implementation.

7.3.1.2. Architectural Model

The architectural model describes the structural organization of the primary components of the proposed CaMPaMF and the relationships between them. It represents a series of structural decisions, such as using the layers architectural style, that aim to satisfy the reusability of the CaMPaMF.

Researchers can extend the constructed architectural model to enhance the architectural design of the CaMPaMF by adding a new component with new functionalities such as by adding a data archiver as a new component that is responsible for storing the collected data for further medical investigations.

Developers can use the constructed architectural model as a solid foundation for efficient development on top of the CaMPaMF by understanding the organization of the CaMPaMF components and relationships among them. For example, the developer can recognize that the context information collector component can be used to collect context information from context data sources through the data source connector component. In this research, the architectural model was used as an input for the framework design and implementation development phases.

7.3.1.3. PIM and PSM

The PIM is a long-lasting reusable model that eliminates the need for redesigning a model when a particular underlying technology is changed and thus reduces

development efforts, time, and cost. It is therefore the best candidate to be used to develop the CaMPaMF. The PSM is generated from the PIM to provide a physical model that is customized to depict the system implementation based on specific technology. C# was used to generate a PSM that can be used to develop mobile applications that can be executed on various platforms such as Android, iOS, and Microsoft Windows Phone.

Researchers can extend the constructed PIM to enhance the design and implementation of the CaMPaMF by adding new interfaces or classes or editing the internal structure of the existing interfaces or classes to add new functionalities such as a new interface to represent environmental location as a new context information type to identify the location of the patient for emergency purposes.

Developers can use the PIM to generate one or more PSM to reflect the continuous changes in the technology, which reduces development efforts, time, and cost. For example, developers can use the PIM to generate a new PSM that is specific to the iOS platform. They can also use the PSM to generate code and thus improve developers' productivity.

7.3.2. Application Framework Reusability Evaluation Approach

This work is the first attempt to evaluate the reusability of a CaMPaMF based on multiple reusability evaluation approaches. By applying multiple reusability evaluation approaches to the CaMPaMF a more extensive idea of its reusability can be obtained compared to applying a single approach. In this research five reusability evaluation approaches were used: evaluating the applicability of design guidelines;

using a reusability model; prototyping; amount of reuse calculation; and using expert review.

First, evaluating the applicability of design guidelines aims to insure a common language for communication between the CaMPaMF authors and the CaMPaMF users, thus confirming the CaMPaMF's reusability. In this approach, the CaMPaMF was analysed based on 211 design guidelines comprised of 61 design rules, 10 globalization rules, 16 interoperability rules, 2 mobility rules, 23 naming rules, 16 performance rules, 3 portability rules, 21 security rules, 20 security transparency rules, and 39 usage rules. The results showed that the CaMPaMF satisfies 98.1% of these guidelines, which confirms the CaMPaMF's reusability.

Second, a reusability model tests the quality factors that affect reusability. In this approach, the framework reusability model introduced in [10] was adopted for evaluating the reusability of the CaMPaMF because it is the only tested model that has addressed the special characteristics of application framework reusability. As shown in Figure 2.1, this model is divided into four levels: factor; design principle; design rule; and metric. On the first level, two factors were identified that affect reusability: flexibility and understandability. On the second level, three design principles were identified that affect flexibility and understandability: modularity, simplicity, and abstraction. On the third level, three groups of design rules were identified: complexity, coupling, and cohesion, which include seven, four and three design rules respectively. These design rules affect the design principles on the second level, as complexity and coupling affect all of the three design principles, while cohesion affects the abstraction design principle only. On the fourth level, three groups of

software metrics were identified: complexity, coupling, and cohesion, which include seven, four and three metrics respectively mapped one to one onto the design rules in the second level. The results showed that 53 out of 1232 measurement values were outlier values, or 4.3%. According to [10], if the outlier value percentage less than 30% then there is no need for refactoring. All the outlier value percentages were less than 30%. Therefore, the CaMPaMF is reusable and there is no need for refactoring.

Third, the prototyping approach aims to provide a proof of concept towards illustrating the CaMPaMF's reusability. In this approach, the CaMPaMF was reused successfully to develop three CaMPaMS, which are a hypertension CaMPaMS, a diabetes CaMPaMS, and an epilepsy CaMPaMS.

Fourth, the amount of reuse calculation aims to measure how much reuse is achieved when developing CaMPaMS on top of the CaMPaMF. The amount of reuse was measured by multiple metrics, each of which represent different points of view that complement each other to provide a complete picture of the effects of reuse. In this research, the amount of reuse was calculated for the three CaMPaMS prototypes by three metrics: the reuse level metric; the reuse frequency metric; and the reuse size and frequency metric. This yielded the average values of 0.88, 0.87, and 0.72 respectively. These values together reflect a high amount of reuse.

Fifth, using expert review aims to confirm the CaMPaMF's reusability in terms of three reusability aspects: design rules; design principles; and factors that can affect software reusability. The results showed that the majority of the experts agreed that the proposed CaMPaMF has acceptable complexity, coupling, cohesiveness,

modularity, simplicity, abstraction. Additionally, all of the experts agreed that the proposed CaMPaMF has acceptable flexibility, understandability, and reusability without modification.

Researchers can use this multiple reusability evaluation approach to evaluate the reusability aspects of their frameworks. This approach also holds significant potential for the discovery of new approaches.

7.4. Research Limitations

The primary limitation of this research was the use of simulated sensors due to the high cost, developmental complexity, and detailed technical specifications of biomedical sensors.

7.5. Future Research

Based on the scope and limitations of this research, there are a many possible directions for future research using the CaMPaMF. This research focused on evaluating the CaMPaMF's reusability. However, future research could evaluate the functionality of the CaMPaMF such as of the context monitoring query evaluation strategy of CaMPaMF. Additionally, this research use simulated sensors and future research could be carried out using real biomedical sensors. The process of analysing wireless sensors' biomedical signals was not covered by this research and future studies could extend the CaMPaMF data converter to be able to analyse these signals.

In the future, the researcher plans to use the CaMPaMF described in this thesis to develop real CaMPaMS for monitoring patients with various diseases that are used by real patients and evaluated by healthcare professionals.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA: Pearson, 2011.

[2] M. Fayad, D. S. Hamu, and D. Brugali, "Enterprise Frameworks Characteristics, Criteria, and Challenges," *Communications of the ACM*, vol. 43, pp. 39-46, October 2000.

[3] J. Sametinger, *Software Engineering with Reusable Components*. Berlin, Germany: Springer, 1997.

[4] H. Mili, A. Mili, S. Yacoub, and E. Addy, *Reuse-Based Software Engineering: Techniques, Organizations, and Controls*: Wiley, 2001.

[5] C. W. Krueger, "Software Reuse," *ACM Computing Surveys*, vol. 24, pp. 131-183, 1992.

[6] J. S. Poulin, "Measuring Software Reusability," in *3rd International Conference on Software Reuse: Advances in Software Reusability*, Rio de Janeiro , Brazil 1994, pp. 126-138

[7] S. Maggo and C. Gupta, "A Machine Learning Based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software," *International Journal of Information Technology and Computer Science*, vol. 6, pp. 1-13, January 2014.

[8] F. Taibi, "Reusability of Open-Source Program Code: A Conceptual Model and Empirical Investigation," *ACM SIGSOFT Software Engineering Notes*, vol. 38, pp. 1-5, July 2013.

[9] K. Cwalina and B. Abrams, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .Net Libraries*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2009.

[10] K. Erni and C. Lewerentz, "Applying Design-Metrics to Object-Oriented Frameworks," in *3rd International Software Metrics Symposium*, Berlin, Germany, 1996, pp. 64-74.

[11] R. N. Ferri, R. N. Pratiwadi, L. M. Rivera, M. Shakir, J. J. Snyder, D. W. Thomas, *et al.*, "Software Reuse Metrics for an Industrial Project," in *4th International Software Metrics Symposium*, Albuquerque, NM, 1997, pp. 165-173.

[12] D. Hristov, O. Hummel, M. Huq, and W. Janjic, "Structuring Software Reusability Metrics for Component-Based Software Development," in *7th International Conference on Software Engineering Advances*, Lisbon, Portugal, 2012, pp. 421-429.

[13] Fazal-e-Amin, A. K. Mahmood, and A. Oxley, "Reusability Assessment of Open Source Components for Software Product Lines," *International Journal on New Computer Architectures and Their Applications*, vol. 1, pp. 519-533, 2011.

[14] S. Sagar, N. W. Nerurkar, and A. Sharma, "A Soft Computing Based Approach to Estimate Reusability of Software," *ACM SIGSOFT Software Engineering Notes*, vol. 35, pp. 1-5, July 2010.

[15] J. Bosch, P. Molin, M. Mattsson, P. Bengtsson, and M. Fayad, "Framework Problems and Experiences," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 55-82.

[16] R. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* vol. 2. Reading, MA: Addison-Wesley, 2000.

[17] H. Hasan, "Information Systems Development as a Research Method," *Australasian Journal of Information Systems*, vol. 11, pp. 4-13, 2003.

[18] W. Frakes and C. Terry, "Software Reuse: Metrics and Models," *ACM Computing Surveys*, vol. 28, pp. 415-435, June 1996.

[19] P. L. Roden, "An Examination of Stability and Reusability in Highly Iterative Software," Doctor of Philosophy, Computer Science, University of Alabama, Huntsville, AL, 2008.

[20] W. Zhang and M. Kim, "What Works and What Does Not: An Analysis of Application Frameworks Technology," *Journal of Business Systems, Governance and Ethics*, vol. 1, pp. 15-26, November 2006.

[21] E. J. Posnak, R. G. Lavender, and H. M. Vin, "An Adaptive Framework for Developing Multimedia Software Components," *Communications of the ACM*, vol. 40, pp. 43 - 47, October 1997.

[22] J. v. Gurp and J. Bosch, "Role-Based Component Engineering," in *Building Reliable Component-Based Software Systems*, I. Crnkovic and M. Larsson, Eds., ed Boston, MA: Artech House, 2002, pp. 135-154.

[23] M. Fayad, D. C. Schmidt, and R. E. Johnson, "Application Frameworks," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 3-28.

[24] M. E. Markiewicz and C. J. P. d. Lucena, "Object Oriented Framework Development," *Crossroads*, vol. 7, pp. 3-9, 2001.

[25] M. Morisio, D. Romano, and I. Stamelos, "Quality, Productivity and Learning in Framework-Based Development: An Exploratory Case Study," *IEEE Transactions on Software Engineering*, vol. 28, pp. 876-888, September 2002.

[26] J. Al-Dallal and P. Sorenson, "Reusing Class-Based Test Cases for Testing Object-Oriented Framework Interface Classes," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, pp. 169-196, May/June 2005.

[27] R. Neumann, S. Günther, and N. Zenker, "Reengineering Deprecated Component Frameworks: A Case Study of the Microsoft Foundation Classes," in *9th International Conference on Business Informatics*, Vienna, Austria, 2009, pp. 737-748.

[28] M. Mattsson, "Comparison of Three Evaluation Methods for Object-Oriented Framework Evolution," in *Software Evolution and Feedback: Theory and Practice*, N. H. Madhavji, J. C. Fernández-Ramil, and D. E. Perry, Eds., ed West Sussex, UK: Wiley, 2006, pp. 281-312.

[29] D. Parsons, A. Rashid, A. Telea, and A. Speck, "An Architectural Pattern for Designing Component-Based Application Frameworks," *Software: Practice and Experience*, vol. 36, pp. 157–190, February 2006.

[30] A. Tevanlinna, J. Taina, and R. Kauppinen, "Product Family Testing: A Survey," *ACM SIGSOFT Software Engineering Notes,* vol. 29, pp. 12-12, March 2004.

[31] U. Kulesza, V. Alves, A. Garcia, C. J. P. d. Lucena, and P. Borba, "Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming," in *Reuse of Off-the-Shelf Components*. vol. 4039, M. Morisio, Ed., ed Berlin, Germany: Springer, 2006, pp. 231-245.

[32] S. P. Lee, S. K. Thin, and H. S. Liu, "Object-Oriented Application Framework on Manufacturing Domain," *Malaysian Journal of Computer Science,* vol. 13, pp. 56-64, June 2000.

[33] D. C. Schmidt, A. Gokhale, and B. Natarajan, "Leveraging Application Frameworks," *Queue,* vol. 2, pp. 66-75, July/August 2004.

[34] T. C. Oliveira, P. Alencar, and D. Cowan, "ReuseTool—An Extensible Tool Support for Object-Oriented Framework Reuse," *Journal of System and Software,* vol. 84, pp. 2234–2252, December 2011.

[35] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications,* 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2008.

[36] A. Jatain and S. Goel, "Comparison of Domain Analysis Methods in Software Reuse," *International Journal of Information Technology and Knowledge Management,* vol. 2, pp. 347-352, July/December 2009.

[37] H. Gomaa, "Reusable Software Requirements and Architectures for Families of Systems," *Journal of System and Software,* vol. 28, pp. 189-202, March 1995.

[38] M. A. Musen, Y. Shahar, and E. H. Shortliffe, "Clinical Decision-Support Systems," in *Biomedical Informatics: Computer Application in Health Care and Biomedicine,* E. H. Shortliffe and J. J. Cimino, Eds., 3rd ed New York, NY: Springer, 2006, pp. 698-734.

[39] P. C. Tang and C. J. McDonald, "Electronic Health Record Systems," in *Biomedical Informatics: Computer Applicatations in Health Care and Biomedicine,* E. H. Shortliffe and J. J. Cimino, Eds., 3rd ed New York, NY: Springer, 2006, pp. 447-473.

[40] R. M. Gardner and M. M. Shabot, "Patient-Monitoring Systems," in *Biomedical Informatics: Computer Applications in Health Care and Biomedicine*, E. H. Shortliffe and J. J. Cimino, Eds., 3rd ed New York, NY: Springer, 2006, pp. 585-625.

[41] H. S. Chae, J. F. Cui, J. Park, J. Park, and W. J. Lee, "An Object-Oriented Framework Approach to Flexible Availability Management for Developing Distributed Applications," *Journal of Information Science and Engineering,* vol. 25, pp. 1021-1040, July 2009.

[42] A. Valerio, G. Succi, and M. Fenaroli, "Domain Analysis and Framework-Based Software Development," *ACM SIGAPP Applied Computing Review,* vol. 5, pp. 4-15, September 1997.

[43] S. Lee, S. Thin, and H. Liu, "EMAF: An Enterprise Manufacturing Application Framework Integrated Environment," in *Pacific Asia Conference on Information Systems*, Seoul, Korea, 2001, pp. 963-977.

[44] J. Zhang, D. Levy, S. Chen, and J. Zic, "mBOSSS+: A Mobile Web Services Framework," in *IEEE Asia-Pacific Services Computing Conference*, Hangzhou, China, 2010, pp. 91-96

[45] T. Broens, A. V. Halteren, M. V. Sinderen, and K. Wac, "Towards an Application Framework for Context-Aware m-Health Applications," *International Journal of Internet Protocol Technology,* vol. 2, pp. 109-116, February 2007.

[46] A. Esposito, L. Tarricone, M. Zappatore, L. Catarinucci, R. Colella, and A. DiBari, "A Framework for Context-Aware Home-Health Monitoring," *International Journal Autonomous and Adaptive Communications Systems,* vol. 3, pp. 75-91, December 2010.

[47] V. Villarreal, J. Fontecha, R. Hervás, and J. Bravo, "Using and Applying MobiPattern to Design MoMo Framework Modules," in *Ambient Assisted Living*. vol. 6693, J. Bravo, R. Hervás, and V. Villarreal, Eds., ed Berlin, Germany: Springer, 2011, pp. 25-32.

[48] A. Fortier, G. Rossi, S. E. Gordillo, and C. Challiol, "Dealing with Variability in Context-Aware Mobile Software," *Journal of Systems and Software*, vol. 83, pp. 915-936, June 2010.

[49] F. Paganelli and D. Giuli, "An Ontology-based System for Context-Aware and Configurable Services to Support Home-Based Continuous Care," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, pp. 324-333, March 2011.

[50] D. Zhang, Z. Yu, and C. Chin, "Context-Aware Infrastructure for Personalized Healthcare," in *Personalised Health Management Systems: The Integration of Innovative Sensing, Textile, Information and Communication Technologies*. vol. 117, C. D. Nugent, P. J. McCullagh, E. T. McAdams, and A. Lymberis, Eds., ed Washington, DC: IOS Press, 2005, pp. 154-163.

[51] S. Kang, J. Lee, H. Jang, Y. Lee, S. Park, and J. Song, "A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 686-702, May 2010.

[52] M. J. Mitchell, C. Meyers, A. A. Wang, and G. Tyson, "ContextProvider: Context Awareness for Medical Monitoring Applications," in *33rd Annual International Conference IEEE Engineering in Medicine and Biology Society*, Boston, MA, 2011, pp. 5244-5247.

[53] World Health Organization (WHO), *Global Health Risks: Mortality and Burden of Disease Attributable to Selected Major Risks*. Geneva, Switzerland: World Health Organization, 2009.

[54] World Health Organization (WHO), *The World Health Report 2008: Primary Health Care: Now More Than Ever*. Geneva, Switzerland: World Health Organization, 2008.

[55] O. Aziz, B. Lo, A. Darzi, and G. Yang, "Introduction to Body Sensor Networks," in *Body Sensor Networks*, G. Yang, Ed., ed New York, NY: Springer, 2006, pp. 1-39.

[56] M. C. Houston, *Handbook of Hypertension*. Chichester, UK: Wiley-Blackwell, 2009.

[57] N. M. Kaplan and R. G. Victor, *Kaplan's Clinical Hypertension*, 10th ed. Philadelphia, PA: Lippincott Williams & Wilkins, 2009.

[58] R. J. McManus, E. P. Bray, J. Mant, R. Holder, S. Greenfield, S. Bryan, *et al.*, "Protocol for a Randomised Controlled Trial of Telemonitoring and Self-Management in the Control of Hypertension: Telemonitoring and Self-Management in Hypertension," *BMC Cardiovascular Disorders*, vol. 9, pp. 1-21, February 2009.

[59] J. E. Bardram and H. B. Christensen, "Pervasive Computing Support for Hospitals: An overview of the Activity-Based Computing Project," *IEEE Pervasive Computing*, vol. 6, pp. 44-51, January-March 2007.

[60] S. Sneha and U. Varshney, "Enabling Ubiquitous Patient Monitoring: Model, Decision Protocols, Opportunities And Challenges," *Decision Support Systems*, vol. 46, pp. 606-619, February 2009.

[61] V. Villarreal, G. Urzaiz, R. Hervas, and J. Bravo, "Monitoring Architecture to Collect Measurement Data and Medical Patient Control through Mobile Devices," in *5th International Symposium on Ubiquitous Computing and Ambient Intelligence*, Riviera Maya, Mexico, 2011.

[62] Y. Ren, R. W. N. Pazzi, and A. Boukerche, "Monitoring Patients Via a Secure and Mobile Healthcare System," *IEEE Wireless Communications*, vol. 17, pp. 59-65, February 2010.

[63] C. Liu, Q. Zhu, K. A. Holroyd, and E. K. Seng, "Status and Trends of Mobile-Health Applications for iOS Devices: A Developer's Perspective," *Journal of System and Software*, vol. 84, pp. 2022-2033, November 2011.

[64] Z. Lv, F. Xia, G. Wu, L. Yao, and Z. Chen, "iCare: A Mobile Health Monitoring System for the Elderly," in *IEEE/ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing*, Hangzhou, China, 2010, pp. 699-705.

[65] D. Apiletti, E. Baralis, G. Bruno, and T. Cerquitelli, "Real-Time Analysis of Physiological Data to Support Medical Applications," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, pp. 313-321, May 2009.

[66] Y. M. Huang, M. Y. Hsieh, H. C. Chao, S. H. Hung, and J. H. Park, "Pervasive, Secure Access to A Hierarchical Sensor-Based Healthcare Monitoring Architecture in Wireless Heterogeneous Networks," *IEEE Journal on Selected Areas in Communications*, vol. 27, pp. 400-411, May 2009.

[67] V. G. Koutkias, I. Chouvarda, A. Triantafyllidis, A. Malousi, G. D. Giaglis, and N. Maglaveras, "A Personalized Framework for Medication Treatment Management in Chronic Care," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, pp. 464-472, March 2010.

[68] A. Copetti, O. Loques, J. C. B. Leite, T. P. C. Barbosa, and A. C. L. d. Nobrega, "Intelligent Context-Aware Monitoring of Hypertensive Patients," in *3rd International Conference on Pervasive Computing Technologies for Healthcare*, London, UK, 2009, pp. 1-6.

[69] I. Mohomed, A. Misra, M. Ebling, and W. Jerome, "HARMONI: Context-Aware Filtering of Sensor Data for Continuous Remote Health Monitoring," in *6th Annual IEEE International Conference on Pervasive Computing and Communications*, Hong Kong, China, 2008, pp. 248-251.

[70] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "The Requirements to Enhance the Design of Context-Aware Mobile Patient Monitoring Systems Using Wireless Sensors," in *Context-Aware Systems and Applications*. vol. 109, P. C. Vinh, N. M. Hung, N. T. Tung, and J. Suzuki, Eds., ed Berlin, Germany: Springer, 2013, pp. 62-71.

[71] F. C. Delicato, I. L. A. Santos, P. F. Pires, A. L. S. Oliveira, T. Batista, and L. Pìrmez, "Using Aspects and Dynamic Composition to Provide Context-Aware Adaptation for Mobile Applications," in *ACM Symposium on Applied Computing*, Honolulu, HI, 2009, pp. 456-460.

[72] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, *et al.*, "A Survey of Context Modelling and Reasoning Techniques," *Pervasive and Mobile Computing,* vol. 6, pp. 161-180, March 2010.

[73] J. R. Hoyos, J. GarcíaMolina, and J. A. Botía, "MLContext: A Context-Modeling Language for Context-Aware Systems," *Electronic Communications of the EASST,* vol. 28, pp. 1-14, 2010.

[74] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Context-Aware Mobile Patient Monitoring Frameworks: A Systematic Review and Research Agenda," *Journal of Software,* vol. 8, pp. 1604-1612, July 2013.

[75] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Software Frameworks in Biomedical Informatics: A Systematic Review and Research Agenda," *Journal of Software,* 2013.

[76] N. F. Ahmad, D. B. Hoang, and M. H. Phung, "Robust Preprocessing for Health Care Monitoring Framework," in *11th IEEE International Conference on e-Health Networking, Applications and Services*, Sydney, Australia, 2009, pp. 169-174.

[77] T. Laakko, J. Leppänen, J. Lähteenmäki, and A. Nummiaho, "Mobile Health and Wellness Application Framework," *Methods of Information in Medicine,* vol. 47, pp. 217-222, 2008.

[78] J. E. Bardram and T. R. Hansen, "The AWARE Architecture: Supporting Context-Mediated Social Awareness in Mobile Cooperation," in *16th ACM Conference on Computer Supported Cooperative Work*, Chicago, IL, 2004, pp. 192-201.

[79] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Context-Aware Mobile Patient Monitoring Framework Development: An Architectural Design," *Advanced Science Letter,* vol. 20, pp. 293-297, January 2014.

[80] G. Arango, "Domain Analysis Methods," in *Software Reusability*, W. Schäfer, R. Prieto-Díaz, and M. Matsumoto, Eds., ed New York, NY: Ellis Horwood, 1994, pp. 17-49.

[81] E. S. d. Almeida, J. C. C. P. Mascena, A. P. C. Cavalcanti, A. Alvaro, V. C. Garcia, S. R. d. L. Meira, *et al.*, "The Domain Analysis Concept Revisited: A Practical Approach," in *Reuse of Off-the-Shelf Components*. vol. 4039, M. Morisio, Ed., ed Berlin, Germany: Springer, 2006, pp. 43-57.

[82] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, *et al.*, "On Extracting Feature Models From Product Descriptions," in *6th International Workshop on Variability Modeling of Software-Intensive Systems*, Leipzig, Germany, 2012, pp. 45-54.

[83] T. Jeon, S. Lee, and H. Seung, "Increasing the Testability of Object-Oriented Frameworks with Built-in Tests," in *Advanced Internet Services and Applications*. vol. 2402, W. Chang, Ed., ed Berlin, Germany: Springer, 2002, pp. 873-881.

[84] W. Pree, *Design Patterns for Object-Oriented Software Development*. Wokingham, UK: Addison-Wesley, 1995.

[85] H. A. Schmid, "Framework Design by Systematic Generalization," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 353-378.

[86] G. Miller, J. McGregor, and M. Major, "Capturing Framework Requirements," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 309-323.

[87] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Context-Aware Mobile Patient Monitoring Framework Development: A Detailed Design," *IERI Procedia*, vol. 4, pp. 155–167, December 2013.

[88] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Feature Model to Design Application Framework for Context-aware Mobile Patient Monitoring Systems," in *2nd IEEE International EMBS Conference Biomedical Engineering and Sciences*, Langkawi, Malaysia, 2012, pp. 72-77.

[89] L. Chung and J. C. S. d. P. Leite, "On non-Functional Requirements in Software Engineering," in *Conceptual Modeling: Foundations and Applications*. vol. 5600, A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. Yu, Eds., 1st ed New York, NY: Springer, 2009, pp. 363-379.

[90] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA: Addison Wesley, 2003.

[91] W. Raghupathi and A. Umar, "Exploring a Model-Driven Architecture (MDA) Approach to Health Care Information Systems Development," *International Journal of Medical Informatics*, vol. 77, pp. 305-314, May 2008.

[92] P. Bourque, F. Robert, J. M. Lavoie, A. Lee, S. Trudel, and T. C. Lethbridge, "Guide to the Software Engineering Body of Knowledge (SWEBOK) and the Software Engineering Education Knowledge (SEEK) - A Preliminary Mapping," in *10th International Workshop on Software Technology and Engineering Practice*, Québec, Canada, 2004, pp. 8-23.

[93] G. Cardino, F. Baruchelli, and A. Valerio, "The Evaluation of Framework Reusability," *ACM SIGAPP Applied Computing Review*, vol. 5, pp. 21-27, September 1997.

[94] G. Sindre, R. Conradi, and E.-A. Karlsson, "The REBOOT Approach to Software Reuse," *Journal of Systems and Software*, vol. 30, pp. 201-212, September 1995.

[95] G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Software Components," *Computer*, vol. 24, pp. 61-70, February 1991.

[96] ISO, IEC, and IEEE, *Systems and Software Engineering - Vocabulary*. Piscataway, NJ: IEEE Computer Society, 2010.

[97] E. S. d. Almeida, A. Alvaro, V. C. Garcia, J. C. C. P. Mascena, V. A. d. A. Burégio, L. M. d. Nascimento, *et al.*, *CRUISE: Component Reuse in Software Engineering*. Recife, Brazil: CESAR, 2007.

[98] L. Chou, J. Sun, and M. Chen, "A New Application Framework for Intelligent Surveillance Sensor Networks," in *3rd International Conference on*

International Information Hiding and Multimedia Signal Processing, Kaohsiung, Taiwan, 2007, pp. 589-591.

[99] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[100] D. C. Schmidt and F. Buschmann, "Patterns, Frameworks, and Middleware: Their Synergistic Relationships," in *25th International Conference on Software Engineering*, Portland, OR, 2003, pp. 694-704.

[101] G. Polancic, R. V. Horvat, and I. Rozman, "Improving Object-Oriented Frameworks by Considering the Characteristics of Constituent Elements," *Journal of Information Science and Engineering,* vol. 25, pp. 1067-1085, July 2009.

[102] I. Crnkovic, B. Hnich, T. Jonsson, and Z. Kiziltan, "Basic Concepts in CBSE," in *Building Reliable Component-Based Software Systems*, I. Crnkovic and M. Larsson, Eds., ed Norwood, MA: Artech House, 2002, pp. 3-22.

[103] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, *et al.*, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2011.

[104] R. E. Johnson, "Components, Frameworks, Patterns," *ACM SIGSOFT Software Engineering Notes*, vol. 22, pp. 10-17, May 1997.

[105] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. New York, NY: Addison-Wesley, 2011.

[106] D. Ranjan and A. K. Tripathi, "Variability-Based Models for Testability Analysis of Frameworks," *Journal of Software Engineering and Applications*, vol. 3, pp. 455-459, May 2010.

[107] G. Larsen, "Designing Component-Based Frameworks Using Patterns in the UML," *Communications of the ACM*, vol. 42, pp. 38-45, October 1999.

[108] G. Froehlich, H. J. Hoover, L. Liu, and P. Sorenson, "Hooking into Object-Oriented Application Frameworks," in *19th International Conference on Software Engineering*, Boston, MA, 1997, pp. 491-501.

[109] A. Sutcliffe and G. Papamargaritis, "Applying the Domain Theory to Design for Reuse," *BT Technology Journal*, vol. 22, pp. 104-115, April 2004.

[110] J. A. McCall, P. K. Richards, and G. F. Walters, *Factors in Software Quality* vol. 1. Rome, NY: US Rome Air Development Center, 1977.

[111] J.-M. Morel and J. Faget, "The REBOOT Environment," in *2nd International Workshop on Software Reusability*, 1993, pp. 80-88.

[112] A. Rosel and K. Erni, "Experiences with the Semantic Graphics Framework," in *Implementing Application Frameworks: Object-Oriented Frameworks at Work*, M. E. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 629-658.

[113] R. P. E. Esilva and E. C. Freiberger, "Metrics to Evaluate the Use of Object Oriented Frameworks," *Computer Journal*, vol. 52, pp. 288-304, 2009.

[114] H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components," in *9th International Software Metrics Symposium*, Sydney, Australia, 2003, pp. 211-223.

[115] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin, Germany: Springer, 2006.

[116] D. Soni, R. Shrivastava, and M. Kumar, "A Framework for Validation of Object-Oriented Design Metrics," *International Journal of Computer Science and Information Security*, vol. 6, pp. 46-52, December 2009.

[117] T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. Hoboken, NJ: Wiley, 2006.

[118] Wikipedia. (2014, April). *Eclipse*. Available: http://en.wikipedia.org/wiki/Eclipse_(software)

[119] Wikipedia. (2014, April). *Java Platform*. Available: http://en.wikipedia.org/wiki/Java_platform

[120] Wikipedia. (2014, April). *.NET Framework*. Available: http://en.wikipedia.org/wiki/.NET_Framework

[121] S. Lopes, A. Tavares, J. Monteiro, and C. Silva, "Instantiation of a Classification System Framework that Facilitates Reuse," *Journal of Software*, vol. 2, pp. 57-69, October 2007.

[122] J. v. Gurp and J. Bosch, "Design, Implementation and Evolution of Object Oriented Frameworks: Concepts and Guidelines," *Software: Practice and Experience*, vol. 31, pp. 277-300, 2001.

[123] S. Srinivasan, "Design Patterns in Object-Oriented Frameworks," *Computer*, vol. 32, pp. 24-32, January 1999.

[124] T. C. Shan and W. W. Hua, "Taxonomy of Java Web Application Frameworks " in *IEEE International Conference on e-Business Engineering*, Shanghai, China, 2006, pp. 378-385.

[125] W. Pree, "Essential Framework Design Patterns," *Object Magazine*, vol. 7, pp. 34-37, 1997.

[126] H. A. Schmid, "Systematic Framework Design by Generalization," *Communications of the ACM*, vol. 40, pp. 48-51, October 1997.

[127] H. Mili, M. Fayad, D. Brugali, D. Hamu, and D. Dori, "Enterprise Frameworks: Issues and Research Directions," *Software: Practice and Experience*, vol. 32, pp. 801-831, 2002.

[128] S. Demeyer, T. D. Meijler, O. Nierstrasz, and P. Steyaert, "Design Guidelines for 'Tailorable' Frameworks," *Communications of the ACM*, vol. 40, pp. 60-64, October 1997.

[129] J. M. Neighbors, "Draco: A Method for Engineering Reusable Software Systems," in *Software Reusability*. vol. 1, T. Biggerstaff and A. Perlis, Eds., ed New York, NY: ACM Press, 1989, pp. 295-319.

[130] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison Wesley, 2000.

[131] A. Sturm, D. Dori, and O. Shehory, "The Application-Based Domain Analysis Approach and its Object-Process Methodology Implementation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 18, pp. 1115-1142, December 2008.

[132] N. S. Gill and P. Tomar, "Modified Development Process of Component-Based Software Engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 35, pp. 1-6, 2010.

[133] A. v. Deursen and P. Klint, "Domain-Specific Language Design Requires Feature Descriptions," *Journal of Computing and Information Technology*, vol. 10, pp. 1-17, 2002.

[134] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. Boston, MA: McGraw Hill, 2010.

[135] IEEE, "IEEE Standard for Information Technology - System and Software Life Cycle Processes - Reuse Processes," ed. New York, NY: IEEE Computer Society, 2010, pp. 1-51.

[136] G. Succi, A. Valerio, T. Vernazza, M. Fenaroli, and P. Predonzani, "Framework Extraction with Domain Analysis," *ACM Computing Surveys*, vol. 32, p. 12, March 2000.

[137] R. Prieto-Díaz, "Historical Overview," in *Software Reusability*, W. Schäfer, R. Prieto-Díaz, and M. Matsumoto, Eds., ed New York, NY: Ellis Horwood, 1994, p. 160.

[138] M. Aksit, F. Marcelloni, and B. Tekinerdogan, "Developing Object-Oriented Framworks Using Domain Models," *ACM Computing Surveys*, vol. 32, March 2000.

[139] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer, *Software & Systems Requirements Engineering: In Practice*. New York, NY: McGraw-Hill, 2009.

[140] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing Cardinality-Based Feature Models and their Specialization," *Software Process: Improvement and Practice*, vol. 10, pp. 7-29, March 2005.

[141] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Pittsburgh, PA, Technical Report CMU/SEI90TR021, November 1990.

[142] C. Kästner, T. Thüm, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, *et al.*, "FeatureIDE: A Tool Framework for Feature-Oriented Software Development," in *31st International Conference Software Engineering*, Vancouver, BC, 2009, pp. 611–614.

[143] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J. Rigault, "Modeling Context and Dynamic Adaptations with Feature Models," in *4th International Workshop Models*, Denver, colo, 2009, pp. 89-98.

[144] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2013.

[145] L. Bass, R. Nord, W. Wood, D. Zubrow, and I. Ozkaya, "Analysis of Architecture Evaluation Data," *Journal of Systems and Software*, vol. 81, pp. 1443-1455, 2008.

[146] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns* vol. 1st. Chichester, UK: Wiley, 1996.

[147] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing* vol. 4th. Chichester, UK: Wiley, 2007.

[148] M. Kircher and P. Jain, *Pattern-Oriented Software Architecture: Patterns for Resource Management* vol. 3rd. Chichester, UK: Wiley, 2004.

[149] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects* vol. 2nd. Chichester, UK: Wiley, 2000.

[150] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall, 1996.

[151] J. Kim, S. Park, and V. Sugumaran, "DRAMA: A Framework for Domain Requirements Analysis and Modeling Architectures in Software Product Lines," *Journal of Systems and Software*, vol. 81, pp. 37-55, January 2008.

[152] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages* vol. 5th. Chichester, UK: Wiley, 2007.

[153] D. Garlan and M. Shaw, "An Introduction to Software Architecture," in *Advances in Software Engineering and Knowledge Engineering*, V. Ambriola and G. Tortora, Eds., ed River Edge, NJ: World Scientific, 1993, pp. 1-40.

[154] I. Gorton, *Essential Software Architecture* Berlin, Germany: Springer 2006.

[155] OMG. (2014, March). *MDA - The Architecture of Choice for a Changing World*. Available: http://www.omg.org/mda/

[156] B. Selic, "The Pragmatics of Model-Driven Development " *IEEE Software*, vol. 20, pp. 19 - 25 2003.

[157] OMG, "Mda Guide Version 1.0.1," omg/2003-06-01, June 2003.

[158] D. Gašević, D. Djurić, and V. Devedžić, *Model Driven Engineering and Ontology Development*, 2nd ed. New York, NY: Springer, 2009.

[159] S. Miller, K. Scott, A. Uhl, and D. Weise, *MDA Distilled: Principles of Model-Driven Architecture*. Boston, MA: Addison-Wesley, 2004.

[160] X. Chen, *Developing Application Frameworks in .NET*. Berkeley, CA: Apress, 2004.

[161] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ: Prentice Hall, 2003.

[162] B. Hailpern and P. Tarr, "Model-Driven Development: The Good, the Bad, and the Ugly," *IBM Systems Journal*, vol. 45, pp. 451-461, July 2006.

[163] V. M. Jones, A. v. Halteren, D. Konstantas, I. Widya, and R. Bults, "An Application of Augmented MDA for the Extended Healthcare Enterprise," *International Journal of Business Process Integration and Management*, vol. 2, pp. 215–229, 2007.

[164] J. Al-Dallal, "Estimating the Coverage of the Framework Application Reusable Cluster-Based Test Cases," *Information and Software Technology*, vol. 50, pp. 595-604, May 2008.

[165] R. E. Johnson and V. F. Russo, "Reusing Object-Oriented Designs," University of Illinois, Urbana, IL, Technical Report UIUCDS91-1696, May 1991.

[166] R. Baskerville and A. T. Wood-Harper, "Diversity in Information Systems Action Research Methods," *European Journal of Information Systems*, vol. 7, pp. 90-107, June 1998.

[167] A. Mouttham, L. Peyton, B. Eze, and A. E. Saddik, "Event-Driven Data Integration for Personal Health Monitoring," *Journal of Emerging Technologies in Web Intelligence*, vol. 1, pp. 110-118, November 2009.

[168] A. Rocha, A. Martins, J. C. F. Junior, M. N. K. Boulos, M. E. Vicente, R. Feld, *et al.*, "Innovations in Health Care Services: The CAALYX System," *International Journal of Medical Informatics*, vol. 82, pp. 307–320, November 2013.

[169] V. Villarreal, J. Fontecha, R. Hervas, and J. Bravo, "An Architecture to Development a Ambient Assisted Living Applications: A Study Case in Diabetes," in *5th International Symposium Ubiquitous Computing and Ambient Intelligence*, Riviera Maya, Mexico, 2011.

[170] E. Coiera, *Guide to Health Informatics*, 2nd ed. London, UK: Arnold, 2003.

[171] E. H. Shortliffe and M. S. Blois, "The Computer Meets Medicine and Biology: Emergence of a Discipline," in *Biomedical Informatics: Computer Application in Health Care and Biomedicine*, E. H. Shortliffe and J. J. Cimino, Eds., 3rd ed New York, NY: Springer, 2006, pp. 3-45.

[172] E. H. Shortliffe, "The Science of Biomedical Computing," *Informatics for Health and Social Care*, vol. 9, pp. 185-193, July/December 1984.

[173] J. D. Myers, "Medical Education in the Information Age," in *Symposium on Medical Informatics*, Washington, DC, 1986.

[174] D. A. B. Lindberg, "NLM Long Range Plan," US National Library of Medicine, Bethesda, MD, Panel Report Z 675.M4 N2782L 1986, January 2002.

[175] H. R. Warner, "Medical Informatics: A Real Discipline?," *Journal of the American Medical Informatics Association,* vol. 2, pp. 207-214, July/August 1995.

[176] R. A. Greenes and E. H. Shortliffe, "Medical Informatics: An Emerging Academic Discipline and Institutional Priority," *JAMA,* vol. 263, pp. 1114-1120, February 1990.

[177] M. F. Collen, "The Origins of Informatics," *Journal of the American Medical lnformatics Association,* vol. 1, pp. 91-107, March/April 1994.

[178] D. Mascareñas, E. Flynn, C. Farrar, G. Park, and M. Todd, "A Mobile Host Approach for Wireless Powering and Interrogation of Structural Health Monitoring Sensor Networks," *IEEE Sensors Journal,* vol. 9, pp. 1719-1726, December 2009.

[179] G. Wiederhold and E. H. Shortliffe, "System Design and Engineering in Health Care," in *Biomedical Informatics: Computer Applications in Health Care and Biomedicine*, E. H. Shortliffe and J. J. Cimino, Eds., 3rd ed New York, NY: Springer, 2006, pp. 233-262.

[180] E. S. Nahm, "Innovations in Patient-Monitoring Systems," *American Nurse Today,* vol. 4, pp. 29-30, November/December 2009.

[181] T. Bratan and M. Clarke, "Optimum Design of Remote Patient Monitoring Systems," in *28th IEEE EMBS Annual International Conference*, New York, NY, 2006, pp. 6465-6468.

[182] B. Lin, B. Lin, N. Chou, F. Chong, and S. Chen, "RTWPMS: A Real-Time Wireless Physiological Monitoring System," *IEEE Transactions on Information Technology in Biomedicine,* vol. 10, pp. 647-656, October 2006.

[183] L. D. Hudson, "Monitoring of Critically Ill Patients: Conference Summary," *Respir Care,* vol. 30, pp. 628-636, 1985.

[184] E. Jovanov, A. O. Lords, D. Raskovic, P. G. Cox, R. Adhami, and F. Andrasik, "Stress Monitoring Using a Distributed Wireless Intelligent Sensor System," *IEEE Engineering in Medicine and Biology Magazine,* vol. 22, pp. 49-55, May/June 2003.

[185] R. S. H. Istepanian, E. Jovanov, and Y. T. Zhang, "Guest Editorial Introduction to the Special Section on M-Health: Beyond Seamless Mobility and Global Wireless Health-Care Connectivity," *IEEE Transactions on Information Technology in Biomedicine,* vol. 8, pp. 405-414, December 2004.

[186] W. M. Omar and A. Taleb-Bendiab, "E-Health Support Services Based on Service-Oriented Architecture," *IT Professional,* vol. 8, pp. 35-41, 2006.

[187] M. Trudel, J. A. Cafazzo, M. Hamill, W. Igharas, K. Tallevi, P. Picton, *et al.*, "A Mobile Phone Based Remote Patient Monitoring System for Chronic Disease Management," in *Building Sustainable Health Systems*. vol. 129, K. A. Kuhn, J. R. Warren, and T.-Y. Leong, Eds., ed Washington, DC: IOS Press, 2007, pp. 167-171.

[188] V. Jones, A. v. Halteren, N. Dokovsky, G. Koprinkov, J. Peuscher, R. Bults, *et al.*, "Mobihealth: Mobile Services for Health Professionals," in *M-Health: Emerging Mobile Health Systems,* R. S. H. Istepanian, S. Laxminarayan, and C. S. Pattichis, Eds., ed New York, NY: Springer, 2006, pp. 237-246.

[189] A. B. Waluyo, W. Yeoh, I. Pek, Y. Yong, and X. Chen, "MobiSense: Mobile Body Sensor Network for Ambulatory Monitoring," *ACM Transactions on Embedded Computing Systems,* vol. 10, August 2010.

[190] J. Shen, D. Shih, H. Chiang, and S. Lin, "A Mobile Physiological Monitoring System for Patient Transport," *Journal of High Speed Networks,* vol. 16, pp. 51-68, January 2007.

[191] H. Mei, B. v. Beijnum, P. Pawar, I. Widya, and H. Hermens, "A*-Based Task Assignment Algorithm for Context-Aware Mobile Patient Monitoring Systems," in *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications,* Beijing, China, 2009, pp. 245-254.

[192] E. J. Ko, H. J. Lee, and J. W. Lee, "Ontology-Based Context Modeling and Reasoning for U-HealthCare," *IEICE Transactions on Information and Systems*, vol. E90-D, pp. 1262-1270, August 2007.

[193] A. V. Halteren, R. Bults, K. Wac, D. Konstantas, I. Widya, N. Dokovsky, *et al.*, "Mobile Patient Monitoring: The MobiHealth System," *Journal on Information Technology in Healthcare*, vol. 2, pp. 365–373, 2004.

[194] I. Mohomed, A. Misra, M. Ebling, and W. Jerome, "Context-Aware and Personalized Event Filtering for Low-Overhead Continuous Remote Health Monitoring," in *9th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Newport Beach, CA, 2008, pp. 1-8.

[195] J. Sriram, M. Shin, D. Kotz, A. Rajan, M. Sastry, and M. Yarvis, "Challenges in Data Quality Assurance in Pervasive Health Monitoring Systems," in *Future of Trust in Computing*, D. Gawrock, H. Reimer, A. Sadeghi, and C. Vishik, Eds., ed Berlin, Germany: Vieweg+Teubner, 2009, pp. 129-142.

[196] ZigBee Alliance. (2008, July). *ZigBee Specification*. Available: http://www.zigbee.org

[197] J. Espina, T. Falck, and O. Mülhens, "Network Topologies, Communication Protocols, and Standards," in *Body Sensor Networks*, G. Yang, Ed., ed New York, NY: Springer, 2006, pp. 145-182.

[198] J. A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Milter, and B. Heile, "IEEE 802. 75.4: A Developing Standard for Low-Power Low-Cost Wireless Personal Area Networks," in *IEEE Network* vol. 15, ed, 2001, pp. 12-19.

[199] C. Hofmann, C. Weigand, and J. Bernhard, "Wireless Medical Sensor Network With Zigbee," in *5th International Conference on Electronics, Hardware, Wireless and Optical Communications*, Madrid, Spain, 2006, pp. 12-15.

[200] E. Mattila, I. Korhonen, and N. Saranummi, "Mobile and Personal Health and Wellness Management Systems," in *Pervasive Computing in Healthcare*, J. E. Bardram, A. Mihailidis, and D. Wan, Eds., ed Boca Raton, FL: CRC Press, 2007, pp. 105-134.

[201] J. Pan, S. Li, and Z. Wu, "Towards a Novel In-Community Healthcare Monitoring System Over Wireless Sensor Networks," in *3rd International Conference on Internet Computing in Science and Engineering*, Harbin, China, 2008, pp. 160-165.

[202] N. Oliver, F. FloresMangas, and R. d. Oliveira, "Towards Wearable Physiological Monitoring on a Mobile Phone," in *Mobile Health Solutions for Biomedical Application*, P. Olla and J. Tan, Eds., ed Hershey, PA: IGI Global, 2009, pp. 208-243.

[203] E. Jovanov, A. Milenkovic, C. Otto, and P. C. d. Groen, "A Wireless Body Area Network of Intelligent Motion Sensors for Computer Assisted Physical Rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 2, March 2005.

[204] M. T. Arredondo, S. Guille´n, I. Peinado, and G. Fico, "Scenarios for the Interaction Between Personal Health Systems and Chronic Patients," in *Wearable Monitoring Systems*, A. Bonfiglio and D. D. Rossi, Eds., ed New York, NY: Springer, 2011, pp. 253-276.

[205] F. Meneses and A. Moreira, "Technology Enablers for Context-Aware Healthcare Applications," in *Mobile Health Solutions for Biomedical Applications*, P. Olla and J. Tan, Eds., ed Hershey, PA: Information Science Reference, 2009, pp. 260-269.

[206] I. Martínez, J. Escayola, M. Martínez-Espronceda, L. Serrano, J. D. Trigo, S. Led, *et al.*, "Standard-Based Middleware Platform for Medical Sensor Networks and u-Health," in *17th International Conference on Computer Communications and Networks*, US Virgin Islands, VI, 2008, pp. 714-719.

[207] U. Varshney, "A Framework for Supporting Emergency Messages in Wireless Patient Monitoring," *Decision Support Systems*, vol. 45, pp. 981-996, November 2008.

[208] M. Galarraga, L. Serrano, I. Martínez, and P. d. Toledo, "Review of the ISO/IEEE X73–POCMDC Standard for Medical Device Interoperability," in *Medical and Care Compunetics 3*. vol. 121, L. Bos, L. Roa, K. Yogesan, B.

O'Connell, A. Marsh, and B. Blobel, Eds., ed Washington, DC: IOS Press, 2006, pp. 242 - 256.

[209] A. K. Dey, "Understanding and Using Context," *Personal Ubiquitous Computing,* vol. 5, pp. 4-7, 2001.

[210] B. N. Schilit and M. M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network,* vol. 8, pp. 22 - 32, September/October 1994.

[211] T. Gu, H. K. Pung, and D. Zhang, "Toward an OSGi-Based Infrastructure for Context-Aware Applications," *IEEE Pervasive Computing,* vol. 3, pp. 66-74, October/December 2004.

[212] R. Hervás, J. Bravo, and J. Fontecha, "A Context Model Based on Ontological Languages: A Proposal for Information Visualization," *Journal of Universal Computer Science,* vol. 16, pp. 1539-1555, August 2010.

[213] D. Zhang, B. Adipat, and Y. Mowafi, "User-Centered Context-Aware Mobile Applications—The Next Generation of Personal Mobile Computing," *Communications of the Association for Information Systems,* vol. 24, pp. 27-46, January 2009.

[214] D. Preuveneers, K. Victor, Y. Vanrompay, P. Rigole, M. K. Pinheiro, and Y. Berbers, "Context-Aware Adaptation in an Ecology of Applications," in *Context-aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Application*, D. Stojanovic, Ed., ed Hershey, PA: Inform. Science Reference, 2009, pp. 1-25.

[215] J. P. A. Almeida, M. Iacob, H. Jonkers, and D. Quartel, "Model-Driven Development of Context-Aware Services," in *Distributed Applications and Interoperable Systems*, F. Eliassen and A. Montresor, Eds., ed Birlin, Germany: Springer, 2006, pp. 213-227.

[216] F. Paganelli and D. Giuli, "An Ontology-Based Context Model for Home Health Monitoring and Alerting in Chronic Patient Care Networks," in *21st International Conference on Advanced Information Networking and Applications Workshops*, Ontario, Canada 2007, pp. 838-845.

[217] S. W. Loke, "Context-Aware Artifacts: Two Development Approaches," *IEEE Pervasive Computing*, vol. 5, pp. 48-53, April/June 2006.

[218] N. Kara and O. A. Dragoi, "Reasoning with Contextual Data in Telehealth Applications," in *3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, White Plains, NY, 2007, pp. 69-76.

[219] M. J. v. Sinderen, A. T. v. Halteren, M. Wegdam, H. B. Meeuwissen, and E. H. Eertink, "Supporting Context-Aware Mobile Applications: An Infrastructure Approach," *IEEE Communications Magazine*, vol. 44, pp. 96-104, September 2006.

[220] R. Ashford, P. Moore, B. Hu, M. Jackson, and J. Wan, "Translational Research and Context in Health Monitoring Systems," in *4th IEEE International Conference on Complex, Intelligent and Software Intensive Systems*, Kraków, Poland, 2010, pp. 81-86.

[221] P. Vajirkar, S. Singh, and Y. Lee, "Context-Aware Data Mining Framework for Wireless Medical Application," in *Database and Expert Systems Applications*. vol. 2736, V. Marík, W. Retschitzegger, and O. Štepánková, Eds., ed Berlin, Germany: Springer, 2003, pp. 381-391.

[222] N. BriconSouf and C. R. Newman, "Context Awareness in Health Care: A Review," *International Journal of Medical Informatics*, vol. 76, pp. 2-12, January 2007.

[223] A. K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16, pp. 97-166, December 2001.

[224] N. Roy, G. Pallapa, and S. K. Das, "An Ontology-Driven Ambiguous Contexts Mediation Framework for Smart Healthcare Applications," in *International Conference on PErvasive Technologies Related to Assistive Environments*, Athens, Greece, 2008, pp. 1-8.

[225] D. S. Hamu, "Enterprise Frameworks," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 83-86.

[226] A. B. Waluyo, I. Pek, X. Chen, and W. Yeoh, "Design and Evaluation of Lightweight Middleware for Personal Wireless Body Area Network," *Personal and Ubiquitous Computing,* vol. 13, pp. 509-525, April 2009.

[227] N. Roy, T. Gu, and S. K. Das, "Supporting Pervasive Computing Applications with Active Context Fusion and Semantic Context Delivery," *Pervasive and Mobile Computing,* vol. 6, pp. 21-42, February 2010.

[228] M. D. Rodríguez and J. Favela, "Assessing the SALSA Architecture for Developing Agent-Based Ambient Computing Applications," *Science of Computer Programming,* vol. 77, pp. 46-65, January 2012.

[229] J. Lewandowski, H. E. Arochena, R. N. G. Naguib, and K. Chao, "A Portable Framework Design to Support User Context-Aware Augmented Reality Applications," in *3rd International Conference on Games and Virtual Worlds for Serious Applications*, Athens, Greece, 2011, pp. 144-147.

[230] N. F. Ahmad and D. B. Hoang, "Assistive Health Care Monitoring Framework Using Active Database Approach," in *4th IADIS International Conference e-Health*, Algarve, Portugal, 2009, pp. 19-26.

[231] C. Orwat, A. Graefe, and T. Faulwasser, "Towards Pervasive Computing in Health Care – A Literature Review," *BMC Medical Informatics and Decision Making,* vol. 8, pp. 26-45, June 2008.

[232] U. Varshney, "Managing Wireless Health Monitoring for Patients with Disabilities," *IT Professional,* vol. 8, pp. 12-16, November/December 2006.

[233] P. Kulkarni and Y. Ozturk, "mPHASiS: Mobile Patient Healthcare and Sensor Information System," *Journal of Network and Computer Applications,* vol. 34, pp. 402-417, January 2011.

[234] F. Balagtas-Fernandez, M. Tafelmayer, and H. Hussmann, "Mobia Modeler: Easing the Creation Process of Mobile Applications for Non-Technical Users,"

in *15th International Conference on Intelligent User Interfaces*, Hong Kong, China, 2010, pp. 269-272.

[235] R. F. LeBlond, R. L. DeGowin, and D. D. Brown, *DeGowin's Diagnostic Examination*, 9th ed. New York, NY: McGraw-Hill Medical, 2009.

[236] S. F. Smith and D. Duell, *Clinical Nursing Skills: Nursing Process Model, Basic to Advanced Skills*, 3rd ed. Englwood, NJ: Appleton & Lange, 1992.

[237] F. Paganelli, E. Spinicci, and D. Giuli, "ERMHAN: A Context-Aware Service Platform to Support Continuous Care Networks for Home-Based Assistance," *International Journal of Telemedicine and Applications,* vol. 2008, January 2008.

[238] P. D. Haghighi, B. Gillick, S. Krishnaswamy, M. M. Gaber, and A. Zaslavsky, "Mobile Visualization for Sensory Data Stream Mining," in *2nd International Workshop on Knowledge Discovery from Sensor Data*, Las Vegas, NV, 2008, pp. 85-92.

[239] American Heart Association. (2012, July). *Understand Your Risk for High Blood Pressure*. Available: http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/UnderstandYourRiskforHighBloodPressure/Understand-Your-Risk-for-High-Blood-Pressure_UCM_002052_Article.jsp

[240] M. H. Ellestad, *Stress Testing: Principles and Practice*, 5th ed. New York, NY: Oxford University Press, 2003.

[241] R. L. Bloom, "A Case-Based Approach to Teaching Evidence-Based Practice and Motor Speech Disorders," *Contemporary Issues in Communication Science and Disorders,* vol. 37, pp. 123-130, 2010.

[242] V. Villarreal, J. Laguna, S. López, J. Fontecha, C. Fuentes, R. Hervás, *et al.*, "A Proposal for Mobile Diabetes Self-Control: Towards a Patient Monitoring Framework," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*. vol. 5518, J. Cabestany, I. Rojas, and G. J. Caparrós, Eds., ed Berlin, Germany: Springer, 2009, pp. 869-876.

[243] W. M. Wang, C. F. Cheung, W. B. Lee, and S. K. Kwok, "Knowledge-Based Treatment Planning for Adolescent Early Intervention of Mental Healthcare: A Hybrid Case-Based Reasoning Approach," *Expert Systems,* vol. 24, pp. 232-251, September 2007.

[244] L. Blessing and A. Chakrabarti, *DRM, a Design Research Methodology*. London, UK: Springer, 2009.

[245] C. M. Eckert, P. J. Clarkson, and M. K. Stacey, "The Spiral of Applied Research: A Methodological View on Integrated Design Research," in *14th International Conference on Engineering Design*, Stockholm, Sweden, 2003, pp. 19-21.

[246] R. Chow and W. Jonas, "Beyond Dualisms in Methodology: An Integrative Design Research Medium "MAPS" and some Reflections," in *Design Research Society Conference*, Sheffield, UK, 2008, pp. 1-18.

[247] M. L. Markus, A. Majchrzak, and L. Gasser, "A Design Theory for Systems that Support Emergent Knowledge Processes," *MIS Quarterly,* vol. 26, pp. 179-212, September 2002.

[248] S. T. March and G. F. Smith, "Design and Natural Science Research on Information Technology," *Decision Support Systems,* vol. 15, pp. 251-266, December 1995.

[249] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. London, UK: MIT Press, 1996.

[250] R. Cole, S. Purao, M. Rossi, and M. Sein, "Being Proactive: Where Action Research Meets Design Research," in *26th International Conference on Information Systems*, Las Vegas, NV, 2005, pp. 325-336.

[251] S. Dagtas, Y. Natchetoi, H. Wu, and A. Shapiro, "An Integrated Wireless Sensing and Mobile Processing Architecture for Assisted Living and Healthcare Applications," in *International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*, San Juan, CA, 2007, pp. 70-72.

[252] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly,* vol. 28, pp. 75-105, March 2004.

[253] R. Wieringa, "Design Science as Nested Problem Solving," in *4th International Conference on Design Science Research in Information Systems and Technology*, Malvern, PA, 2009.

[254] V. K. Vaishnavi and William Kuechler Jr., *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Boca Raton, FL: Auerbach Publications, 2008.

[255] J. R. Venable, "The Role of Theory and Theorising in Design Science Research," in *International Conference on Design Science Research in Information Systems and Technology*, Claremont, CA, 2006, pp. 1-18.

[256] S. Easterbrook, J. Singer, M. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds., ed London, UK: Springer, 2008, pp. 285-311.

[257] W. James and G. B. Gunn, *Pragmatism and other Writings*. New York, NY: Penguin Books, 2000.

[258] R. B. Johnson and A. J. Onwuegbuzie, "Mixed Methods Research: A Research Paradigm Whose Time Has Come," *Educational Researcher,* vol. 33, pp. 14-26, October 2004.

[259] J. Brannen, "Mixing Methods: The Entry of Qualitative and Quantitative Approaches into the Research Process," *International Journal of Social Research Methodology,* vol. 8, pp. 173-184, July 2005.

[260] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "Towards Successful Design of Context-aware Application Frameworks to Develop Mobile Patient Monitoring Systems Using Wireless Sensors," in *3rd IEEE Conference Open System*, Kuala Lumpur, Malaysia, 2012, pp. 1-6.

[261] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "A Survey on Success Factors to Design Context-aware Frameworks to Develop Mobile Patient Monitoring Systems," in *3rd IEEE Conference on Open Systems*, Kuala Lumpur, Malaysia, 2012, pp. 1-6.

[262] M. G. Al-Bashayreh, N. L. Hashim, and O. T. Khorma, "A Survey on Success Factors to Design Application Frameworks to Develop Mobile Patient Monitoring Systems," in *2nd IEEE International EMBS Conference Biomedical Engineering and Sciences*, Langkawi, Malaysia, 2012, pp. 57-62.

[263] J. V. Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, and A. Cleven, "Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process," in *17th European Conference on Information Systems*, Verona, Italy, 2009, pp. 2206-2217.

[264] S. Elo and H. Kyngäs, "The Qualitative Content Analysis Process," *Journal of Advanced Nursing*, vol. 62, pp. 107-115, November 2008.

[265] J. Webster and R. T. Watson, "Analyzing the Past to Prepare for the Future: Writing a Literature Review," *MIS Quarterly*, vol. 26, pp. xiii-xxiii, June 2002.

[266] M. Aksit, B. Tekinerdogan, F. Marcelloni, and L. Bergmans, "Deriving Frameworks from Domain Knowledge," in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, M. Fayad, D. C. Schmidt, and R. E. Johnson, Eds., ed New York, NY: Wiley, 1999, pp. 169-198.

[267] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature Modeling Plug-In for Eclipse," in *Eclipse Technology eXchange (ETX) Workshop*, Vancouver, BC, 2004, pp. 67-72.

[268] J. Kim, M. Kim, and S. Park, "Goal and Scenario Based Domain Requirements Analysis Environment," *Journal of Systems and Software*, vol. 79, pp. 926-938, July 2006.

[269] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenarios in System Development: Current Practice," *IEEE Software*, vol. 15, pp. 34-45, March/April 1998.

[270] Brazilian Society of Cardiology, "IV Guideline for Ambulatory Blood Pressure Monitoring / II Guideline for Home Blood Pressure Monitoring," *Arquivos Brasileiros de Cardiologia*, vol. 85, pp. 1-18, February 2005.

[271] H. W. Rodbard, L. Blonde, S. S. Braithwaite, E. M. Brett, R. H. Cobin, Y. Handelsman, *et al.*, "American Association of Clinical Endocrinologists Medical

Guidelines for Clinical Practice for the Management of Diabetes Mellitus," *Endocrine Practice*, vol. 13, pp. 1-68, May/June 2007.

[272] D. Matthews, N. Meston, P. Dyson, J. Shaw, L. King, and A. Pal, *Diabetes: An Overview*. Oxford, UK: Oxford, 2008.

[273] US National Library of Medicine. (2011, July). *Hypoglycemia*. Available: http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0001423/

[274] H. Batteram, H. Benz, T. Broens, P. D. Costa, H. Eertink, L. F. Pires, *et al.*, "AWARENESS Scope and Scenarios," Freeband AWARENESS Enschede, Netherlands AWARENESS/D1.1v2, June 2005.

[275] R. Huis, H. Hermens, and M. Vollenbroek-Hutten, "Tools and Methods for Reliable Measurement of Sensory Input Information: State of the Art and First Selection of Parameters," Freeband AWARENESS AWARENESS/D4.1, November 2004.

[276] K. H. E. Kroemer, H. J. Kroemer, and K. E. Kroemer-Elbert, "Exercise and Work," in *Engineering Physiology*, K. H. E. Kroemer, H. J. Kroemer, and K. E. Kroemer-Elbert, Eds., 4th ed Berlin, Germany: Springer, 2010, pp. 173-198.

[277] A. Bertolino, G. D. Angelis, A. D. Sandro, and A. Sabetta, "Is My Model Right? Let me Ask the Expert," *Journal of System and Software*, vol. 84, pp. 1089-1099, July 2011.

[278] M. Moon, K. Yeom, and H. S. Chae, "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line," *IEEE Transactions on Software Engineering*, vol. 31, pp. 551-569, July 2005.

[279] K. E. Wiegers, *Software Requirements*, 2 ed. Redmond, WA: Microsoft Press, 2003.

[280] Sparx Systems. (2014, January). *Enterprise Architect*. Available: http://www.sparxsystems.com/products/ea/index.html

[281] Sparx Systems, "MDA Overview," 2007.

[282] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, NY: Wiley, 2003.

[283] M. H. Lutz and P. A. Laplante, "C# and the .NET Framework: Ready for Real Time?," *IEEE Software,* vol. 20, pp. 74-80, Jan/Feb 2003.

[284] A. Davies, *Async in C# 5.0*. Sebastopol, CA: O'Reilly, 2012.

[285] S. Olson, J. Hunter, B. Horgen, and K. Goers, *Professional Cross-Platform Mobile Development in C#*. Indianapolis, IN: Wiley, 2012.

[286] G. Shackles, *Mobile Development with C#*, 1st ed. Sebastopol, CA: O'Reilly, 2012.

[287] Xamarin. (2014, April). *Mono*. Available: http://www.mono-project.com

[288] Simple Injector. (2014, April). *Simple Injector*. Available: http://simpleinjector.codeplex.com/

[289] E. Skorve and M. Aanestad, "Bootstrapping Revisited: Opening the Black Box of Organizational Implementation," in *Scandinavian Information Systems Research*. vol. 60, K. Kautz and P. A. Nielsen, Eds., ed Berlin, Germany: Springer, 2010, pp. 111-126.

[290] M. Seemann, *Dependency Injection in .NET*. Shelter Island, NY: Manning, 2012.

[291] M. Gousset. (2013). *Use Code Maps to Understand Code Relationships*. Available: http://visualstudiomagazine.com/articles/2013/04/25/use-code-maps-to-understand-code-relationships.aspx

[292] Microsoft. (2014, June). *Visual Studio Code Metrics Viewer 2013*. Available: http://visualstudiogallery.msdn.microsoft.com/03de6710-4573-460c-aded-96588572dc19

[293] C. Driver and S. Clarke, "An Application Framework for Mobile, Context-Aware Trails," *Pervasive and Mobile Computing,* vol. 4, pp. 719-736, October 2008.

[294] P. Devanbu, S. Karstu, W. Melo, and W. Thomas, "Analytical and Empirical Evaluation of Software Reuse Metrics," presented at the 18th International Conference on Software Engineering, Berlin, Germany, 1996.

[295] Microsoft, *Enterprise Solution Patterns Using Microsoft .NET*. Redmond, WA: Microsoft, 2003.

[296] J. Bishop, *C# 3.0 Design Patterns*. Sebastopol, CA: O'Reilly, 2008.

[297] R. E. Johnson and B. Foote, "Designing Reusable Classes," *Journal of Object-Oriented Programming,* vol. 1, pp. 22-35, June/July 1988.

[298] P. Coad and E. Yourdon, *Object-Oriented Design*. Englewood Cliffs, NJ: Yourdon Press, 1991.

[299] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering,* vol. 20, pp. 476-493, June 1994.

[300] G. Booch, *Object Oriented Design: With Applications*. Redwood City, CA: Pearson, 1991.

[301] T. Korson and J. D. McGregor, "Understanding Object-Oriented: A Unifying Paradigm," *Communications of the ACM,* vol. 33, pp. 40-60, September 1990.

[302] M. F. Kilian, "A Note on Type Composition and Reusability," *ACM SIGPLAN OOPS Messenger,* vol. 2, pp. 24-32, July 1991.