



Algerian Republic Democratic and Popular
Ministry of Higher Education and Scientific Research



University Mohamed El Bachir El Ibrahimi of Bordj Bou Arreridj

Mathematics and Computer Science Faculty

Department of Computer Science

Specialization

Distributed Informatics and Decision-making

Thesis submitted in fulfillment of the requirements for the degree of
Third Cycle LMD Doctorate in Computer science

Conceptual Solutions of Modeling the Adaptive Aspect for Context-Awareness in Ubiquitous Environments

**Issues conceptuelles de modélisation de l'aspect adaptif pour
la sensibilité au contexte en environnement ubiquitaire**

Presented by

Boudjemline Haïder

Committee:

Dr. Nouioua Farid	University of Bordj Bou Arreridj	President
Pr. Khababa Abdallah	University of Setif	Examiner
Dr. Alti Adel	University of Setif	Examiner
Dr. Akhrouf Samir	University of M'sila	Examiner
Pr. Touahria Mohamed	University of Setif	Supervisor
Pr. Boubetra Abdelhak	University of Bordj Bou Arreridj	Co-Supervisor

2018/2019

*This thesis is dedicated to my parents,
to my brothers and sisters,
and to all those who believed in me,
without whose support during the hardest of times,
this work would not have been possible.*

Acknowledgments

First and foremost, my truthful thankfulness goes to the most merciful **ALLAH** for all things he blessed me with throughout my whole life. Without those blessings, I would not be in this position at all.

During my doctoral study, I have been accompanied and supported by many people. It is a pleasure that I have now the opportunity to express my gratitude to all of them.

I owe my most sincere gratitude to my supervisor **Pr. Touahria Mohamed** for his sympathetic personality, enthusiasm, patience, engagement and for his countless hours of reading, useful comments, numerous valuable advices and remarks.

I also feel honored and grateful to thank my co-supervisor **Pr. Boubetra Abdelhak**, who has always kept an eye on the progress of my work and was always available when I needed his advice. His encouragement and guidance were very valuable.

I would like to thank the members of the jury for accepting to evaluate my work: **Dr. Nouioua Farid, Pr. Khababa Abdallah, Dr. Alti Adel, and Dr. Akhrouf Samir**, which means a significant amount of work.

Special thanks to **Kaabeche Hamza** and **Bounouni Mahdi** who provided valuable inputs about my work and encouraged the research collaboration.

Furthermore, I extend my sincere thanks to all my colleagues especially to **Younes Chaouch Islem** and **Loucif Hamza**.

I would like to acknowledge and thank my faculty for allowing me to conduct my research and providing any requested assistance.

Words cannot express how grateful I am to my parents for the sacrifices that they have made on my behalf and their prayers since the day I was born. Also, all appreciation goes to my brothers and sisters for their continuous support and their constant care.

I am also very grateful to all my friends, especially **Yahia, Lokmane, Amine, Mounir, and Hamza** who have always been a constant source of encouragement during the hardest times.

Thank you all

Abstract

With the widespread use of mobile devices, a new generation of ubiquitous applications has emerged in daily life activities, making information available everywhere and at any moment. Context-awareness is the feature that allows these applications to be smart, by enabling them to continuously detect the user's current situation and to assist him or her by adapting the application's behavior accordingly.

In the same vein, designing context-aware applications involves new challenges not present in traditional ones, including the perpetual variations in contextual information and the different interactions with sensors. This has propelled the software engineers to introduce additional requirements analysis and enhanced modeling techniques capable of dealing with the different contextual parameters that may affect the application's behavior.

The Unified Modeling Language (UML) is the most commonly used language to specify, visualize, and document the artifacts of software systems. Nevertheless, existing UML diagrams are too general to describe context-aware systems adequately, they do not offer suitable notations to distinguish context-awareness from other system requirements.

This thesis proposes a new modeling approach called the Context Unified Modeling Language (CUML), which is an extension of UML diagrams to cater for the specification, visualization, and documentation of context-aware computing systems.

In this thesis, we present a number of contributions through which we have succeeded in (i) A novel notion called ContextClass diagram, which extends the UML Class diagram to model the structure of context-aware systems by monitoring the contextual aspects that characterize them. (ii) A novel notion called ContextSequence diagram, which extends the UML Sequence diagram to model the interactions as well as the adaptation behavior of context-aware systems. (iii) A set of concrete modeling editors. (iv) A case study in the healthcare field to demonstrate the pragmatics of the proposed approach.

Keywords: Context-aware Systems, Ubiquitous Environment, Adaptation, Context Modeling, Requirements Analysis, Software Design.

Contents

Acknowledgement	i
Abstract	ii
Table of Contents	iii
Figure Index	x
List of Tables	xi
Glossary of Abbreviations	xii
1 Introduction	1
1.1 Background	2
1.2 Challenges	2
1.3 Scopes and contributions	3
1.4 Case study	4
1.5 Evaluation criteria	4
1.6 Structure of the thesis	5
I STATE OF THE ART	7
2 Literature Review of Context-Aware Computing	8
2.1 Introduction	9
2.2 Context and Context-awareness	9
2.2.1 What is context?	9

2.2.2	Context-awareness	11
2.3	Context categories and characteristics	13
2.4	Adaptation to context	14
2.4.1	Definition of adaptation	15
2.4.2	Adaptation process	15
2.4.3	Classifications of adaptation	16
2.4.4	Utility of adaptation	16
2.5	Context-awareness in ubiquitous environments	17
2.5.1	The origins of context-awareness in ubiquitous computing	17
2.5.2	Early context-aware ubiquitous applications	18
2.5.3	Context-awareness usage in ubiquitous applications	19
2.6	Common context-aware systems architecture	21
2.6.1	Context sensing layer	22
2.6.2	Raw data retrieval layer	23
2.6.3	Preprocessing layer	23
2.6.4	Storage and Management layer	23
2.6.5	Application layer	24
2.7	Context-aware computing research areas	24
2.8	Summary	25
3	The Model Driven Engineering	26
3.1	Introduction	27
3.2	The four-level architecture	27
3.2.1	Basic concepts	27
3.2.2	Why use metamodels?	28
3.2.3	The Meta-Object Facility (MOF)	28
3.2.4	The four-level architecture	29
3.3	The Model-Driven Architecture (MDA)	30
3.3.1	Principles of the MDA	30
3.3.2	Models managed in the MDA	31
3.3.3	MDA process	31
3.3.4	Model transformation techniques in MDA	32

3.3.4.1	The marking technique	32
3.3.4.2	The model merging technique	33
3.4	The Unified Modeling Language (UML)	33
3.4.1	Overview of the UML	33
3.4.2	UML diagrams	34
3.4.2.1	The UML Class diagram	35
3.4.2.2	The UML Sequence diagram	39
3.5	UML extension mechanisms	42
3.5.1	When to extend UML?	42
3.5.2	Lightweight extension (UML Profile)	43
3.5.3	Heavyweight extension (MOF-based metamodel)	44
3.5.4	Lightweight vs. Heavyweight	45
3.6	Summary	46

II CONTRIBUTIONS 47

4 Context Handling Approaches 48

4.1	Introduction	49
4.2	Why is context difficult to handle?	49
4.3	Handling context as a separated concern	51
4.4	Context handling approaches	52
4.4.1	The Context Toolkit	52
4.4.2	Handling context using the MDA	54
4.4.2.1	The marking technique	54
4.4.2.2	The merging technique	55
4.4.3	Handling context using UML Profiles	56
4.5	Discussion and evaluation of the existing approaches	58
4.6	Summary	61

5 Overview of the Context Unified Modeling Language 62

5.1	Introduction	63
5.2	Motivations	63

5.3	Overview of the Context Unified Modelig Language	64
5.4	Our vision of context	65
5.5	Implementation tools	67
5.5.1	Eclipse Ecore tools	67
5.5.2	The UMLet tool	67
5.6	Summary	68
6	Extending the Class Diagram to Model Context-Aware Systems: the ContextClass Diagram	69
6.1	Introduction	70
6.2	Limitations of the traditional UML Class diagram	70
6.3	Extension of the UML Class diagram: the ContextClass diagram	71
6.4	ContextClass diagram metamodel	73
6.5	Ecore implementation: ContextClass hierarchical editor	76
6.6	ContextClass diagram graphical editor	78
6.7	Summary	79
7	Extending the Sequence Diagram to Model Context-Aware Systems: The ContextSequence Diagram	80
7.1	Introduction	81
7.2	Limitations of the traditional UML Sequence diagram	81
7.3	Extension of the UML Sequence diagram: the ContextSequence diagram	82
7.4	ContextSequence diagram metamodel	86
7.5	Ecore implementation: ContextSequence hierarchical editor	88
7.6	ContextSequence diagram graphical editor	91
7.7	Summary	92
8	Case Study: a Smart Blood Pressure Tracker Application	93
8.1	Introduction	94
8.2	Description of the Smart Blood Pressure Tracker	94
8.3	Utility of the Smart Blood Pressure Tracker	96
8.4	Requirements specification	97
8.5	Contextual parameters of the SPBT system	98

8.6	Modeling the SBPT with the ContextClass diagram	99
8.6.1	The SBPT system's structure	99
8.6.2	Modeling the SBPT with the ContextClass graphical editor	100
8.6.3	Modeling the SBPT with the ContextClass hierarchical editor	101
8.7	Modeling the SBPT with the ContextSequence diagram	104
8.7.1	The SBPT system's behavior	104
8.7.2	Modeling the SBPT with the ContextSequence graphical editor	105
8.7.3	Modeling the SBPT with the ContextSequence hierarchical editor	107
8.8	Summary	110
9	Conclusion and Future Perspectives	111
9.1	Research summary	112
9.2	Evaluation criteria revisited	113
9.3	Contribution to knowledge	115
9.4	Future works	116
	Bibliography	117
	Appendix A	130
	Appendix B	132
	List of Scientific Papers	138

Figure Index

2.1	Context-awareness in the computing evolution chain [46]	18
2.2	A typical display of the Active Badge system [54]	19
2.3	Screenshots of the indoor Cyberguide [55]	20
2.4	The outdoor Cyberguide with GPS unit [55]	21
2.5	Layered architecture of context-aware systems [60]	21
3.1	Relationships between system, model, and metamodel [71]	28
3.2	The four level architecture of MOF [75]	29
3.3	Model transformation process [71]	30
3.4	Models and transformation process in the MDA [71]	32
3.5	The organization of UML diagrams [71]	34
3.6	UML Class diagram metamodel [74]	37
3.7	An example of a typical UML Class diagram [74]	38
3.8	UML Sequence diagram metamodel [74]	41
3.9	An example of a typical UML Sequence diagram [74]	42
3.10	Example of UML lightweight extension [75]	43
3.11	Example of UML heavyweight extension [75]	44
3.12	UML Lightweight vs. Heavyweight extensions mechanisms [84]	45
4.1	The Context Toolkit architecture [61]	53
4.2	MDA architecture based on the marking technique [76]	54
4.3	MDA architecture based on the merging technique [76]	55
4.4	The Class stereotypes of the proposed Profile [101]	56
4.5	The Association stereotypes of the proposed Profile [101]	57
4.6	The timeline of the studied context handling approaches	58

5.1	Overview of the Context Unified Modeling Language diagrams	64
5.2	Context and context-aware applications	66
6.1	An example of the ContextClass "Tourist"	72
6.2	ContextClass diagram metamodel definition	73
6.3	Classifier specialization: the ContextClass metaclass	73
6.4	Feature specialization: the ContextualFeature metaclass	74
6.5	ContextClass structure: the monitor metaclass	74
6.6	The metamodel of the ContextClass diagram	75
6.7	ContextClass metamodel implementation under Eclipse	77
6.8	Example of creating a new ContextClass model	78
6.9	The ContextClass diagram graphical editor interface	79
7.1	An example of an adapt combined fragment in a ContextSequence diagram	84
7.2	ContextSequence diagram for the Temperature Control System	85
7.3	ContextSequence diagram metamodel definition	86
7.4	Lifeline specialization: the SensorLifeline metaclass	87
7.5	InteractionFragment specialization: the AdaptationCombinedFragment metaclass	87
7.6	Constraint specialization: the ContextualConstraint metaclass	88
7.7	The metamodel of the ContextSequence diagram	89
7.8	ContextSequence metamodel implementation under Eclipse	90
7.9	Example of creating a new ContextSequence model	91
7.10	The ContextSequence diagram graphical editor interface	92
8.1	A smartwatch with a built-in heart rate sensor	95
8.2	The infrastructure of the Smart Blood Pressure Tracker system	95
8.3	The ContextClass graphical editor: Modeling the SBPT's structure	101
8.4	The ContextClass Eclipse editor: creating new ContextClasses and associations	101
8.5	The ContextClass Eclipse editor: adding new properties and operations	102
8.6	The ContextClass Eclipse editor: defining the type of a new monitor	103
8.7	The ContextClass Eclipse editor: defining the influenced operations for a monitor	103
8.8	The ContextSequence graphical editor: Modeling the SBPT's behavior	106
8.9	The ContextSequence Eclipse editor: Defining message proprieties	107

- 8.10 The ContextSequence Eclipse editor: Defining an adaptation combined fragment 108
- 8.11 The ContextSequence Eclipse editor: Modeling the hypertensive crisis scenario 109

List of Tables

2.1	The main physical sensor types [64]	22
3.1	The types of relationships in UML Class diagrams [74]	38
4.1	Summary of the studied context handling approach	60
6.1	Recapitulation of the new metaclasses of the ContextClass Diagram	76
7.1	ContextSequence diagram interaction notations	83
7.2	Recapitulation of the new metaclasses of the ContextSequence Diagram	88
8.1	The monitors of each ContextClass of the SBPT system	100
8.2	Specification of the exchanged messages in the SBPT system	104

Glossary of Abbreviations

3G	3 rd Generation
4G	4 th Generation
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
CAMEL	Context-Awareness ModEling Language
CAProf	Context-Aware Profile
CIM	Computation Independent Model
CM	Context Model
CMP	Context Modeling Profile
CSOA	Context-aware Service Oriented Architecture
CUML	Context Unified Modeling Language
DSL	Domain-Specific Language
DSML	Domain-Specific Modeling Language
ECA	Enterprise Collaboration Architecture
EDOC	External Document
EMF	Eclipse Modeling Framework
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
ID	Identifier
IR	Infrared
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MDIA	Model-Driven Integration Architecture

MOF	Meta-Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
OMT	Object-Modeling Technique
OOSE	Object-Oriented Software Engineering
PC	Personal Computer
PDA	Personal Digital Assistant
PDM	Platform Description Model
PIM	Platform Independent Model
PSM	Platform Specific Model
RFID	Radio Frequency Identification
SBPT	Smart Blood Pressure Tracker
SP	Smartphone
SW	Smartwatch
SDLC	Software Development Life Cycle
TCS	Temperature Control System
UML	Unified Modeling Language
UV	Ultraviolet
WiFi	Wireless Fidelity

Chapter 1

Introduction

*"If we knew what it was we were doing,
it would not be called research, would it?"*

Albert EINSTEIN

Contents

1.1	Background	2
1.2	Challenges	2
1.3	Scopes and contributions	3
1.4	Case study	4
1.5	Evaluation criteria	4
1.6	Structure of the thesis	5

1.1 Background

The last few years have shown a growing evolution in mobile technologies, the proliferation of smartphones and other handheld devices has prominently led to the emergence of a new commonly acknowledged research field called ubiquitous computing. Indeed, the future of computer science was marked by Mark Weiser's vision [1] who defines a new technology that can help people over the course of the day in a graceful and a transparent way: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". Still today, this vision has bred the seed of an overwhelming number of research studies with the aim of realizing Weiser's vision, by dint of new applications that can take advantage of the emerging mobile technologies.

Given the particularity of ubiquitous environments which are often characterized by dynamicity and users' mobility, a noteworthy way to provide enhanced functioning of ubiquitous applications is to make them "context-aware" by attributing them capacities to gather information about their environments and to react according to the collected information.

The first generations of ubiquitous applications were just location-aware, they detect the user's current location and provide adapted services accordingly. Subsequently, recent years have seen many applications coming to perceive a huge amount of data about the user's context and use it to make improved adaptations in response.

1.2 Challenges

Adapting applications to contextual variations leads to an enhancement of the user's satisfaction. With this in mind, many software designers and developers are skeptical and concerned because of the challenges that exacerbate the development of applications with such feature. Over the last years, meaningful effort has been devoted to the handling of the contextual elements of which the changes may require the application to adapt itself accordingly.

Regarding the software development lifecycle, early solutions applied the notion of context on finished applications, by trying to continuously adapt them to new contextual situations, whereas other approaches tended to introduce context during the development phase using a set of specific development techniques.

Meanwhile, to the authors' best knowledge, few publications which get into the issue of considering context since earlier stages of the software development lifecycle. In connection therewith, software requirements analysis and modeling are mandatory in application development process, they play a crucial role in preventing software projects from failure by simulating and capturing complete description about how systems are expected to perform.

To a great extent, advances in the field of software engineering offered a variety of modeling languages to design systems. The Unified Modeling Language (UML) is a graphical language commonly used to specify, visualize, and document the artifacts of software systems; in spite of that, UML is an all-purpose modeling language, which means that the current form of its diagrams is sufficient to represent traditional systems concerns, but does not have notations to specify context-awareness requirements conveniently.

1.3 Scopes and contributions

This thesis focuses on an increasingly common challenge of requirements engineering and modeling of context-aware systems. Our key purpose is to provide a novel approach for designing self-adaptive applications by extending the Unified Modeling Language with new notations to cater for the specification, visualization and documentation of context-aware systems.

After discussing the limitations of the existing context handling approaches, we present our proposal that we called Context Unified Modeling Language (CUML), and which consists in extending the UML Class and Sequence diagrams for context-aware applications modeling.

By extending the existing notations of the UML Class diagram, we aim to support the design of context-aware systems' structure and prepare the context-awareness requirements for the implementation stage. Hence, the novel graphical notations help to clarify the structural modeling of context-aware applications by highlighting how these are linked to context parameters and how may the values of those parameters affect the application.

Extending UML notations for the traditional Sequence diagram aims to increase designers' attention towards context-awareness through two concepts: the first one offers dedicated notations to model the different interactions that convey contextual information from sensors; the second one concerns the adaptation behavioral aspect, it depicts the automatic adaptation actions to be performed depending on context variations.

Another advantageous contribution of this thesis is the implementation of the proposed modeling approaches by creating hierarchical and graphical modeling tools. This may enable exploiting the expressiveness of our modeling approach by simulating and documenting the structure and behavior of context-aware systems.

1.4 Case study

To better illustrate the purpose of this thesis work, we rely on an application example in the healthcare field. Healthcare applications have become popular around the world, the shift towards them can improve the management of hospital workflows and support clinical communication between providers and patients.

Our case study is of interest to hypertensive patients and is presented as a Smart Blood Pressure Tracker. Hypertension, also known as *the silent killer*, it has no symptoms, but it's a major risk for heart disease and stroke, for this, the application is able of screening blood pressure values regularly to decrease hypertension crisis damage probabilities. Also, the Smart Blood Pressure Tracker is purposefully intended to provide an automatic system for the daily measurement and surveillance of the patient's blood pressure and also to enhance the monitoring and the reporting of hypertensive patients records, everywhere and at any time.

We chose this example because it expresses the main characteristics of a context-aware system in a ubiquitous environment, also, it shows the necessity and the interest of having automatic adaptation reactions to new contextual situations, and which should better be taken into account at the design stage of the development lifecycle.

Exploiting the proposed CUMML diagrams, we aim to demonstrate our modeling approach in rich scenarios by modeling the various application needs, and show how to apply the novel diagram notations in practical ways.

1.5 Evaluation criteria

In light of the above objectives, this research sets out to answer the following questions which stand for its measures of success:

Q1. What is the originality of the CUMML proposal and what contributions does it afford over the existing context handling approaches?

- Q2.** How can software designers achieve efficient translation of context into models using CUML diagrams?
- Q3.** How to use CUML to properly integrate the contextual parameters and their influence on applications when modeling context-aware systems' structure?
- Q4.** How does CUML enable software designers to effectively specify details about context changes events and interactions methods between applications and context sensors?
- Q5.** How can CUML diagrams be used to describe autonomous adaptation behavior of context-aware applications?
- Q6.** How is it possible for applications modelers to concretely build and share their models using CUML tools?
- Q7.** Are the resulting models conform to specific syntax rules? How to ensure that?

1.6 Structure of the thesis

This thesis is organized into nine chapters with the following objectives:

The introduction is **Chapter 1**, it gives some hints to the readers by presenting an overview of the document. The remainder of this document has two parts. The first one comprises the state of the art, it includes basic concepts and terminology used throughout this manuscript, as well as related approaches. It is divided into three chapters:

- **Chapter 2** introduces the reader to context, context-awareness, and adaptation, starting from a historical perspective. The chapter then outlines some fundamental properties in the domain, such as the possible categorizations of context as well as the common architecture of context-aware systems.
- **Chapter 3** concentrates on introducing some modeling concepts which are requisite to understand the rest of the thesis, it focuses on standards proposed by the OMG (Object Management Group) such as the Model-Driven Architecture and the Unified Modeling Language.
- **Chapter 4** provides an overview of the importance of requirements engineering in context-aware systems development, presents concepts around the separation of concerns, and investigates some previous works on handling context, by standing out the advantages and the weaknesses of each approach.

The second part of the manuscript presents our contributions, it is divided into four chapters:

- **Chapter 5** gives an overview of the CUML modeling approach by presenting our vision of context, the general methodology in which we proceeded, as well as the tools we used to elaborate modeling editors of our proposal.
- In **Chapter 6**, we start by addressing the limitations of the traditional UML Class diagram, afterward, we present the concept of ContextClass diagram to model the structure of context-aware applications.
- **Chapter 7** presents the limitations of the existing UML Sequence diagram, before defining the concept of the ContextSequence diagram to model the behavior of context-aware applications.
- **Chapter 8** demonstrates the expressiveness of the proposed extensions using a real-world case study of a Smart Blood Pressure Tracker, we describe its different contextual aspects using the CUML diagrams.
- Summary of contributions, thesis's findings, along with some possible directions for future research are drawn in **Chapter 9**.

Part I

STATE OF THE ART

Chapter 2

Literature Review of Context-Aware Computing

"The science of today is the technology of tomorrow"

Edward TELLER

Contents

- 2.1 Introduction 9
- 2.2 Context and Context-awareness 9
- 2.3 Context categories and characteristics 13
- 2.4 Adaptation to context 14
- 2.5 Context-awareness in ubiquitous environments 17
- 2.6 Common context-aware systems architecture 21
- 2.7 Context-aware computing research areas 24
- 2.8 Summary 25

2.1 Introduction

The continuous process of hardware miniaturization together with the improvements in the field of software engineering is leading to the birth of smart environments in which users can search for, access, and exchange information whenever and wherever. Context-awareness enables these applications to assist the user by adjusting their behavior every time when the context of use changes. Grasping context characteristics is thus crucial in developing self-adaptive applications.

This chapter lays the groundwork for the thesis by examining the main properties of context-awareness in ubiquitous environments, it starts by defining the terms of context, context-awareness, and adaptation, then explores some fundamentals of the domain, including the common architecture of context-aware systems and some possible categorizations of context.

2.2 Context and Context-awareness

Throughout this thesis, we will make use of a set of recurrent terms to discuss related works and to describe novel concepts as well. This section analyses what the community understands by the terms *context* and *context-awareness*.

2.2.1 What is context?

The Oxford Dictionary¹ gives a general definition for context as "The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood".

According to the Cambridge Dictionary², context is defined as "The situation within which something exists or happens, and that can help explain it".

When browsing through the scientific literature from a span of disciplines, we realize that many definitions were given to the term *context*, researchers in diverse domains tried to define it in their own way depending on the necessities and the investigated field. In the following, we present a non-exhaustive list of context definition proposals ordered chronologically:

¹<https://en.oxforddictionaries.com>

²<https://dictionary.cambridge.org>

- Schilit and Theimer [2] were the first to define context by describing it as "locations, identities of nearby people, objects, and changes to those objects over time".
- Schilit et al. [3] considered context as "the constantly changing execution environment".
- More precisely, Brown et al. [4] enumerated context as "location, identities of the people around the user, the time of day, season, temperature, etc.".
- Pascoe [5] defined context as "the subset of physical and conceptual states of interest to a particular entity".
- Brézillon and Pomerol [6] referred to context as "all the knowledge that constrains a problem solving at a given step without intervening in it explicitly".
- Schmidt [7] considered context as "the knowledge about the state of the user and device, including surroundings, situation and tasks", he emphasized on the fact that context is more than location.
- An interesting approach has been proposed by Dix et al. [8] in which context is divided into four dimensions: system, infrastructure, domain, and physical context.
- One of the most complete definitions was given by Dey and Abowd [9], they describe it as "any information that can be used to characterize the situation of an entity. An entity should be treated as anything relevant to the interaction between a user and an application, such as a person, a place, or an object, including the user and the application themselves".
- By contrast, Winograd [10] saw the definition of Dey and Abowd as too general, so he specified context as an operational term: "something is context because of the way it is used in interpretation, not due to its inherent properties".
- In agreement with Mostéfaoui et al. [11], context generally refers to what surrounds the center of interest, provides additional sources of information "where, who, what" and increases understanding.
- Lemlouma [12] viewed context as "the set of all environment's information that can influence on the process of adaptation and content transmission to the end user".
- Chaari et al. [13] considered that the definition given by Dey may be a source of conflicts since it does not distinguish what exactly belongs to context from the application's internal elements, they proposed to define context as "the set of parameters which are external to the application and which can influence on its behavior by defining new views on its data and its services. These parameters have a dynamic aspects that allows them to evolve during the execution time".

- Strassner et al. [14] used the following definition of context: "is a collection of measured (the facts as perceived) and inferred knowledge (resulting from a learning and computational reasoning applied to past and present contexts) that describe the state and environment in which an entity exists or has existed".

By analyzing the previous definitions of the term *context*, one can notice that its meaning has drastically evolved following the advances in application development, after being related to location and identity of users and objects, the term has expanded later to include more environmental information such as the computing and the physical environment.

2.2.2 Context-awareness

Since the term *context-awareness* highly depends on *context*, context-awareness definitions can therefore be diverse. In the following, we are interested in presenting the foremost definitions given to the terms *context-awareness*, *context-aware computing*, *context-aware systems*, and *context-aware applications*.

- The originators of the term *context-awareness* are Schilit and Theimer [2] who in 1994 defined it as "the ability of a mobile user's applications to discover and react to changes in the environment they are situated in".
- Next, Schilit et al. [3] stated that context-awareness enables systems to adapt themselves according to the location of use, the collection of nearby people, hosts, accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes in it.
- Brown et al. [4] defined context-aware applications as those that change their behavior according to the user's context.
- As reported by Ryan et al. [15], context-awareness is a term that describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user's identity.
- Salber et al. [16] defined context-awareness to be "the ability to provide maximum flexibility of a computational service based on real-time sensing of context".
- Lieberman and Selker [17] indicated that, unlike traditional applications which acquire explicit input data (from the user) to produce explicit output, a context-aware application also takes into account implicit data (generally sensed by its self).

- Korkea-Aho [18] argue that "a system is context-aware if it can extract, interpret and use context information and adapt its functionality to the current context of use".
- Dey [19] developed another definition by stating that "a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task".
- Burrell and Gay [20] gave another definition stating that "context-aware computing is the use of environmental characteristics such as the user's location, time, identity, and activity to inform the computing device so that it may provide information to the user relevantly to the current context".
- In [21] it was shown that "applications are context-aware if they are able to use context to adapt the presentation of hypermedia".
- For Rohn [22], "a system is context-aware if it can extract, interpret and use context information and adapt its functionality to the current context of use".
- Barkhuus [23] claims that "context-awareness is the application's ability to detect and react to environment variables autonomously".
- Chaari et al. [13] characterized context-aware systems by their capacity to sense the user's situation in his environment, and to consequently adapt all the system's behavior to the related situation, in terms of services, data, and interfaces, and without explicit user intervention.
- According to Abbas and his colleagues [24], context-aware systems are capable of returning the user with information relevant and adapted to his needs and his context that influences his behavior during its interaction with information systems.
- Christopoulou [25] stated that a mobile application is context-aware if it uses context to provide relevant information to users or to enable services for them, relevancy depends on a user's current task and profile.
- Sindico [26] defined context-awareness as "the ability of a system to change its behaviors depending on the characteristics of the environment".

Throughout the previous definitions, we notice that the context-awareness can be characterized by two common aspects, the ability to sense context and the ability to exploit the sensed context; in other words, a context-aware system does not solely sense the environment, it is also able to react to environmental changes.

Along with that, the term *context-aware* has become somewhat synonymous with other terms such as: *adaptive* [27], *reactive* [28], *responsive* [29], *situated* [30], *context-sensitive* [31] and *environment-directed* [32].

2.3 Context categories and characteristics

Not only is difficult to reach an agreement in what context is, but also on how can it be categorized. A clear and accurate classification of context is highly needed to sort out a variety of contexts, to enable application designers to choose what context to use in their applications, and to determine what context-aware behaviors to support [9].

The context of use can be classified from different perspectives, the list below summarizes a few well-known forms of context categorization given in the current literature:

- Bill Schilit [3] divided context into three categories: *computing context* (computing and network situation), *user context* (user's location and other user related situations), and *physical context* (physical environment situation).
- Chen and Kotz [33] elaborated two categories: *active context* which directly influences on the application behavior, and *passive context* which is necessary but not critical for the application, the latter presents it to an interested user to retrieve later.
- Petrelli et al. [34] defined the *material context* (location, machine, existing platform), and the *social context* (social aspects) such as the relationship between the individuals.
- Gwizdka [35] presented two categories: *internal context* which represents the user's state, and the *external context* which contains the state of the environment.
- Dey [19] classified context into two categories: *primary context* which encompasses location, identity, time and activity information, and *secondary context* which can be derived from the former one.
- Hofer et al. [36] established two categories for context: *physical context* which can be retrieved with physical sensors, and *logical context* that contains interaction information (emotional state of the user, his objectives, etc.).
- Henriksen [37] has shown that context can be categorized as:
 - *Sensed context* is directly detected by sensors such as temperature, humidity, etc.
 - *Static context* is the information that does not change over time such as the identifi-

- cation of sensors from the manufacturer, person identification, etc.
- *Profiles context* means the information that can evolve over time with low frequency such as the location of the sensor, the status of the person, etc.
 - *Derived context* means the information that is computed by using primary context such as the distance between two sensors.
- Razzaque et al. [38] proposed a six categories classification:
 - *User context*: allows getting user’s information, such as his profile. It encompasses information on his identity, his relationships with other users, etc.
 - *Physical context*: allows to embed physical environment information, such as location, humidity, temperature, noise level, etc.
 - *Network context*: provides the relevant network information, such as: network connectivity, bandwidth, protocols.
 - *Activity context*: repertories the events which occurred in the environment, such as a user movement or a temperature variation.
 - *Device context*: allows identifying the usable devices as well as their information (ID, location, battery level, available resources, etc.)
 - *Service context*: holds information about the functionalities offered by the system.
 - Authors in [39] gave a more detailed classification by identifying several parameters that may affect the current situation of a user: *environment* (time, location, etc.), *application* (software, hardware, etc.), and the *user himself* (identity, preferences, etc.).
 - Geihs and Wagner [40] indicate that contextual information includes the state of *the execution environment*, the state of *the computing device*, and the *user preferences*.
 - Sánchez-Pi et al. [41] considered that context parameters include *physical location*, *device type*, *user role*, *time*, *temperature* and *available battery*.

None of those presented types of classification can be considered as the best since each one is suitable and reasonable in a specific situation. Hence, context can be handled from the desired perspective using a certain classification scheme.

2.4 Adaptation to context

The concept of *adaptation* was and still is a subject of numerous studies. This section sets forth the attention that has been paid to put the adaptation forward in context-aware systems.

2.4.1 Definition of adaptation

Generally, *adaptation* equals the ability of systems to use context for providing appropriate responses to users, in the following we present the most interesting definitions of this term:

- Dey et al. [42] introduced the notion of adaptation to be the work leading to the automation of a software system based on knowledge of the user's context.
- Layaïda [43] referred the adaptation to the automatic or semi-automatic means that allow content to be usable on devices with various characteristics and resources.
- According to Dey and Abowd [9], the adaptation of an application to a particular context is the activity or process that the application needs to perform in order to satisfy the requirements posed by the context.
- Satyanarayanan has shown the importance of adaptation in ubiquitous computing in [44] by stating that "the adaptation is necessary when there is a significant mismatch between the supply and demand of a resource".
- Capra et al. [45] defined adaptation as "the ability of the application to alter and reconfigure itself as a result of context changes, to deliver the same service in different ways when requested in different contexts and at different points in time".

2.4.2 Adaptation process

To accomplish the objective of providing efficient adaptations and maintaining the execution in diverse situations, the context-aware system must [46]:

- Perceive the necessary contextual information using various sensors.
- Translate this sensed information into an adequate format.
- Interpret and aggregate low-level context information to generate a high-level one (e.g., conversion of geographic coordinates into street names).
- Automatically trigger adaptation actions based on the context information and the monitoring of actions.
- Make the information accessible to other applications, the context management model should therefore handle context in a reusable manner.

2.4.3 Classifications of adaptation

In this passage, readers can explore the essence of the classification of the existing adaptation works elaborated by Chassot [47] and summarized by Chaari [48]. This classification depends on the way in which it is applied, and how resources are utilized, as an instance of the authors' study, the resource may be a document, an image, a video or even an application.

- **Static (manual) vs. dynamic (automatic):** *static* or *manual* adaptation consists in preparing several versions of the same resource so that it can be directly used in its new context, this strategy presents a simple and effective solution but has the drawback of the inability to support new contexts. *Dynamic* adaptation performs transformations on the resource following the current context, although, adaptation problems may arise when providing a result which may not be relevant for the related context.
- **Vertical (centralized) vs. horizontal (distributed):** the adaptation can concern a resource which is local to a machine, or distributed on several ones. In the case where the resource is *distributed*, the adaptation process can be centralized or distributed over the different peers where the parts of the resource have been stored. The *centralized* solution is easier to implement and gives better performances in terms of access time.
- **Behavioral vs. architectural:** adaptation can be described as *behavioral* when it alters the behavior of the resource without affecting its internal structure, the new resource remains directly exploitable and does not require reevaluations since the structure is the same. In contrast, adaptation is *architectural* when the internal structure of the resource can be adapted according to the users' needs, this solution offers a high degree of adaptability, but remains difficult to implement.

2.4.4 Utility of adaptation

When environments are dynamically changing, often unavoidably, the adaptation becomes an attractive option, if not a requirement. Amidst the elementary services that context-aware systems provide to users are [46]:

- **Context triggered action:** triggers are set up on certain events to enable context-aware systems taking automatic actions in response to these events, e.g., given that a user is in a meeting and there is a phone call for him/her, then the call should be routed to voicemail.

- **Multi-facet command processing:** commands issued by the user can produce different results depending upon the context in which they were issued. For instance, a web browser responds according to the context of use (available resources: memory, screen size, and resolution).
- **Location-based proximity:** information about the location of the user and objects located nearby can be given high priority, this would be particularly useful when using specific devices with limited resources. For example, when the user wishes to print something, a nearby printer would be proposed.
- **Metadata tagging:** context information can be attached to existing information as metadata about physical and virtual objects to give descriptive information. For instance, when a user records an audio clip on a handheld device, the system can attach the current context (date, time, current activity, etc.) to the recorded clip for easy retrieval and indexing.
- **Collaborative computing:** voluntary based mission-oriented context-aware and dynamic communities of computing entities that perform tasks on behalf of users autonomously. This kind of applications exists in many domains such as campus management, health care, telemedicine, and crisis management.

2.5 Context-awareness in ubiquitous environments

The aim behind the taking into account of the context of use is to improve the quality of the provided services by adjusting them relevantly to the user's current context. The context-awareness feature has been integrated into various computing domains namely the *cloud computing* [49], *the internet of things* [50], in *healthcare* [51], and in *web service systems* as well [52]. In this thesis, we are interested in one of the foremost application fields that takes advantage of context-awareness which is the ubiquitous computing.

2.5.1 The origins of context-awareness in ubiquitous computing

Traditional applications are unaware of the context of execution, they do not discern what the user is doing, where is he, who is nearby and other information related to the user's environment, they just take the explicit input from the user (e.g., pointing to a menu item), process it, and then output the result [38].

In the article "The computer of the 21st century" [1], Mark Weiser shaped the vision of ubiquitous computing as an omnipresent infrastructure for information and communication technologies of which the purpose is to create a device-rich computing environment able to orchestrate devices and provide services to users at anytime and anywhere. Context-awareness thus seems to be one of the main ingredients to realize the idea of ubiquitous computing.

Deemed as computing of the next generation, ubiquitous computing is aimed to change the way mobile devices behave. The basic idea is to instrument the physical world around us with various kinds of sensors, actuators, and tiny computers, the massive amount of information can then be collected and processed by computer systems, enabling them to deduce the user's situation and act correspondingly [38], as a result, mobile users need to keep accessing the different information and services while moving from one location to another. **Figure 2.1** shows the flow in the evolution chain from centralized computing as presented in [44], [46], [53].

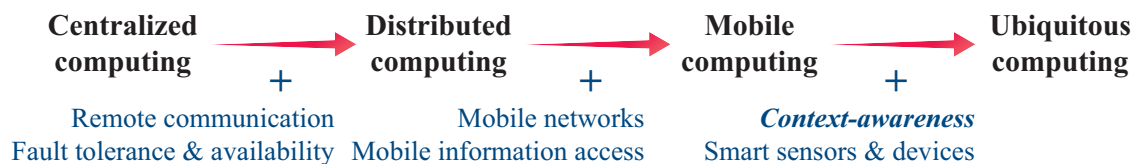


Figure 2.1. Context-awareness in the computing evolution chain [46]

2.5.2 Early context-aware ubiquitous applications

Admittedly, first context-aware applications were office and meeting tools; the reason is that most computers were used in an office environment, also, because it was easier to obtain context information (location) in a limited and controllable area (an office).

In accordance with several studies, researchers agree that the first work in the domain of ubiquitous context-awareness computing was the Olivetti's Active Badge system [54], developed in 1992, just one year after Weiser's vision of "The Computer for the 21st century" [1]. With the Active Badge system, employees could be located inside an office building, and phone calls could be directed to the closest phone to the called person. To meet this objective, each employee wore a badge that periodically transmitted infrared signals. A network of sensors placed around the building picked up the signals, and a central location server polled the sensors. In this way, the telephone receptionist could easily find out in which room a person was located and forward the call to that room's phone, such cases are shown in **Figure 2.2** below.

ORL/STL Active Badge Project					
Name	Location	Prob.	Name	Location	Prob.
P Ainsworth	X343 Accs	100%	J Martin	X310 Mc Rm	100%
T Blackie	X222 DVI Rm.	80%	O Mason	X307 Lab	77%
M Chopping	X410 R302	TUE.	D Milway	X307 Drill	AWAY
D Clarke	X316 R321	10:30	B Miners	X202 DVI Rm.	10:40
V Falcao	X218 R435	AWAY	P Mital	X213 PM	11:20
D Garnett	X232 R310	100%	J Porter	X398 Lib.	100%
J Gibbons	X0 Rec.	AWAY	B Robertson	X307 Lab	100%
D Greaves	X304 F3	MON.	C Turner	X307 Lab.	MON.
A Hopper	X434 AH	100%	R Want	X309 Meet. Rm.	77%
A Jackson	X308 AJ	90%	M Wilkes	X300 MW	100%
A Jones	X210 Coffee	100%	I Wilson	X307 Lab.	100%
T King	X309 Meet. Rm.	11:20	S Wray	X204 SW	11:20
D Liouplis	X304 R311	100%	K Zielinski	X402 Coffee	100%

12.00 1st January 1990

Figure 2.2. A typical display of the Active Badge system [54]

The Georgia Tech Cyberguide project [55], [56] was built in the mid-nineties as a mobile context-aware tour guide to help tourists with information based on their position and orientation. Initial prototypes of the Cyberguide were designed to assist visitors on a tour of the Graphics, Visualization and Usability Center during monthly open house sessions. The prototypes worked on an Apple MessagePad and used infrared beacons for user positioning, whereas location sensing in the outdoor version was based on GPS coordinates. Both Cyberguide indoor and outdoor systems make use of users' environment context data: in the indoor application, users are associated with areas inside the building, while in the outdoor one, the user position is described through GPS coordinates. **Figure 2.3** and **Figure 2.4** depict respectively a screenshot of the indoor Cyberguide and the outdoor version with GPS unit.

2.5.3 Context-awareness usage in ubiquitous applications

Context-aware applications have been making major technological advances with smart mobile devices that help users to achieve many business processes, Schilit [57] listed few examples of basic ubiquitous application uses:

- Helping to navigate the computerized world by displaying the interesting objects, both nearby and far away.

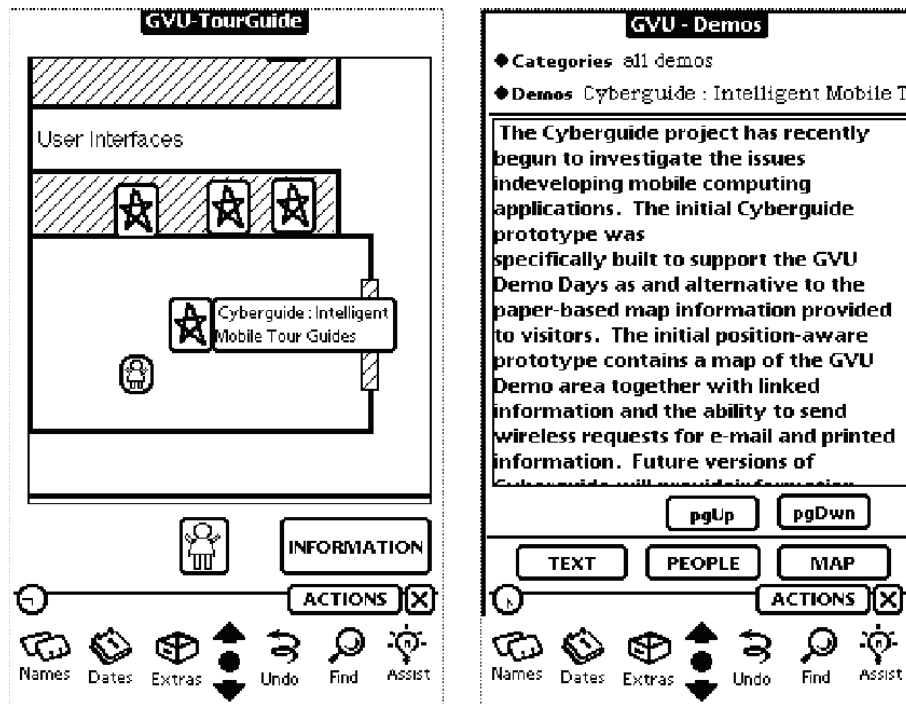


Figure 2.3. Screenshots of the indoor Cyberguide [55]

- Keeping a record of objects and persons, for use by applications such as activity-based information retrieval.
- Detecting specific information, for example, electronic messages left for the user or public perusal.
- Keeping a lookout for nearby devices that can be used opportunistically by applications, such as additional display terminals in a room.
- Detecting nearby people, objects, or services that are relevant to reminders or actions set to be triggered by their presence.
- Tracking a particular object as it moves around a region, examples include tracking a co-worker we wish to talk to or tracking the office coffee cart.
- Tracking objects with a specified set of attributes in a particular region, an example is tracking all members of a workgroup.
- Finding the nearest object to a specified location that meets specific constraints, examples include finding the nearest printer or administrator.
- Monitoring the activity at a particular location, a typical example is keeping track of available terminals at one's current location.

The reader is referred to [18] for more details about different ubiquitous applications.

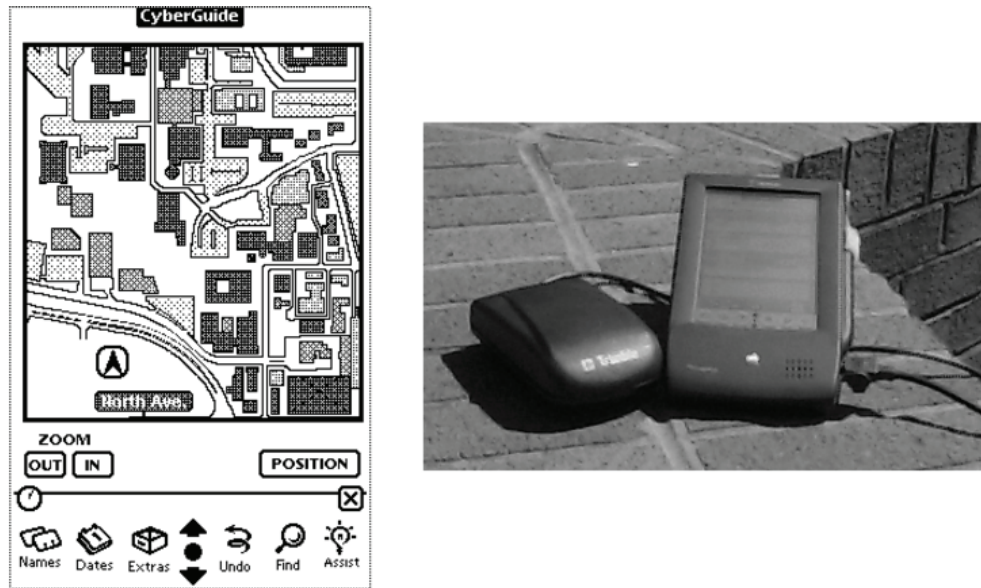


Figure 2.4. The outdoor Cyberguide with GPS unit [55]

2.6 Common context-aware systems architecture

Generally speaking, the architecture is required for systems representation and implementation, it is an abstraction used for generalizing systems without showing the implementation's details [58]. Regarding context-aware systems, many layered-architectures have been proposed during the last years, they differ in functional range, location and naming of layers [59].

A common architecture in modern context-aware systems is identifiable when analyzing the various design approaches. The layered architecture, as depicted in **Figure 2.5**, is primarily based on five main layers, and augments layers for detecting and using context functionalities by adding other interpreting and storage layers [60]-[62].

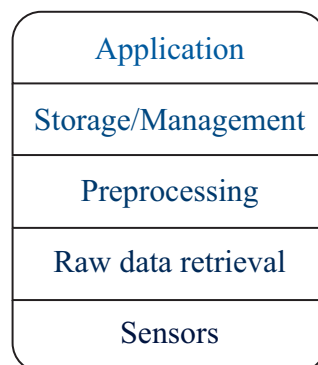


Figure 2.5. Layered architecture of context-aware systems [60]

2.6.1 Context sensing layer

Consists of a set of sensors, it is worth to note that a sensor does not only refer to hardware sensing device but also to any source which may provide useful context information. Regarding the way data is captured, sensors can be classified into three groups: *physical*, *logical*, and *virtual* [63].

- Physical sensors: the most frequent type of sensors, many hardware devices are available nowadays and are capable of providing contextual information by capturing almost any physical data (e.g., location, acceleration, temperature, and much more). **Table 2.1** shows some examples of physical sensors.

Table 2.1. The main physical sensor types [64]

Type of context	Available sensors
Light	Photodiodes, color sensors, IR and UV sensors
Visual context	Various cameras
Audio	Microphones
Motion, acceleration	Mercury switches, angular sensors, accelerometers, motion detectors, magnetic fields
Location	Outdoor: Global Positioning System (GPS), Global System for Mobile Communications (GSM); Indoor: Active Badge system
Touch	Touch sensors implemented in mobile devices
Temperature	Thermometers
Physical attributes	Biosensors to measure skin resistance, blood pressure

- Virtual sensors: provide contextual information through software applications or services, for instance, we can determine an employee's location using a virtual sensor by browsing an electronic calendar, a travel-booking system, or emails, etc.. Also, the user's activity can be sensed by checking for mouse-movement and keyboard input.
- Logical sensors: use several information sources, and combine physical and virtual sensors with additional information from databases or various other sources, for example, a logical sensor can be constructed to detect an employee's current position by analyzing logins at desktop PCs and a database mapping of devices to location information.

2.6.2 Raw data retrieval layer

This layer is dedicated to the retrieval of raw context data; it is based on a set of drivers (for physical sensors) and APIs (for virtual and logical sensors). By using interfaces, sensors perceiving the same type of context become exchangeable, for instance, it becomes possible to replace an RFID (Radio Frequency Identification) system by a GPS (Global Positioning System) without any major modification in the current and upper layers.

2.6.3 Preprocessing layer

The preprocessing layer is responsible for interpreting sensed contextual information. Usually, sensors return technical data that are not appropriate to be directly used by applications, hence, this layer includes *transformation* operations (*extraction* and *quantization*) to raise the abstraction level, this can be seen when the name of the room the person is in may be more significant and used than the exact GPS coordinates.

Also, this layer includes another process called *aggregation* or *composition*, which consists in combining single sensed context atoms into meaningful high-level information. For instance, a system is capable of determining whether a client is situated indoor or outdoor by analyzing temperature and light, or whether a person is currently attending a meeting by capturing noise level and location.

2.6.4 Storage and Management layer

The fourth layer is responsible for organizing the stored data in a specific structure, Strang and Linnhoff-Popien [53] summarized the most relevant data structure approaches in the literature. This layer also allows clients to access contextual information in two different ways, *synchronous* and *asynchronous*. In the synchronous method, the client side is polling the server for context changes via remote method calls; it sends a message requesting some data and waits until it receives the server's answer. The asynchronous mode works via subscriptions; each client subscribes to specific events it is interested in, on the occurrence of one of these events, the client is simply notified. Generally, the asynchronous approach is more suitable due to rapid changes in the underlying context; the polling technique is more resource intensive as context data has to be requested quite often [60].

2.6.5 Application layer

The client is realized in the fifth layer, reactions to different events and context changes are implemented in this layer depending on the related application's nature. For example, if the light sensor of the user's mobile devices detects bad illumination, text may be displayed in higher color contrast.

2.7 Context-aware computing research areas

Context-aware computing implicates various concerns that need separate attention. Major difficulties exacerbating the development of context-aware applications are briefly discussed below [46], [65].

- **Context acquisition and management:** context data obtained from diverse sources may be uncertain, ambiguous or imperfect. In most cases, sensor error or out of date data gives rise to some uncertainty about sensed context, handling this uncertainty problem before using the context data is thus needed.
- **Context modeling:** existing models to represent context vary in many aspects such as the expressiveness, the scalability, and the ease with which real-world concepts can be captured. It is important to have a standardized context model to facilitate context interpretation, sharing, and semantic interoperability.
- **Context repository:** there exist two approaches for storing context data, *centralized* and *distributed*. The *centralized* one guarantees the integrity of context but suffers from the traditional problem of centralized data repositories like a single point of failure. A *distributed* strategy gives autonomy and allows mobility, although, in this approach, context repositories should provide a solution to merge interrelated information and enables further data retrieval.
- **Context query:** to explore stored context data across context repositories, we need a high-level mechanism to issue queries transparently to end users. Context query poses design issues such as context query language and event notification.
- **Context aggregation:** composite context is a high-level context that aggregates multiple atomic contexts. Hence, a context aggregator is needed to retrieve meta-information from the repository about the specific context providing a related composite context.

- **Context reasoning/inferring:** the context interpretation leverages reasoning/learning techniques to deduce implicit high-level context from explicit low-level one. Intelligent systems can learn the pattern of users' actions from historical sequences of context data and then use this pattern to predict next event.
- **Context discovery and delivery:** context discovery aims to locate and access context sources. Issues of context discovery include service description, advertisement and event subscription. Context delivery performs the job of delivering appropriate context to the applications; it includes registration, query and notification services.
- **Security and privacy:** context-aware systems security challenges in the form of privacy, integrity, and trust. Privacy enables protecting context resources from unauthorized entities. Integrity focuses on guaranteeing that the provided context information has not been corrupted by a third party. Trust deals with a respect for common security policy and a common goal.

Amongst the mentioned challenges, this thesis addresses the issue of context modeling, apart from the existing models of context, the lack of formalism for the taking into account of context-awareness and context handling persists.

2.8 Summary

By and large, from the previous definitions of context and for our observation of this new dimension, context-aware computing is an environment in which applications can sense and take advantage of a variety of contextual information. Context-awareness in ubiquitous environments allows not only to take account of the users' requests but also to adapt the applications' behavior according to the variations in the parameters that compose the context of execution.

Adaptation of applications can be a reality if context modeling and management were properly put in place, for this purpose, meaningful research efforts have been devoted to introducing methods for sensing, representing, and interpreting contextual information. Nevertheless, we still need to improve the handling of context, and therefore, to provide a more efficient adaptation of applications.

Chapter 3

The Model Driven Engineering

"software is a great combination between artistry and engineering"

Bill GATES

Contents

3.1	Introduction	27
3.2	The four-level architecture	27
3.3	The Model-Driven Architecture (MDA)	30
3.4	The Unified Modeling Language (UML)	33
3.5	UML extension mechanisms	42
3.6	Summary	46

3.1 Introduction

Assuredly, designing complex systems urged the software engineering community to introduce sophisticated paradigms and languages to deal with the different design difficulties. Indeed, the Model Driven Engineering (MDE) [66] has emerged as a software paradigm which relies on models to cope with the increasing complexity of software design and development.

In this vein, the Object Management Group (OMG) is famous for its running activities in diverse domains linked to modeling. The outcomes of these activities have already gained important notoriety; they serve as the foundation for a lot of forthcoming normative recommendations for products and processes [67]. The interest of this chapter is to walk the reader through some fundamental OMG's standards.

3.2 The four-level architecture

Models play a crucial role in the MDE, as in all engineering. This section clarifies basic terms including models and metamodels as used throughout this manuscript [68]-[70].

3.2.1 Basic concepts

- *A system*: a collection of parts and relationships among them organized to accomplish some purpose. It may include software, enterprises, a computer program, etc.
- *A model*: an information selectively representing some aspect of a system based on a specific set of concerns. It may represent business, software, or domain-specific aspects of a system.
- *Model-driven*: an approach which relies on models to develop a system, it supports the design, the construction, and the deployment of systems.
- *Modeling language*: any model needs to be expressed in a way that communicates system's information among involved stakeholders. Achieving this implies that the structure, terms, notations, syntax, and semantics are well defined and consistently represented.
- *Metamodel*: there is a certain circularity to models and modeling languages. The best way to express a modeling language is as a model, this is called a metamodel. A metamodel is, therefore, a model that defines a modeling language and is in turn expressed using a modeling language.

3.2.2 Why use metamodels?

If a given model is understandable by who created it, it should also be interpretable by the machine that executes it; therefore, the model must conform to a clearly defined set of rules to verify its consistency and to automate its interpretation, these rules are called the metamodel.

In other words, if modeling is a technique for designing systems using some predefined concepts, then metamodeling is a technique for defining the concepts to be used for modeling systems. Metamodeling, therefore, enables the flexibility needed to provide the indispensable means for designing applications. It is important to understand that there is no universal metamodel which can describe all systems, a metamodel is defined for a specific purpose, and so, that there exist a multitude of metamodels. **Figure 3.1** illustrates the relationships between *system*, *model*, and *metamodel*.

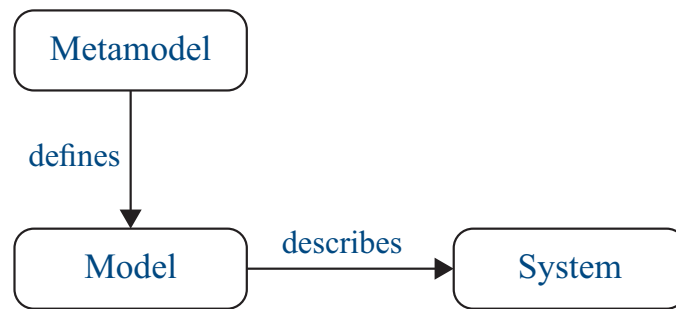


Figure 3.1. Relationships between system, model, and metamodel [71]

3.2.3 The Meta-Object Facility (MOF)

Not solely models, metamodels are in turn conform to a meta-metamodel, and so forth. To stop this infinitely recursive scheme, a meta-metamodel which defines itself is needed, such language is the Meta-Object Facility (MOF) standardized by the OMG [72], it includes basic concepts such as *Class* and *Relationship* that support the definition of many concepts including the definition concepts themselves.

The Meta Object Facility (MOF) [72], [73] represents the core of metamodeling aspects at the OMG. The UML (Unified Modeling Language) [74], for example, is definable as a meta-model based on MOF concepts.

3.2.4 The four-level architecture

The principle of metamodeling is based on the architecture given in **Figure 3.2**, in which we distinguish four modeling levels successively labeled from *M0* to *M3* [75]:

- *M0* (instances level): this level corresponds to the information of the described real-world system. It contains class instances, such as instances of people (*Amine*). This model is consistent with the *M1* model.
- *M1* (model level): includes the description of the real-world information grouped as a model, it encompasses the definition of classes, a class *Person* with a *name* attribute of *String* type in our example. This model is consistent with the *M2* metamodel level.
- *M2* (metamodel level): holds the description of the structure and semantics of the *M1* level grouped within metamodels. In our example, this level contains the definition of *Class*, *Attribute*, and *Type*.
- *M3* (meta-metamodel level): is the level describing the structure and semantics of the *M2* level grouped into a meta-metamodel. It includes the concepts of *Metaclass* and *Metarelationship* and thus can be self-defining.

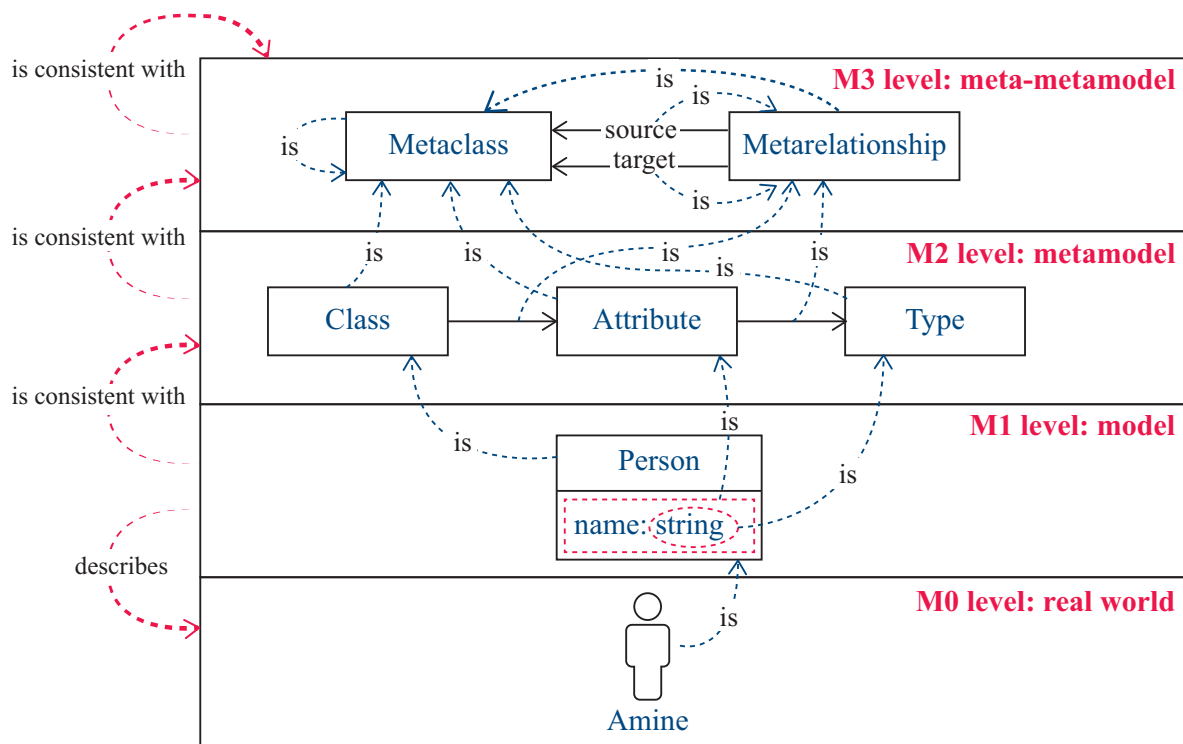


Figure 3.2. The four level architecture of MOF [75]

3.3 The Model-Driven Architecture (MDA)

The importance of model engineering in software development projects is rapidly growing; this is a reason why the OMG is now centering its activities on providing suitable solutions for software developers, they launched the Model-Driven Architecture (MDA) [70] to support the model-driven engineering of software systems.

3.3.1 Principles of the MDA

The OMG introduced the MDA approach [70] as a new paradigm to support the development of software systems. It relies on an automatic process carried out by a succession of *model transformations* to generate elements of the system from these models. **Figure 3.3** illustrates the process of model transformation.

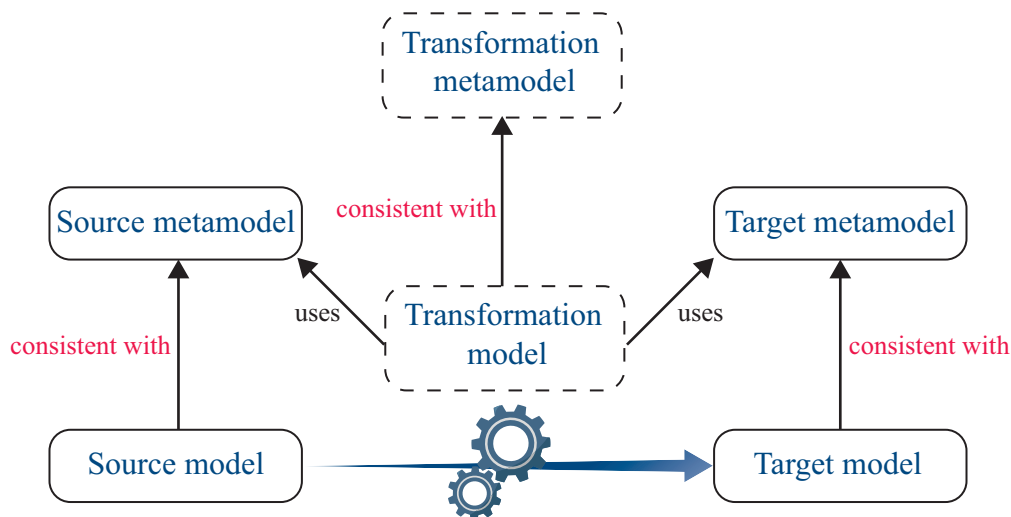


Figure 3.3. Model transformation process [71]

The MDA relies also on the notion of the *separation of concerns* between the business logic (system's functionalities) and the implementation logic (development platform constraints). This can be achieved by creating system models separately from the execution platforms, this separation automates the process of models manipulation and leads to the automatic generation of the application's code.

3.3.2 Models managed in the MDA

The types of the models manipulated in the MDA are the following [70]:

1. **Computation Independent Model (CIM):** this model shows neither the details of the system's structure nor its implementation, it just highlights information about the environment and the system's requirements. This model helps to reduce the system's complexity and gives a vision about the requirements of a particular environment.
2. **Platform Independent Model (PIM):** includes only information about the system's functionalities without implementation's technical details, consequently, a *PIM* model must be platform-independent so that it can be used later by different platforms.
3. **Platform Description Model (PDM):** information related to the execution platform are described in this model, it provides the technical concepts and the services provided by the target platform.
4. **Platform Specific Model (PSM):** this model combines the specifications of the *PIM* model and the details which specify how a system uses a particular platform. It corresponds to a phase of design and implementation.

3.3.3 MDA process

The development process in the MDA can be recapped in the following steps [70], [71]:

1. Generation of the *CIM* model: this phase corresponds to a requirement gathering phase while ignoring the structure and the functionalities of the system.
2. Construction of the *PIM* model: a phase of analysis and design, in which we use the UML to express the *PIM* model independently of any execution platform details.
3. Enriching the *PIM* model: with successive refinement, by incrementally detailing the system's behavior, while eliminating the platform related details.
4. Transformation of the *PIM* into *PSM*: choosing the appropriate execution platform, then generating the corresponding *PSM* model using specific transformation techniques. A single *PIM* can be used to provide multiple *PSM* models for different execution platforms.
5. Code generation: there are several detail levels of the *PSM* model. The first level results from the transformation of the *PIM* and the *PDM* models. The most detailed level is a generated code in a specific language such as Java, C ++, to name but a few.

Figure 3.4 summarizes the above process. Model transformations of the same type (e.g., from *CIM* to *CIM*) are intended to refine the model by completing, specializing, or filtering it. The *CIM* to *PIM* transformation is used to move from the requirement view to a preliminary functional view independently of any implementation details. The transformation of the *PIM* to *PSM* allows specializing the former model in accordance with the target execution platform based on the information provided by the *PDM* model. The transformation of the *PSM* into code allows generating the executable code of the application after compilation.

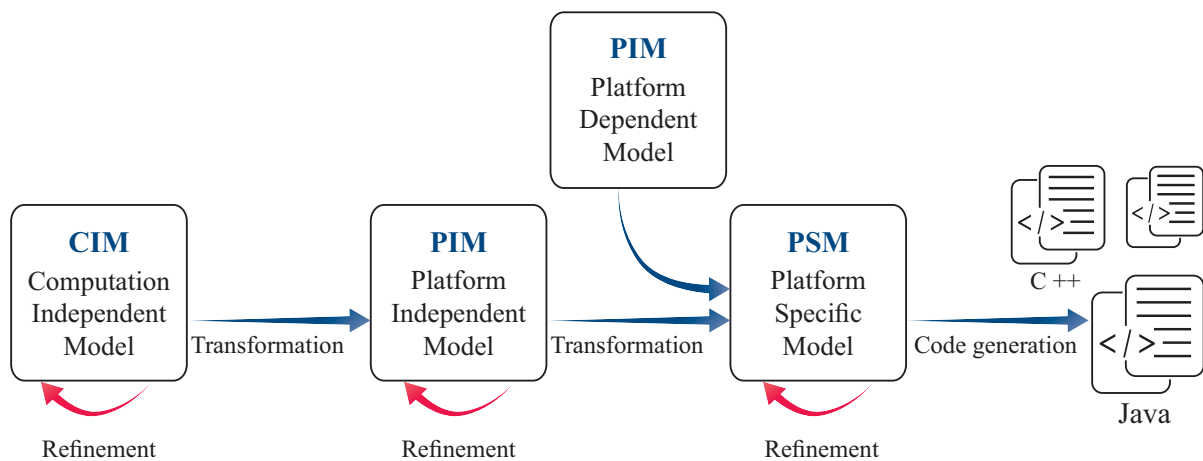


Figure 3.4. Models and transformation process in the MDA [71]

3.3.4 Model transformation techniques in MDA

Among the different model transformation techniques proposed by the OMG, we would like to explain these which we judge necessary to understand the remainder of this thesis [70], [76], [77].

3.3.4.1 The marking technique

Used to transform a *PIM* model to a *PSM* model through *annotations* (marks), these latter are extracted from the *mapping* which represents the transformation rules of the target platform. The annotations are first applied to the *PIM* elements to obtain a new model so-called *marked PIM*. Then, each rule of the *mapping* is applied on the related element of the *marked PIM* to obtain an element of the *PSM*.

3.3.4.2 The model merging technique

Four steps are needed when wanting to merge a source model (*PIM*) with another model (to be defined) to get a target one (*PSM*).

- Comparison: identifying correspondences between elements of the models to be merged.
- Conformance checking: eliminating the conflicts between the identified elements.
- Merging: applying the merging tools (graph-based algorithms, UML models, etc.).
- Reconciliation and restructuring: fixing the resulting model by cleaning all inconsistencies.

3.4 The Unified Modeling Language (UML)

Over the past few years, the Unified Modeling Language (UML) [74] has become a de-facto standard for modeling diverse systems. In this section, we are about to present the main concepts of the Class and Sequence diagrams.

3.4.1 Overview of the UML

Originally, UML was called unified as it resulted from the combination of three leading methods: Booch [78], the object-modeling technique (OMT) [79], and the object-oriented software engineering (OOSE) [80]. It started yet to incorporate a number of best practices from modeling language design, object-oriented programming, and architectural description languages.

In its latest revisions, UML has been significantly enhanced with precise definitions of its abstract syntax; this has made of it a more modular language with improved capability for modeling large-scale systems, covering all structural components and dynamic behaviors.

The UML language is defined by a two-part standard, the infrastructure which describes the UML common kernel for all the UML diagrams, whereas the superstructure specifies the detail of each diagram separately. A formal definition through a set of MOF-based metamodels specifies the abstract syntax of the UML; it defines the UML modeling concepts, their attributes, relationships, as well as the rules for combining these concepts to construct partial or complete UML models [74].

3.4.2 UML diagrams

A diagram is the graphical presentation of a set of elements which aims to visualize a system from different perspectives [81]. Following this, the UML is wealthy in notations; it allows describing various aspects of the system being modeled thanks to the variety of its offered diagrams.

As depicted in **Figure 3.5**, UML includes fourteen diagrams organized into two main groups: seven structural diagrams which describe the static structure of systems (*Class diagram*, *Deployment diagram*, *Composite Structure diagram*, *Component diagram*, *Object diagram*, *Package diagram*, and *Profile diagram*), and seven behavioral diagrams that describe the systems' dynamic behavior (*Activity diagram*, *State Machine diagram*, *Use Case diagram*, *Communication diagram*, *Interaction Overview diagram*, *Sequence diagram*, and *Timing diagram*) [82].

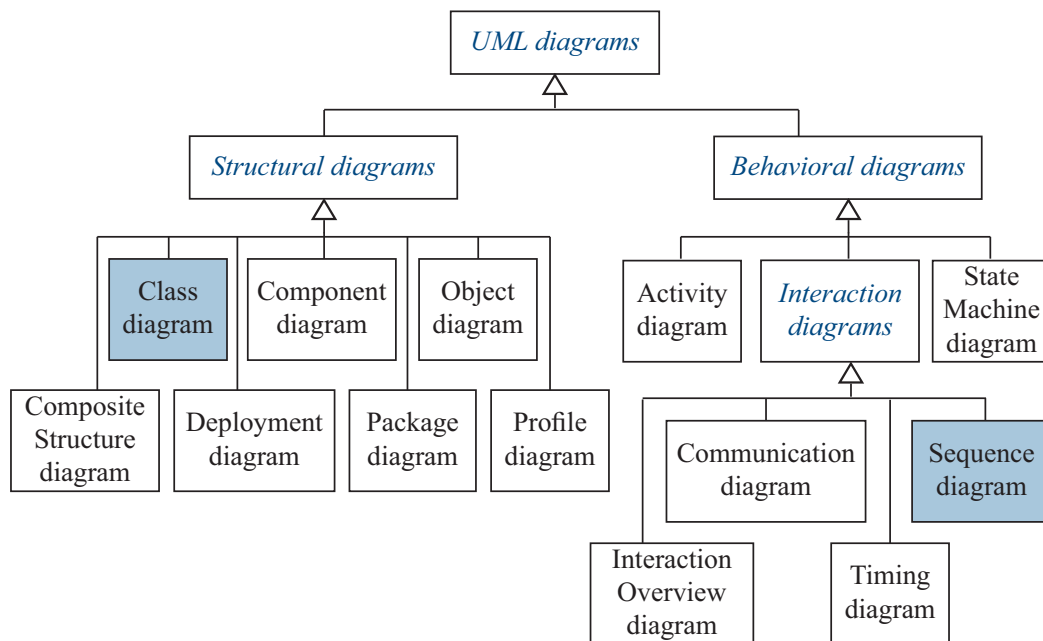


Figure 3.5. The organization of UML diagrams [71]

It is worth mentioning that reading and understanding the contribution part of this thesis requires a thorough knowledge of both the Class and Sequence diagrams.

3.4.2.1 The UML Class diagram

Certainly, Class diagrams are the mainstay of object-oriented analysis and design; they are used in a broad range of modeling concepts. Class diagrams describe the types of objects in the system and the various kinds of static relationships that exist among them.

In the following, we start by presenting the concepts that define the Class diagram through a MOF-based description of its metamodel (*M2* level); next, we outline the main graphical notations for a UML Class diagram (*M1* level).

UML Class diagram metamodel (*M2* level)

The metaclasses that constitute the Class diagram's metamodel are described below [74]:

- *Element*: is an abstract metaclass with no superclass. It is used as the common superclass for all metaclasses in the infrastructure library.
- *Classifier*: is a classification of instances that have features in common, it is a namespace whose members can include features. *Classifier* is an abstract metaclass.
- *Feature*: declares a behavioral or structural characteristic of instances of classifiers. *Feature* is an abstract metaclass.
- *StructuralFeature*: is a feature of a classifier which specifies the structure of the classifier's instances. *StructuralFeature* is an abstract metaclass.
- *BehavioralFeature*: is a feature of a classifier that specifies an aspect of the behavior of its instances. *BehavioralFeature* is an abstract metaclass.
- *Class*: describes a set of objects that share the same specifications of features, constraints, and semantics, it is a kind of classifier whose features are attributes and operations. Attributes of a class are represented by instances of *Property* that are owned by the class.
- *Property*: it is a structural feature, a property related to a class by *ownedAttribute* represents an attribute that is shared by all instances of that class. When a property is related to an association by *memberEnd* then it represents an end of the association.
- *Operation*: is a behavioral feature of a class that specifies the name, type, parameters, and constraints for invoking an associated behavior.
- *Parameter*: a specification of an argument used to pass information into or out of an invocation of a behavioral feature. It has a type and may have a multiplicity and an optional default value.

- *ParameterDirectionKind*: an enumeration type that defines literals used to specify the direction of parameters. The literals are: *in* (input), *out* (output), *inout* (both input and output), and *return*.
- *Relationship*: a concept that specifies some relationship between elements, it references one or more related elements. *Relationship* is an abstract metaclass.
- *Association*: specifies a semantic relationship between typed instances. It has at least two ends represented by properties connected to the type of the end. Its instances are called links.
- *Generalization*: or specialization, is a taxonomic relationship between a general classifier (parent) and a specific one (child), the specific classifier inherits the features of the general one.
- *AggregationKind*: a special kind of association representing a structural relationship between a whole and its parts, *AggregationKind* is an enumeration type that specifies the literals for defining the kind of aggregation of which the literals are: *none* (no aggregation), *shared* (shared aggregation), and *composite* (the property is aggregated compositely).
- *VisibilityKind*: an enumeration type that defines literals to determine the visibility of elements in a model. The literal values are: + (public), - (private), (package), and (protected).

Figure 3.6 depicts a simplified metamodel for the UML Class diagram.

UML Class diagram notations (*MI* level)

The following list aims to give a comprehensive account of the graphical notations that may be used in Class diagrams at model level (*MI*).

- *Class*: the basic unit of a Class diagram. Graphically, a class is rendered as a rectangle, divided into three compartments containing: the name of the class, its attributes, and its operations.
- *Property*: a class may have several attributes like it may have no one at all. An attribute is characterized by a name, a visibility, a type, a multiplicity, and a default value. Graphically, attributes are listed in the second compartment just below the class name's one.
- *Operation*: defines the actions that a class knows to carry out, it is characterized by a name, a visibility, a return type, and is related to a set of parameters. Graphically, operations are listed in the third compartment below the attributes.

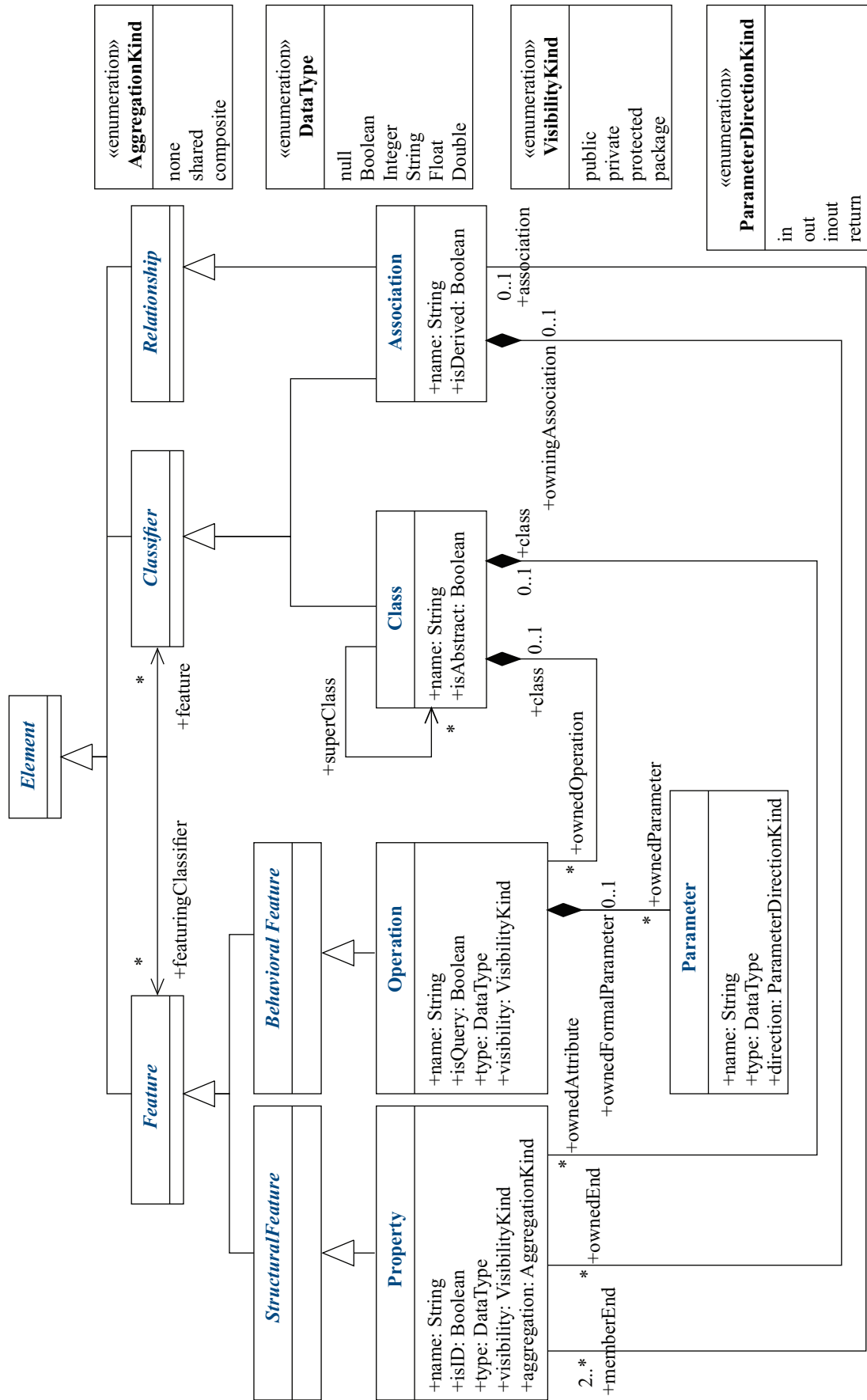


Figure 3.6. UML Class diagram metamodel [74]

- *Parameters*: operation's parameters are notated using a name, a type, a default value, and a direction which takes its value from the *ParameterDirectionKind* enumeration and that indicates the parameter's type.
- *Association*: it is rendered as a solid line between two classes, directed from the source class to the target class as it can also be bidirectional. An association has a name and can show multiplicities at both ends of the line.
- *Generalization*: rendered as a solid line with a hollow arrowhead pointing to the parent.
- *Aggregation and composition*: composite aggregation is depicted as a binary association decorated with a filled black diamond at the aggregate (whole) end, while a shared aggregation is presented as an unfilled diamond.

Table 3.1 presents the different types of relationships used to connect classes, whereas **Figure 3.7** exhibits an example of a UML Class diagram.

Table 3.1. The types of relationships in UML Class diagrams [74]

Relationship	Graphical notation
Association	————
Generalization	————>
Aggregation	————◇
Composition	————◆

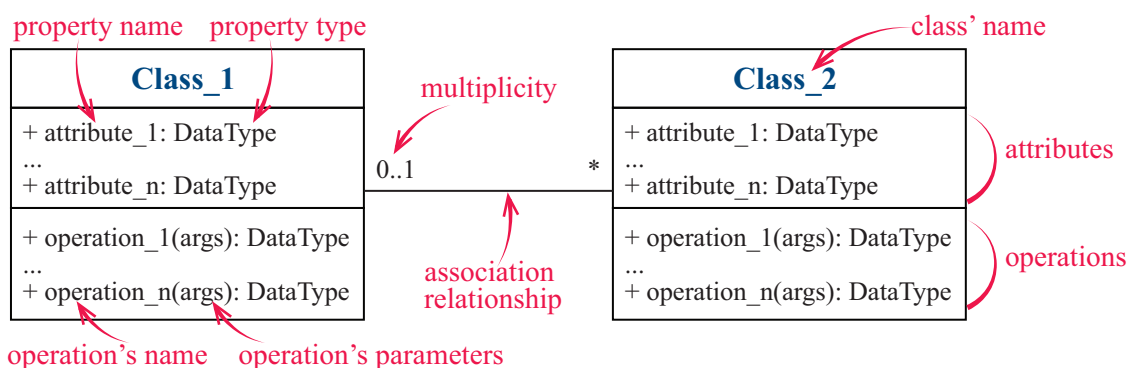


Figure 3.7. An example of a typical UML Class diagram [74]

3.4.2.2 The UML Sequence diagram

Interaction diagrams gained increasing importance when modeling systems behavior; they describe sequences of messages exchanged between objects in the order they occur. The great strength of Sequence diagrams is how clearly they can describe the collaboration behavior of several objects within a use case.

In what follows, we present the structure of the Sequence diagram by explaining the meta-classes that compose its metamodel (*M2* level), afterward, we describe the corresponding graphical notations to draw a coherent Sequence diagram (*M1* level).

UML Sequence diagram metamodel (*M2* level)

The descriptions of the metaclasses that constitute the Sequence diagram metamodel are [74]:

- *Lifeline*: this metaclass represents an individual participant in the interaction.
- *ExecutionSpecification*: or activation, it is a specification of the execution of a unit of behavior within the lifeline which shows that the participant is active in the interaction.
- *OccurrenceSpecification*: the basic semantic unit of interactions. The sequences of occurrences specified by them are the meanings of interactions.
- *Interaction*: is a unit of behavior. It comprises a set of messages exchanged among objects in some roles to accomplish a purpose.
- *InteractionFragment*: an abstract notion of the most general interaction unit. It is a piece of interaction. Each interaction fragment is conceptually like an interaction by itself.
- *CombinedFragment*: defines an expression of interaction fragments. It is specified by an interaction operator and corresponding interaction operands. The semantics of a combined fragment depend upon its interaction operator.
- *InteractionOperand*: represents one operand of the expression given by the enclosing *CombinedFragment*. An *InteractionOperand* is an interaction fragment with an optional guard expression. An *InteractionOperand* may be guarded by an *InteractionConstraint*.
- *InteractionOperatorKind*: an enumeration designating the different kinds of operators of combined fragments. The literal values of this enumeration are *alt*, *opt*, *par*, *loop*, *critical*, *neg*, *assert*, *strict*, *seq*, *ignore*, and *consider*.
- *InteractionConstraint*: is a Boolean expression that guards an interaction operand.

- *Message*: defines a particular communication between Lifelines of an Interaction. It is a specification of communication between objects that convey information with the expectation that activity will ensue.
- *MessageKind*: an enumerated type that determines the message's type whether it is *complete*, *lost*, *found*, or *unknown*.
- *MessageSort*: an enumerated type that identifies the type of communication action that was used to generate the message. The literal values of this enumeration are *synchCall*, *asynchCall*, *asynchSignal*, *createMessage*, *deleteMessage*, and *reply*.

Figure 3.8 shows an excerpt of the metamodel for the UML Sequence diagram.

UML Sequence diagram notations (*MI* level)

Below, we present the graphical elements of Sequence diagrams at model level (*MI*).

- The *Lifeline* is shown using a rectangle followed by a vertical dashed line.
- The *ExecutionSpecification* (activation) represented as a thin rectangle on the lifeline.
- The notation for a *CombinedFragment* in a Sequence diagram is a solid-outline rectangle. The operator is shown in a pentagon in the upper left corner of the rectangle. The operands of a combined fragment are shown by tiling its graph region using dashed horizontal lines to divide it into regions corresponding to the operands.
- Conditional behavior can be expressed in different ways: a condition written above the message signifies that the message is only sent if the condition is met, to show that several messages are conditionally sent under the same condition an *opt* operand is used, whereas an *alt* operand is used to indicate several alternative interactions.
- To show a loop, preceding a message with an asterisk symbol means that the message is sent repeatedly as long as the condition written above it holds, whereas using a *loop* operand means that multiple messages are sent in the same iteration.
- A message is shown as a line from the sender to the receiver. The form of the line or arrowhead reflect its kind: *stick arrowheads* show asynchronous messages, *filled arrowheads* show synchronous messages, and *dashed lines* represent reply messages. If a caller sends a synchronous message, it must wait until the message is done. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.

Figure 3.9 shows an example of a typical Sequence diagram.

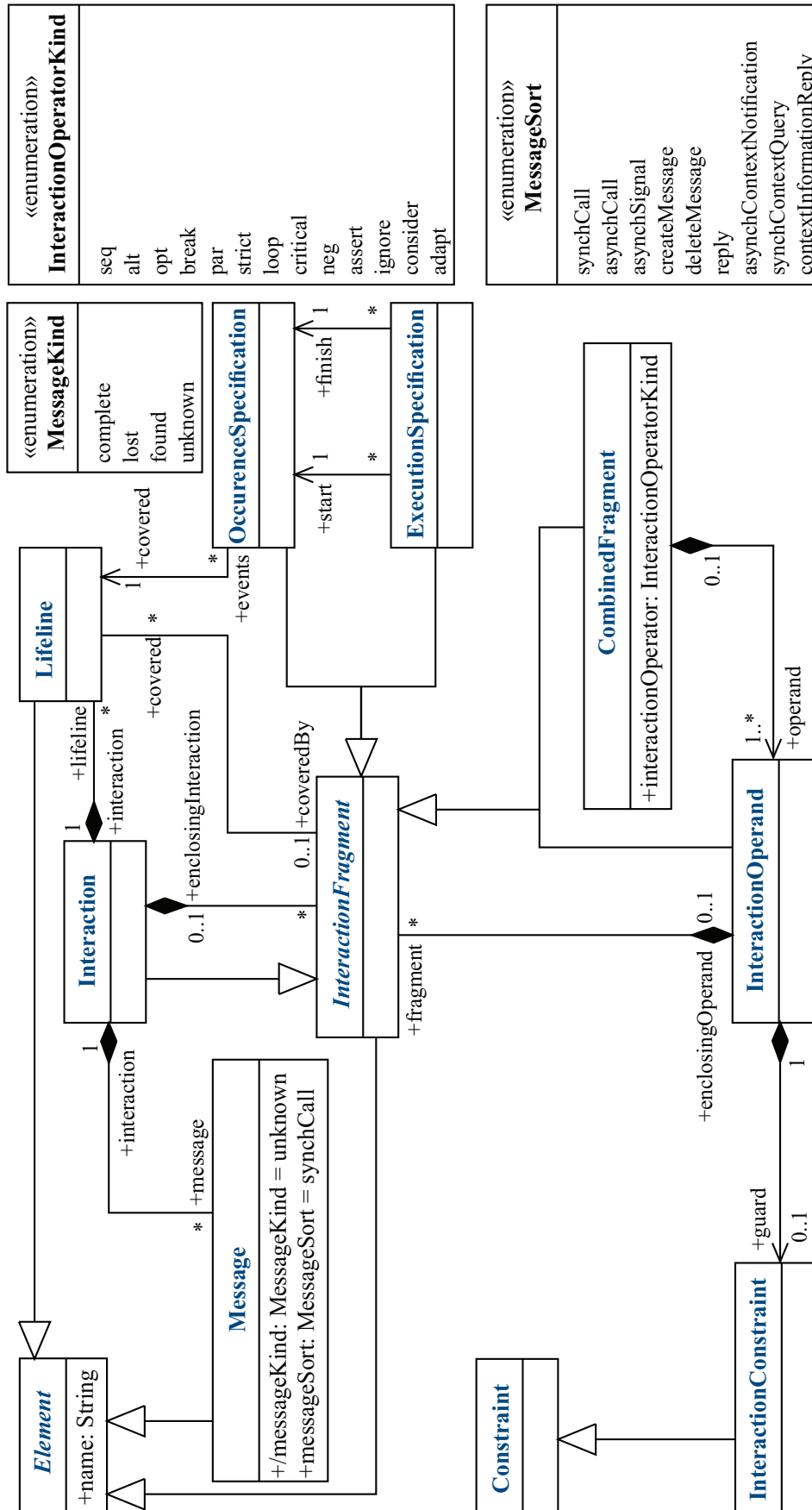


Figure 3.8. UML Sequence diagram metamodel [74]

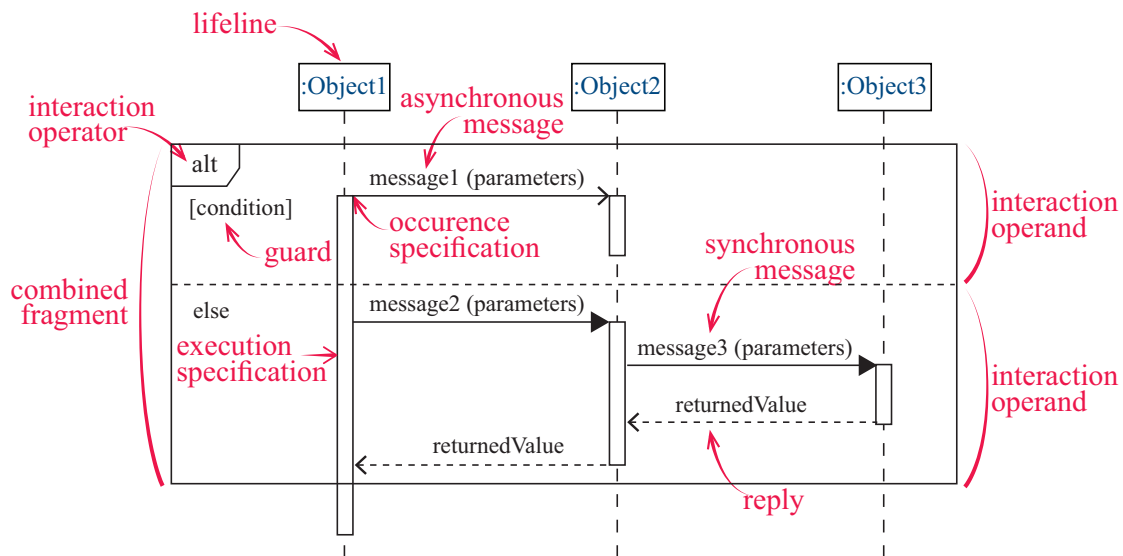


Figure 3.9. An example of a typical UML Sequence diagram [74]

3.5 UML extension mechanisms

UML diagrams aim to provide a universal language that can be used for various modeling purposes. But in spite of this, there exist some situations and fields where this language is not entirely adequate to be used in its standard form, i.e., the UML's native syntax and primitive semantics may be insufficient to express particular domain needs.

The OMG devoted plenty of efforts to enable software designers to overcome the above limitations, they put forward two possible mechanisms to extend UML, *UML Profiles* (lightweight extension) and *MOF-based metamodels* (Heavyweight extension) [83]-[85].

3.5.1 When to extend UML?

In line with Andrea Sindico [26], there are many reasons why one may want to create domain-specific customizations:

- Giving a terminology that is adapted to a particular domain.
- Giving a syntax for constructs that do not have an existing notation.
- Giving a different notation for already existing symbols.
- Adding semantics that is left unspecified in the metamodel.
- Adding semantics that do not exist in the metamodel.
- Adding constraints that restrict the way of using the metamodel.

3.5.2 Lightweight extension (UML Profile)

This mechanism allows adjusting UML to a wide range of domains through the definition of *Profiles*. In this method, a copy of the native UML metamodel is manipulated in a read-only mode; all the imported elements are thus used at the *M1* level without neither modifying their characteristics nor introducing new UML metamodel elements at the *M2* level [74], [85].

UML Profiles are based on three main concepts:

- *Stereotype*: extends an existing metaclass of UML to change its semantics, this allows to create new kinds of concepts for a particular domain. A stereotype can include *tagged values* (properties) and *constraints*. UML also allows associating a shape (icon and/or frame) to the stereotype to distinguish it graphically from the original concept.
- *Tagged value*: extends the properties of a UML stereotype. Tagged values are a way to annotate features that are additional to the extended UML concept allowing to create new information in the stereotype's specification.
- *Constraint*: extends the semantics of a UML building block permitting to add new rules or to modify existing ones. We can use the OCL (Object Constraint Language) [86] to define some rules in order to constrain the UML metamodel.

Figure 3.10 depicts an example of a UML lightweight extension.

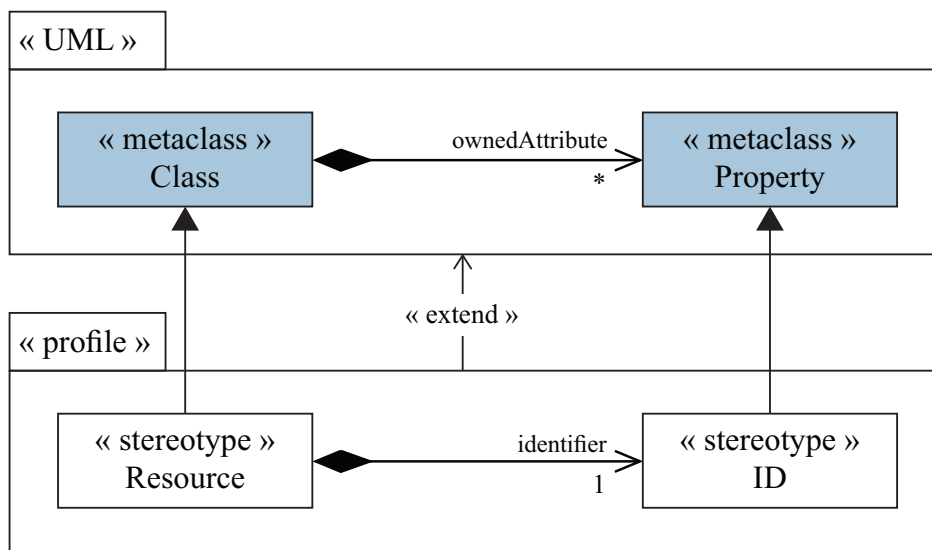


Figure 3.10. Example of UML lightweight extension [75]

3.5.3 Heavyweight extension (MOF-based metamodel)

The heavyweight extension mechanism is based on manipulating a copy of the UML's metamodel or at least some parts of it in a read-write mode. In other words, this mechanism makes possible to import elements of the UML metamodel through the merge operation and to add, delete, or modify some characteristics. All changes are thus made directly on a copy of the concerned metaclasses at the *M2* level [84].

This approach has been followed by many researchers in various domains such as [87]-[90], and also in other standards and languages including the CWM (Common Warehouse Metamodel) [91].

Figure 3.11 depicts an example of a UML heavyweight extension.

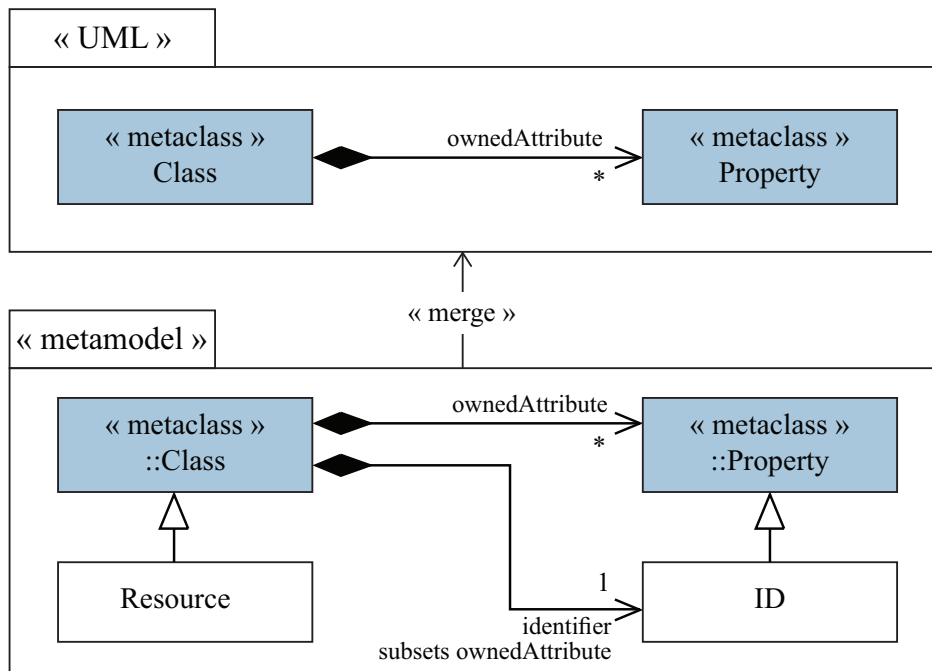


Figure 3.11. Example of UML heavyweight extension [75]

3.5.4 Lightweight vs. Heavyweight

Collectively, the lightweight extension mechanism helps to customize UML for specific needs; it lets UML adapt to the different new software technologies by including new building blocks. Despite this, it may not provide such elegant and perfectly fitting notations as may be required for particular systems [83], this is since defining Profiles does not apply a true modification in the UML metamodel. Instead, it is just a read-only import; it tailors the UML metamodel for different platforms or domains without conflict with the standard semantics [90].

The self-defined MOF-based metamodel can be as communicative as needed [92], it produces notations that will seamlessly match the concepts and nature of the target application domain. However, the downside to the heavyweight extension is that creating a tailor-made language is more laborious than constructing a UML Profile [83]. Also, the metamodel approach does not have enough supporting tool as compared with UML Profile [85], which requires the need to develop a proper editor of the new modeling language that can be exploitable by its end users [26].

Indeed, each extension alternative has its advantages and disadvantages; therefore, it may not be easy to decide when to create a new MOF-based language and when to define just a UML Profile. However, this can be answered by analyzing the domain space of the target Domain Specific Language (DSL). As can be seen in **Figure 3.12**, if there is much overlap between the concepts in UML and those within the DSL then a Profile-based solution is preferred. In contrast, if there is only a little overlap then favor the MOF-based solution [83], [84].

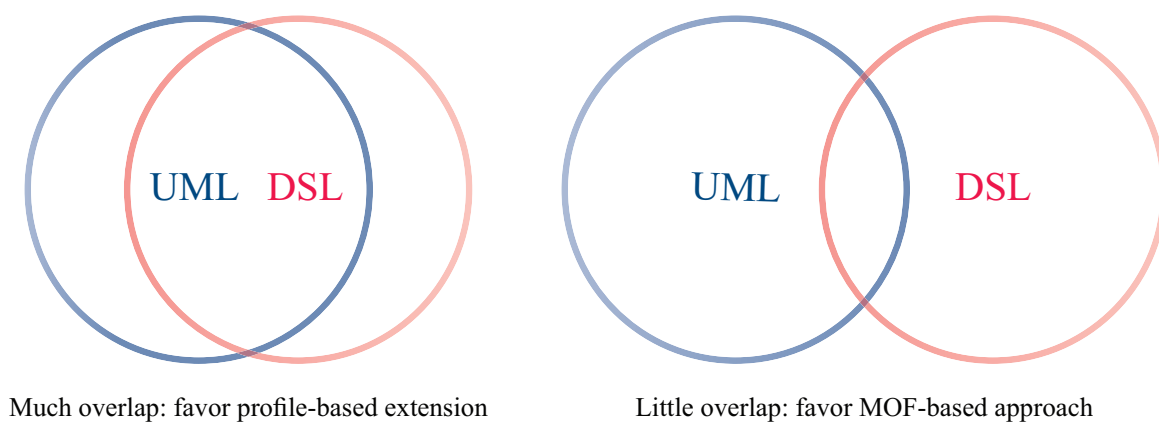


Figure 3.12. UML Lightweight vs. Heavyweight extensions mechanisms [84]

3.6 Summary

In this chapter, we highlighted some of the Object Management Group's standards and which are vital for understanding the rest of this thesis. The MDA approach is a rich software design approach for the development of software systems based on designing models and applying several transformation techniques.

Correspondingly, the Unified Modeling Language (UML) has emerged as the most popular modeling language among software engineers, it includes modeling nomenclatures for the design of complex software systems, both structurally and behaviorally, through different types of diagrams. Nevertheless, the fact that UML is a general-purpose notation may limit its suitability for modeling some specific domains, for which domain-specific languages may be more appropriate. The Object Management Group provides extension mechanisms to tackle this issue; they allow the customization of its syntax and semantics to adapt it to specific application domains.

Part II

CONTRIBUTIONS

Chapter 4

Context Handling Approaches

"As we advance in life we learn the limits of our abilities"

Henry FORD

Contents

4.1	Introduction	49
4.2	Why is context difficult to handle?	49
4.3	Handling context as a separated concern	51
4.4	Context handling approaches	52
4.5	Discussion and evaluation of the existing approaches	58
4.6	Summary	61

4.1 Introduction

By means of the study we conducted, the main challenge for context-aware computing is to transfer our real life's environment into a virtual one. In recent decades, there have been several studies focusing on enabling context-aware applications to interact with our lives in ways that are easier to manage, leading to many opportunities to succeed in tasks.

In point of fact, special software engineering approaches and techniques have been proposed to support the development of self-adaptive applications by handling the various environmental factors that surround the applications' execution.

This chapter starts by investigating the encountered issues when conceiving context-aware applications, followed by an overview of some key software engineering concepts. Afterward, we present the literature's main context handling approaches in light of which we establish a comparison by standing out the advantages and weaknesses of each approach.

4.2 Why is context difficult to handle?

In line with the literature, there exist many reasons that make the task of handling context very strenuous. Dey and Abowd outlined in [9] the following characteristics of context:

- Context must be abstracted to make sense to the application, for example, longitude and latitude coordinates should be provided as higher-level information street or building names.
- Context information is perceived from different sensors; it requires dedicated hardware (e.g., a GPS receiver) that must be programmed in order to interface the sensors with the application.
- Context is acquired from distributed sources; this may require the system to gather information throughout the environment and to calculate interpolations of position data, or similar.
- Context sensing technologies may introduce uncertainty, the process of sensing for certain properties may provide results with an associated level of probability.
- Context is dynamic; therefore, changes in the environment must be detected and consequently, applications must be able to adapt to the constantly changing context conditions.

Furthermore, Bettini et al. [93] discussed the main aspects that characterize context-aware systems and which should be taken into account when developing applications:

- **Heterogeneity:** designers manage a variety of context sources which differ in their update frequency. Sensors observe certain states of the physical world and provide fast access. Information provided by the user (profiles or preferences) is updated rarely, but context data obtained from databases or digital libraries (geographic maps) is often static.
- **Mobility:** when context-aware applications are mobile, i.e., running on a mobile device or depend on mobile sensors -which is the case of ubiquitous applications- the context information provisioning must be adaptable to the changing environment.
- **Relationships and dependencies:** there exist various relationships between types of contextual information that need to be handled to ensure correct behavior of the applications. One such relationship is dependency whereby entities may depend on other context parameters, for example, a change to the value of one parameter (network bandwidth) may impact the values of some properties (remaining battery power).
- **Timeliness:** context-aware applications may need access to past states (context histories) and future states (prognosis); therefore, timeliness is another feature of context information that needs to be handled by context models.
- **Imperfection:** due to its dynamic and heterogeneous nature, context information may be of variable quality as it may be incomplete or even incorrect. The sensed values may become useless if the physical world changes rapidly comparing to the updates frequency; therefore, designers must consider these problems to achieve appropriate adaptations.
- **Usability of modeling formalisms:** the important features of modeling formalisms are the ease with which designers can translate real-world concepts into modeling constructs.
- **Efficient context provisioning:** efficient access to context can be a burdensome requirement to meet in the presence of large models and numerous data objects.

In light of the foregoing, building context-aware applications is an inherently complex task, not only need the developers to be concerned only about the main functionality of the application, but also they have to understand which context parameters influence the application functionality, how application variants depend on these parameters, and which action should be activated under specific context conditions. Thus, building such applications requires specific engineering and run-time support [40].

4.3 Handling context as a separated concern

Complex systems are in general characterized by several heterogeneous aspects. Because the global view is very complex and tricky to understand, a system should no more be seen as an indivisible whole; instead, it is more practical to divide it into several simple subsystems, each of which represents only a single aspect of the system. This separation enables to handle each aspect independently of the others; it helps to distinctly overcome the problems encountered in every aspect by reducing design complexity and enhancing the problem understanding [94].

For many years, this strategy has attracted the software designers to organize and divide applications into a set of simpler modules that are easier to handle [95]. Since then, the separation of concerns appears in different phases of the software lifecycle, these concerns may be functional, technical, or related to any other aspect of the application's elements.

Regarding the domain of context-awareness computing, as believed by authors in [9], context is onerously considered in applications because of the lack of a common way to acquire and handle context separately. Also, Sindico [26] indicated that the taking into account of context-awareness can be very tricky because of its tendency to crosscut the other concerns of the system in which it is introduced.

Therefore, applying the separation of concerns on context-aware systems development requires a thorough study of the contextual parameters to be treated independently. Isolating the contextual constraints may bring significant support to engineers by eliminating dependencies between the system's functionalities and adaptations, increasing the designers' awareness towards context, and reducing costs and delays caused by later requirement changes.

In this regard, context parameters may be dynamic and may change during the execution, so they must be transparent for the user since they are not significant as application data. An instance of these parameters characterizes a context situation which does not modify the application data but may lead to process them in a different way [96].

In the opinion of authors in [97], the application should be designed in a context independent way, i.e., by abstracting it from the different contexts in which it will be used. In such a way, a designer should be able to identify the data which are inherently associated with the application and to distinguish them from the data which specify the context.

To achieve the above-stated goal, the boundary between application data and context data must be clearly defined; it may depend on the application domain since some data which is at the application level in one domain can be seen as context data in another domain. For example, GPS localization is part of application data in a traffic regulation system but is part of context data in a telemedicine application.

4.4 Context handling approaches

The software development lifecycle (SDLC) aims to increase systems quality and performance through successive phases such as *requirements analysis*, *modeling*, *coding*, *testing*, and *maintenance* [98]-[100].

Unlike traditional ones, developing self-adaptive applications requires enhanced skills and delicate carefulness for additional concerns related to context and context-awareness. Indeed, it is strongly needed to revisit existing development approaches and to introduce additional concepts to ensure more efficient adaptation.

Meaningful research works have been led in the field of context handling. Researchers in the field of software engineering have adopted different strategies in their attempts to handle context efficiently. The literature shows several publications which addressed the issue in various ways; by introducing context in different phases, and by using multiple techniques and mechanisms. The approaches of interest are mainly the *Context Toolkit* [61], the *MDA-based* approaches [76], and the *UML Profile-based* approaches [101].

4.4.1 The Context Toolkit

Dey et al. built the *Context Toolkit* [61] to ease the deployment and to support the rapid development of context-aware applications. They used the separation of concerns to isolate the context acquisition from the use of context in applications. Applications can thus use context without worrying about the details of sensors.

The Context Toolkit was inspired from the GUI tools (Graphical User Interface) which ensure the mediation between applications and users, analogously, the mediation in the Context Toolkit is performed between the application and the collected contextual information through the following components: *Context-Widgets*, *Context-Interpreters*, and *Context-Aggregators*.

- Context-Widgets are software components that provide applications with access to context information and reusable elements for context detection while hiding the details of context sensing.
- Context-Interpreters are used by Context-Widgets to abstract or interpret low-level context into higher-level information and communicate it to Context-Aggregators, e.g., a Context-Widget provides location information as latitude and longitude, but the application requires this information in the form of a street name; also information about identity, location, and sound level can be used to interpret whether a meeting is taking place.
- Context-Aggregators concatenate Context-Widgets data into an abstraction unit and provide applications with simplified contextual information. A Context-Aggregator is therefore similar to a Context-Widget, the difference between the two is that the former gathers multiple elements of the context (from multiple Context-Widgets). Aggregators provide an additional separation of concerns between how context is acquired and how it is used.

Figure 4.1 represents the Context Toolkit architecture.

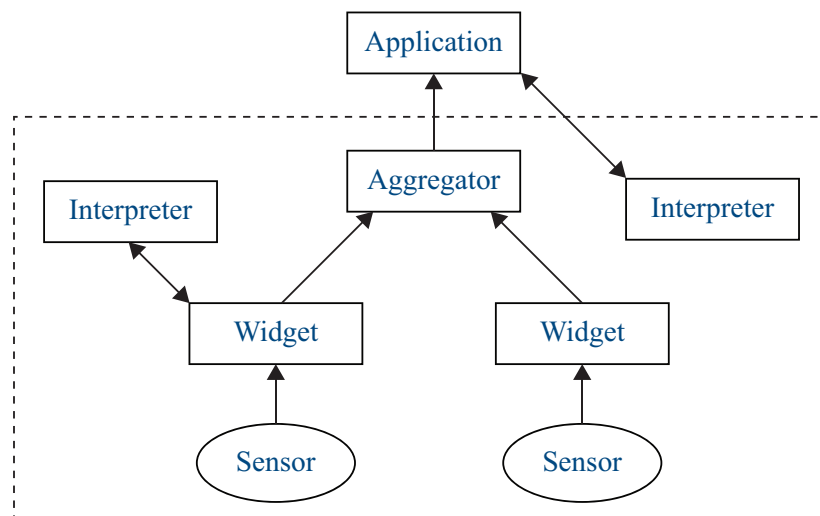


Figure 4.1. The Context Toolkit architecture [61]

Alongside that, Chen and Kotz [33] presented an interesting study which consists of capturing the different contextual information to be treated and modeled (location model, data structures). The authors proposed two types of approaches, one based on a centralized architecture using a centralized context server that provides the contextual information to the applications, and the other is based on a distributed architecture in which the required contextual information is hosted on multiple distributed sites.

4.4.2 Handling context using the MDA

Authors in [76] developed a new technique to handle context using the MDA approach. As detailed in Section 3.3.1, the MDA is based on model transformations and the separation of concerns (business and technical), but the notion of context is not explained and taken in charge by the original lifecycle of the MDA [76].

For it, the authors relied on the MDA approach while resorting to the separation of concerns to isolate the contextual constraints from the business and the technical aspects; they justified their choice with the constant changes of the surrounding objects which can directly affect the state of the current situation of a user (time, location, etc.), and as a consequence, changes in one element of context may involve the transition of this context toward another state of the user's current situation. Because of these changes, it is not reasonable to remake the whole application development process, but only the part that deals with such constraints.

Authors consider introducing the contextual information in the cycle of life of the MDA during the *PIM-PSM* transformation through the marking and model merging techniques.

4.4.2.1 The marking technique

By using the marking technique (see Section 3.3.4.1), the set of the *mapping* rules of the target platform can be enriched with contextual information, it means that this *mapping* will not include the platform rules solely, but it will nourish itself of another source which is the context.

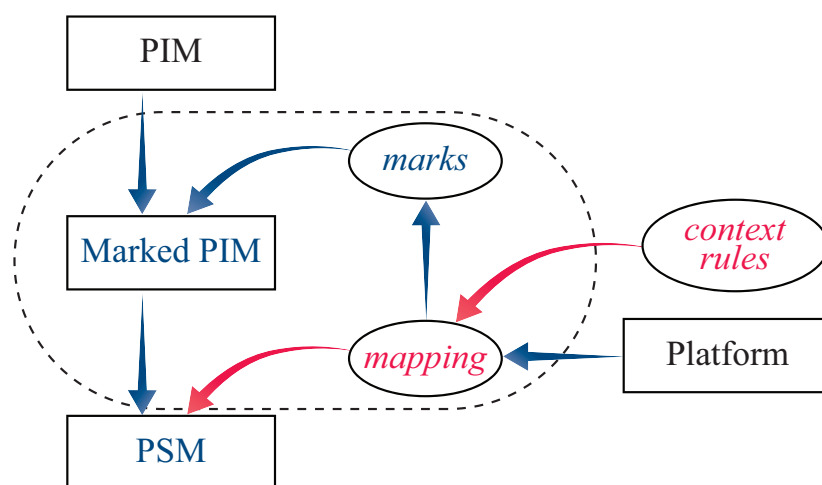


Figure 4.2. MDA architecture based on the marking technique [76]

As illustrated in **Figure 4.2**, from the mapping, the authors used the *marks* (or annotations) to mark the elements of the *PIM* model in order to get a *marked PIM*. Next, they applied the rules of the mapping (platform and context) of the resulting *marked PIM* to obtain a *PSM* model. The latter encompasses all technical specifications of the platform as well as the contextual specifications and system behaviors.

4.4.2.2 The merging technique

Authors used the MDA's merging technique (see Section 3.3.4.2) to merge the *PIM* model with a novel model containing all the system's contextual information; therefore, they consider adding a preliminary step of context modeling which will result in a contextual model (*CM*), in the latter, all contextual constraints are formalized. The following step consists in applying the merging of the resulting *CM* model with the *PIM* model to build of the target *PSM* model.

Figure 4.3 illustrates how to handle context with the MDA's merging technique.

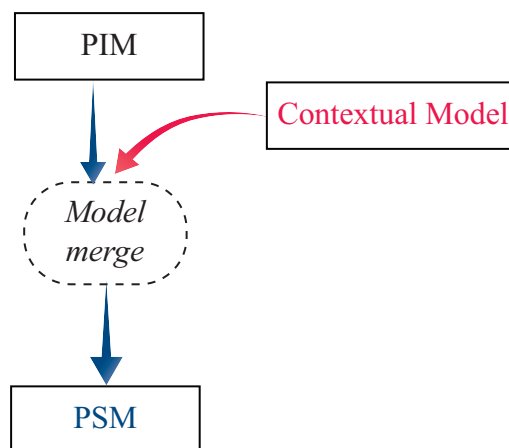


Figure 4.3. MDA architecture based on the merging technique [76]

In the literature, other works moved toward the use of the MDA approach for building attractive, efficient and adaptive applications. Ou et al. [102] outlined the value of the MDA in the development of context-aware applications, they conceived a contextual model using ontologies, then they propose an MDIA architecture (Model Driven Integration Architecture) dedicated to the implementation of context-aware applications. Ceri et al. [103] applied the MDA approach for developing context-aware web applications and focus on context adaptation actions and context data representation and management. Vale and Hammoudi developed in [104] a Context-aware Service Oriented Architecture (CSOA) using MDA principles and relied on EDOC-ECA viewpoints.

4.4.3 Handling context using UML Profiles

A quite recent work of Benselim and Seridi-Bouchelaghem [101] proposed a lightweight extension of UML to support the modeling of context-aware applications including the contextual aspects that characterize the user's current situation. On the report of the authors, this situation may be influenced by several factors and constraints related to the *user* himself, to the *application*, or to the *environment* [39]; using the new vision of concerns separation, authors isolated these contextual elements from the global aspects of the system.

The proposed UML Profile was conducted by introducing a set of *stereotypes*, *tagged values*, and *constraints*. *Stereotypes* allow to define new meanings for existing UML concepts, *constraints* specify restrictions of semantics for each new element, while *tagged values* are attached to a stereotype to indicate its attributes. In order to take all the contextual components into account, the authors started by defining the stereotypes *ContextClass* and *ContextAssociation* which are obtained by extending the UML metaclasses *Class* and *Association* respectively.

The stereotype *ContextClass* aims to represent the context of a system and is composed of a set of elements called *ContextElement*. **Figure 4.4** shows some stereotypes that are associated with the *ContextElement* stereotype and that define all constraints related to context, e.g., the type of a contextual element is presented by the stereotype *ContextType*, and is provided by one of the following entities: the user, the application, the environment, or the behavior.

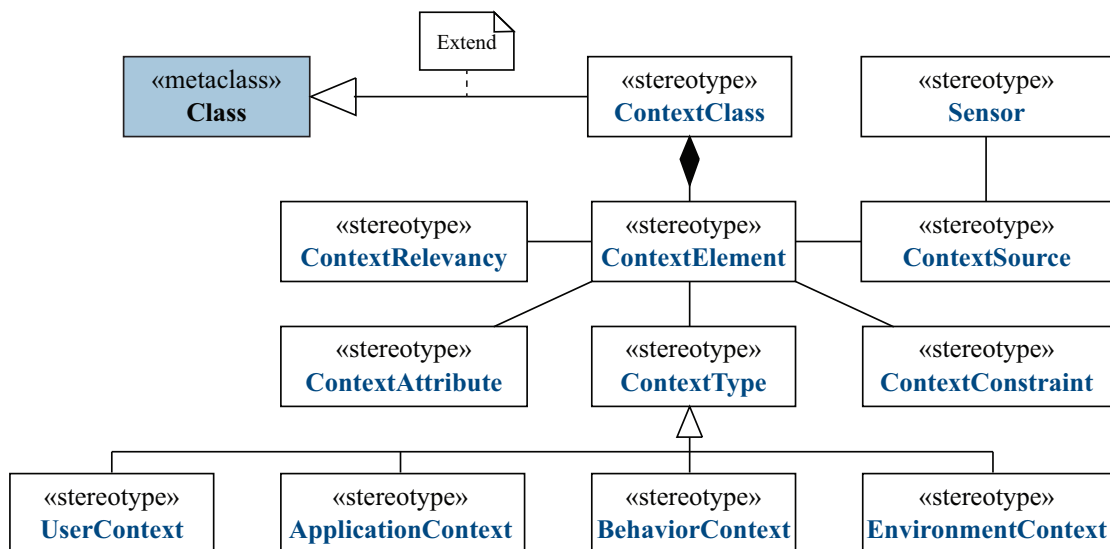


Figure 4.4. The Class stereotypes of the proposed Profile [101]

Also, the stereotype *ContextAssociation* can be specialized in order to get specific relationships such as *CtxtAssociation*, *CtxtGeneralization* or *CtxtComposition*. These association stereotypes represent respectively a simple association, a generalization, or a composition relationship between two *Class* stereotypes (see **Figure 4.5**).

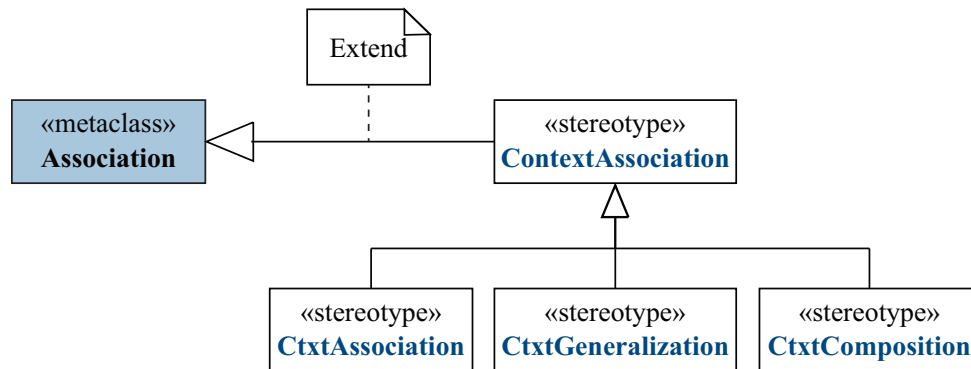


Figure 4.5. The Association stereotypes of the proposed Profile [101]

Constraints are used to extend the UML semantics by providing conditions used as restrictions during the modeling process; they can be expressed by using natural language or using OCL (Object Constraint Language)[86]. Lastly, tagged values are attached to stereotypes and used to specify attributes for them.

Another great deal of research works used the UML lightweight extension mechanism to handle context, Simons [105] proposed the Context Modeling Profile (CMP), which is a UML lightweight extension for context models in mobile distributed systems which can be integrated into many UML modeling tools. A case study of meeting system is utilized to illustrate the approach. Sindico and Grassi [106] defined a UML extension called CAMEL (Context-Awareness ModELing Language), which is a domain-specific modeling language. CAMEL can be used to enrich a UML model of an application with elements related to context and context-dependent behavior, enabling software engineers to handle context-awareness concerns.

Al-alshuhai and Siewe [107]-[109] introduced new annotations at model level (*MI*) to make UML diagrams context sensitive. The context-aware use case diagram [107] specifies the context information upon which the system's functions depend, the context-aware activity diagram [108] enables the representation of context objects, context constraints and adaptation activities, while the context-aware class diagram [109] enables the representation of context information that affect the behaviors of a class.

4.5 Discussion and evaluation of the existing approaches

The first interesting work to handle context was the Context Toolkit, it presents an unavoidable way for the understanding of adaptation principles. The Context Toolkit can serve as support for finished applications by adapting them to the context through three main steps: capturing context, interpreting it, and providing it to the application [76].

Next, another innovative work we investigated is the MDA-based approach in which authors employed model transformation techniques to embed context in applications since the phase of development. Lastly, we examined UML Profile-based approaches in which contextual information are included since the modeling phase by resorting to the UML lightweight extension mechanism.

When analyzing the studied approaches regarding the SDLC phases in which the notion of context was introduced, one may notice that the trend heads toward handling context in earlier phases: first applied on finished applications (Context Toolkit), then during the development (MDA-based approaches), and lastly at the modeling phase (Profile-based approaches). This stems from the fact that context is like any other software requirement, the earlier we handle it, the best we save costs, efforts, and delays [99], [100], [110]. **Figure 4.6** shows the timeline of the studied context handling approaches as well as their distribution with regard to the phases of context embedding.

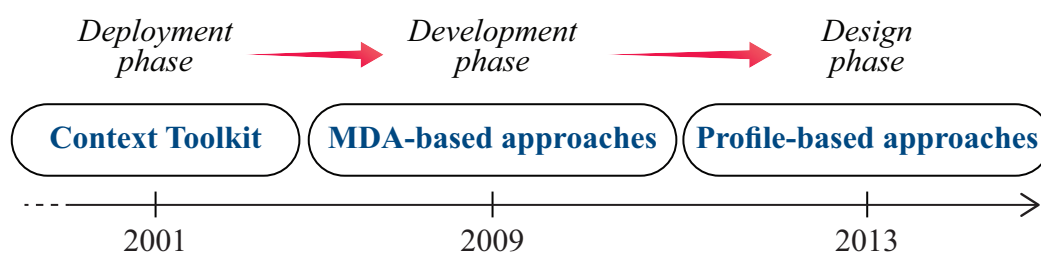


Figure 4.6. The timeline of the studied context handling approaches

The Context Toolkit introduces the concept of abstraction clearly, *Widgets* allow simple access to sensors, *Interpreters* give meaning to the captured context, while *Aggregators* serve to link Widgets to the application. In spite of that, the data structure used in this approach is based on $\langle key - value \rangle$ models which limits the expressivity especially for complex systems [53]. Also, following Rey and Coutaz [111], the infrastructure of the Context Toolkit does

not explicitly support metadata to inform the application on the quality and/or accuracy of the provided information. Furthermore, as the same drawback of the majority of initial adaptation works, the Context Toolkit considers the applications as a finished product and tries to adapt their execution to the captured context information [76], this has led to the need of introducing the context in an earlier development phase.

Approaches presented in [76] and [112] distinguish themselves by introducing the context during the development phase and not after. The authors proposed a new vision of the Model-Driven Architecture and demonstrated how it is possible to isolate the contextual constraints from the business and the technological aspects; this separation allows managing the context in a separate branch without restarting the entire development process even if the contextual constraints change. Nevertheless, the authors themselves were not completely satisfied with their proposal given that the UML concepts used in the Context Model (*CM*) are not sufficient to describe all contextual constraints accurately [101]; on the other side, MDA-based approaches can be criticized as they require its practitioners to be very experienced in model transformation techniques [113], [114]. Hence, that points to the need for a universal language, capable of representing the contextual information independently of development technologies starting from the modeling phase.

The UML lightweight extension presented in [101] came to help handling context since the modeling stage by relying on the separation of concerns to isolate the context from the global aspects of the system. The authors proposed a set of stereotypes to model the contextual elements; also, they attached some tagged values and constraints to the proposed stereotypes. Nonetheless, defining Profiles does not apply a true modification in the UML metamodel, it just adapts the UML concepts for different domains [74], [84], in other words, a Profile is not a new element, its expressiveness is consequently constrained by the model element it specializes [115]. Thereupon, it seems lacking to create stereotypes like *ContextClass* and to refer it to the UML metaclass *Class*, in such manner, objects would unavoidably have the same properties and relationships defined in the traditional metaclass *Class* (attributes and operations), whereas the constructs to be managed define additional features [114]. Another key limitation of this approach is that lightweight extension mechanisms do not offer the possibility to specify behavior neither do they for modifying existing structures [84].

Hence, we strongly believe that the contextual features must be included in the UML meta-model as first-class entities and not as stereotypes. UML heavyweight extension offers the possibility to introduce totally new constructs in the metamodel level ($M2$), and new corresponding graphical notations in the model level ($M1$), this allows to define new concepts, structures, behavior descriptions, and relationships that may help to overcome related works limitations and to cover the adaptive aspect accurately.

Table 4.1 summarizes the studied context handling approach, by addressing their advantages and their limitations.

Table 4.1. Summary of the studied context handling approach

Approach	Context Toolkit	MDA-based	UML Profile-based
<i>Phase of introducing context</i>	Deployment phase	Development phase	Design phase
<i>The used techniques / concepts</i>	Widgets, Interpreters, and Aggregators	PIM, PSM, PDM models, MDA's marking and merging techniques	UML Profiles, stereotypes, constraints, and tagged values
<i>Expressiveness level of the context model</i>	<key - value> model is clear and simple but not suitable for complex systems	The standard UML concepts (used in the Context Model CM) do not support all aspects of the context adequately	Its expressiveness is constrained by the model element it specializes (native class and association metaclasses)
<i>Main advantage</i>	eases the deployment and to support the rapid development of context-aware systems	adds context at the development phase avoids restarting the development process because of context changes	helps modeling context-aware applications with specific notation since the system design stage
<i>Main limitations</i>	introduces the notion of context at late stages of development process	requires the developers to be experienced in development processes and technologies like model transformation	Profiles do not offer the possibility to specify behavior neither do they for modifying existing structures

4.6 Summary

On the whole, handling context and adaptation assumptions leads to further challenges and requirements. The literature shows a variety of approaches which aim to introduce the notion of context to cope with the perpetual context variations, and thus, to adapt the application's behavior correspondingly.

Analyzing the presented studies helped us raising two important notes, the first one regards the software development lifecycle phases in which context has been introduced, we found that application developers are tending to anticipate context handling since earlier design phases. The second note addresses the way in which context has been embedded into the development lifecycle; it shows the limitations of the related approaches as well as the necessity to define novel concepts to specify context-awareness requirements appropriately.

Chapter 5

Overview of the Context Unified Modeling Language

"The ultimate task of the architect is to dream, otherwise nothing happens"

Oscar NIEMEYER

Contents

5.1	Introduction	63
5.2	Motivations	63
5.3	Overview of the Context Unified Modelig Language	64
5.4	Our vision of context	65
5.5	Implementation tools	67
5.6	Summary	68

5.1 Introduction

Analyzing various context handling approaches has led us to find out some interesting issues to be broached. On that account, our current research focuses on putting forward a novel approach to ease the design of self-adaptive applications.

The objective of this chapter is to shed light on our contributions; it starts by bringing to the fore the factors that motivated us to propose new concepts to support context-aware systems design. Afterward, we introduce the Context Unified Modeling Language (*CUML*) [114] as heavyweight extensions for the UML Class and Sequence diagrams. Finally, we present the tools that we used to operationalize our proposal into concrete modeling editors.

5.2 Motivations

Increasingly, software engineering needs to accommodate and evolve with the emerging technologies and the related features that they can afford. Notwithstanding, the way application programmers can effectively manage and use contextual information is still a challenge, software designers thus devoted and still are investing considerable efforts to provide advanced methods for describing context-aware systems appropriately [116], [117]. On these grounds, we summed up the previous chapter with two inspiring factors:

1. In which software development lifecycle phase it is more advantageous to embed context?
Regarding this issue, we found that application developers are tending to anticipate context handling since earlier design phases. Admittedly, it is worth considering the context since the phases of requirements analysis and design by documenting the structure and simulating the behavior of context-aware applications before the implementation phase.
2. What is the most suitable strategy to introduce the context in applications development?
At this juncture, we outlined the complications of solutions requiring developers to manipulate laborious model transformation methods as well as the lack of expressiveness of other approaches. This has led us to think about the necessity to conceive a Domain-Specific Modeling Language of which the concepts are well-fitting to express how do applications interact with context sources, and how are they affected by the perpetual context changes.

From this perspective, our purpose is to provide universal tools capable of coping with the context-awareness concerns since earlier design phases and independently of any development processes techniques. More precisely, we aim at exploiting the UML's heavyweight extension mechanism to enrich its diagrams (*M1* level) with new specialized means and primitives tailored for monitoring how applications are linked to context parameters and how the values of these parameters may affect the application behavior.

5.3 Overview of the Context Unified Modelig Language

The UML's heavyweight extension mechanism has always been a solution for software engineering experts to create Domain-Specific Modeling Languages by importing native UML elements in aggregation with new concepts to complement native ones' limitations [87]-[91].

In the following chapters of this thesis we will present heavyweight extensions for the UML Class and Sequence diagrams that we named *ContextClass* and *ContextSequence* diagrams respectively. These extensions aim to support the modeling of context-aware systems structure and behavior and are actually an ongoing work toward the realization of a complete framework so-called Context Unified Modeling Language (CUML) [114].

Figure 5.1 depicts the overall scheme of the Context Unified Model diagrams.

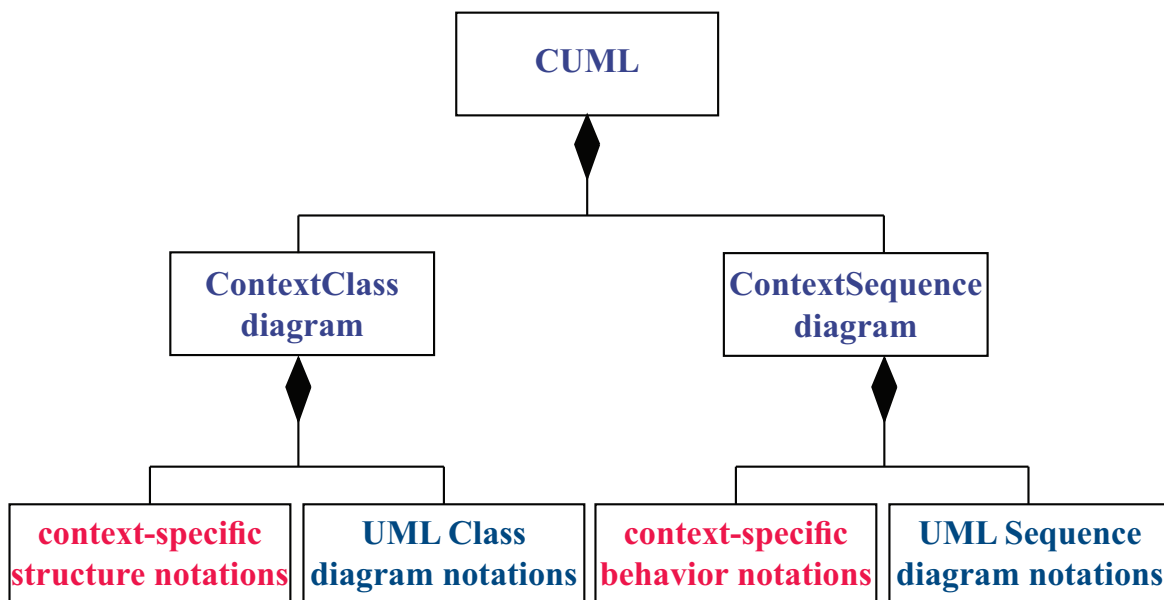


Figure 5.1. Overview of the Context Unified Modeling Language diagrams

As an extension of the UML Class diagram, the *ContextClass* diagram is set to map out the structure of context-aware applications not only by describing objects, attributes, and operations; but also by specifying relationships between the application itself and the contextual parameters that may alter its execution.

From a behavioral point of view, our solution builds on the *ContextSequence* diagram. The latter extends the UML Sequence diagram to capture the behavior of context-aware systems by depicting the flow of events including context-dependent adaptation actions and interactions with context sources.

In furtherance of achieving the purposed extensions correctly, we will build on the following methodology to extend each diagram:

- First, we start by digging into the limitations that may prevent UML's standard notations from describing the context-awareness concerns adequately.
- Second, we explain the customization processes and what diagram notations have been introduced at the model level (*M1*).
- Next, to ensure the syntactic correctness of future models, we elaborate corresponding MOF-based schemes at the metamodels (*M2*).
- At last, we bolster the proposed modeling concepts by providing hierarchical and graphical modeling editors for each diagram, this will ease the understanding and the academic use of the approach on the one hand, and on the other hand, to allow sharing real models between software designers and developers.

5.4 Our vision of context

Context-aware systems were originally founded to monitor the context and then act accordingly without any human mediation. The aim of having an autonomous system is to reduce the user intervention, easing its use and decreasing user distraction [58]. Considering the definitions discussed in Chapter 2, we may conclude that the term *context* is almost synonymous to the implicit parameters that may drag a change in the conditions of execution, and therefore, may alter the application's behavior (**Figure 5.2**).

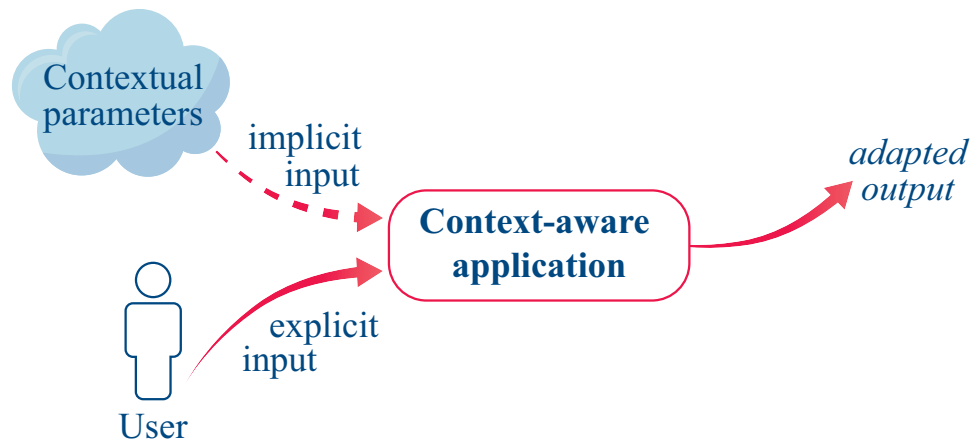


Figure 5.2. Context and context-aware applications

In this section, we will put the basic stones of our work by defining the parameters that we take under consideration in context-aware applications design.

1. **Location:** refers to the user's location in a particular situation. Due to the mobility of users, the application must adjust its behavior by customizing services related to language, currency, units of measure, etc.
2. **Time:** this parameter includes time, date, seasons, etc.. The application can therefore provide several indications of reminders, schedules, and deadlines.
3. **Environment:** this parameter encompasses elements related to the state of the execution environment such as light, sound, temperature, and atmospheric pressure. Significant changes in one of these elements can modify the application execution accordingly.
4. **Device:** the state of computing devices (computing and memory capacity, battery status, etc.). Users might access their applications from wireless portables, via stationary embedded devices, or from traditional workstations connected to a local area network.
5. **Network:** it includes several network parameters such as connectivity, bandwidth, signal level, network protocols, etc.
6. **Nearby persons:** people who are in the user's area may be considered as relevant contextual input, it may be in the form of individuals, colleagues, or neighbors.
7. **Surrounding objects:** close objects can also influence the application's behavior, for example, alerting the user when he starts walking away from his wallet.
8. **User:** the application can perform dynamic adaptations by considering the state of the user himself as well as his activity: walking, running, cycling, driving, etc.

9. **Profile:** the application can refine its behavior by taking advantage of information related to the user's age, gender, and occupation in order to furnish relevant services adaptations.
10. **Preferences:** context-aware applications can adapt their behavior depending on users preferences which may encompass default display preferences or priorities for specific operating modes.
11. **History:** the application can personalize the suggested services according to a user's history, for example, it may recommend services in accordance with the most visited places, comments, shares, polls, and ratings.

5.5 Implementation tools

To put our modeling proposal into practice, and to overcome the lack of tools in heavy-weight extension approaches [26], [85], we used specific metamodeling and drawing tools to elaborate proper hierarchical and graphical editors.

5.5.1 Eclipse Ecore tools

For both diagram extensions, we will present implementations of their metamodels under the Eclipse Modeling Framework [118]. The EMF defines a metamodeling environment and code generation facility for building application tools based on a structured data model.

When working with EMF, we start by drawing then validating our metamodel through a proprietary language called Ecore, the latter includes the necessary constructs to draw a coherent MOF-based metamodel (*EClass*, *EAttribute*, *EOperation*, *ESuperType*, *EReference*, *EEnumeration*, etc.). After that, we proceed to generate the corresponding hierarchical modeling editor which can hereafter be integrated into Eclipse as a new EMF model creation plugin. More implementation details can be found in **Appendix A**.

5.5.2 The UMLet tool

UMLet is a free, open-source, and multiplatform UML tool with a simple user interface. It enables to draw different UML diagrams, and also to share them using Eclipse. It is based on text-formatting codes to modify the basic shapes with decorations and annotations. The main interface of UMLet is straightforward, it is composed mainly of three panels: a *property* panel, a *source code* panel, and a *preview* panel.

One of the greatest features of UMLet is the possibility to customize it by introducing not only new graphical notations but also new diagram types. This can be done by directly editing the text in the property panel or from the source code panel by modifying the code that is responsible for interpreting the property text and for drawing the element. The preview panel allows displaying the changes results in real-time [119]-[121]. Further explanations can be found in **Appendix B**.

5.6 Summary

All things considered, the main concern of this chapter was to present a brief overview of the upcoming thesis' contributions. We started by revealing the design challenges that motivated us to use the UML heavyweight extension mechanism as a solution to overcome existing context handling approaches limitations. Next, we introduced the Context Unified Modeling Language (*CUML*) as a novel approach to support the design of self-adaptive systems. Afterward, we explained our vision regarding the contextual parameters that may affect the application's adaptation behavior. Eventually, we found it would be worth presenting some tools that we used to implement our modeling proposal into concrete editors allowing to explore its expressiveness in a visual manner.

Chapter 6

Extending the Class Diagram to Model Context-Aware Systems: the ContextClass Diagram

*"We can't solve problems by using the same kind of thinking
we used when we created them"*

Albert EINSTEIN

Contents

6.1	Introduction	70
6.2	Limitations of the traditional UML Class diagram	70
6.3	Extension of the UML Class diagram: the ContextClass diagram	71
6.4	ContextClass diagram metamodel	73
6.5	Ecore implementation: ContextClass hierarchical editor	76
6.6	ContextClass diagram graphical editor	78
6.7	Summary	79

6.1 Introduction

In pursuance of conceiving context-aware applications appropriately, software designers must devote more attention for describing applications dependencies to the contextual parameters that surround the application execution. Indeed, as the most popular static diagram amidst software engineers, the UML Class diagram is quite sufficient for describing the structure of traditional computing systems, but seem limited in the case of context-aware ones.

To overcome the latter, this chapter presents a heavyweight extension of the UML Class diagram to ease the modeling of context-aware systems' structure. We start by recalling the necessity of introducing novel modeling notations; then we define the concepts of the ContextClass diagram as well as its corresponding modeling tools.

6.2 Limitations of the traditional UML Class diagram

As the backbone for the design of any system's structure, the UML Class diagram is a modeling standard used to identify the static aspect of systems by mapping their entities as classes and connecting them via different relationships.

In traditional systems, entities are characterized by several structural and behavioral aspects which are respectively presented as attributes (*name*, *age*, etc.) and operations (*getName()*, *submit()*, etc.). Typically, the traditional Class diagram has powerful concepts to model such aspects by using classes, attributes, operations, and relationships between classes.

On the flip side, entities evolving in context-aware systems are not solely characterized with structural and behavioral features, but they are also affected by a couple of contextual parameters including location, time, environment, etc.. This constitutes a new aspect which in turn needs be considered for the sake of ensuring better control of the whole system, and consequently to provide valuable adaptation services [114].

As emphasized by authors in [38], most of the existing models fail to represent dependencies between the diverse context information and to utilize these dependency relationships. Also, Bettini et al. [93] showed that there exist various relationships between types of contextual parameters that need to be handled to ensure correct behavior of the applications, one such relationship is dependency whereby entities may depend on other context parameters.

Geihs and Wagner [40] asserted that designers must understand which context parameters influence the application functionality and how application variants depend on these parameters.

As an illustration, given a tourist who uses a smart application to seek a reservation in the closest hotel to his current position, this functionality doesn't depend only on explicit entries provided by the user himself (e.g., price, room type), but also on the user's location as an implicit input contextual information to favor closest hotels first, i.e., the provided services when the user is located in somewhere are different from the services when the user is elsewhere, therefore, we can say that the input location information is autonomously altering the output provided service. Another example can be found in [93], when a change to the value of one parameter (network bandwidth) may automatically impact the values of some properties (remaining battery power).

In this sense, the traditional Class diagram including the structure of the class itself (attributes and operations) are likely insufficient to express how applications are linked to the context parameters and how the values of these parameters may positively affect the application structure and behavior, this calls to think about the necessity to introduce distinct modeling concepts to specify these new relationships.

6.3 Extension of the UML Class diagram: the ContextClass diagram

Our goal is to enlarge the semantics of the Class diagram so that it can depict further information about context and context-awareness. To do so, our proposal consists in enriching the class' structure with a novel construct capable of monitoring how applications are linked to the contextual parameters and how the values of these parameters may affect the application behavior, we call this new construct the *monitor*.

Contextual aspects will be handled by means of the *monitor*, it has the real task of revealing the contextual parameters that may have an influence on objects states and behaviors, as pointed earlier, considerable variations in the contextual parameters entail changes in objects states (properties) and alter the ways in which objects behave (operations). With this in mind, we characterize the *monitor* by two concepts: a contextual parameter type and the set of the influenced attributes and/or the operations. The monitor type takes its value from the enumeration that contains the context parameters of our proposal (see Section 5.4).

Wherefore setting up such a new feature rather than resorting to the original structural and behavioral ones comes down to the particularity of the contextual aspect that we are aiming to append to the inner structure of the class, and which cannot be presented as ordinary attributes or operations. Moreover, suchlike dedicated notation permits to achieve a clear separation between traditional (non-contextual) and contextual concerns.

Henceforth, the original notion of *Class* will be substituted by *ContextClass*, and the diagram will be called the *ContextClass* diagram, it will be constituted of a set of *ContextClasses*, and relationships among them. Each *ContextClass* will thus be composed of four compartments:

1. The top compartment holds the name of the ContextClass.
2. The second compartment encompasses the attributes.
3. The third one carries the operations.
4. The bottom compartment contains the monitors.

Figure 6.1 illustrates the general structure of a ContextClass through the previous example of the tourist.

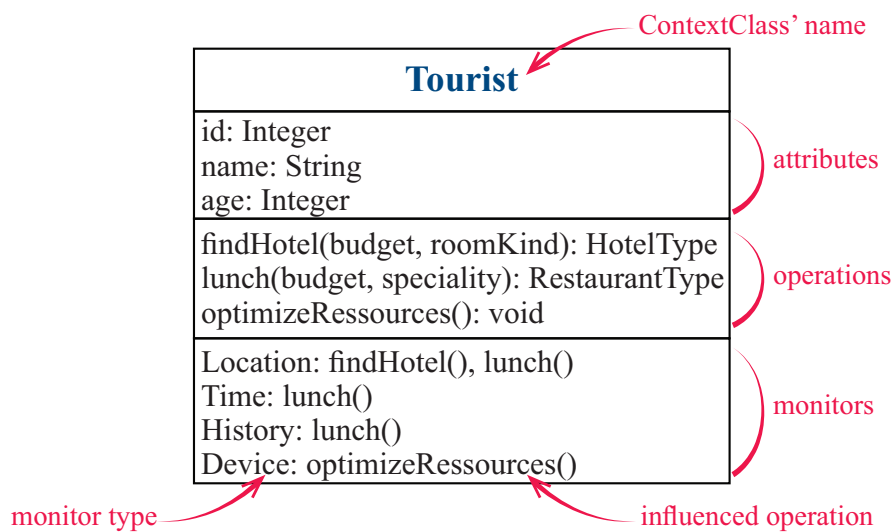


Figure 6.1. An example of the ContextClass "Tourist"

The relationships used in ContextClass models remain unchanged, we rely on the same relationships of the traditional Class diagram (*association*, *composition*, *aggregation*, and *specialization*). A full example of a ContextClass diagram will be shown in Chapter 8.

6.4 ContextClass diagram metamodel

The MOF-based definition of the ContextClass diagram consists in introducing new constructs related by inheritance at least to one element of the UML metamodel (*M2* level). Some metaclasses will have their own features and relationships among each other, whereas some features will redefine and subsume UML-related features (see **Figure 6.2**).

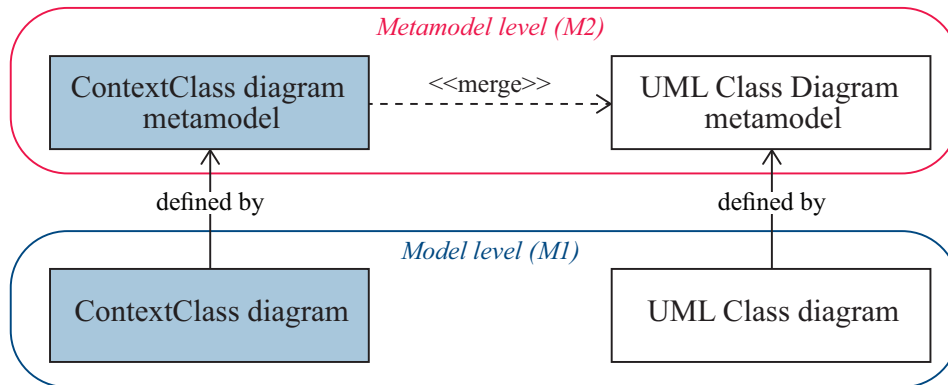


Figure 6.2. ContextClass diagram metamodel definition

In consonance with the UML definition [74], *Element* is an abstract metaclass with no superclass, all modeling concepts are specializations of it. The metaclass *Classifier* is related to the metaclasses *StructuralFeature* and *BehavioralFeature*. *StructuralFeature* is a generalization of *Property* and *BehavioralFeature* is a generalization of *Operation*.

As can be seen from **Figure 6.3** to **Figure 6.5**, our contribution is shaped in three new metaclasses (*ContextClass*, *ContextualFeature*, and *Monitor*), as well as an enumeration (*ContextualParameter*), all depicted in blue.

The *ContextClass* metaclass substitutes the older structure of *Class* and extends the metaclass *Classifier* (**Figure 6.3**); therefore, it will be associated with the native metaclasses *StructuralFeature* and *BehavioralFeature*. Similarly to *Class*, the structural features of *ContextClass* instances are attributes and their behavioral features are operations.

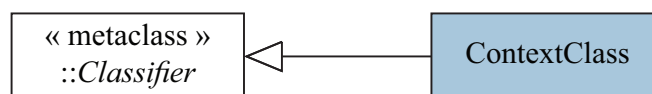


Figure 6.3. Classifier specialization: the ContextClass metaclass

The other new metaclass is *ContextualFeature*, it specializes the *Feature* metaclass (**Figure 6.4**) and is the superclass of the metaclass *Monitor*, the latter is defined as a new component of *ContextClass* via a composition relationship (**Figure 6.5**).

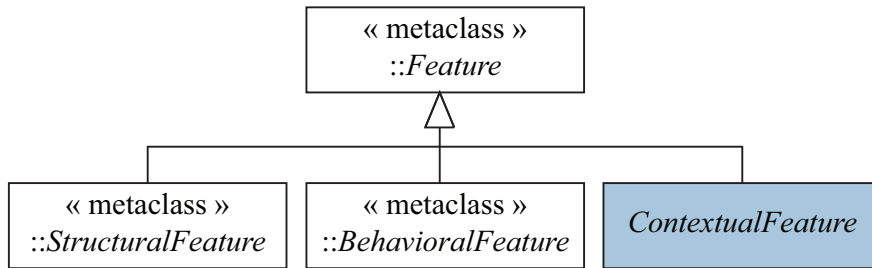


Figure 6.4. Feature specialization: the ContextualFeature metaclass

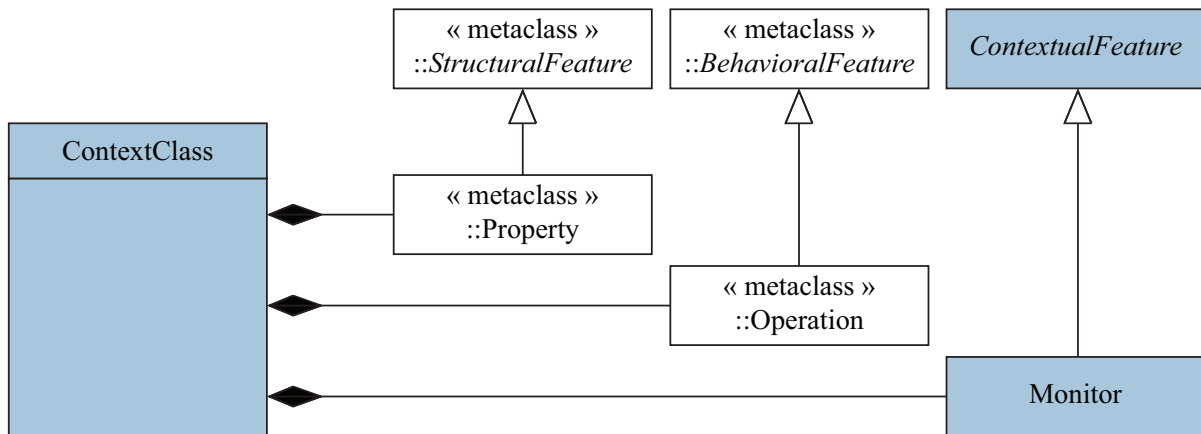


Figure 6.5. ContextClass structure: the monitor metaclass

A *monitor* has a type that takes its value from the *ContextParameter* enumeration and also has target meta-relationships with the set of attributes and operations that are influenced by it (*influencedState* and *influencedBehavior*). The *ContextParameter* enumeration encompasses the contextual parameters of our proposal: *Location*, *Time*, *Environment*, *Device*, *Network*, *User state*, *Profile*, *Preferences*, *History*, *Surround objects*, and *Nearby persons*.

Table 6.1 recaps the new metaclasses generalizations, attributes, and associations, whereas **Figure 6.6** depicts the full MOF-based metamodel of the ContextClass diagram.

Table 6.1. Recapitulation of the new metaclasses of the ContextClass Diagram

Metaclass	Generalizations	Attributes	Associations
ContextualFeature	Feature (from UML)	No additional attributes	No additional associations
ContextClass	Classifier (from UML)	name: String[0..1] isAbstract: Boolean	/superclass: ContextClass[*] ownedAttribute: Property[*] ownedOperation: Operation[*] ownedMonitor: Monitor[*]
Monitor	ContextualFeature	type: ContextualParameter[1]	contextClass: ContextClass[0..1] influencedState: Property[*] influencedBehavior: Operation[*]

6.5 Ecore implementation: ContextClass hierarchical editor

In the interest of providing a practical tool for describing the structure of context-aware systems, we created a ContextClass modeling editor by setting up a description of its metamodel under the Eclipse Modeling Framework (see **Appendix A**). **Figure 6.7** depicts the metamodel drawn under Eclipse correspondingly to the MOF-based metamodel illustrated in **Figure 6.6**.

Instantiating the drawn metamodel will offer the possibility to generate an hierarchical modeling editor which can be integrated into Eclipse as a new Eclipse plugin called ContextClass, as given in **Figure 6.8**. Further explanation on how can this editor be used to model context-aware systems structures will be presented in Chapter 8.

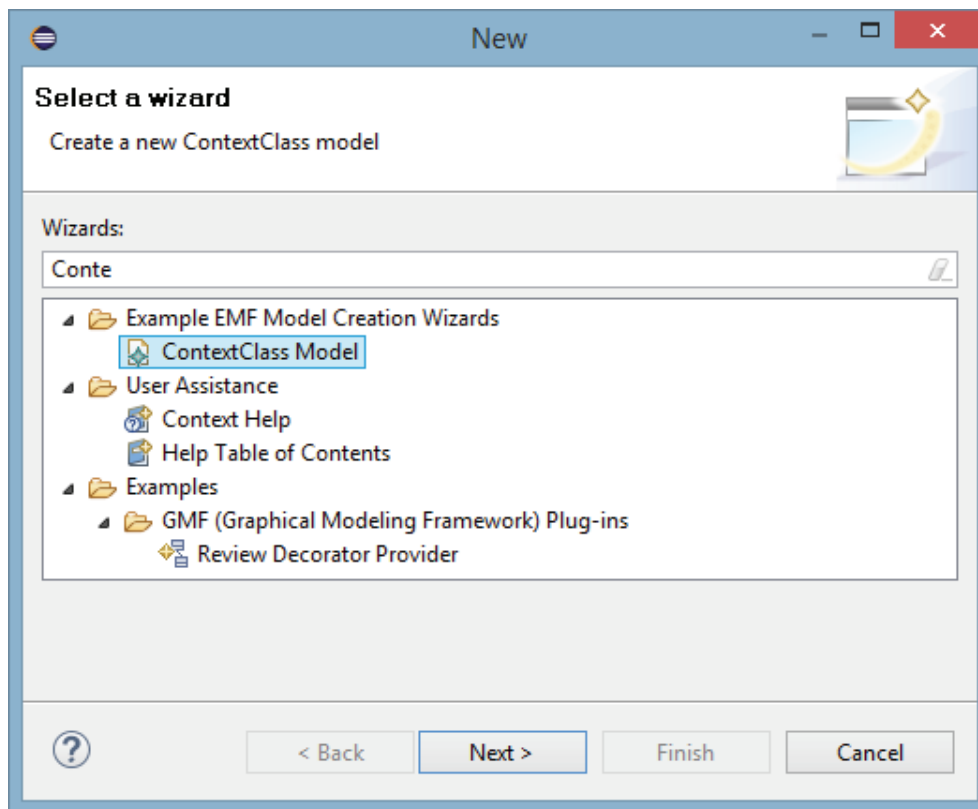


Figure 6.8. Example of creating a new ContextClass model

6.6 ContextClass diagram graphical editor

We created a graphical modeling tool for the ContextClass diagram under UMLet. As depicted in **Figure 6.9**, this editor is based on three main panels: *tools* panel, *properties* panel, and the *drawing space* panel.

1. In the tools panel, we inserted the necessary graphical notations to draw a complete ContextClass diagram; it encompasses the main element which is *ContextClass* (including *Monitor*) as well as the relationships (*generalization*, *composition*, *aggregation*, and *association*).
2. To add an element from the available tools panel to the drawing space panel we simply double-click on it.
3. From the command line in the properties panel, we can set up the properties of each ContextClass by entering the list of attributes, operations, and monitors, followed by a double dash " - " to create separations between compartments.

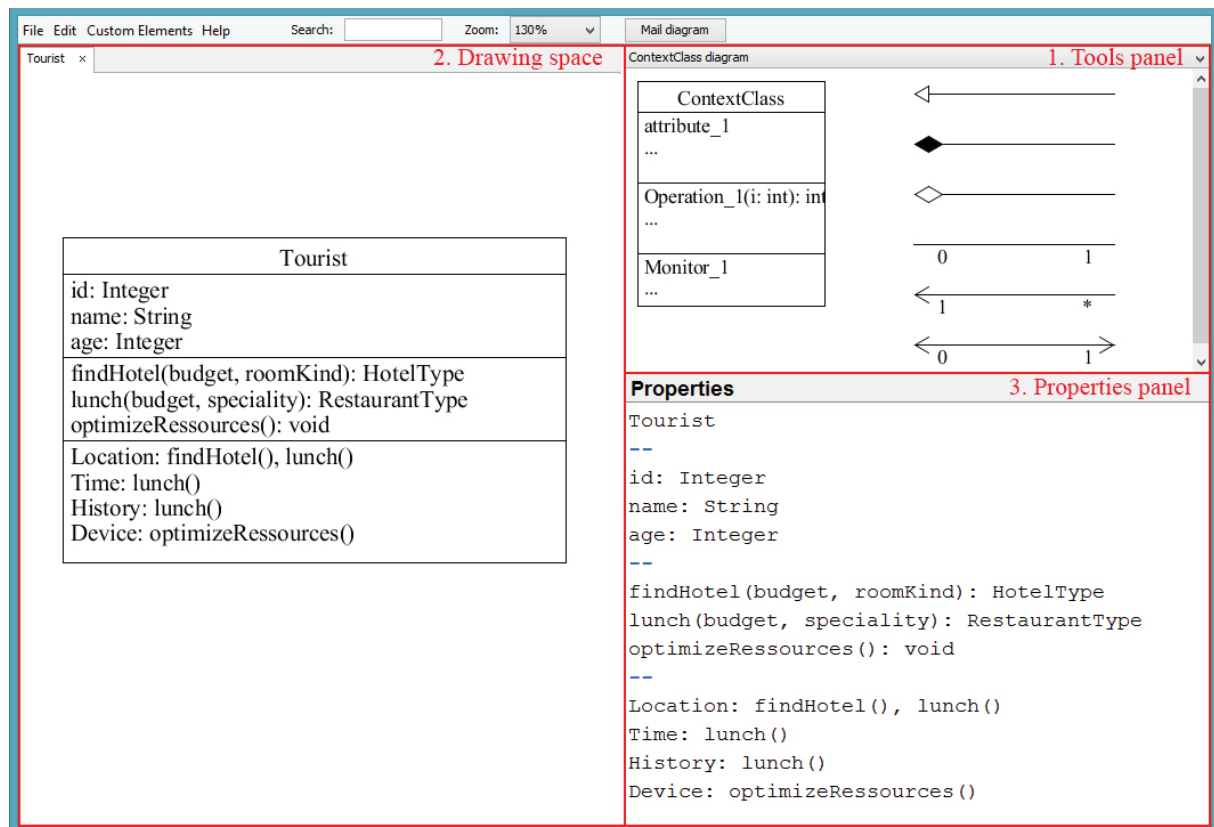


Figure 6.9. The ContextClass diagram graphical editor interface

6.7 Summary

Context-aware systems structure simulation is a substantial step toward depicting the external factors connected with the application. Exploiting the UML heavyweight extension mechanism, we presented a new form of the UML Class diagram by enriching it with new syntax and semantics to surmount its inabilities of expressing contextual information.

Besides the structural and behavioral features that characterize the system's entities, the ContextClass diagram depicts supplementary information by describing the contextual aspects. This is achieved by monitoring the dependencies between the parameters of context on the one hand, and on the other hand the properties and operations susceptible to be influenced once the context of use changes. In such manner, we are not only highlighting the contextual circumstances but also specifying which part of the system is affected by the related parameter.

At last, we implemented our modeling approach in concrete modeling editors to support academicians and software engineers understanding and handling context-awareness concerns using our extension proposal.

Chapter 7

Extending the Sequence Diagram to Model Context-Aware Systems: The ContextSequence Diagram

*"Serendipity is the faculty of finding things
we did not know we were looking for"*

Glauco ORTOLANO

Contents

7.1	Introduction	81
7.2	Limitations of the traditional UML Sequence diagram	81
7.3	Extension of the UML Sequence diagram: the ContextSequence diagram	82
7.4	ContextSequence diagram metamodel	86
7.5	Ecore implementation: ContextSequence hierarchical editor	88
7.6	ContextSequence diagram graphical editor	91
7.7	Summary	92

7.1 Introduction

Undoubtedly, developing context-aware systems implicates the consideration of many external aspects such as the perpetual variations in the contextual information and the different interactions with sensors. In fact, application designers use the UML behavioral diagrams to capture the interactions and to simulate the behavior of systems before the construction stage, however, these diagrams notations may need some enhancement to enable them specifying communications between the user, the system, and sensors.

This chapter presents a heavyweight extension of the UML Sequence diagram to enable software designers documenting and simulating models before the developers implement them. We start by illustrating the usefulness of modeling context-awareness concerns with distinct modeling notations, before presenting the concepts of ContextSequence diagram as well as its corresponding modeling tools.

7.2 Limitations of the traditional UML Sequence diagram

Documenting and simulating the needs of applications is requisite to manage their behavior as well as their interactions with the world. In this sense, the UML Sequence diagram describes the flow of events between objects leading to the desired outcome through different kinds of messages, also, it offers the possibility to specify conditional flow by virtue of its operands which are suitable for indicating actions to be performed under particular constraints.

Nonetheless, the standard notations of the UML Sequence diagram are too general to model self-adapting systems, these latter are characterized by special kinds of features that need advanced handling:

- As presented in the survey of Baldauf and Dustdar [60], context-aware systems don't solely implicate interactions among its participants but also interactions with context sources with which it must establish and maintain connections to obtain relevant contextual information; these interactions specify the communication policies of how the system requests and acquires information from sensors. Hence, we should clarify the interaction modeling between context-aware systems and context sources.

- Chaari and his colleagues [97] elucidated the necessity to design applications in a context independent way, i.e., a designer should identify the data which is related to the application distinctively from the data which specify the context. In this sense, in addition to actions executed according to functional constraints, context-aware applications also perform autonomous adaptation actions which are automatically triggered on the occurrence of context change events, their behavior is therefore affected and controlled by constraints of implicit contextual information. To do so, the boundary between application constraints and contextual constraints must be clearly specified depending on the application domain, for instance, a GPS localization represents an application data in a traffic regulation system but is part of context data in a telemedicine application.

Arguably, the Sequence diagram in its current form neither offers constructs to fully represent interactions with sensors nor provides proper notations for describing autonomous adaptation actions. Therefore, we need to specify the real behavior of context-aware systems visually by introducing explicit notations to represent perpetual context changes events, adaptation activities, and interactions between systems and context sources.

7.3 Extension of the UML Sequence diagram: the ContextSequence diagram

In this section, we introduce the concepts of the *ContextSequence diagram*. In addition to the traditional notations presented in Chapter 3, we will define new ones to overcome the modeling limitations of the traditional UML Sequence diagram. The proposed notations will offer software designers the possibility to achieve a clear separation of concerns between functional requirements and context-awareness requirements, and will be addressed through two main aspects: interactions with sensors and adaptation actions.





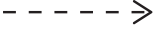
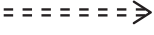
1. Interactions with sensors:

Together with the traditional interactions defined by the Sequence diagram messages (*synchronous*, *asynchronous*, and *replies*), context-aware systems comprise new kind of communications with sensors from which they acquire the relevant contextual information.

Conforming to Baldauf and Dustdar [60], there exist two possible ways of gaining access to contextual information sources: *synchronous* and *asynchronous*. The synchronous method allows querying the context source for specific information in a pull-based manner, therefore, the context source receives a message requesting some information and answers by delivering its value to the requester. Alternatively, the asynchronous method works via subscriptions, using this method enables the application to subscribe to specific events which it is interested in, then gets informed on occurrence of one of these events without continuously repeating context queries.

Our solution proposes to include these types of interaction in future models by defining novel notations to help software engineers specifying context-dependent messages distinctively from traditional ones. Graphically, contextual exchanges messages will be presented as double lines with the related message above as follows: *stick arrowhead* to represent *asynchronous notification* from context sources, *filled arrowheads* to express *synchronous context queries*, and *dashed lines* for *contextual information reply* messages. Interaction notations of the ContextSequence diagram are summarized in **Table 7.1**.

Table 7.1. ContextSequence diagram interaction notations

Traditional UML notations	Contextual interactions notations
Asynchronous message 	Asynchronous context notification 
Synchronous message 	Synchronous context query 
Message reply 	Context information reply 

2. Adaption behavior:

The promise of context-awareness is to enable applications to take actions autonomously by qualifying them to sense the user's context and to adapt their behavior appropriately. Considering the prevalent parameters that constitute the context of execution, we aim to introduce a novel notation to visualize the adaptation actions which are performed automatically according to the variations of a specific contextual parameter.

As can be seen in **Figure 7.1**, the novel notation consists of a dedicated combined fragment so-called *AdaptationCombinedFragment* with a new operand which we name *adapt*. The adaptation combined fragment is divided into one or more fragments called adaptation operands. Each *AdaptationOperand* corresponds to a specific contextual constraint and holds the set of messages which are sent once this constraint is met. In this manner, this allows achieving a clear separation between traditional systems' functionalities (non-contextual constraints) and automatic adaptation behaviors (context-dependent constraints). Also, we can help developers distinguishing Combined Fragments from Adaptation Combined Fragments at model level (*MI* level) through the use of annotations.

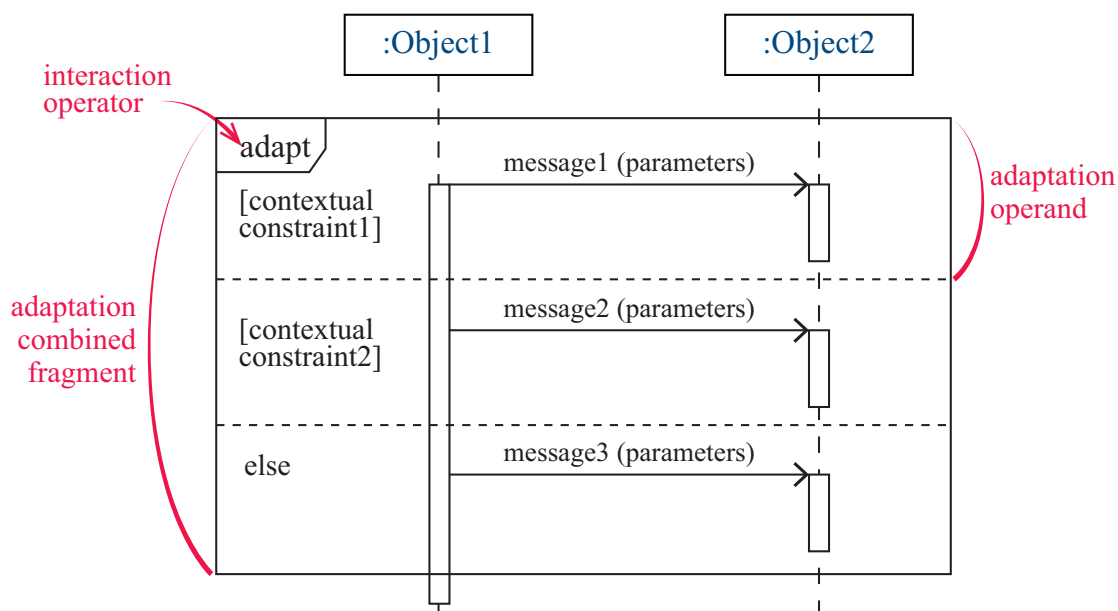


Figure 7.1. An example of an adapt combined fragment in a ContextSequence diagram

In furtherance to demonstrate the proposed concepts, we consider a ContextSequence diagram for the Temperature Control System (TCS) [122], the TCS ensures users' comfort by automatically adjusting rooms temperature. The system acquires the current room's temperature by requesting its value from temperature sensors, afterward, a controller applies automatic adaptation actions by switching between heating, warming, or cooling modes.

We address both of system's interactions and adaptation behavior by describing them through a ContextSequence (**Figure 7.2**). In addition to traditional interactions (*switchOn()*, *setPreferences()*, and *activate()*), we distinguish those which convey particular contextual in-

formation of the environment's temperature, *getTemperature()* and *return(temp)* are exactly the kind of interactions described by Baldauf and Dustdar [60], and therefore, are modeled respectively as *Synchronous context query* and *Context information reply*. Besides, we used an *AdaptationCombinedFragment* to specify the system's automatic adaptation; this behavior manifests in switching modes autonomously depending on contextual constraints of temperature values ranges.

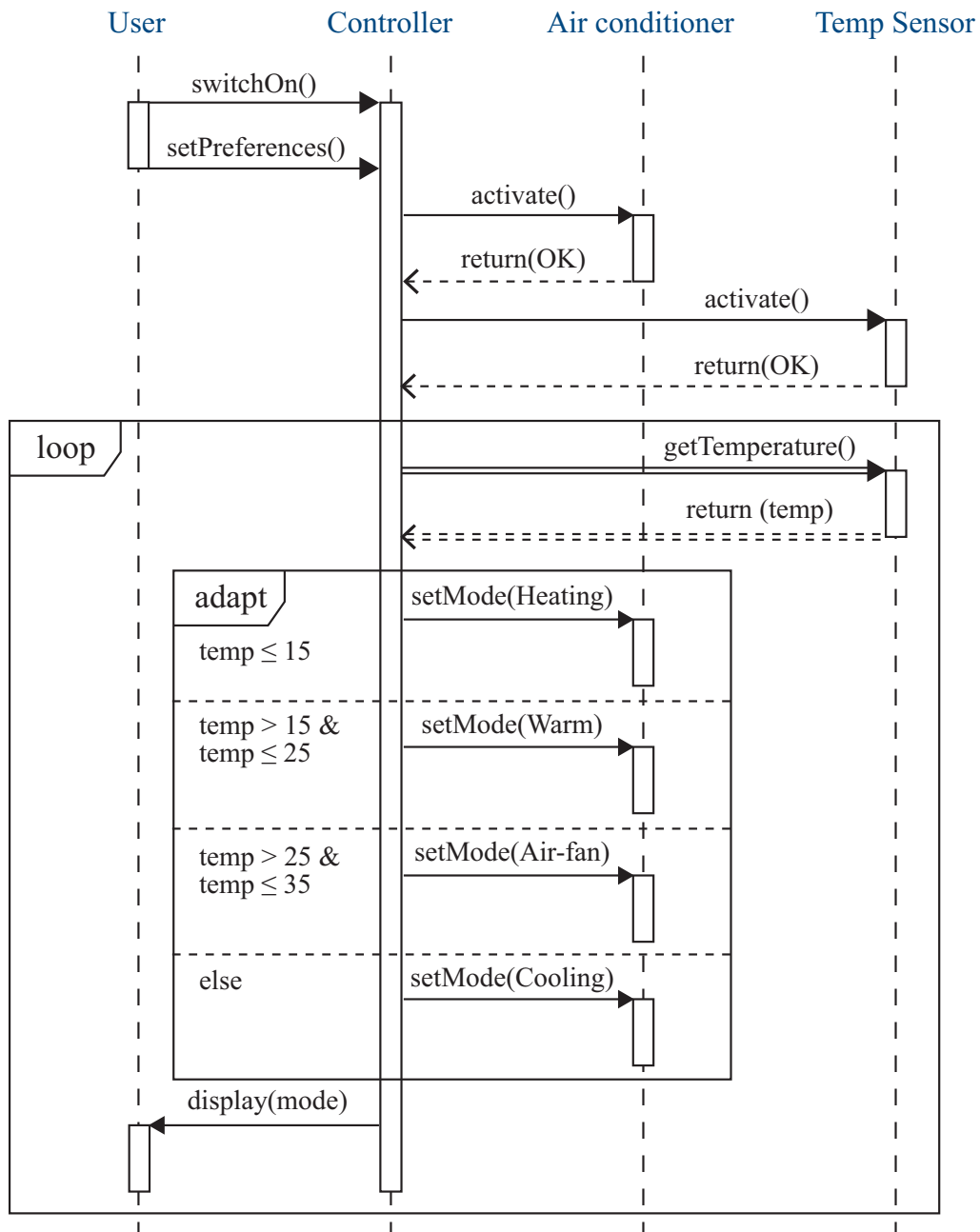


Figure 7.2. ContextSequence diagram for the Temperature Control System

7.4 ContextSequence diagram metamodel

Up to a point, future ContextSequence diagrams must syntactically conform to some rules. In this section, we present a MOF-based metamodel of the ContextSequence diagram by depicting UML's native metaclasses as well as the new modeling concepts that we presented previously (see **Figure 7.3**).

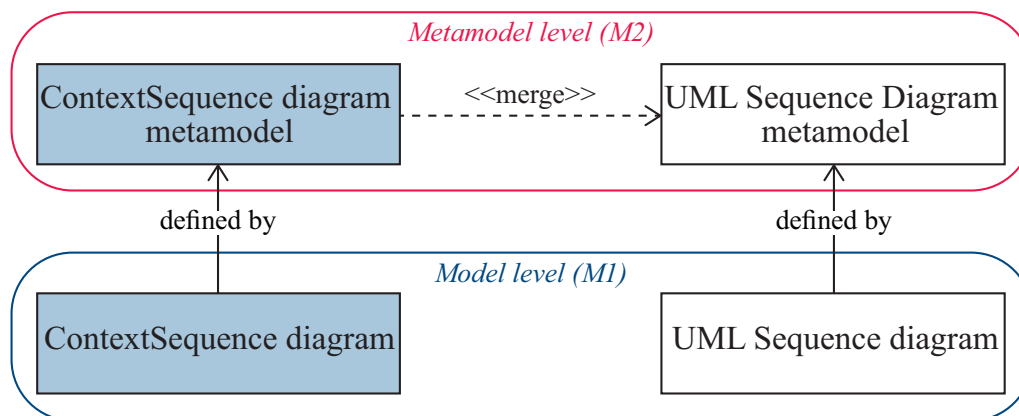


Figure 7.3. ContextSequence diagram metamodel definition

From **Figure 7.4** to **Figure 7.6**, we present the constructs of the ContextSequence diagram metamodel including the metaclasses and enumerations that we introduced (colored in blue).

InteractionFragment is an abstract notion of the most general interaction unit, it is a piece of an interaction. Each interaction fragment is conceptually like an interaction by itself. *Interaction* comprises the set of messages exchanged among lifelines [74].

The metaclass *Message* defines a particular communication between lifelines, a message is characterized by *messageKind* and *messageSort* properties which take their values respectively from the *MessageKind* and the *MessageSort* enumerations. Originally, the literal values of the *MessageSort* enumeration are only: *synchCall*, *asynchCall*, *asynchSignal*, *createMessage*, *deleteMessage*, and *reply*; we enriched this enumeration by adding the three types of messages introduced by Baldauf [60]: *asynchContextNotification*, *synchContextQuery*, and *contextInformationReply*.

The metaclass *Lifeline* represents an individual participant in the interaction, whereas the metaclass *sensorLifeline* is the specialization of *Lifeline*, its instances represent context sources distinctively from the system participants (see **Figure 7.4**).

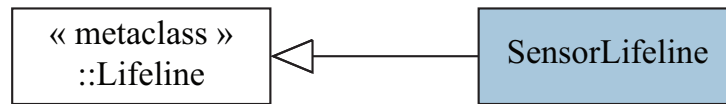


Figure 7.4. Lifeline specialization: the SensorLifeline meta-class

Following the UML specification [74], combined fragments of traditional Sequence diagrams are defined by the *CombinedFragment* meta-class, while the fragments that compose it are presented by *InteractionOperand*. Analogously, our proposed adaptation combined fragment concept is defined by the *AdaptationCombinedFragment* meta-class while the inner fragments that compose it are instances of the meta-class *AdaptationOperand* (see **Figure 7.5**).

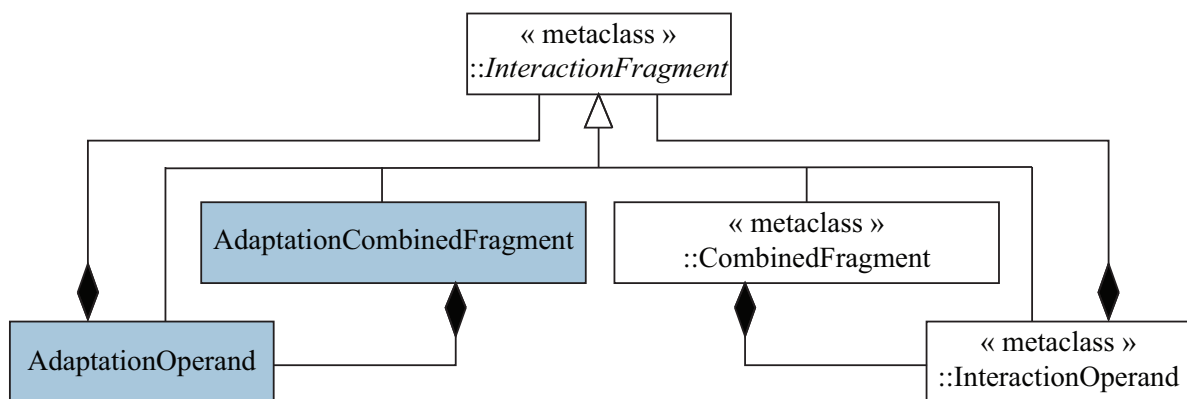


Figure 7.5. InteractionFragment specialization: the AdaptationCombinedFragment meta-class

To designate its semantic, the meta-class *CombinedFragment* is characterized by the property *interactionOperator*, this latter takes its value from the literals of the *InteractionOperatorKind* enumeration: *seq*, *alt*, *opt*, *break*, *par*, *strict*, *loop*, *critical*, *neg*, *assert*, *ignore*, and *consider*.

The *interactionOperator* attribute of the *AdaptationCombinedFragment* is always set to *adapt*, we added this literal to the *InteractionOperatorKind* enumeration to express the semantics of the adaptation behavior.

As illustrated in **Figure 7.6**, *InteractionOperand* is guarded by an *InteractionConstraint*, whereas *AdaptationOperand* is guarded by a *ContextualConstraint*, the latter corresponds to one of the contextual parameters of our proposal (see Section 5.4).

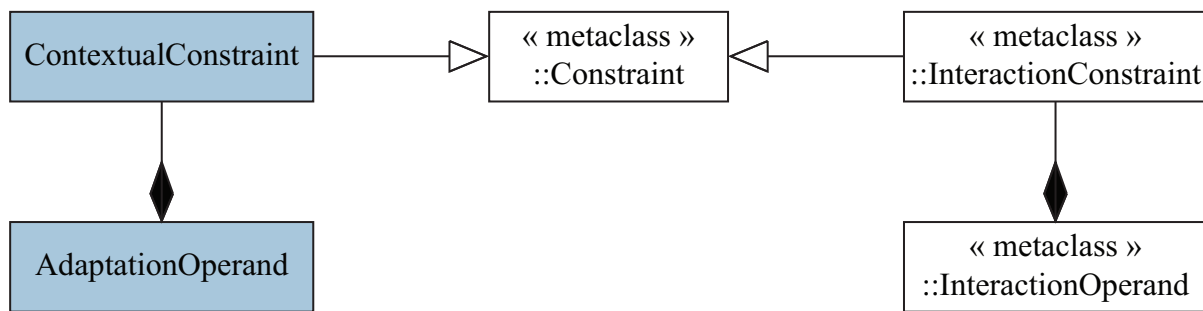


Figure 7.6. Constraint specialization: the ContextualConstraint metaclass

Table 7.2 recaps the properties of the metaclasses that we introduced as new concepts of the ContextSequence diagram, whereas **Figure 7.7** depicts its full MOF-based metamodel.

Table 7.2. Recapitulation of the new metaclasses of the ContextSequence Diagram

Metaclass	Generalizations	Attributes	Associations
SensorLifeline	Lifeline (from UML)	No additional attributes	No additional associations
AdaptationCombined Fragment	InteractionFragment (from UML)	interactionOperator: InteractionOperatorKind	operand:AdaptationOperand[1..*]
AdaptationOperand	InteractionFragment (from UML)	No additional attributes	fragment:InteractionFragment[*] guard:ContextualConstraint[0..1]
ContextualConstraint	Constraint (from UML)	Type: ContextParameter	No additional associations

7.5 Ecore implementation: ContextSequence hierarchical editor

As depicted in **Figure 7.8**, we implemented the ContextSequence diagram metamodel in Eclipse (details can be found in **Appendix A**), this metamodel holds the rules to verify the correctness of the future models. Next, we have generated the corresponding hierarchical editor as an Eclipse plugin called ContextSequence, as illustrated in **Figure 7.9**. More illustrative examples of using this editor will be presented in Chapter 8.

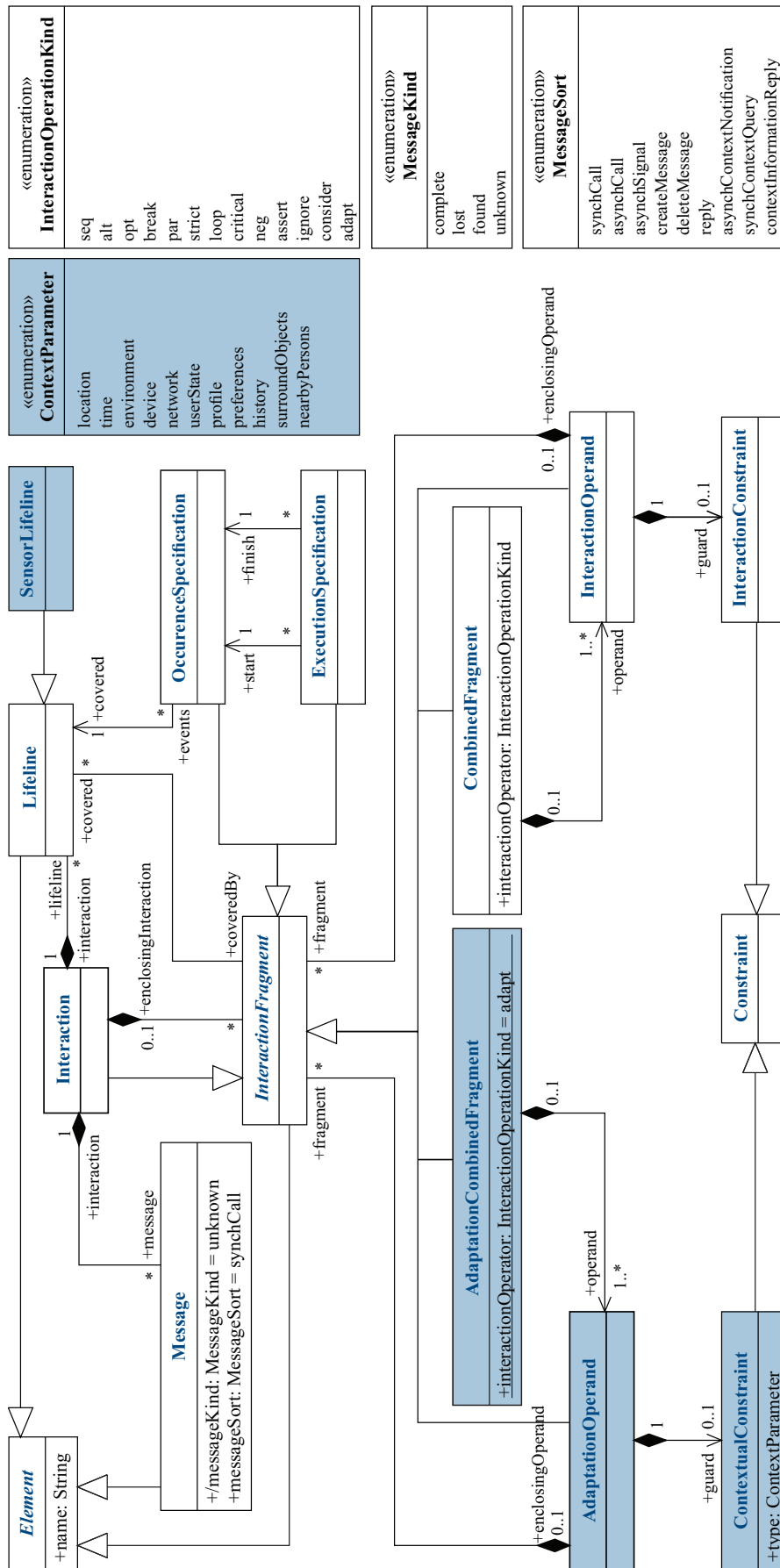


Figure 7.7. The metamodel of the ContextSequence diagram

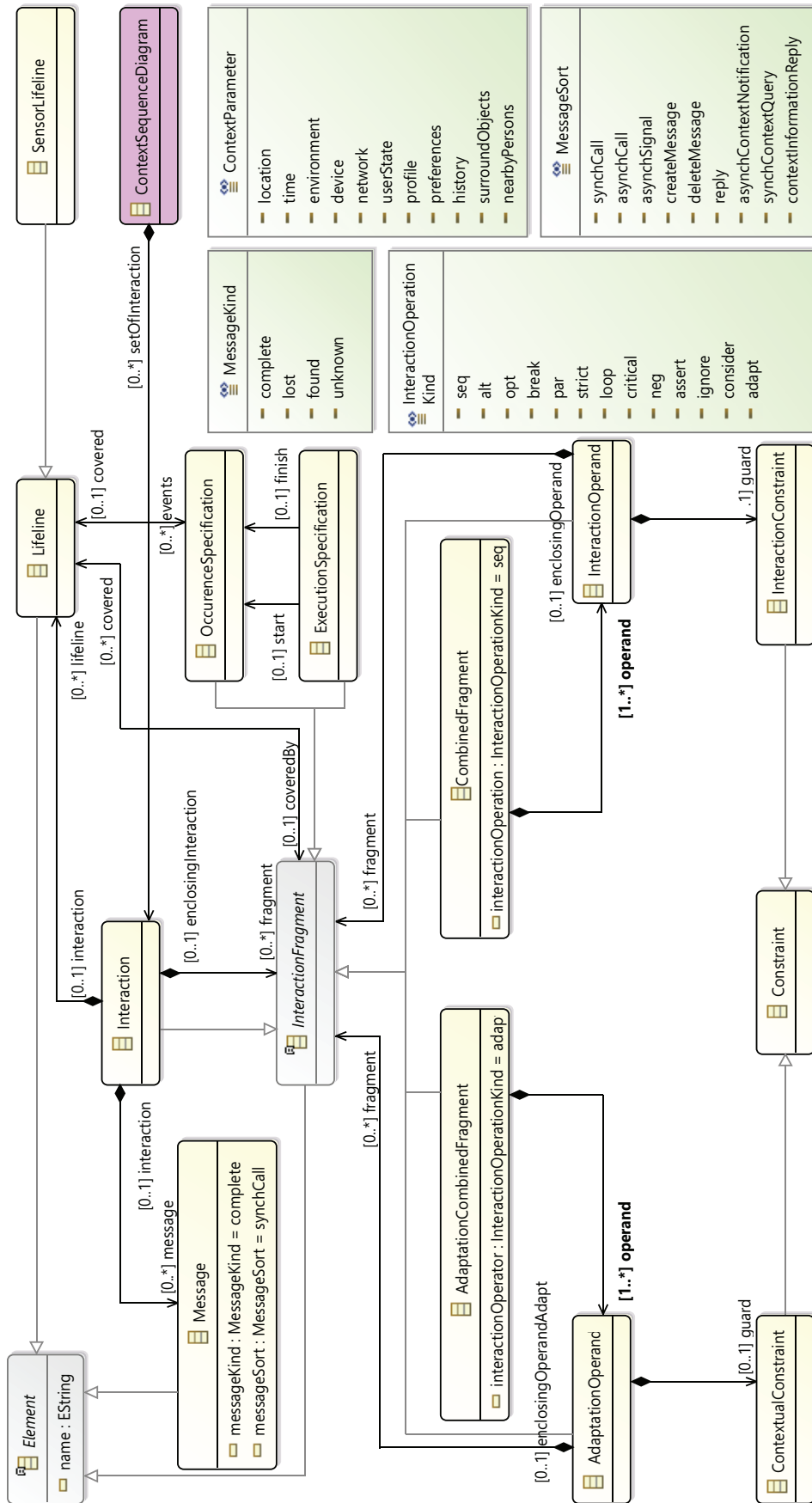


Figure 7.8. ContextSequence metamodel implementation under Eclipse

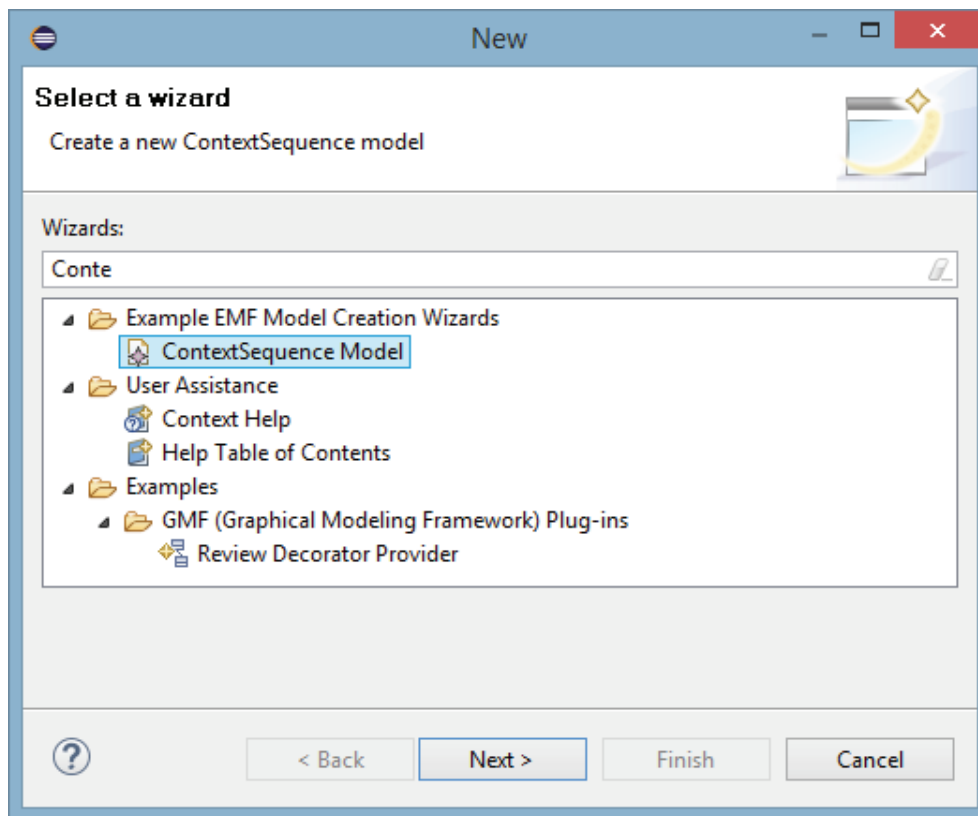


Figure 7.9. Example of creating a new ContextSequence model

7.6 ContextSequence diagram graphical editor

To expedite the description of context-aware systems behavior graphically, we built an editor for the ContextSequence diagram under UMLet by embedding the necessary drawing primitives. Further explanations can be found in **Appendix B**.

As presented in **Figure 7.10**, the editor is based on three main panels:

1. Tools panel: this panel contains all the necessary notations to draw a ContextSequence diagram (*lifeline*, *activation*, *traditional interactions*, *contextual interactions*, *traditional combined fragments*, and *adaptation combined fragments*).
2. Drawing panel: the space where to draw the diagram's elements.
3. Proprieties panel: enables to fill the properties of the drawn elements via the command line. As shown in the TCS example, we can indicate the contextual constraints for an adaptation combined fragment, following each contextual constraint with a "-" symbol allows creating separations between adaptation operands. Regarding messages, we can select the convenient message sort and determine its direction with "_LtR" or "_RtL".

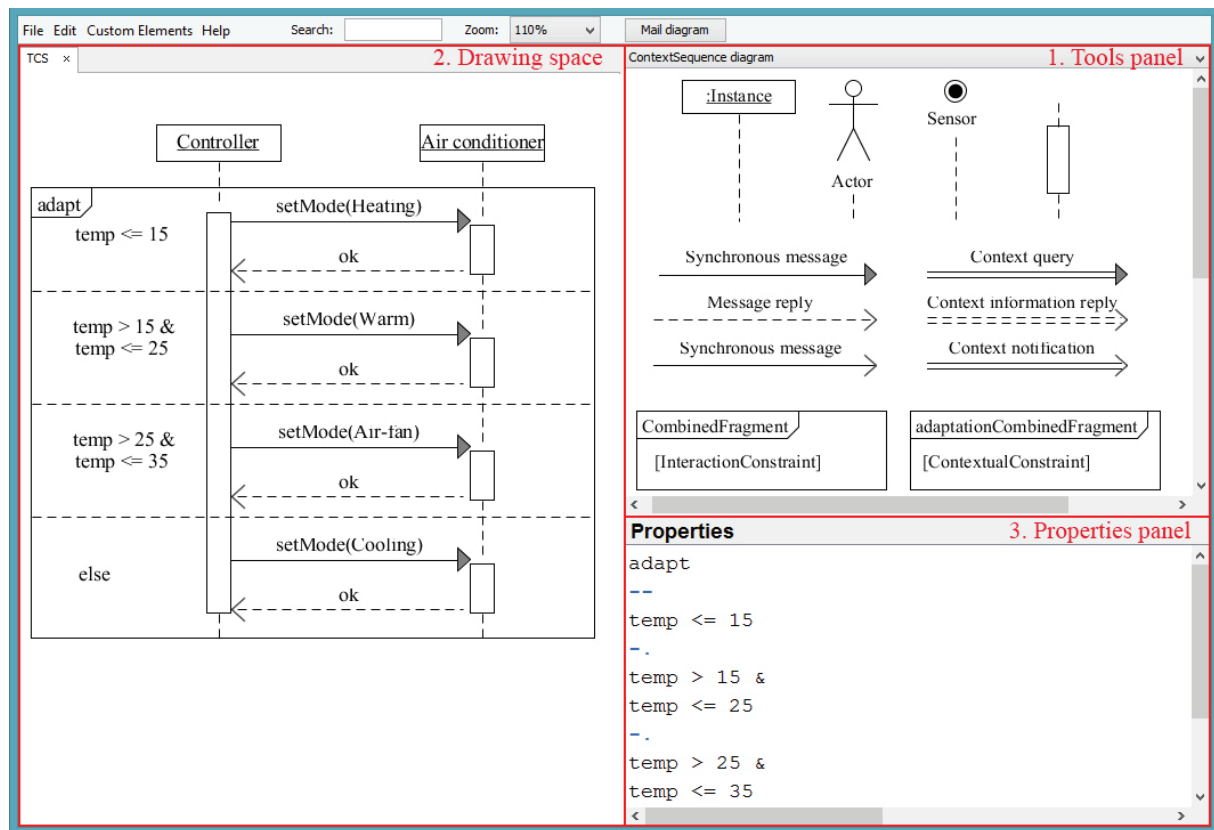


Figure 7.10. The ContextSequence diagram graphical editor interface

7.7 Summary

Altogether, simulation of context-aware systems behavior specifies the purpose of any interaction with an external part and identifies the different options for applications responses; however, it may be a challenging task especially when the output is affected by the constantly changing contextual parameters. Software designers aim to elaborate refined models conducive to increase adaptation's quality and performance, and also to reduce time and cost risks.

This chapter was devoted to presenting the ContextSequence diagram as a heavyweight extension of the UML Sequence diagram. The new diagram affords dedicated notations to help increasing designers' awareness towards the different interactions that convey contextual information from sensors. Moreover, the ContextSequence diagram also depicts the automatic adaptation actions which are performed under specific contextual constraints.

For the purpose of supporting software engineers and academic institutions simulating and documenting the behavior of context-aware systems, we operationalized our modeling approach in concrete editors.

Chapter 8

Case Study: a Smart Blood Pressure Tracker Application

"Don't find fault, find a remedy"

Henry FORD

Contents

8.1	Introduction	94
8.2	Description of the Smart Blood Pressure Tracker	94
8.3	Utility of the Smart Blood Pressure Tracker	96
8.4	Requirements specification	97
8.5	Contextual parameters of the SPBT system	98
8.6	Modeling the SBPT with the ContextClass diagram	99
8.7	Modeling the SBPT with the ContextSequence diagram	104
8.8	Summary	110

8.1 Introduction

The maturity of ubiquitous applications has made them available in an increasing number of areas. In this vein, diverse assistive technologies can afford health monitoring services by perceiving biological and physiological data of an individual and providing supports for diagnosis, therapy, prevention and early detection of diseases.

To evaluate our modeling approach in rich scenarios, we condense into investigating a case study in the healthcare field involving a Smart Blood Pressure Tracker application. We begin this chapter by describing the application's functionalities and infrastructure; followed by its requirements specifications as well as the contextual aspects that characterize it; afterward, we proceed to the modeling of the structure and the behavior of the SBPT system using the ContextClass and the ContextSequence diagrams respectively.

8.2 Description of the Smart Blood Pressure Tracker

The popularity of healthcare applications has increased beyond expectations, they help improving hospitals workflow management as well as clinical communications between providers and patients. Reinhold Haux [123] described health-enabling applications as "technologies that include wearable devices, such as micro-sensors embedded in textiles and personal computers. These technologies are aimed at making it easier for individuals to monitor and maintain their own health while enjoying lives in normal social settings".

To demonstrate the pragmatics of our modeling approach, our self-proposed case study is of interest to hypertensive patients and is presented as a Smart Blood Pressure Tracker (SBPT). Because the daily measurement of blood pressure may be very bothersome, the SBPT application is intended to provide an automatic system for the daily measurement and the regular screening of the patient's blood pressure. Also, the application purposes to decrease hypertension crisis damage probabilities by enhancing the monitoring and the reporting of critical health conditions of hypertensive patients, everywhere and at any time.

The SBPT consists in outfitting the patient with a smartwatch and a smartphone. The former is endowed with built-in sensorial features to measure the patient's blood pressure periodically and to emit the gotten values to his smartphone via a Bluetooth connection. The

smartphone merges all the day's blood pressure values into a single report and uploads it via Internet to a remote data server from which the doctor can consult all his patients' daily reports, using any device from everywhere (home, hospital, office, etc.).



Figure 8.1. A smartwatch with a built-in heart rate sensor

In the case of hypertensive crisis, the application detects it and performs an emergency call to the nearest rescue services, whereas the location of the user is known via his smartphone GPS sensor. Moreover, the application is capable of considering the user's profile and can afford dietary advice accordingly. The system is also capable of capturing the environmental conditions by displaying weather warnings and includes a timely reminder system for medication.

Figure 8.2 depicts the elements that constitute the infrastructure of the SBPT system.

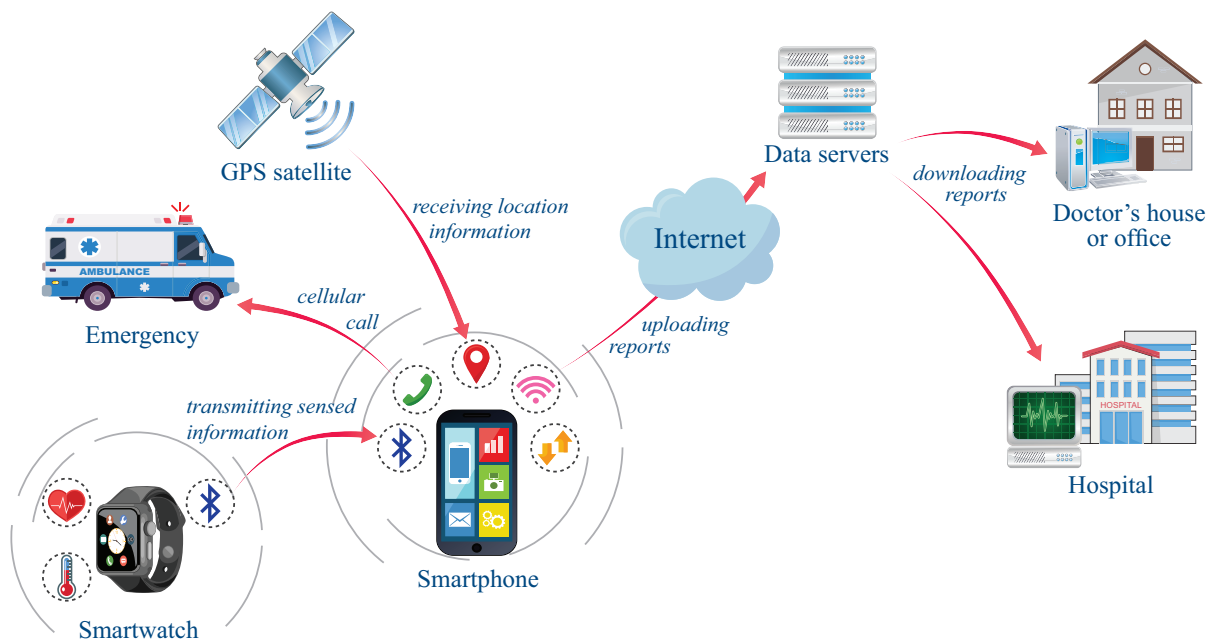


Figure 8.2. The infrastructure of the Smart Blood Pressure Tracker system

8.3 Utility of the Smart Blood Pressure Tracker

With the evolving climate of healthcare, rapidly developing technology, and emphasis on delivering patient-centered care, blood pressure telemonitoring is a promising tool to help patients achieve optimal blood pressure control.

The SBPT application leverages different types of contextual information to provide automatic adaptation services; it is beneficial for many reasons:

- Blood pressure telemonitoring offers an automatic measurement and remote monitoring of blood pressure by gathering the patient's blood pressure values and delivering them to the doctor, without bothering the patients to visit hospitals and measure their blood pressure manually.
- May encourage more appropriate resource utilization by curtailing the need for unnecessary in-person clinic visits (e.g., visits solely for a blood pressure check), while simultaneously alerting needed visits when a patient's blood pressure is out of target range.
- Accelerate the speed at which a patient achieves their target blood pressure goals. Patients can alter their health behaviors or have adjustments made in their medication regimen between visits, avoiding the need to wait months between visits for adjustments.
- Home-based monitoring may also alert the provider of new changes in a patient's health that may manifest with uncontrolled blood pressure.
- Hypertension is not called the "*silent killer*" for no reason, it has no symptoms, but it's a major risk for heart disease and stroke, for this, the application is able to detect sudden hypertension states, and performs automatic calls for the nearest emergency services.
- To ensure better safety of patients, the application is also capable of taking into account the environmental conditions like the temperature by displaying weather warnings.
- The application provides dietary advice according to the patient's profile including his age, gender, job, and his medical history.
- The medicine reminder tracks the patient's prescriptions and reminds him/her when it's time for a refill.

8.4 Requirements specification

When smart applications are introduced into the domain of health care, their well-functioning is critical. To guarantee an enhanced performance of such applications, several properties need to be addressed since the phases of requirements analysis and modeling. The various functionalities involved in the Smart Blood Pressure Tracker application make the design and development tasks complicated due to many considerations:

- Unlike traditional systems where input information is provided manually by the user, the Smart Blood Pressure Tracker is context-aware; therefore, it collects information of users and their surrounding environment continuously and makes adaptation decisions proactively.
- The environment is highly dynamic, the system must perform transparent adaptations in response to context variations, i.e., without distracting the user from his tasks.
- The application is surrounded by several contextual parameters that vary and influence the application's behavior perpetually; thus, these influencing relationships must be considered as a key concern.
- The user is nomad and carries his smartwatch and smartphone with him; also, the doctor needs to keep accessing all his patients' information any time and from anywhere (home, hospital, office).
- The smartwatch is supplied with temperature and heart rate sensors, the system queries them at precise times to monitor the patient's temperature and blood pressure respectively.
- The user's smartwatch is paired with his smartphone via a Bluetooth connection to emit the values of measured blood pressure. Both must be reasonably near one another.
- The Smartphone is responsible for delivering the daily gathered information to the doctor via Internet, calling the emergency services through cellular networks, locating the user via the embedded GPS sensor, and for alerting the patient for medication time.
- As mentioned previously, healthcare applications' functioning is critical, they may become unavailable due to malfunction or network failure, the system must watch over the availability of services in all situations.
- The system can take advantage of information related to the patient's profile to improve the quality of the provided services, including age, gender, and medical history.

- Each patient has a specific healthcare level related to his current state and profile, this property determines several parameters such as the schedules and the timetables for blood pressure measurements and is defined and updated continuously by the doctor.

8.5 Contextual parameters of the SPBT system

The parameters that constitute the contextual aspect of our application and which may alter its adaptation behavior are:

1. Location: longitude and latitude coordinates are usually used to provide users with location-aware services such as locating the patient in a case of critical health state.
2. Time: defines the schedules and the timetables for blood pressure measurements as well as the medication reminders.
3. Environment: high temperature and bad environmental conditions may be hurtful for hypertensive patients, the application reacts by alerting the user with weather warnings.
4. Device: the application must maintain the availability of information regardless of the used device, for example, the doctor can receive and consult all his patients daily reports, using any device (laptop, tablet, PDA, etc.).
5. Network: the application should manage to keep the system functional, for example, if the ADSL connection (Asymmetric Digital Subscriber Line) is down, the doctor still can receive the patients' reports by toggling towards a 4G connection automatically.
6. Surrounding objects: the application is aware of near objects, the smartwatch emits the measured blood pressure value to the smartphone as soon as they are closer to each other.
7. User's state: the state of the patient is an essential information source that may alter the way in which the application behaves (emergency alerts).
8. User's profile: age, gender, and job are significant factors for hypertension disease, as an example, stressful occupation strains situations which can cause blood pressure spikes, this requires to monitor the patient's state in a more accurate way.
9. User's history: the patient's medical history may be used as additional information to provide supplementary adaptation services, for example, hypertensive patients who suffer kidney failure or spinal disc herniation need extra care.

8.6 Modeling the SBPT with the ContextClass diagram

8.6.1 The SBPT system's structure

For the purposes of modeling this example with our proposed approach, we should:

1. define the involved ContextClasses, their attributes, and their operations;
2. determine the context monitors for each ContextClass;
3. draw the corresponding ContextClass diagram using the graphical editor; and
4. instantiate the ContextClass diagram metamodel under Eclipse.

At first, we consider the four following ContextClasses: *Doctor*, *Patient*, *SmartWatch* and *SmartPhone*.

A doctor can determine the medical profile for a given patient by inquiring his past medical history and by examining his current health state (*defineProfile()* operation), he can thus prescribe the adequate medication for the patient (*prescribeMedic()*). The patient's profile, history, state, and information are presented as attributes.

The SmartWatch is responsible for carrying out low-level context sensing, it measures the patient's blood pressure and his skin temperature continuously (*measurePressValue()* and *measureSkinTemperature()* operations), it emits the measured values as well as their measuring time to the smartphone as soon as they are close to each other (*emitPressValue()*). The smartwatch sounds alarm reminders for medication (*alarmMedic ()*) and for weather warnings to inform the patient of hot or cold weather (*weatherWarning ()*).

The smartphone receives the measured blood pressure values from the smartwatch (*getPressValue()*), gathers them into a single report with the related date (*makeReport()*) and delivers it to the Doctor's data server (*deliverReport()*).

If a measured blood pressure value represents a critical health status, the smartwatch detects it (*signalER()*) and orders the smartphone to perform a call to the closest emergency services automatically, according to the patient's location (*callER()*).

Later on, we determine the context monitors for each ContextClass by enumerating the contextual parameters and the influenced operation and/or attributes for each one. Monitors

aim at highlighting which context parameters may alter the attributes values or operations behaviors for the corresponding ContextClass. If we take the ContextClass *SmartWatch* as an example, it is fed with four kinds of context monitors: *Time*, *Surrounding Objects*, *User state*, and *Environment*. For instance, the *weatherWarning()* operation depends on the environment, and its behavior is altered depending on environmental variations. Just in the same way, for the *SmartPhone* ContextClass, *callER()* may operate in different ways; thereby, the called emergency services are selected according to the patient's location.

Table 8.1 summarizes the context monitors for each ContextClass.

Table 8.1. The monitors of each ContextClass of the SBPT system

ContextClass	context monitors	influenced properties	influenced operations
SmartWatch	Time	-	alarmMedic()
			measurePressValue()
	Surrounding Objects	-	emitPressValue()
	User state	-	signalER()
	Environment	-	weatherWarning()
SmartPhone	Time	-	deliverReport()
	Network	-	deliverReport()
	Surrounding Objects	-	getPressValue()
	Location	-	callER()
Patient	User state	healthCareLevel	-
	Profile	healthCareLevel	-
Doctor	Location	-	receiveReport()
	Device	-	receiveReport()
	Network	-	receiveReport()

8.6.2 Modeling the SBPT with the ContextClass graphical editor

Figure 8.3 depicts the full ContextClass diagram of our case study drawn under the UMLet-based graphical editor. In addition to attributes and operations, the model also represents the contextual aspects clearly by means of the monitors.

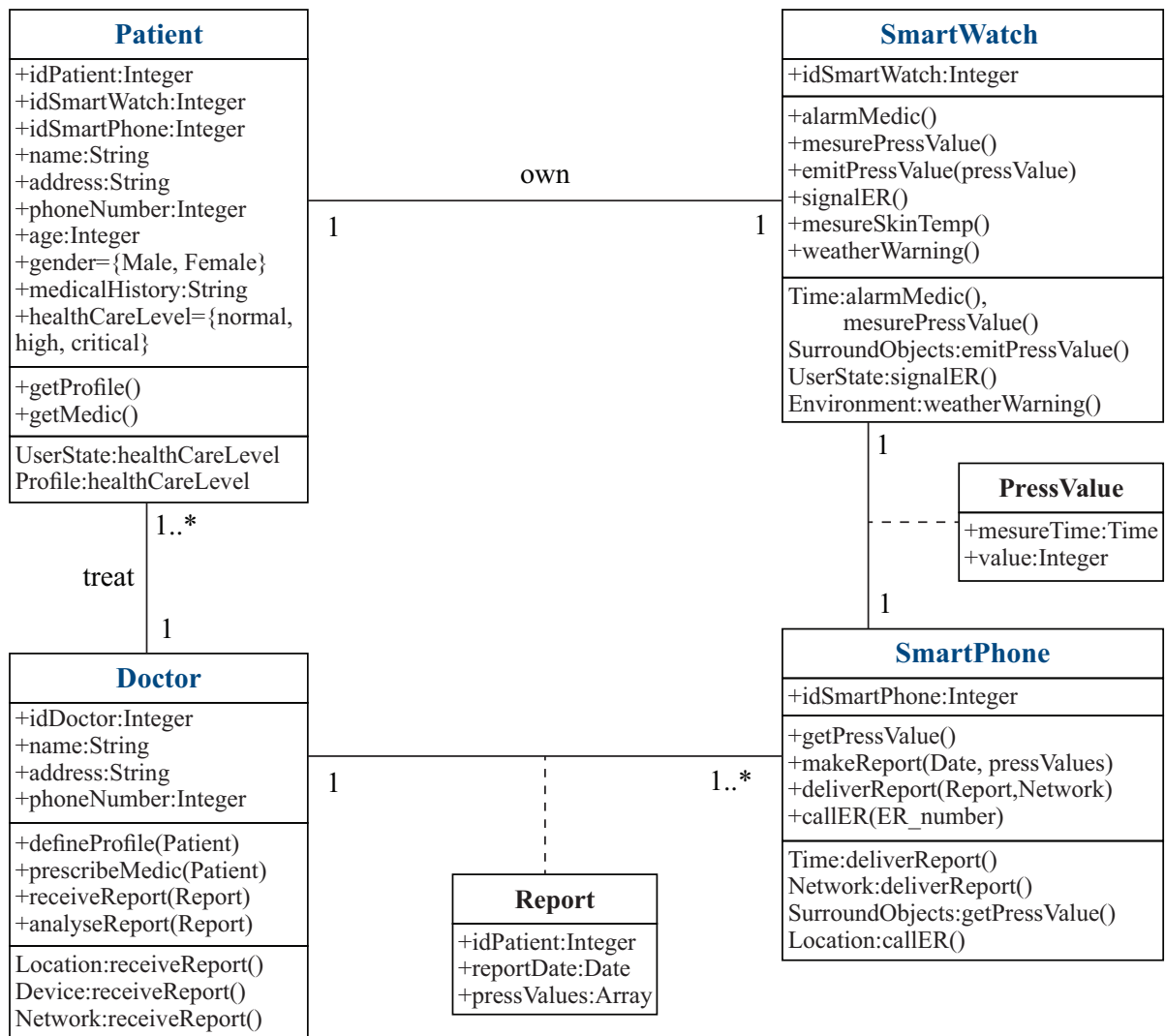


Figure 8.3. The ContextClass graphical editor: Modeling the SBPT’s structure

8.6.3 Modeling the SBPT with the ContextClass hierarchical editor

In this subsection, we will describe the SBPT system using the ContextClass Eclipse editor to explore its capabilities, we start by modeling the ContextClasses that intervene in our example as well as the associations between them as shown in **Figure 8.4**.

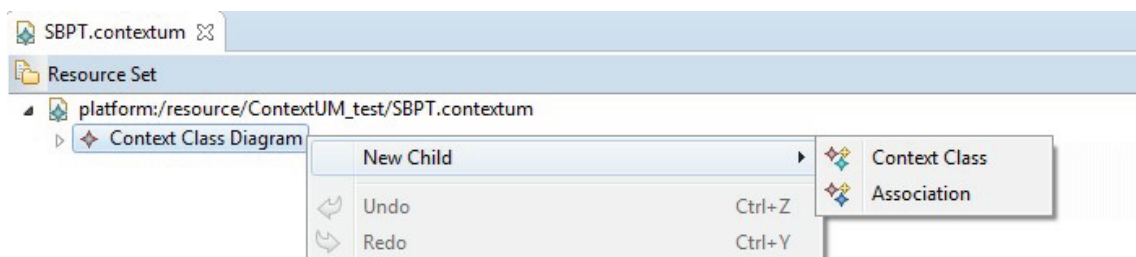


Figure 8.4. The ContextClass Eclipse editor: creating new ContextClasses and associations

Once the considered ContextClasses have properly been created, we add for each one the corresponding properties, operations and parameters of operations. **Figure 8.5** depicts an example showing the properties and operations of the *SmartWatch* ContextClass.

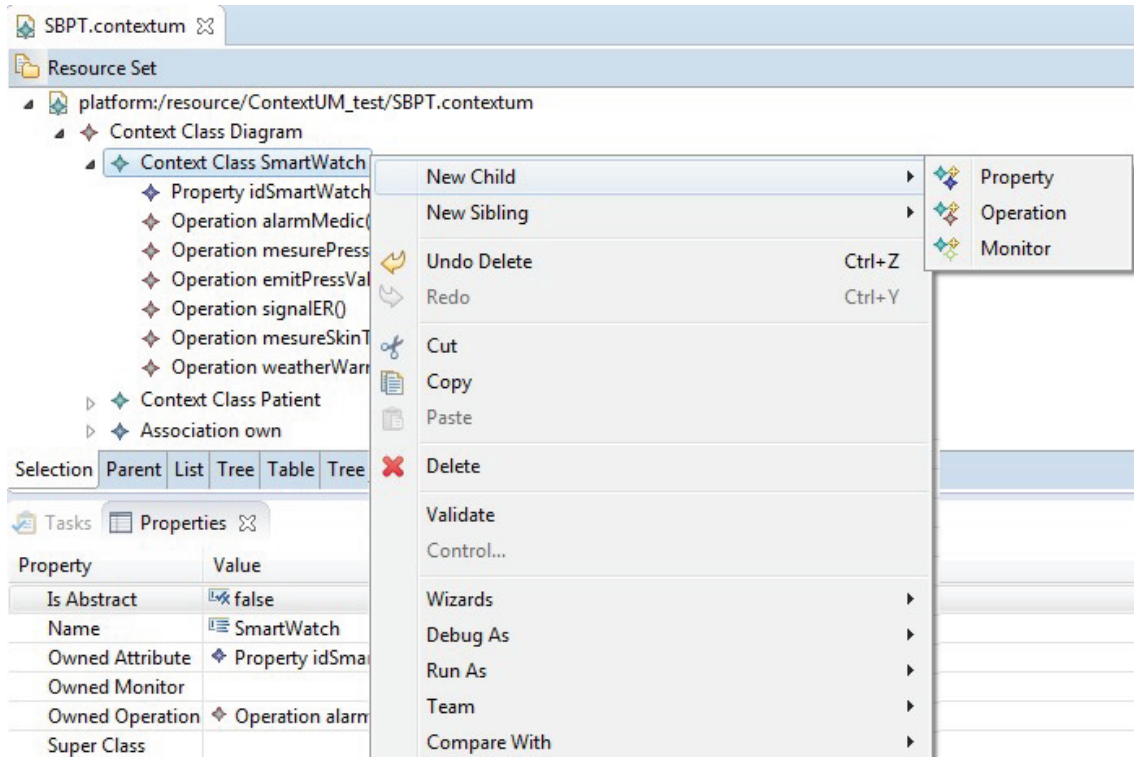


Figure 8.5. The ContextClass Eclipse editor: adding new properties and operations

We proceed to model the contextual information by adding context monitors. For each one of those, we specify the context type from a combo box list (**Figure 8.6**).

Also, we match the monitors to the corresponding influenced operations and/or attributes for each ContextClass. **Figure 8.7** shows an example of matching the operation *weatherWarning()* as an influenced behavior for the monitor *Environment*. The modeling process is completed when all the attributes, operations and monitors for all the ContextClasses are identified.

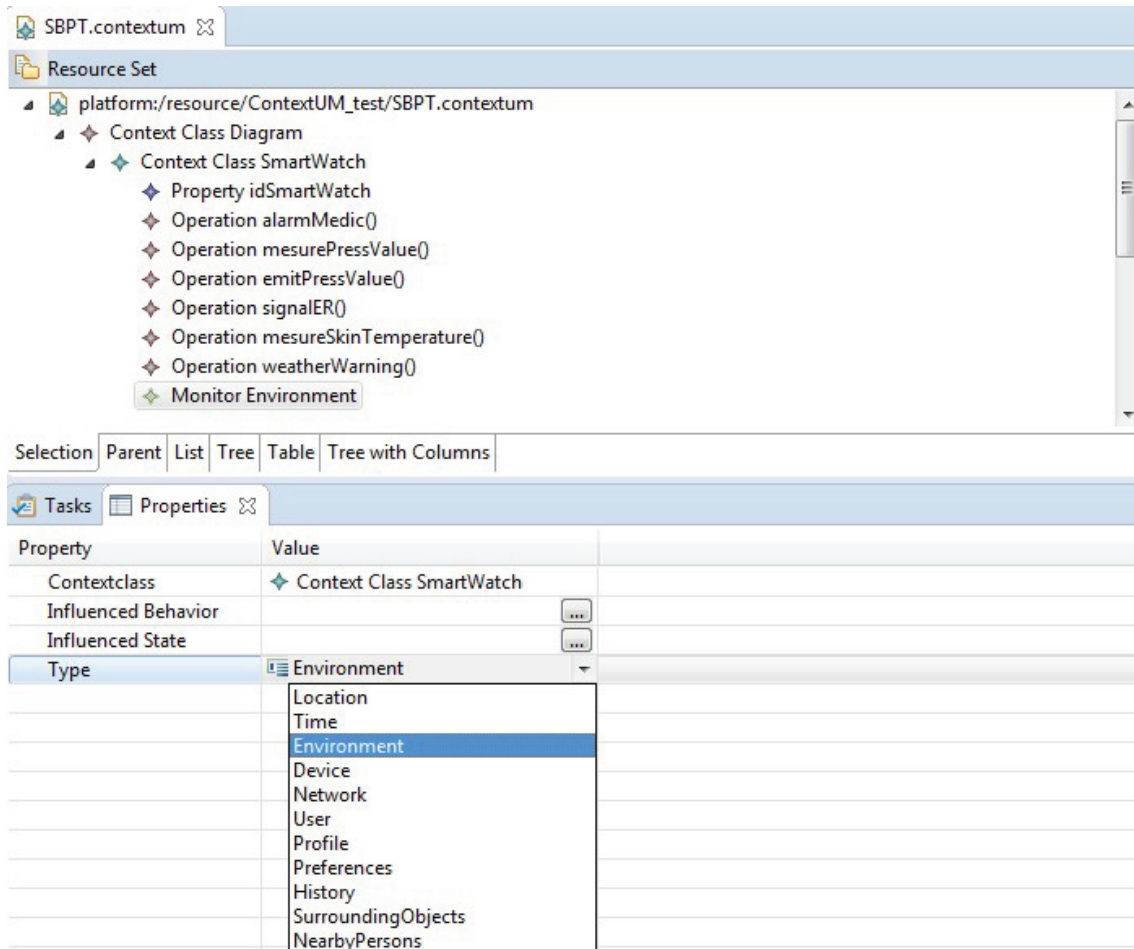


Figure 8.6. The ContextClass Eclipse editor: defining the type of a new monitor

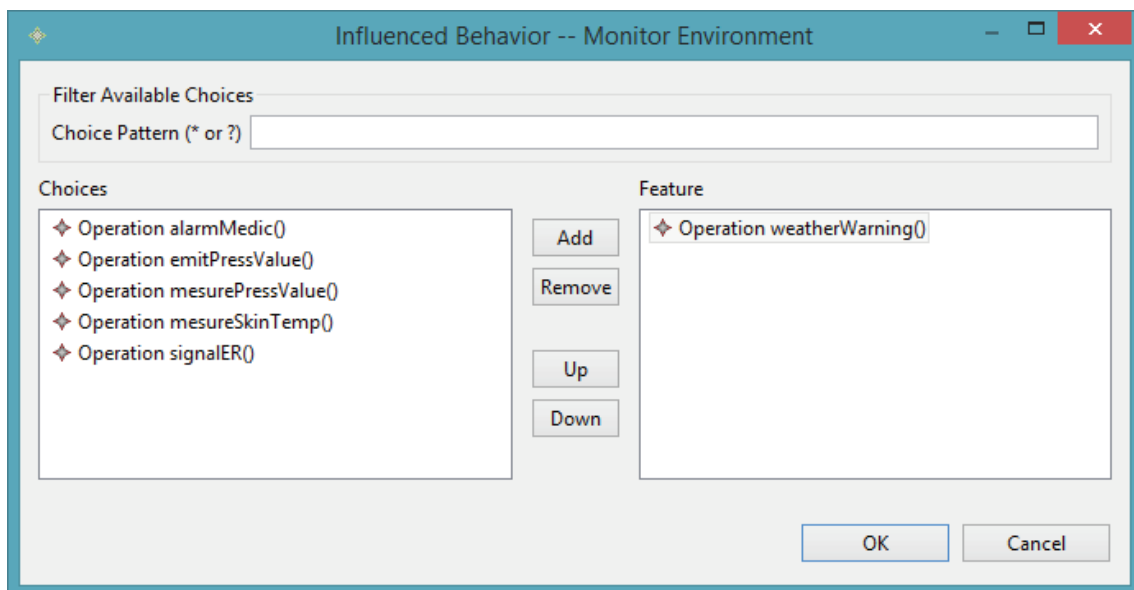


Figure 8.7. The ContextClass Eclipse editor: defining the influenced operations for a monitor

8.7 Modeling the SBPT with the ContextSequence diagram

8.7.1 The SBPT system's behavior

In pursuance of describing the behavior and the interactions of our case study system using the proposed ContextSequence diagram we start by defining the objects that may participate in the different use case scenarios, and that may interact by sending or receiving messages. These participants include sensors which provide contextual information for the system's objects which consume this information.

We take the emergency use case as an example; in addition to traditional interactions of switching on and connecting devices (*switchOn()* and *connect()*), we distinguish those which are related to contextual information exchanges (blood pressure and location queries and replies). The system interrogates both the smartphone GPS and the smartwatch blood pressure sensors to acquire the relevant contextual information in a Query-Reply method. **Table 8.2** summarizes the interactions in the emergency use case by specifying the source, the target and the sort for each one.

Table 8.2. Specification of the exchanged messages in the SBPT system

Message	Source	Target	Message sort
switchOn	User	SW_BloodPressureSensor	Asynchronous message
switchOn	User	SPhone	Asynchronous message
connect	SPhone	SW_BloodPressureSensor	Synchronous message
connected	SW_BloodPressureSensor	SPhone	Message reply
getPressValue	SPhone	SW_BloodPressureSensor	Context Query
pressValue	SW_BloodPressureSensor	SPhone	Context Information Reply
getLocation	SPhone	SP_GPSSensor	Context Query
location	SP_GPSSensor	SPhone	Context Information Reply
call	SPhone	Emergency	Asynchronous message
makeReport	SPhone	Sphone	Synchronous message
deliverReport	SPhone	Server	Synchronous message
receipt	Server	SPhone	Message reply

Besides, the SBPT system includes a set of functions that compose its behavior, among these, we can name the weather warnings, the dietary advice, and the medicine reminder services.

If we take the telemonitoring functionality as an example, the application's automaticity manifests in measuring and delivering blood pressure reports to doctors; in addition, when the user's health state is critical, the system reacts autonomously by performing an emergency call to the closest rescue services according to the user's position, whereas the latter is known via his smartphone GPS sensor. This adaptation depends on contextual parameters variations (user's state and location), and therefore, is performed automatically and without asking the user's guidance.

Also, the system must look up and switch between Internet connections whenever one of them keeps dropping off, e.g., auto-switching from WiFi to 4G mobile data when ADSL connection goes down. Other adaptation scenarios consist in locating the doctor and delivering reports to the closest associated device of his, alerting the patient for medication time and for bad weather.

8.7.2 Modeling the SBPT with the ContextSequence graphical editor

Exploiting the clarity the ContextSequence diagram drawing tool, we try to model the blood pressure telemonitoring and hypertensive crisis scenarios (**Figure 8.8**). The future diagram will feature all the interactions described in **Table 8.2** including the novel double-lined notations for highlighting contextual information traffic.

As well as that, the resulting diagram will describe the system's self-adaptation in the case of hypertensive crisis and of switching between Internet connections, to model such context-dependent behavior, we use the novel notation of *AdaptationCombinedFragment*, which encompasses the automatic actions to be performed depending on the user's context.

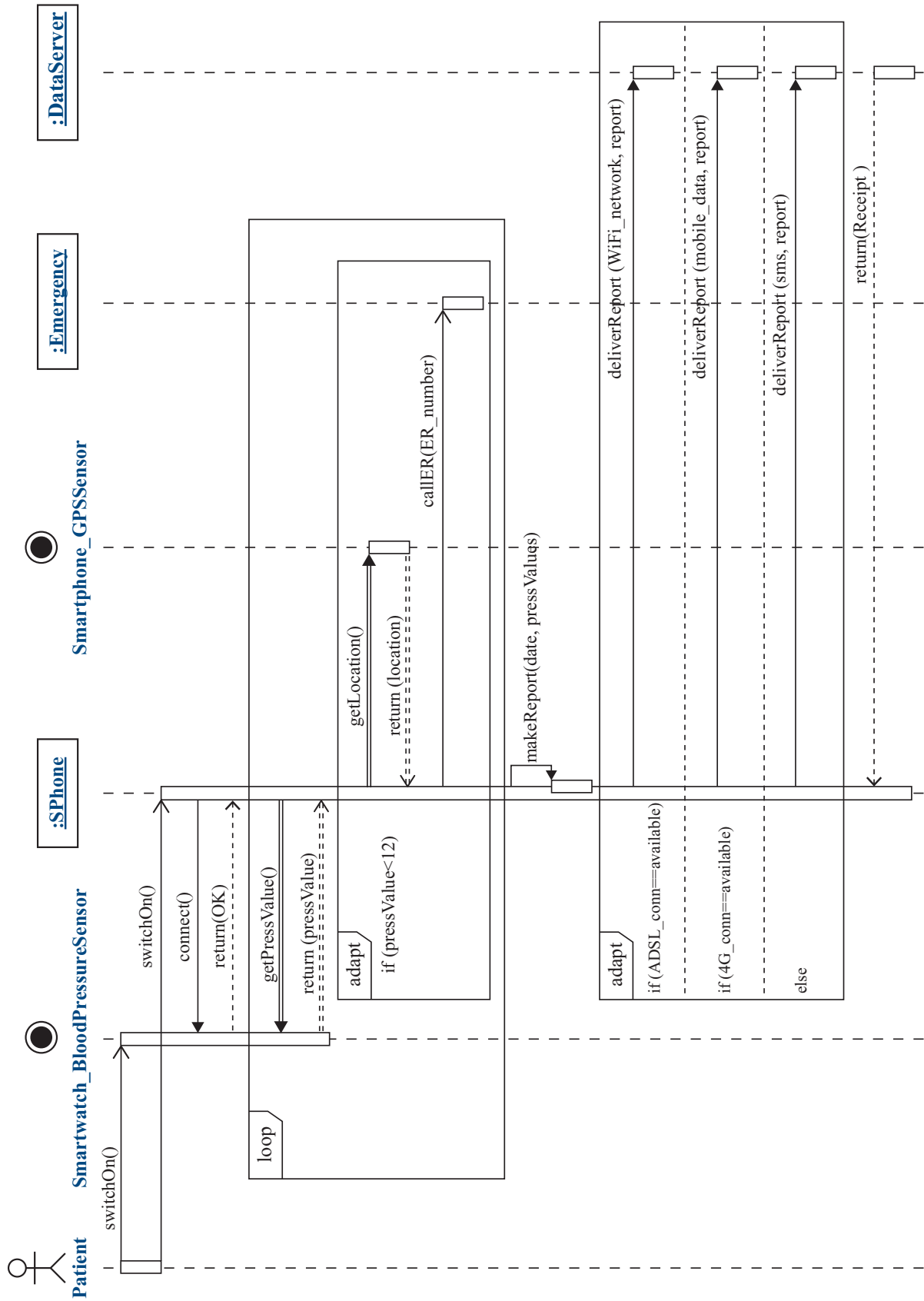


Figure 8.8. The ContextSequence graphical editor: Modeling the SBPT’s behavior

8.7.3 Modeling the SBPT with the ContextSequence hierarchical editor

In this subsection, we present a description of the hypertensive crisis scenario by instantiating the ContextSequence metamodel under the Eclipse hierarchical editor. We can create lifelines, sensor lifelines, interactions, combined fragments, adaptation interaction fragments, and messages. Each of these latter can be described by specifying his name, kind, and sort as well as the interaction to which it belongs.

Figure 8.9 shows an example of changing the sort of the message *getLocation(patient)* to *synchContextQuery*.

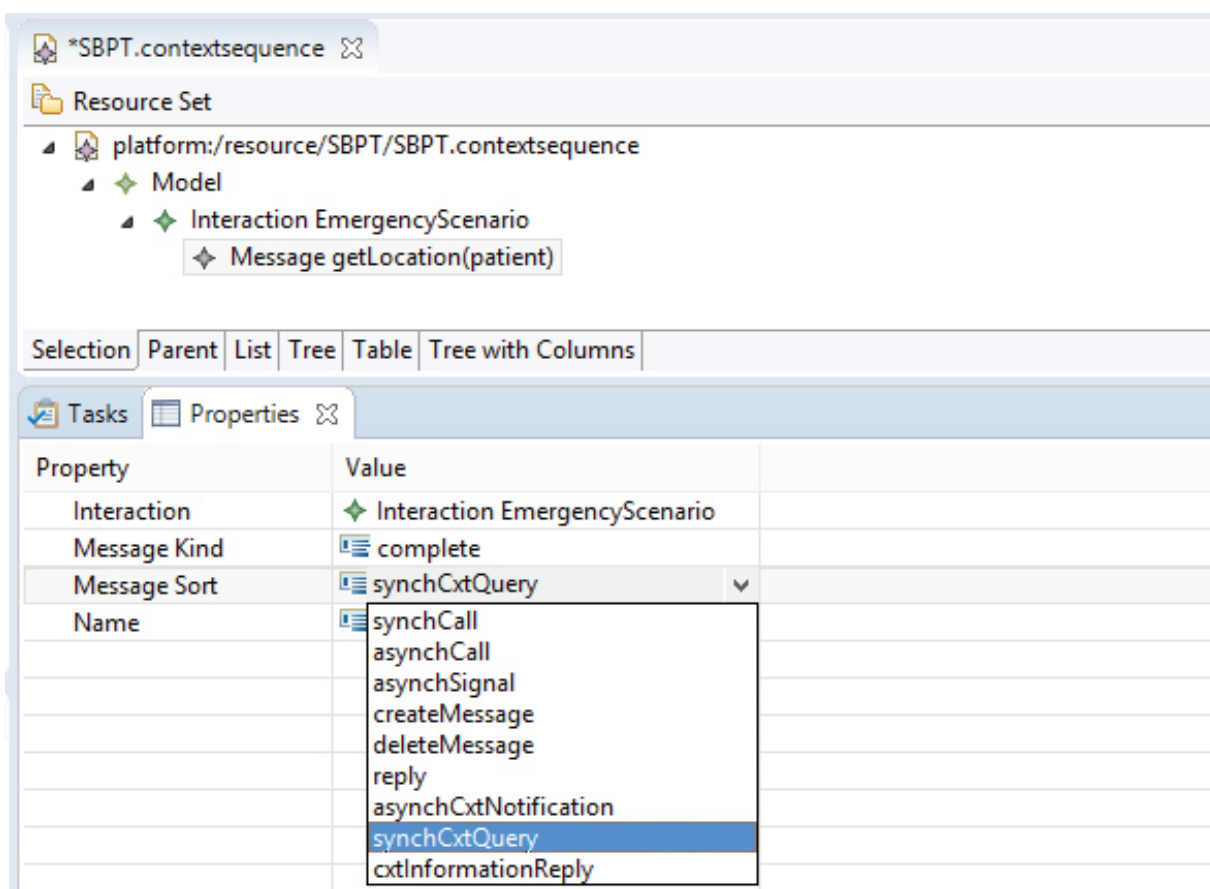


Figure 8.9. The ContextSequence Eclipse editor: Defining message proprieties

Regarding the complex structures, we take the example of the adaptation combined fragment, we can identify the lifelines it covers (the property *Covered*), the inner fragments (*Enclosing Operand*), the fragment by whom it is enclosed (*Enclosing Interaction*), and the kind of the interaction operator (*Interaction Operator*) as shown in **Figure 8.10**.

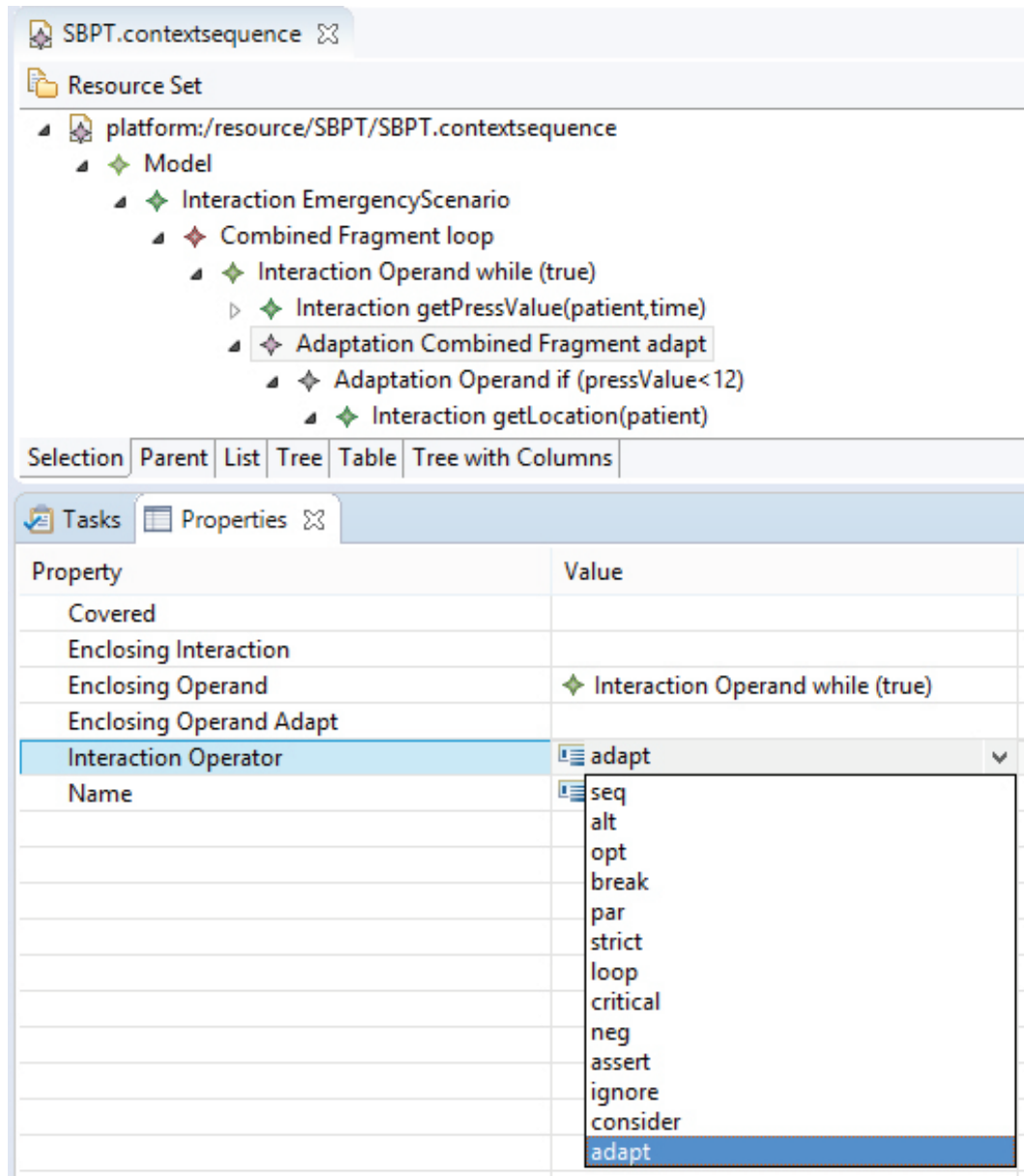


Figure 8.10. The ContextSequence Eclipse editor: Defining an adaptation combined fragment

In the meantime, the modeling process is finished once all elements are added correctly.

Figure 8.11 presents the ContextSequence model of the hypertensive crisis scenario.



Figure 8.11. The ContextSequence Eclipse editor: Modeling the hypertensive crisis scenario

8.8 Summary

In the interest of instantiating the concepts of our modeling approach proposal, we relied on a real context-aware application of a Smart Blood Pressure Tracker, which is able to monitor blood pressure values regularly to decrease hypertension crisis damage probabilities. This example holds the main adaptation requirements of a context-aware system in ubiquitous environments.

Using the CUMML diagrams, our objective was to clarify how to apply the new diagram notations in practical ways.

By virtue of the ContextClass diagram, we showed that objects are characterized by not only attributes and operations, but also by some additional features related to context, these features describe the relationships between the application and the contextual parameters of which the variation may have a significant impact on the applications itself.

The ContextSequence diagram shows clearly the contextual information flows within a system including the methods of communication between context-aware systems and sensors, and how do the formers acquire the relevant contextual information from the latters. It also indicates the output adaptation behaviors which are executed depending on contextual variation events.

Chapter 9

Conclusion and Future Perspectives

"Once we accept our limits, we go beyond them"

Albert EINSTEIN

Contents

9.1	Research summary	112
9.2	Evaluation criteria revisited	113
9.3	Contribution to knowledge	115
9.4	Future works	116

9.1 Research summary

Context-awareness feature has been gaining huge importance in recent years, it becomes the main ingredient to enable next generation applications to fulfil the needs of making the information available everywhere and at any moment, this requires to sense different information from the environment and use the gotten information to adapt applications behavior automatically. Conversely, software engineers are still facing difficulties when designing context-aware systems, existing modeling approaches and tools do not deal with all context-awareness requirements in an adequate manner.

Throughout the present research work, we recalled the fundamental concepts of the domain, as well as several modeling standards which we judged requisite to familiarize the reader of this thesis to the work it covers. Before moving to the main contributions of our work, we believed it was mandatory to investigate why requirements analysis and modeling are the foundation of context-aware applications development, afterward, we dedicated our attention to present and assess the foremost research works on context handling.

In this thesis we suggested a powerful modeling approach based on UML heavyweight extension, called Context Unified Modeling Language, which has the ultimate purpose of decaying the gap between the real world and models by depicting structures, interactions, and behavior of context-aware systems with suitable notations. In particular, the results of the CUML language present two diagrams we called ContextClass and ContextSequence diagrams, these latter attempt to provide meaningful modeling concepts to cater for the description of future applications, and therefore, helping developers to better understand the contextual constraints and ensure higher adaptation levels. For the sake of verifying the consistency of the future models, the new notations and relationships for each diagram are defined through a MOF-based description of their metamodels.

The ContextClass diagram has been presented as a heavyweight extension of the UML Class diagram, to ease the design of context-aware systems structure. It suggests a new component called ContextClass which extends and replaces the old Class structure. The ContextClass includes beneath the attributes and the operations a new compartment we named monitor, which reveals how applications depend on context parameters and how the values of these parameters may influence the related object's attributes and/or operations.

The thesis contributed with another diagram which extends the UML Sequence diagram to enable the modeling of context-aware systems behavior. The ContextSequence diagram is capable of depicting interactions with context sources, adaptation actions, as well as their constraints using two main concepts. The first one consists in capturing the different contextual information acquisition and distinguishing them from traditional interactions by modeling them with distinct notations. The second concept tackles the adaptation behavior, the use of a dedicated combined fragment enables software engineers to specify adaptation actions and their constraints to be performed in reaction to changes in the application's environment.

Another notable contribution of the thesis lies in the modeling editors that we elaborated. The hierarchical and the graphical tools we presented aim to improve the usability and to explore the semantics of our proposal visually.

Lastly, we demonstrated the usefulness of our method through a real-world case study in the healthcare domain. The Smart Blood Pressure Tracker allowed us to show how the CUMML notations can be applied to model context-aware systems. We used the ContextClass diagram to describe the structure of the SBPT and prepare the context-awareness requirements for the implementation stage by monitoring the contextual aspects that characterize the system. We used the ContextSequence diagram to depict the different interactions with sensors by specifying the context acquisition methods, and also, to model the adaptations which are performed automatically by the system in reaction to contextual variations.

Summing up the results, it can be concluded that CUMML models represent several potentialities of use, they provide improved ways to model context-aware systems structure and behavior using semantic-rich modeling concepts, and which are able to refine context models for different uses.

9.2 Evaluation criteria revisited

Context-aware applications are a new generation of sophisticated technologies of which the interest is to make users' lives better and safer. This research has investigated the context-awareness requirements of such applications and has suggested a new approach to cater for the specification, visualization, and documentation of the structure and the behavior of context-aware applications.

The solution presented in this thesis is assessed by answering the research questions that have been formulated in Chapter 1.

Q1. What is the originality of CUMML and what contributions does it afford among the existing context handling approaches?

A1. To the best of our knowledge, this is the first study to deal with context-awareness requirements by proposing heavyweight extensions to UML diagrams, that is to say, our solution presents an innovative way to effectively customize and enrich native UML diagrams, by applying true modification on their metamodel.

Q2. How can software designers achieve efficient translation of context into models using CUMML diagrams?

A2. CUMML models enable higher abstraction levels, they include simple and explicit notations capable of expressing complex contextual aspects of the system. Describing contextual parameters influence on the application and how does the latter reacts is therefore possible.

Q3. How to use CUMML to properly integrate the contextual parameters and their influence on applications when modeling context-aware systems' structure?

A3. Using the ContextClass diagram, we established new notation called monitor, the latter helps describing the different influence relationships between the application and the diverse context parameters.

Q4. How does CUMML enable software designers to effectively specify details about context changes events and interactions methods between applications and context sensors?

A4. The novel notations of the ContextSequence diagram can help enormously in distinguishing traditional messages from context information conveying messages. They can also depict the context acquisition methods (synchronous or asynchronous).

Q5. How can CUMML diagrams used to describe autonomous adaptation behavior of context-aware applications?

A5. We can describe adaptation behavior by using dedicated notations, the adaptation combined fragment captures automatic adaptation behaviors distinctly while specifying the contextual events that constrain them.

Q6. How is it possible for applications modelers and academic institutions to concretely build and share their models using CUMML?

A6. We created a set of hierarchical and graphical modeling editors that help designers exploiting the CUMML expressiveness through the creation of concrete models.

Q7. Are the resulting models conform to specific syntax rules? How to ensure that?

A7. Among the advantages of using MOF-based metamodels resides in the ability to define the rules that verify the correctness of the future models, otherwise they will be ad hoc. Our approach distinguishes itself by being defined through a MOF-based metamodels for both ContextClass and ContextSequence diagrams. This method is an effective way to enforce the formalisms and the syntax convention, all models are thus conform to the related metamodel.

9.3 Contribution to knowledge

This thesis' findings can be summarized as follows:

- Conducts a literature review about context-aware computing and adaptive systems.
- Analyzes different OMG standards including the MDA approach, UML Class and Sequence diagrams, as well as UML extension mechanisms.
- Provides thorough study of several software engineering concepts such as the separation of concerns and requirement engineering novelties in context-aware systems.
- Presents a set of significant research works that attempt to handle context, and specifies their assets and weaknesses.
- Demonstrates how application developers are tending to anticipate context handling since earlier design phases.
- Suggests a definition for the notion of context by enumerating several parameters of which the changes can be significantly influencing on the application's autonomy.
- Addresses the limitations preventing the existing UML Class and Sequence diagrams from modeling context-awareness concerns in an accurate manner.
- Proposes an extension of the UML Class diagram to describe the structure of context-aware systems by monitoring the contextual aspects that characterize them.
- Presents an extension of the UML Sequence diagram to capture the behavior and the interactions of context-aware systems with clear models.
- Furnishes hierarchical and graphical modeling editors for the two proposed diagrams.
- Suggests a considerable real-life case study presented as a smart system for tracking users' blood pressure, and investigates the system's requirements.
- At last, the thesis demonstrates the feasibility of the proposed approach by applying the ContextClass and ContextSequence diagrams notations on the case study example.

9.4 Future works

The advent of smart technologies is changing the ways of producing and consuming information. Context-sensitive computing is the first channel affected by these changes, and as a result, the outlook is enormous. Our main purpose is to reduce the gap between the users' needs and the conceptual model of the system to be created.

On the basis of the promising findings presented in this thesis, further study would be of interest. In our future research we intend to concentrate on four areas:

- Improve the semantics of the novel concepts by including constraints expressed in the Object Constraint Language (OCL), this may support the validation of the proposed approach, moreover, we may validate it by testing it in rich scenarios in different domains (medical, university, banking sector).
- UML includes other important types of diagrams. Our future work involves extending other UML diagrams with new concepts and vocabularies to cover supplementary views of context-aware systems including the Activity Diagram and the Use Case Diagram. This will make CUMML a more complete tool for specifying and designing context-aware systems.
- Another possible perspective concerns the elaboration of a development framework. Requirements analysis and modeling are the foundation of context-aware applications development, but the implementation phase is also necessary to generate executable code. We aim to enforce our proposal by employing integrated development environment tools (IDEs) such as Eclipse, Netbeans, and Visual Studio, to facilitate the development of adaptive applications
- Last but not least, by dint of the present findings and the envisioned works, the next stage of our research will be to concretize the project of the Smart Blood Pressure Tracker into a real context-aware application that has the ultimate purpose of helping hypertensive patients with real-time blood pressure tracking.

Bibliography

- [1] Weiser, M. (1991), "The Computer for the 21st Century", *Scientific American*, Vol. 265 No. 3, pp. 94-105.
- [2] Schilit, B.N. and Theimer, M.M. (1994), "Disseminating active map information to mobile hosts", *IEEE Network*, Vol. 8, No. 5, pp. 22-32.
- [3] Schilit, B.N., Adams, N.L. and Want, R. (1994), "Context-aware computing applications", *IEEE 1st Workshop on Mobile Computing Systems and Applications (WMCSA 1994)*, IEEE, Santa Cruz, CA, December 8-9, pp. 85-90.
- [4] Brown, P.J., Bovey, J.D. and Chen, X. (1997), "Context-Aware Applications: From the Laboratory to the Marketplace", *IEEE Personal Communications*, Vol. 4, No. 5, pp. 58-64.
- [5] Pascoe, J. (1998), "Adding generic contextual capabilities to wearable computers", In *Second International Symposium on Wearable Computers*, IEEE Computer Society, Pittsburgh, USA, October, Vol. 44, pp. 92-99.
- [6] Brézillon, P. and Pomerol, J. C. (1999), "Contextual knowledge sharing and cooperation in intelligent assistant systems", *Le Travail Humain*, Vol. 62, No. 3, pp. 223-246.
- [7] Schmidt, A. (2000), "Implicit human computer interaction through context", *Personal Technologies*, Vol. 4, No. 2-3, pp. 191-199.
- [8] Dix, A., Rodden, T., Davies, N., Trevor, J., Friday, A. and Palfreyman, K. (2000), "Exploiting space and location as a design framework for interactive mobile systems", *ACM Transactions on Computer Human Interaction (TOCHI)*, Vol. 7, No. 3, pp. 285-321.

- [9] Dey, A.K. and Abowd, G.D. (2000), "Towards a better understanding of context and context-awareness", In *CHI 2000's Workshop on The What, Who, Where, When, Why and How of Context-awareness*, ACM Press, The Hague, Netherlands, April, pp.1-6.
- [10] Winograd, T. (2001), "Architectures for Context", *Human-Computer Interaction*, Vol. 16, No. 2, pp. 401-419.
- [11] Mostéfaoui, K., Pasquier-Rocha, J. and Brézillon, P. (2004), "Context-aware computing: a guide for the pervasive computing community", In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (IPCS'04)*, USA, p. 39-48.
- [12] Lemlouma, T. (2004), "Architecture de négociation et d'adaptation de Services Multimédia dans des Environnements Hétérogènes", Doctoral dissertation, Institut National Polytechnique de Grenoble-INPG, France.
- [13] Chaari, T., Laforest, F. and Flory, A. (2005), "Adaptation des applications au contexte en utilisant les services Web", In *Proceedings of the 2nd French-speaking conference on mobility and ubiquity computing (UbiMob '05)*, ACM, Grenoble, France, June, p. 111-118.
- [14] Strassner, J., Meer, S., Sullivan, D.O. and Dobson, S. (2009), "The use of context-aware policies and ontologies to facilitate business-aware network management", *Journal of Network and Systems Management*, Vol. 17, No. 3, pp. 255-284.
- [15] Ryan, N.S., Pascoe, J and Morse, D.R. (1998), "Enhanced reality fieldwork : the context-aware archaeological assistant", In *Computer Applications in Archaeology*, British Archeological Reports, Tempus Reparatum, pp. 1-9.
- [16] Salber, D., Dey, A.K. and Abowd, G.D. (1998), "Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm", Georgia Tech GVU Technical Report GIT-GVU-98-01, Georgia Institute of Technology.
- [17] Lieberman, H. and Selker, T. (2000), "Out of context : computer systems that adapt to, and learn from, context", *IBM Systems Journal*, Vol. 39, No. 3.4, pp. 617-632.
- [18] Korkea-Aho, M. (2000), "Context-aware applications survey", available at: www.cse.tkk.fi/fi/opinnot/T-110.5190/2000/applications/context-aware.html (accessed April 2018).

- [19] Dey, A.K. (2001), "Understanding and using context", *Personal and Ubiquitous Computing*, Vol. 5 No. 1, pp. 4-7.
- [20] Burrell, J. and Gay, G. K. (2001), "Collectively defining context in a mobile, networked computing environment", In *CHI'01 extended abstracts on Human factors in computing systems*, ACM, Seattle, USA, March, pp. 231-232.
- [21] Cheverest, K., Mitchell, K. and Davies, N. (2002), "The role of adaptive hypermedia in a context-aware tourist guide", *Communications of the ACM*, Vol. 45, No. 5, pp. 47-51.
- [22] Rohn, E. (2003), "Predicting context aware computing performance", *Ubiquity*, pp. 1-17.
- [23] Barkhuus, L. (2003), "Context information vs. sensor information: A model for categorizing context in context-aware mobile computing", *SIMULATION SERIES*, Vol. 35, No. 1, pp. 127-133.
- [24] Abbas, K., Verdier, C. and Flory, A. (2007), "Exploiting profile modeling for web-based information systems", In *International Conference on Web Information Systems Engineering*, Springer, Berlin, Germany, December, pp. 313-324.
- [25] Christopoulou, E. (2009), "Context as a necessity in mobile applications", In *Mobile Computing: Concepts, Methodologies, Tools, and Applications*, IGI Global, pp. 65-83.
- [26] Sindico, A. (2009), "Model Driven Development of Context Aware Software Systems", Doctoral thesis, Università Degli Studi Di Roma "Tor Vergata", Roma, Italy.
- [27] Brown, M. (1996), "Supporting User Mobility", In *Mobile Communications, The International Federation for Information Processing (IFIP)*, Springer, Boston, USA, pp. 69-77.
- [28] Cooperstock, J., Tanikoshi, K., Beirne, G., Narine, T. and Buxton, W (1995), "Evolution of a Reactive Environment", In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '95)*, ACM Press/Addison-Wesley Publishing Co., pp. 170-177.
- [29] Elrod, S., Hall, G., Costanza, R., Dixon, M. and Des Rivieres, J. (1993), "Responsive Office Environments", *Communications of the ACM (CACM)*, Vol. 36, No. 7, pp. 84-85.

- [30] Hull, R., Neaves, P. and Bedford-Roberts, J. (1997), "Towards Situated Computing", In *1st International Symposium on Wearable Computers*, IEEE, Boston, USA, October, pp. 146-153.
- [31] Rekimoto, J., Ayatsuka, Y. and Hayashi, K. (1998), "Augment-able Reality: Situated Communication through Physical and Digital Spaces", In *2nd International Symposium on Wearable Computers*, IEEE, Pittsburgh, USA, October, pp. 68-75.
- [32] Fickas, S., Korteum, G. and Segall, Z. (1997), "Software Organization for Dynamic and Adaptable Wearable Systems", In *1st International Symposium on Wearable Computers*, IEEE, Boston, USA, October, pp. 56-63.
- [33] Chen, G. and Kotz, D. (2000), "A survey of context-aware mobile computing research", Technical Report TR2000-381, Department of Computer Science, Dartmouth College.
- [34] Petrelli, D., Not, E., Strapparava, C., Stock, O. and Zancanaro, M. (2000), "Modeling Context is Like Taking Pictures", In *CHI 2000's Workshop on The What, Who, Where, When, Why and How of Context-awareness*, ACM Press, The Hague, Netherlands, April.
- [35] Gwizdka, J. (2000), "What's in the Context?", In *CHI 2000's Workshop on The What, Who, Where, When, Why and How of Context-awareness*, ACM Press, The Hague, Netherlands, April.
- [36] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J. and Retschitzegger, W. (2003), "Context-awareness on mobile devices-the hydrogen approach", In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, IEEE, Hawaii, January, p. 10.
- [37] Henricksen, K. (2003), "A framework for context-aware pervasive computing applications", Queensland: University of Queensland.
- [38] Razzaque, M.A., Dobson, S. and Nixon, P. (2006), "Categorization and modelling of quality in context information", In *Proceedings of the IJCAI 2005 Workshop on AI and Autonomic Communications*, Edinburgh, Scotland, August 2005.

- [39] Benselim, M.S. and Seridi-Bouchelaghem, H. (2011), "Development of context-aware applications in ubiquitous information systems", *Proceeding of the 13th International Conference on Enterprise Information Systems (ICEIS vol. 3)*, Beijing, June 8-11, pp. 223-228.
- [40] Geihs, K. and Wagner, M. (2012), "Context-awareness for self-adaptive applications in ubiquitous computing environments", *International Conference on Context-Aware Systems and Applications (ICCASA 2012)*, Springer, Ho Chi Minh City, November 26-27, pp. 108-120.
- [41] Sánchez-Pi, N., Carbó, J. and Molina, J.M. (2012), "A knowledge-based system approach for a context-aware system", *Knowledge-Based Systems*, Vol. 27, pp. 1-17.
- [42] Dey, A.K., Abowd, G.D. and Wood, A. (1999), "CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services", *Knowledge-Based Systems*, Vol. 11, No. 1, pp. 3-13.
- [43] Layaïda, N. (1999), "Adaptabilité. Pistes d'étude pour la définition d'une infrastructure d'accès au contenu multimédia pour des machines hétérogènes", Technical Report. Grenoble : INRIA Rhône-Alpes, France, October.
- [44] Satyanarayanan, M. (2001), "Pervasive computing: Vision and challenges", *IEEE Personal communications*, Vol. 8, No. 4, pp. 10-17.
- [45] Capra, L., Emmerich, W. and Mascolo, C. (2001), "Reflective middleware solutions for context-aware applications", In *International Conference on Metalevel Architectures and Reflection*, Springer, Berlin, Germany, September, pp. 126-133.
- [46] Dejene Ejigu, D. (2007), "Services pervasifs contextualisés: modélisation et mise en uvre", Doctoral dissertation, Villeurbanne, INSA, France.
- [47] Chassot, C., Drira, K. and Guennoun, K. (2006), "Towards autonomous management of QoS through model-driven adaptability in communication-centric systems", *International Transactions on Systems Science and Applications (ITSSA)*, Vol. 2, No. 3, pp. 255-264.

- [48] Chaari, T. (2007), "Adaptation d'applications pervasives dans des environnements multi-contextes", PhD thesis, L'institut national des sciences appliquées de Lyon, laboratoire LIRIS, France.
- [49] Fernando, N., Loke, S.W. and Rahayu, W. (2013), "Mobile cloud computing: A survey", *Future generation computer systems*, Vol. 29, No. 1, pp. 84-106.
- [50] Perera, C., Zaslavsky, A., Christen, P. and Georgakopoulos, D. (2014), "Context aware computing for the internet of things: A survey", *IEEE communications surveys tutorials*, Vol. 6, No. 1, pp. 414-454.
- [51] Bricon-Souf, N. and Newman, C.R. (2007), "Context awareness in health care: A review", *International Journal of Medical Informatics*, Vol. 76, No. 1, pp. 2-12.
- [52] Truong, H. L. and Dustdar, S. (2009), "A survey on context-aware web service systems", *International Journal of Web Information Systems*, Vol. 5, No. 1, pp. 5-31.
- [53] Strang, T. and Linnhoff-Popien, C. (2004), "A Context Modeling Survey", In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management (UbiComp 2004)*, Nottingham, September 7, pp. 31-41.
- [54] Want, R., Hopper, A., Falcao, V. and Gibbons, J. (1992), "The Active Badge Location System", *ACM Transaction on Information Systems*, Vol. 10, No. 1, pp. 42-47.
- [55] Abowd, G.D., Atkeson, C.G., Hong, J.I., Long, S., Kooper, R. and Pinkerton, M. (1997), "Cyberguide: A mobile context-aware tour guide", *Wireless Networks*, Vol. 3, No. 5, pp. 421-433.
- [56] Long, S., Kooper, R., Abowd, G.D. and Atkeson, C.G. (1996), "Rapid Prototyping of Mobile Context-aware Applications: The Cyberguide Case Study", In *2nd International Conference on Mobile Computing and Networking (MobiCom'96)*, ACM, White Plains, USA, November, pp. 97-107.
- [57] Schilit, W.N. (1995), "A system architecture for context-aware mobile computing", Doctoral dissertation, Columbia University.
- [58] Alegre, U., Augusto, J.C. and Clark, T. (2016), "Engineering context-aware systems and applications: A survey", *Journal of Systems and Software*, Vol. 117, pp. 55-83.

- [59] Temdee, P. and Prasad, R. (2018), "Context-Aware Communication and Computing: Applications for Smart Environment", 1st ed. Springer International Publishing.
- [60] Baldauf, M. and Dustdar, S. (2007), "A survey on context-aware systems", *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4, pp. 263-277.
- [61] Dey, A.K., Abowd, G.D. and Salber, D. (2001), " A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications" , *Human-Computer Interaction*, Vol. 16 No. 2, pp. 97-166.
- [62] Ailisto, H., Alahuhta, P., Haataja, V., Kyllonen, V. and Lindholm, M. (2002), "Structuring context aware applications: Five-layer model and example case", In *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing*, Goteborg, Sweden, pp. 1-5.
- [63] Indulska, J. and Sutton, P. (2003), "Location management in pervasive systems", In *Proceedings of the Australasian Information Security Workshop (CRPITS'03)*, Adelaide, Australia, pp.143-151.
- [64] Schmidt, A. and Van Laerhoven, K. (2001), "How to build smart appliances?", *IEEE Personal Communications*, Vol. 8, No. 4, pp. 66-71.
- [65] Rabail, T. (2013), "Context aware mobile computing as a challenge for developers and software engineers: a review", *European Scientific Journal*, Vol. 4, pp. 534-542.
- [66] Bézivin, J. (2004), "In Search of a Basic Principle for Model Driven Engineering", *The European Journal for the Informatics Professional*, Vol. 5, No. 2, pp. 21-24.
- [67] Bézivin, J. (1998), "On different interoperability modes in software engineering: The case of modeling activities at OMG", In *Proceedings of Software Engineering'98*, Paris, December.
- [68] Bézivin, J. and Gerbé, O. (2001), "Towards a precise definition of the OMG/MDA framework", In *16th Annual International Conference on Automated Software Engineering (ASE)*, IEEE, San Diego, USA, November, pp. 273-280.
- [69] Favre, J.M. (2004), "Towards a basic theory to model driven engineering", In *3rd Workshop on Software Model Engineering (WISME)*, pp. 262-271.

- [70] OMG MDA (2014), *OMG MDA Guide revision 2.0*, available at: www.omg.org/cgi-bin/doc?ormsc/14-06-01 (accessed September 2016).
- [71] Benyahia, A. (2012), "Contribution à la mise en uvre d'un moteur d'exécution de modèles UML pour la simulation d'applications temporisées et concurrentes", Doctoral dissertation, Supélec, France.
- [72] Object Management Group (2013), *Meta Object Facility (MOF) specification, version 2.4.1*, available at: www.omg.org/spec/MOF/2.4.1/ (accessed September 2016).
- [73] Crawley, S., Davis, S., Indulska, J., McBride, S. and Raymond, K. (1997), "Meta-meta is better better!", In *Proceedings of the IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'91)*, Cottbus, Germany, September.
- [74] Object Management Group (2011), *Unified Modeling Language (UML) specification, version 2.4.1*, available at: www.omg.org/spec/UML/2.4.1/ (accessed September 2016).
- [75] Taha, S. (2008), "Modélisation conjointe logiciel/matériel de systèmes temps réel", Doctoral dissertation, Lille 1, France.
- [76] Benselim, M.S. and Seridi-Bouchelaghem, H. (2009), "Contextual adaptation of ubiquitous information systems", *IEEE International Conference on Multimedia Computing and Systems (ICMCS'09)*, IEEE, Ouarzazate, April 2-4, pp. 17-22.
- [77] Kolovos, D. S., Paige, R. F. and Polack, F. A. (2006), "Merging Models with the Epsilon Merging Language (EML)", In *International Conference on Model Driven Engineering Languages and Systems*, Springer, Berlin, Germany, October, pp. 215-229.
- [78] Booch, G. (1994), "The Booch method: process and pragmatics", In *Object development methods*, SIGS Publications, pp. 149-166.
- [79] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.E. (1991), "Object-oriented modeling and design", Englewood Cliffs, NJ: Prentice-hall, Vol. 199, No. 1.
- [80] Jacobson, I. (1993), "Object-oriented software engineering: a use case driven approach", Pearson Education India.

- [81] Booch, G., Rumbaugh, J. and Jacobson, I. (2005), "The Unified Modeling Language User Guide", 2nd ed., Addison-Wesley Professional.
- [82] Fowler, M. (2004), "UML distilled: a brief guide to the standard object modeling language", 3rd ed., Addison Wesley Professional.
- [83] Fuentes-Fernández, L. and Vallecillo-Moreno, A. (2004), "An Introduction to UML Profiles", *UML and Model Engineering*, vol. 2, pp. 6-13.
- [84] Bruck, J. and Hussey, K. (2007), "Customizing UML: which technique is right for you?", International Business Machines Corporation, available at: www.eclipse.org/modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html (accessed September 2016).
- [85] Magableh, A.A., Shukur, Z. and Ali, N.M. (2012), "Heavy-Weight and Light-weight UML Modelling Extensions of Aspect-Orientation in the Early Stage of Software Development", *Journal of Applied Sciences*, Vol. 12, No. 21, pp. 2195-2201.
- [86] Object Management Group (2014), Object Constraint Language (OCL) specification, version 2.4, available at: www.omg.org/spec/OCL/2.4/ (accessed September 2016).
- [87] Pérez-Martínez, J.E. (2003), "Heavyweight extensions to the UML metamodel to describe the C3 architectural style", *ACM SIGSOFT Software Engineering Notes*, Vol. 28, No. 3, p. 5.
- [88] Da Silva, V.T. and De Lucena, C.J.P. (2004), "Extending UML to Model Multi-Agent Systems", Computer Science Department, Pontifical Catholic University, Rio de Janeiro, Brazil.
- [89] Burgués, X., Franch, X. and Ribó, J.M. (2005), "A MOF-compliant approach to software quality modeling", In *International Conference on Conceptual Modeling*, Springer, Klagenfurt, Austria, October, pp. 176-191.
- [90] Al-Kady, M., Bahgat, R. and Fahmy, A. (2008), "A UML Heavyweight Extension for MAS Modeling", In *The 8th International Conference on Quality Software*, IEEE, August, pp. 435-440.

- [91] Object Management Group (2003), Common Warehouse Metamodel (CWM) specification, version 1.1, available at: www.omg.org/spec/CWM/1.1/ (accessed September 2016).
- [92] Rui, W., Xiao-Guang, M., Zi-Ying, D. and Yan-Ni, W. (2009), "Extending UML for aspect-oriented architecture modeling", *Proceedings of the 2nd International Workshop on Computer Science and Engineering*, IEEE, Qingdao, China, October, pp. 362-366.
- [93] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. (2010), "A survey of context modelling and reasoning techniques", *Pervasive and Mobile Computing*, Vol. 6 No. 2, pp. 161-180.
- [94] Taconet, C. (2011), "Intergiciels pour la sensibilité au contexte en environnement ubiquitaire", Habilitation Dissertation, Université d'Évry Val d'Essonne, Évry, France.
- [95] Parnas, D. (1972), "On the criteria to be used in decomposing systems in modules", *Communication on the ACM*, Vol. 15, No. 12, pp. 1053-1058.
- [96] Chaari, T., Laforest, F. and Celentano, A. (2004), "Design of context-aware applications based on web services", Technical Report RR-2004-033, INSA Lyon, France.
- [97] Chaari, T., Laforest, F. and Celentano, A. (2008), "Adaptation in context-aware pervasive information systems: the SECAS project", *International Journal of Pervasive Computing and Communications*, Vol. 3, No. 4, pp. 400-425.
- [98] Bassil, Y. (2012), "A simulation model for the waterfall software development life cycle", *International Journal of Engineering Technology*, Vol. 2, No. 5, pp. 1-7.
- [99] Breitman, K., Leite, J. and Finkelstein, A. (1999), "The World a Stage: A Survey of Requirements Engineering Using a Real-life Case Study", *Journal of the Brazilian Computer Society*, Vol. 6, No. 1, pp. 13-37.
- [100] Finkelstein, A. and Kramer, J. (2000), "Software Engineering: a roadmap", A. Finkelstein Ed., ACM Press, pp. 3-24.
- [101] Benselim, M.S. and Seridi-Bouchelaghem, H. (2013), "Extending UML class diagram notation for the development of context-aware applications", *Journal of Emerging Technologies in Web Intelligence*, Vol. 5 No. 1, pp. 35-44.

- [102] Ou, S., Georgalas, N., Azmoodeh, M., Yang, K. and Sun, X. (2006), "A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development", In *European Conference on Model Driven Architecture-Foundations and Applications*, Springer, Bilbao, Spain, July, pp. 188-197.
- [103] Ceri, S., Florian, D., Matera, M. and Facca, F. (2007), "Model driven development of context-aware web applications", *ACM Transactions in Internet Technology (TOIT)*, Vol. 7, No. 1, p. 2.
- [104] Vale, S. and Hammoudi, S. (2008), "Context-aware Model Driven Development by Parameterized Transformation", *Proceedings of the First International Workshop on Model Driven Interoperability for Sustainable Information Systems (MDISIS'08)*, Montpellier, France, June, pp. 121-133.
- [105] Simons, C. (2007), "CMP: a UML context modeling profile for mobile distributed systems", *IEEE Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007)*, IEEE, Hawaii, January 3-6, pp. 289b-289b.
- [106] Sindico, A. and Grassi, V. (2009), "Model driven development of context aware software systems", *International Workshop on Context-Oriented Programming (COP 2009)*, ACM, Genoa, July 7, p. 7.
- [107] Al-alshuhai, A. and Siewe, F. (2015), "An extension of the use case diagram to model context-aware applications", In *SAI Intelligent Systems Conference (IntelliSys)*, IEEE, London, November, pp. 884-888.
- [108] Al-alshuhai, A. and Siewe, F. (2015), "An Extension of UML Activity Diagram to Model the Behaviour of Context-Aware Systems", In *IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, IEEE, Liverpool, October, pp. 431-437.
- [109] Al-alshuhai, A. and Siewe, F. (2015), "An Extension of Class Diagram to Model the Structure of Context-Aware Systems", In *The Sixth International Joint Conference on Advances in Engineering and Technology (AET-2015)*.

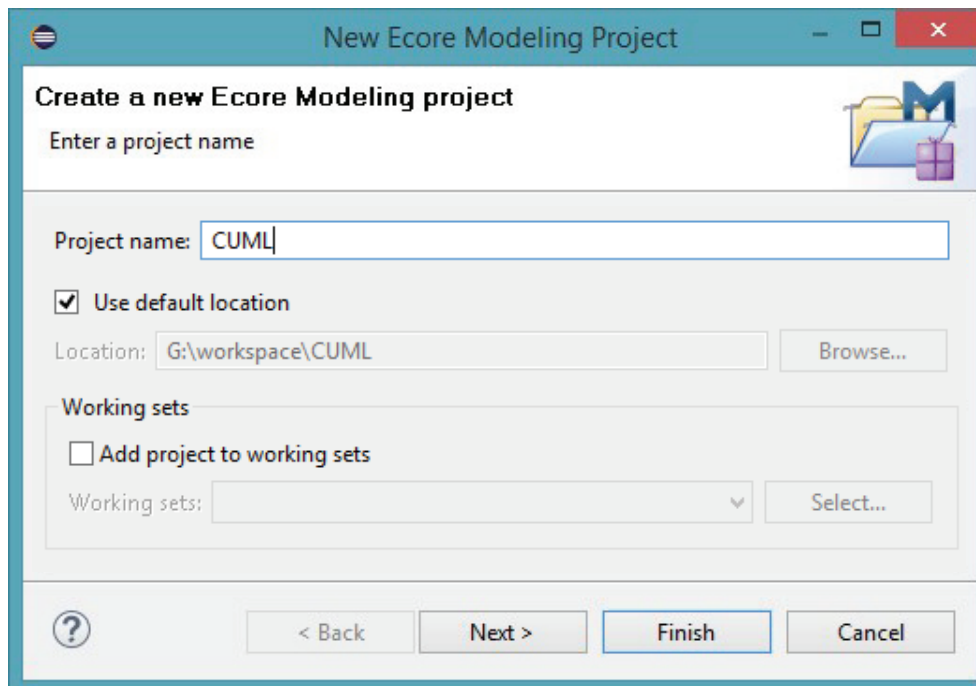
- [110] Hong, D., Chiu, D.K. and Shen, V.Y. (2005), "Requirements elicitation for the design of context-aware applications in a ubiquitous environment", In *Proceedings of the 7th international conference on Electronic commerce*, ACM, Xi'an, China, August, pp. 590-596.
- [111] Rey, G. and Coutaz, J. (2004), "Le contexteur : capture et distribution dynamique d'information contextuelle", In *Proceedings of the 1st French-speaking conference on mobility and ubiquity computing (UbiMob '04)*, ACM, Nice, France, June, pp. 131-138.
- [112] Benselim, M.S. (2009), "Une approche pour le développement d'applications sensibles au contexte", *The 27th congress of INFORSID*, Toulouse, May 26-29, pp. 479-480.
- [113] Ambler, S.W. (2004), "Are You Ready For the MDA?", available at: www.agilemodeling.com/essays/readyForMDA.htm (accessed June 2017).
- [114] Boudjemline, H., Touahria, M., Boubetra, A. and Kaabeche, H. (2017), "Heavyweight extension to the UML class diagram metamodel for modeling context aware systems in ubiquitous computing", *International Journal of Pervasive Computing and Communications*, Vol. 13, No. 3, pp. 238-251.
- [115] Misbhauddin, M. and Alshayeb, M. (2015), "UML model refactoring: a systematic literature review", *Empirical Software Engineering*, Vol. 20, No. 1, pp. 206-251.
- [116] Oh, Y., Schmidt, A. and Woo, W. (2007), "Designing, Developing and Evaluating Context-Aware Systems", In *Proceedings of the International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, IEEE, Seoul, South Korea, April, pp. 1158-1163.
- [117] Milner, R. (2001), "Bigraphical Reactive Systems: Basic Theory", Technical Report 523, University of Cambridge, Computer Laboratory, England.
- [118] Eclipse Modeling Framework (EMF) (2016), available at: www.eclipse.org/modeling/emf/ (accessed September 2016).
- [119] UMLet 14.2 (2017), available at: www.umlet.com (accessed April 2018).

- [120] Auer, M., Tschurtschenthaler, T. and Biffel, S. (2003), "A flyweight UML modelling tool for software development in heterogeneous environments", In *Proceedings of the 29th EUROMICRO Conference*, IEEE, Belek-Antalya, Turkey, September, pp. 267-272.
- [121] Auer, M., Meyer, L. and Biffel, S. (2007), "Explorative UML Modeling-Comparing the Usability of UML Tools", In *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS'07)*, Madeira, Portugal, June, pp. 466-473.
- [122] Singhala, P., Shah, D. and Patel, B. (2014), "Temperature Control using Fuzzy Logic", *International Journal of Instrumentation and Control Systems*, Vol.4, No.1.
- [123] Haux R. (2006), "Individualization, globalization and health about sustainable information technologies and the aim of medical informatics", *International Journal of Medical Informatics*, Vol. 75, No. 12, pp. 795-808.

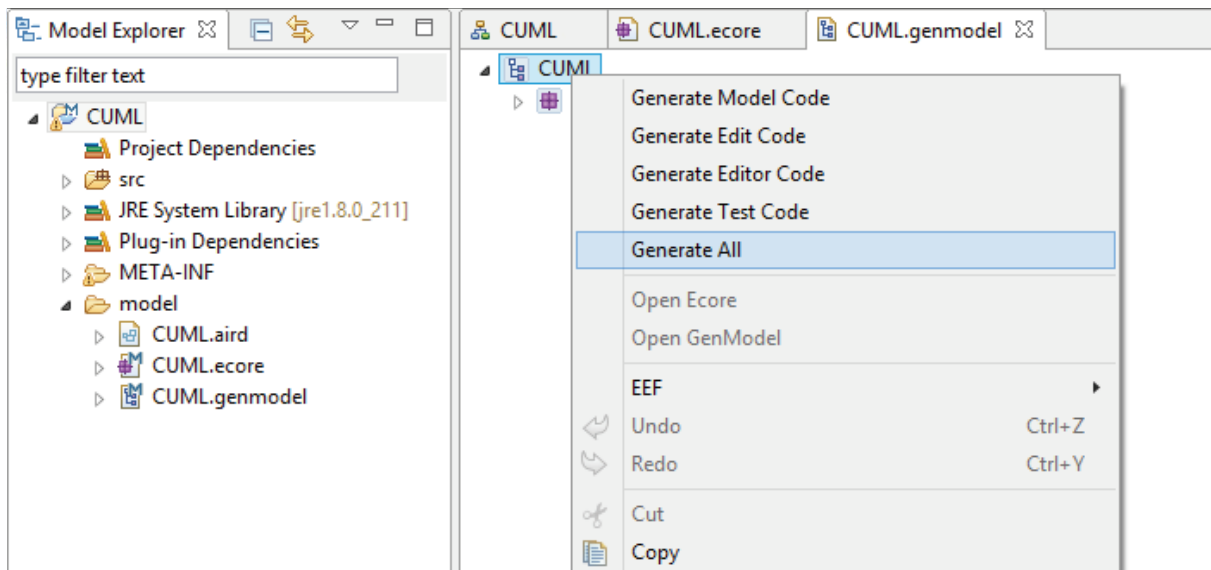
Appendix A

Implementing metamodels and creating modeling editors in Eclipse

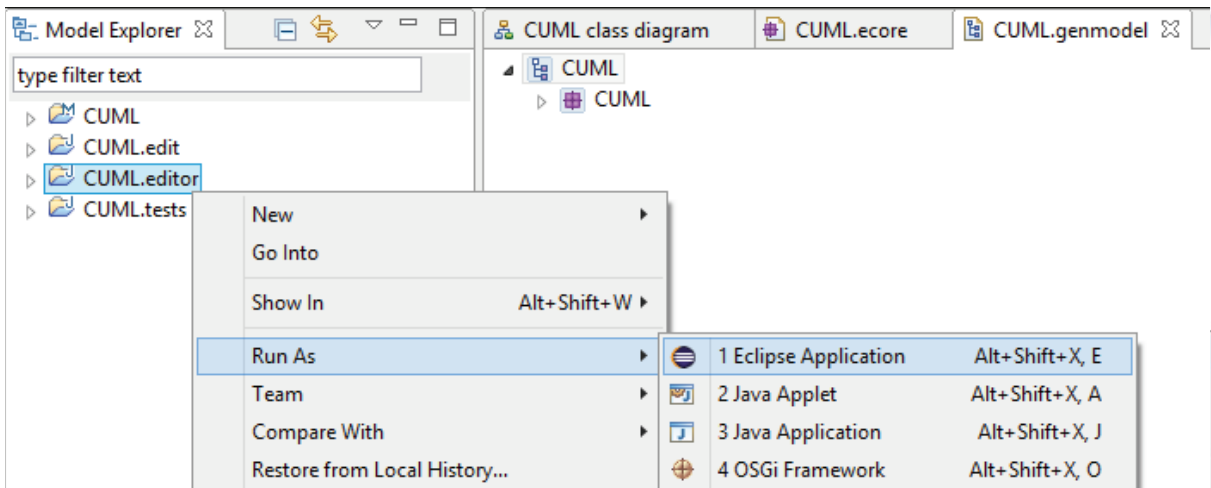
In the following, sample Eclipse screenshot showing the easiest way to generate modeling editors based on self-created metamodels. We start by creating a new *Ecore Modeling Project* named *CUML* for instance, this will create *CUML.aird*, *CUML.core*, and *CUML.genmodel* files.



We proceed to draw the metamodel inside the *CUML.aird* design panel using constructs such as *EClass*, *EAttribute*, *EOperation*, *ESuperType*, *EReference*, and *EEnumeration*, then we validate the *CUML.core* file. Afterward, we generate the corresponding modeling editor using the *CUML.genmodel*.



Once generated, the modeling editor can be launched by running it as an Eclipse Application.



Appendix B

Creating new primitive shapes in UMLet

To illustrate how to introduce new shapes to UMLet, the following set of Java codes shows how to draw the shapes of ContextSequence diagram's interactions.

Asynchronous message

```
    setLineType(0);
drawLine(5, 20, width-2, 20);
setLineType(0);
int y=textHeight();
for(String textline : textlines) {
    if (textline.equals("_LtR")){
        drawLine(width-10, 11, width, 20);
        drawLine(width-10, 29, width, 20);}
}

    for(String textline : textlines) {
        if (textline.equals("_RtL")){
            drawLine(10, 11, 0, 20);
            drawLine(10, 29, 0, 20);}
    }
for(String textline : textlines) {
    y += printCenter(textline,10);
    break;
}
```

Synchronous message

```
    int y=textHeight();
    Polygon p = new Polygon();
    for(String textline : textlines) {
        if (textline.equals("_LtR")){
            setBackgroundColor("Black");
            setLineType(0);
            drawLine(0, 21, width-10, 21);
            setLineType(0);
            p.addPoint(width-10, 11);
            p.addPoint(width-10, 29);
            p.addPoint(width, 20);
            drawPolygon(p);
        }
    }
    for(String textline : textlines) {
        if (textline.equals("_RtL")){
            setBackgroundColor("Black");
            setLineType(0);
            drawLine(10, 21, width, 21);
            setLineType(0);
            p.addPoint(10, 11);
            p.addPoint(10, 29);
            p.addPoint(0, 20);
            drawPolygon(p);
        }
    }
    for(String textline : textlines) {
        y += printCenter(textline,10);
        break;
    }
}
```

Message reply

```
    setLineType(1);
drawLine(5, 20, width-5, 20);
setLineType(0);
int y=textHeight();
for(String textline : textlines) {
    if (textline.equals("_LtR")){
        drawLine(width-10, 11, width, 20);
        drawLine(width-10, 29, width, 20);}
}
for(String textline : textlines) {
    if (textline.equals("_RtL")){
        drawLine(10, 11, 0, 20);
        drawLine(10, 29, 0, 20);}
}
for(String textline : textlines) {
    y += printCenter(textline,10);
    break;
}
```

Context notification

```
    setLineType(0);
drawLine(5, 18, width-5, 18);
setLineType(0);
drawLine(5, 23, width-5, 23);
setLineType(0);
int y=textHeight();
for(String textline : textlines) {
    if (textline.equals("_LtR")){
        drawLine(width-10, 11, width, 20);
        drawLine(width-10, 29, width, 20);}
}
for(String textline : textlines) {
    if (textline.equals("_RtL")){
        drawLine(10, 11, 0, 20);
        drawLine(10, 29, 0, 20);}
}
for(String textline : textlines) {
    y += printCenter(textline,10);
    break;
}
```

Context query

```
int y=textHeight();
Polygon p = new Polygon();
for(String textline : textlines) {
    if (textline.equals("_LtR")){
        setBackgroundColor("Black");
        setLineType(0);
        drawLine(5, 18, width-10, 18);
        setLineType(0);
        drawLine(5, 23, width-10, 23);
        setLineType(0);
        p.addPoint(width-10, 11);
        p.addPoint(width-10, 29);
        p.addPoint(width, 20);
        drawPolygon(p);
    }
}
for(String textline : textlines) {
    if (textline.equals("_RtL")){
        setBackgroundColor("Black");
        setLineType(0);
        drawLine(10, 18, width, 18);
        setLineType(0);
        drawLine(10, 23, width, 23);
        setLineType(0);
        p.addPoint(10, 11);
        p.addPoint(10, 29);
        p.addPoint(0, 20);
        drawPolygon(p);
    }
}
for(String textline : textlines) {
    y += printCenter(textline,10);
    break;
}
}
```

Context information reply

```
    setLineType(1);
drawLine(5, 18, width-5, 18);
setLineType(1);
drawLine(5, 23, width-5, 23);
setLineType(0);
int y=textHeight();
for(String textline : textlines) {
    if (textline.equals("_LtR")){
        drawLine(width-10, 11, width, 20);
        drawLine(width-10, 29, width, 20);}
}
for(String textline : textlines) {
    if (textline.equals("_RtL")){
        drawLine(10, 11, 0, 20);
        drawLine(10, 29, 0, 20);}
}
for(String textline : textlines) {
    y += printCenter(textline,10);
    break;
}
```

List of Scientific Papers

Journals' papers

- Boudjemline, H., Touahria, M., Boubetra, A. and Kaabeche, H. (2017), Heavyweight extension to the UML class diagram metamodel for modeling context aware systems in ubiquitous computing, *International Journal of Pervasive Computing and Communications*, Vol. 13, No. 3, pp. 238-251.
- Boudjemline, H., Touahria, M., Boubetra, A. and Kaabeche, H., Heavyweight Extension to the UML Sequence Diagram for Modeling Context-Aware Systems, *submitted*.

National conferences

- Boudjemline, H., Touahria, M. and Boubetra, A. (2013), La sensibilité au contexte dans les environnements ubiquitaires, *JIDID Doctoral day, 1st Ed., BBA University*, December, 15th.
- Boudjemline, H., Touahria, M. and Boubetra, A. (2014), Conception des applications sensibles au contexte dans des environnements ubiquitaires avec l'approche orienté-modèle (MDA), *JIDID Doctoral day, 2nd Ed., BBA University*, December, 18th.
- Boudjemline, H., Touahria, M. and Boubetra, A. (2015), Vers une extension de UML pour la modélisation de l'aspect adaptatif d'un système sensible au contexte, *JIDID Doctoral day, 3rd Ed., BBA University*, December, 17th.
- Boudjemline, H., Touahria, M. and Boubetra, A. (2016), Extending UML class diagram notations for modeling context-aware systems in ubiquitous environments, *JIDID Doctoral day, 4th Ed., BBA University*, June, 7th.

ABSTRACT

With the widespread use of mobile devices, a new generation of ubiquitous applications has emerged in daily life activities, making information available everywhere and at any moment. Context-awareness is the feature that allows these applications to be smart, by enabling them to adapt their behavior according to the user's current situation. In the same vein, designing context-aware applications involves new challenges not present in traditional ones, this has propelled the software engineers to introduce additional requirements analysis and enhanced modeling techniques capable of dealing with the different contextual parameters that may affect the application's behavior. The Unified Modeling Language (UML) is the most commonly used language to specify, visualize, and document the artifacts of software systems. Nevertheless, existing UML diagrams are too general to describe context-aware systems adequately.

This thesis proposes a new modeling approach so-called CUML (Context Unified Modeling Language) to cater for the specification, visualization, and documentation of context-aware computing systems. We present a number of contributions through which we have succeeded in (i) A novel notion called ContextClass diagram, which extends the UML Class diagram to model the structure of context-aware systems by monitoring the contextual aspects that characterize them. (ii) A novel notion called ContextSequence diagram, which extends the UML Sequence diagram to model the interactions as well as the adaptation behavior of context-aware systems. (iii) A set of modeling editors. (iv) A case study in the healthcare field to demonstrate the pragmatics of our approach.

Keywords: Context-aware Systems, Ubiquitous Environment, Adaptation, Context Modeling, Requirements Analysis, Software Design.

RESUME

Avec l'utilisation répandue des dispositifs mobiles, une nouvelle génération d'applications ubiquitaire est apparue dans les activités quotidiennes des utilisateurs, rendant ainsi l'information disponible partout et à tout moment. La sensibilité au contexte est la propriété qui permet à ces applications de devenir intelligentes, en leur permettant d'adapter leurs comportements selon la situation actuelle de l'utilisateur. Dans ce sens, la conception d'applications sensibles au contexte comprend de nouveaux défis par rapport aux applications traditionnelles, ce qui a incité les ingénieurs du génie logiciel à introduire des analyses des besoins et des techniques de modélisation améliorées dans le but de gérer les différents paramètres contextuels susceptibles d'affecter le comportement de l'application. Le langage UML (Unified Modeling Language) est le langage le plus couramment utilisé pour spécifier, visualiser et documenter les artefacts des systèmes logiciels. Néanmoins, les diagrammes existants d'UML sont trop généraux pour décrire les systèmes sensibles au contexte correctement.

L'objectif de cette thèse est d'offrir une nouvelle approche de modélisation appelée CUML (Context Unified Modeling Language) pour aider à la spécification, la visualisation, et la documentation des systèmes sensibles au contexte. Nous présentons l'ensemble des contributions aux quelles nous avons abouti (i) Une nouvelle notion appelée diagramme de ContextClass, qui étend le diagramme de Classes de UML pour modéliser la structure des systèmes sensibles au contexte en spécifiant les aspects contextuels qui les caractérisent. (ii) Une nouvelle notion appelée diagramme de ContextSequence, qui étend le diagramme de Séquence de UML pour modéliser les interactions ainsi que le comportement d'adaptation des systèmes sensibles au contexte. (iii) Un ensemble d'éditeurs de modélisation graphiques et hiérarchiques. (iv) Une étude de cas dans le domaine médical pour démontrer la pragmatique de notre approche.

Mots-clés : Systèmes Sensible au Contexte, Environnement Ubiquitaire, Adaptation, Modélisation du Contexte, Analyse des Besoins, Conception de Logiciel.

ملخص

مع الانتشار الواسع لاستخدام الأجهزة المحمولة ، اندمج جيل جديد من التطبيقات في الأنشطة اليومية للمستخدمين مما أسهم بشكل كبير في جعل المعلومات متوفرة في كل مكان وفي أي زمان. إدراك سياق الاستعمال هو الخاصية التي تتيح لهذه التطبيقات أن تصير ذكية، و هذا برصدها المستمر للوضع الحالي للمستخدم وتكييف سلوكياتها بشكل تلقائي تماثياً مع تحولات سياق الاستعمال. في هذا الصدد ، يشتمل تصميم التطبيقات المدركة للسياق على تحديات جديدة مقارنة بالتطبيقات التقليدية وهو ما دفع مهندسي البرمجيات إلى اللجوء لتحليل الاحتياجات وتقنيات نمذجة متطورة من أجل التحكم في مختلف معلومات السياق التي من شأنها أن تؤثر على سلوك التطبيقات. لغة النمذجة UML (Unified Modeling Language) هي اللغة الأكثر استخداماً لتحديد، تصوير، وتوثيق أعمال البرمجيات. لكن بالرغم من ذلك ، فإن رموز مخططاته الحالية غير دقيقة بما يكفي لوصف الأنظمة المدركة للسياق بشكل صحيح.

تقترح هذه الأطروحة نهجاً جديداً للنمذجة يسمى اللغة الموحدة لنمذجة السياق (Context Unified Modeling Language) والذي يرمي إلى تسهيل عمليات وصف، تصوير، وتوثيق الأنظمة المدركة للسياق. نقدم مجموعة من المساهمات التي توصلنا إليها (أ) مخطط جديد يدعى ContextClass diagram، والذي يُعتبر كامتداد لمخطط UML Class diagram لتصميم بنية الأنظمة المدركة للسياق من خلال تحديد الجوانب السياقية التي تُميزها. (ب) مخطط جديد يسمى مخطط ContextSequence diagram، و الذي يمثل امتداد لمخطط UML Sequence diagram لنمذجة التبادلات مع مصادر معلومات السياق بالإضافة لسلوك التأقلم للأنظمة المدركة للسياق. (ج) مجموعة من أدوات النمذجة. (د) دراسة حالة في المجال الطبي لتبيان براغماتية النهج المقترح.

كلمات مفتاحية : الأنظمة المدركة للسياق، البيئة دائمة الوجود، التكيف، نمذجة السياق، تحليل المتطلبات، تصميم البرامج.