Middlesex University London

# A software development framework for context-aware systems

Unai Alegre-Ibarra

**Supervisors:**
Prof. Juan Carlos Augusto
Dr. Carl Evans

**Middlesex University**
Department of Computer Science, Faculty of Science & Technology

*A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctor of Philosophy.*

## Acknowledgments

*I take this opportunity to thank all those people without whom this thesis would not have been possible.*

*En primer lugar, quisiera agradecer a mis padres, por todos los sacrificios hechos para que yo pudiera tener una oportunidad, por la educación y los valores que me han brindado y sin los cuales el camino hubiera sido mucho mas difícil. Sois y seréis siempre un ejemplo a seguir.*

*To Carl Evans and Franco Raimondi for their guidance and support throughout this whole process.*

*A tí Rocío por tu paciencia, por todos los buenos consejos y los sacrificios que has hecho por que yo pueda tener esta oprotunidad. A toda mi familia por su apoyo incondicional durante todo este tiempo, y en especial a Marisa por acogerme de la forma en que lo has hecho.*

*To all the colleagues who I encountered throughout this experience, and made London a warmer place to be. Especially, to Pragya, Javier, Poujha, Alexandra, Dean, Andras, Olanna and Diego. Ta nola ez, Esti ta Imanoli, nigatik egin duzuen guztiagatik, eta zuen bihotzarekin Londres pixkat jasangarriagoa egiteagatik. Londresen elkartu ginen euskal lagun taldeari, eta azkenik, Gasteizko kuadrillari etxera itzuli izan naizen bakoitzean eskeini didazuen beroagatik.*

*I was awarded by Middlesex University with funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n 610840.*

## Abstract

The beginning of the new century has been characterised by the miniaturisation and accessibility of electronics, which has enabled its widespread usage around the world. This technological background is progressively materialising the future of the remainder of the century, where industry-based societies have been moving towards information-based societies. Information from users and their environment is now pervasively available, and many new research areas have born in order to shape the potential of such advancements. Particularly, context-aware computing is at the core of many areas such as Intelligent Environments, Ambient Intelligence, Ambient Assisted Living or Pervasive Computing. Embedding contextual awareness into computers promises a fundamental enhancement in the interaction between computers and humans. While traditional computers require explicit commands in order to operate, contextually aware computers could also use information from the background and the users to provide services according to the situation. But embedding this contextual awareness has many unresolved challenges. The area of context-aware computing has attracted the interest of many researchers that have presented different approaches to solve particular aspects on the implementation of this technology. The great corpus of research in this direction indicates that context-aware systems have different requirements than those of traditional computing. Approaches for developing context-aware systems are typically scattered or do not present compatibility with other approaches. Existing techniques for creating context-aware systems also do not focus on covering all the different stages of a typical software development life-cycle. The contribution of this thesis is towards the foundation layers of a more holistic approach, that tries to facilitate further research on the best techniques for developing these kinds of systems. The approach presents a framework to support the development not only with methodologies, but with open-source tools that facilitate the implementation of context-aware systems in mobile and stationary platforms.

# Contents

# List of Figures

# List of Tables

# Definitions

# Part I

# Opening

Chapter 1

# Introduction

## 1.1 Introduction

Just as the industrial revolution brought the industrial age, the digital revolution is abruptly giving rise to the information age. Emerging technology breakthroughs have made accessible a wide range of heterogeneous devices, which are not necessarily bound to a desktop anymore. Computing can occur in tablets, smartphones or laptops, as well as in everyday objects with integrated circuits. Electronics are now easily available, and people have quickly assimilated these new technologies into the fabric of their being. Global positioning systems, microphones, cameras, accelerometers, gyroscopes, ambient light and proximity detectors are some examples of sensors that are already part of people's daily life, as they are embedded by default into the most commonly used devices. In the information age, data is originated by almost any digital process, from sensors to social media exchanges or website communications. The third wave of computing [6] is now here, and digital information is now everywhere.

Communication and understanding in human interactions happen naturally. Even when some information is omitted, humans can effortlessly figure out meaning from the context. For instance, imagine two people at home. One announces to the other: "I am going out now, to buy it at the store". The listener can interpret what the speaker is going to buy, as they are both holding the flyer of a product. After noticing the speaker leaving without it, the listener does not require a great effort to remind him: "Do not forget your wallet". Both have a natural and deep contextual understanding. They know that buying requires money that is usually kept in a wallet and can use this information to help each other. But the interaction between humans and computers happens differently, as computers are of a radically different nature than humans. When requiring a computing service, a user typically has to make a request for it, and explicitly provide all the information required. This makes the interaction artificial for the users, especially in cases where the same information is requested repeatedly, and there is a poor communication interface. Inspired by human abilities, context-aware computing [7][8] aims to dynamically and non-intrusively provide services according to the situational needs and preferences of the users by using context information. Such an enhancement is in the essence of new emerging areas such as

Ubiquitous & Pervasive Computing, Ambient Intelligence, Ambient Assisted Living or Intelligent Environments, which have the potential of materialising significant improvements in society. Intelligent houses, autonomous driving cars, or systems to support carers looking after patients, can facilitate the creation of more efficient and enhanced hospitals or better public transport. Technology can be made available for groups that find it confusing or difficult to use like the elderly population and people with disabilities. The elder population can extend their independence at home by the intelligent assistance of different technologies [9] [10] [11]. People with disabilities or chronic diseases can also achieve more independence and be more integrated into society, mainstream education and work thanks to these new technologies [12] [13] [14].

In the current information age, it is becoming more and more feasible to obtain and use digital data about the context. Companies and researchers are now much more likely to adopt the trend of aiming to provide more usable services to the end-users by using this information. The acceleration of technological progress has opened the discussion as to whether or not the mid-century will be the point where human intelligence will be exceeded by machine intelligence [15] [16]. Although a number of context-aware systems (C-AS) have been created, context-aware computing is still in its infancy, and many of the existing projects are still laboratory-based prototypes. In projects that involve the development of contextual awareness, the real abilities exhibited by the final system can often differ from the stakeholder expectations [17]. The development of C-AS is difficult because of the following challenges:

Chg1 Context conceptualisation: Context is a term that most people tacitly understand, but find difficult to elucidate [18]. The dynamic nature of context complicates its programming in a computing system, for which it is required to be more static. There is no consensus on its definition, and context is understood as a reflection of the concerns in the particular area that approaches the notion. Sometimes is also difficult to delineate a boundary between context and other system information.

Chg2 Diversity: The field of context-awareness itself is very broad and interdisciplinary, while context-aware solutions are quite ad-hoc. Typically, a context-aware feature (or aspect) is added on top of an existing system or functionality, making its implementation intrinsically dependent on the system within which it is going to be implemented. As systems vary significantly with respect to the implementation of context-aware features, a consequence is that their development has followed a series of ad-hoc paths. Existing tools and methods provide a solution specific problem and are typically disconnected with other research works.

Chg3 Context information management: Data is acquired from multiple and distributed resources, which makes it difficult to ensure its quality and authenticity. Besides, sensors can provide inaccurate, overlapping, contradictory or even missing data [19]. After the information is sensed, it needs to be translated into usable values that are stored in models, which can often involve a trade-off between expressiveness and complexity. This is often also called "low-level" context. Based on the modelled data, new knowledge and understanding is obtained. Reasoning techniques will be applied, transforming "low-level" context into "high-level" context. Therefore, the accuracy of context information created at a higher architectural layer will depend on the correctness of information created at a lower layer. Then, all the produced information needs to be timely distributed to the part of the system that will be consuming it. It is also important to mention that this information might contain some user private data, raising privacy and ethical concerns.

Chg4 Cost: Writing a context-aware system without having an underlying software framework that is bespoke to context-awareness is a hard process. For the way in which the context information needs to be managed, systems usually depend on some sort of infrastructure to support the information handling. One must develop the code to communicate with some external sensing and reasoning system which should be scalable to support the use of different kinds of sensors through its life-cycle. Creating and maintaining these sorts of infrastructures requires a considerable investment of resources, making the creation of context-awareness more expensive than creating traditional software [20].

Chg5 Reliability: Usually, context-aware systems require complex technologies which enable them to obtain higher order knowledge from sensors in order to trigger different services. The more contexts that the system is aware of, the more complex will be the system, and therefore, the more complex it will be to ensure that the system will not exhibit any undesirable behaviour. Likewise, it is difficult to have high quality context information that reflects an accurate representation of reality, making it difficult to reason and adapt accordingly to the way that the user might desire.

Chg6 Change: The dynamic nature of context makes these systems demand high adaptability. The requirements of such systems have a tendency to rapidly change [21]. The context information management infrastructure needs to be both flexible and scalable, enabling the addition and removal of new sensors or high-level context inferences. The introduction of additional context information may trigger requirements for further developmental change. Entirely new services, actuators, or graphical user interface variations are some of the examples of things that might be required for these changes.

It needs to be mentioned that the scope of this thesis is not aimed at completely resolving all the existing challenges in the field. However, these challenges are used as a starting point to guide the research of this thesis.

## 1.2 Supporting a software development methodology for context-aware systems

The challenges in the creation of context-aware systems make their development different from that of traditional computing. Literature shows scattered and disconnected research extending existing software development methods, indicating that there is a need for a more holistic approach in the development of context-aware systems. A development process with so many challenges requires a set of governing methods, rules and principles to maximise the chances of creating a successful and reliable system. There are several methodologies, frameworks, and toolkits that have been tried to address this problem. Although the state-of-the-art is presented in Chapter 3, the following section presents a few selected examples in order to illustrate common issues that provide some context for the themes that are addressed by this thesis.

The Context Modelling Language (CML) [22] was created as a tool to assist designers exploring and specifying the design and implementation of context-aware applications with an emphasis on rapid prototyping. The authors propose a set of conceptual models to support the development process, including context modelling techniques, a preference model for requirements representation and two programming models. The methodology they present has three main steps: Analysis, infrastructure customisation and testing. During the analysis the general goal is to document the functionality and requirements of the application, including the identification of context information, its quality and the sources which are suitable. The infrastructure customisation is a tool-supported step that helps to personalise some context components, identifying new situations and its changes in the databases. Finally, the testing is divided into three stages. First traditional white box testing methods are applied (T1), to then perform black box testing, which uses specially constructed sets of context information (T2). Finally, the application is evaluated in the field, using a realistic hardware environment.

Later on, OPEN [23] was introduced, an ontology-based cooperative programming framework for the rapid prototyping, sharing, and personalisation of context-aware applications for users with diverse technical skills. To meet diverse developer requirements in the development and customisation of context-aware applications, it implements four programming modes with diverse complexity: 1) Incremental mode, for high-level users, which supports the creation of new context-aware applications; 2) Context Interpreter, to reason sensor data using different reasoning techniques; 3) Composition mode, a programming mode for middle-level users; and 4) Parametrisation mode, for low-level users, to enable them customise existing applications.

The MUSIC [24] framework and methodology facilitates the development of adaptive applications in open, heterogeneous ubiquitous computing environments. The methodology proposed has five main stages. The first stage starts with an analysis of resources, context dependencies and context adaptations. Then, during the modelling stage, the variability and the domain model are created. The first stage specifies application variants and their dependencies on the context and resources while the second stage provides semantic information on the execution domain with respect to the context and resource entities as well as the service landscape. Once the model is ready, appropriate transformation tools generate the adaptation model that publishes the design elements into the middleware. Finally, the testing, validation and operation of the code adapting to contextual changes in the desired manner.

And more recently, DiaSuite [25] is a tool-supported development suite that uses a software design approach to driving the development process in the domain of Sense-Compute-Control (SCC) applications. The first step of the method consists of creating a taxonomy of entities. Then, the application logic is decomposed into context and control operators. These first two steps constitute a description of the system in the DiaSpec language. After this, its compiler generates a dedicated Java modelling framework to support the programmer in the implementation of the application. Then, the program is simulated in a smart-home environment scenario. During this step, the tester can develop simulated entities by extending the corresponding abstraction provided by the generated programming framework. DiaSuite also provides support for deploying, maintaining and evolving the system.

Augusto [26] also highlights the need for a more tailored software development process to the area of Intelligent Environments. Although the aim of this thesis is to address the more general topic of context-aware computing, this work has been influenced by, and has synergies with these works [26] [27]. Other development environments and methodologies [28] [29] [30] [31] [32] oriented to similar areas such as Pervasive Computing, Ambient Intelligence [33] and Intelligent Environments [34] also provide support to some extent for context-awareness development. Nevertheless, these approaches share other concerns that are not essential to context-awareness, such as services or ubiquitous features.

## 1.3 Problem statement

There are several methodologies, frameworks and toolkits that have been presented for supporting the development of context-aware systems. Table 1.1 presents an analysis based on the contents of their corresponding publications. The analysis summarises, to the best of our knowledge, the support for the following feature categories: a) The scope of the approach, aiming to analyse if the work is focused just on context-awareness, or it also has influence from other areas; b) Whether or not the approaches take into account user needs, preferences and limitations; c) Whether or not the approaches are able to guide developers with respect to identifying context and context-triggered services; d) To what extent the approaches give coverage to the most common development stages of an application's life-cycle; e) Tool support for developers, and; f) Whether or not the code of those tools is publicly and freely available to any researcher that might be interested in extending it. The analysis concludes that there is no unified software development methodology for the creation of context-aware systems, which has:

- Focus on context-awareness: The development process of a context-aware system has different needs to that of traditional software [Chg1] [Chg2] [Chg3] [Chg4] [Chg5] [Chg6]. Although context-awareness is key for Ubiquitous Computing, Ambient Intelligence, and Intelligent Environments, it constitutes a small part of the whole approach. An approach that is narrowed to context-awareness will make it easier to create some principles that give response to the specific needs of this field, enabling other fields at the same time. Table 1.1 Column (4) shows the specific field in which the research group approaches context-awareness. Although all methodologies provide support to a certain degree for developing context-awareness, in some cases methodologies pay more attention to other field-related aspects and oversimplify the real complexity of context and context-awareness.
- User-centred perspective: Context-aware computing looks for a natural human-computer interaction, where the main purpose is making the computer disappear from the user perception. Therefore, it is inherently user-centred, where the needs, preferences, and limitations of the stakeholders need to be taken into account. Table 1.1 Column (5) conveys whether or not the methodological approach has the philosophy of developing systems which take into account the user needs, preferences and limitations. Only three of the approaches have a user-centred perspective, but none of them include this philosophy as an essential aspect of the conceptualisation of context and context-awareness.
- Development life-cycle coverage: There is a need to have an even more holistic approach to the development of C-AS. Generally, existing methodologies and frameworks do not cover the most common stages of the development life-cycle

of a system, focusing mostly on design and implementation. Not taking into account some stages like requirements elicitation makes it more likely for developers to be uninformed about what to program and increases the chances of producing an incorrect system. Enabling a complete life-cycle can also give a better response to the dynamic nature of context, that demands frequent changes in the systems [Chg6]. Stages like requirements elicitation and maintenance gain more importance. Finally, design verification also takes an important role in producing more reliable systems [Chg5]. Table 1.1 Column (7) collates information about the methodological coverage throughout the different development life-cycle stages. It can be observed that most of the methodologies focus on the design and development stages, paying less attention to other essential development stages such as requirements elicitation. Even methodologies that include support for eliciting requirements do not provide it for other stages like deployment or maintenance.

- Developer guidance: Some approaches present very sophisticated frameworks and tools for the creation of C-AS, predominantly focused on the increase of development speed of highly routine contextual situations. However, they do not include anything to inform developers about what contextual situations are appropriate for the system under development, and what information can determine a contextual situation and what services it will trigger [35] [1]. Table 1.1 Column (6) expresses whether or not the methodology has any support for: (a) enumerating the set of contextual states that may exist; (b) knowing what information could accurately determine a contextual state within a set, and (c) stating what appropriate action should be taken from a particular state, or (d) another type of context discovery guidance for developers. As it can be observed, developers have very little guidance on what contexts need to be implemented and why.
- Open-source tool support: One of the challenges in the development of C-AS is how expensive it is to develop these systems [Chg4] and their specific needs for information handling [Chg3]. For this reason, it is desirable to have a set of tools and infrastructure that helps to reduce development time and cost of implementation. Although almost half of the analysed methodologies provide

tool support (Table 1.1 Column (8)), these tools are usually hard to find online. As far as the author of this thesis is aware, only one methodology has released the code so that it can be improved and extended by other researchers, DiaSuite [36].

## 1.4 Objectives

The broader goal of this research thesis is to:

*Lay the foundations of a user-centred and open-source tool-supported framework for aiding the development of context-aware systems throughout the most common stages of their development life-cycle.*

The philosophy of this dissertation is to provide a contribution that is an integration of its parts in terms of addressing several phases within an overall software engineering process. The approach bases a development model on the standard phases of existing methods and tools, respectively enhancing them with respect to the needs of context-aware systems. The core development phases are addressed with a view to enhancing rather than having a deep focus on a particular stage to a depth of a thesis itself. This dissertation targets the following research goals:

- Conceptualisation: A conceptualisation of context-aware systems that facilitates its development by focusing on the creation of more usable context-aware systems, by taking into account the following aspects: (a) The specific development needs of these systems; (b) The state-of-the-art limitations of these systems; and (c) the needs and preferences of the end-user stakeholders (user-centred perspective).
- Life-cycle coverage: The assembly of a framework which embeds the conceptualisation across the most common phases of a development process. By doing so, the aim is to achieve an integrated consistency across those phases towards the achievement of a methodology that holistically addresses the concerns related to the development of more usable context-aware systems.

- Open-source approach: The area of context-aware computing is still in its infancy, and achieving a methodology which can create perfectly mature context-aware systems is out of the scope of this thesis. The aim is to contribute to the

| No. | Name | Year | Reference | Scope | User-centred perspective | Developer Guidance | Development Life-cycle Coverage | Tool Support | Open source |
|---|---|---|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
| (a) | ISAMadapt | 2002 | [28] | Percom | - | - | Des, Dev | ✓ | - |
| (b) | CML | 2006 | [22] | C-A in Percom | ~ | - | Des, Dev, M | ✓ | - |
| (c) | CAMUS | 2007 | [29] | C-A in Ubicomp | ~ | - | Des, Dev | - | - |
| (d) | MUSIC | 2012 | [24] | Ubicomp | ~ | - | R, Des, Dev, M | ✓ | ~ |
| (e) | OPEN | 2011 | [23] | C-A | ✓ | ~ | Des, Dev | - | - |
| (f) | CA-PSCF | 2010 | [30] | C-A in Pervcom | ~ | - | Des, Dev | ✓ | - |
| (g) | - | 2010 | [33] | AmI | ✓ | ~ | Des, Dev | - | - |
| (h) | MIRIE | 2013 | [34] | IE | - | - | Des | - | - |
| (i) | REUBI | 2013 | [32] | Percom | - | - | R, Des, Dev | ~ | - |
| (j) | PerDe | 2014 | [31] | Percom | ✓ | ~ | Des, Dev | ✓ | - |
| (k) | DiaSuite | 2014 | [25] | SCC | - | - | Des, Dev, M | - | - |

[4] *Scope:* "C-A" = Context-Awareness, "Percom" = Pervasive Computing, "Ubicomp" = Ubiquitous Computing, "AmI" = Ambient Intelligence, "IE" = Intelligent Environments, SCC = "Sense Compute Control".

[5] *User-centred perspective:* "✓" = Has a user-centred perspective, "~" = Takes into account the user preferences, "-" = Does not have a user-centred perspective.

[6] *Developer guidance:* "~"= Offers developer some development support for context discovery [35] [1]., "-" = Does not offer any development support.

[7] *Coverage of development stages:* "R"= Requirements elicitation, "Des" = Analysis and Design, "Dev" = Development, "M" = Maintenance.

[8] *Tool support:* "✓"= Offers tool support, "-" = Does not offer tool support, "~" = It has tool support for some aspects of the methodology but not all of them.

[9] *Open source:* "-" = The code is not publicly available.

**Table 1.1: Comparison of current methodologies, frameworks and toolkits for the development of context-aware systems, with the required characteristics as introduced in Section 1.3**

foundations of an engineering process that will hopefully be able to do so in the future. Therefore, it is also important to highlight the open-source philosophy of this thesis, for which all the tangible source outcomes are publicly available to facilitate its study, adoption, and extension in the future.

## 1.5 Case Study

The whole framework presented in this thesis utilises a total of eight diagram formats, each of which requires the introduction of its corresponding meta-model. In order to facilitate the understanding of the most technical aspects, this section introduces the case study which has been used as a validation example for this thesis. In each of these five chapters, activities of the methodology are illustrated with an example which is related to this case study. Particularly, when presenting the meta-model of each diagram, its corresponding example will be presented, to illustrate the usage of these diagrams to the Reader.

The case study introduced in this section is based on the insights gained during the development of the EU funded POSEIDON project [14] [37]. The project name stands for PersOnalised Smart Environments to increase Inclusion of people with DOwn's syNdrome, and is particularly focused on using smart assistive technologies in order to foster the independence of people with this condition. During the POSEIDON project, several meetings, discussions, interviews and questionnaires were conducted, in which representatives from the Down´s Syndrome Association, carers or parents of a person Down´s syndrome, and people with Down´s syndrome (PDS) participated. These interactions happened in the United Kingdom, Germany and Norway. Some outcomes from these meetings have been referenced for complementing the case study presented in this chapter.

### 1.5.1 Users with special needs

Down's syndrome (DS) is a neuro-developmental disorder, which is caused by the presence of either a copy, or part copy, of chromosome 21 [38]. As part of the POSEIDON project research efforts, an evaluation of the particular needs of people with this disability was conducted. The aim of the following section is to illustrate, to the Reader, the relevant insights to the examples that are presented in this chapter as part of the validation of the framework introduced in this dissertation. This section mainly summarises the results obtained from [39]. The term PDS, will be used in this section to refer to people with Down's Syndrome which are part of the research sample of the mentioned investigation.

PDS have two main challenges which are relevant to this case study: lack of independence and difficulties with regard to integration. The results of the background study revealed that PDS can only complete a few daily tasks on their own. This lack of independence is also related to their inability to integrate socially. The study shows that PDS often experience low social participation. Although PDS show a slightly better integration at work or school, their involvement in leisure activities is more difficult for them. Lack of integration in leisure activities especially holds for the group of youngest PDS. As most of PDS have leisure activities outside home, a key enabler for their integration is their ability to displace. The ability of travelling independently enables better access to a person's community, friends, and activities, giving the person self-determination and quality of life [40]. In some cases, the lack of independence is linked to the challenges presented when travelling alone. Due to challenges in independent travel, especially when travelling to leisure activities, they mostly need to be driven by a relative, walk accompanied by someone or take a taxi. This dependence also impacts the life quality of their family members, as this dependence put a strain on other individuals. Their lack of independence for travelling, affects negatively to their low social integration and vice-versa, creating a negatively feedbacked circle [38].

It is important to mention two main aspects that give an opportunity for taking on these two main challenges. The first aspect is that PDS can often complete tasks by themselves, provided that they have a set of instructions they can follow. The second aspect is that the majority of PDS already use information technologies in their daily life[1], which can be used for assisting and giving them guidance. The aim of the POSEIDON project is to foster the independence of people with Down's syndrome through the use of smart and assistive technologies which are personalised to their special needs and preferences. The hypothesis of the project is that by enabling the inde-

[1]It is also important to mention that half of PDS can use these devices without any help [39].

pendence of individuals, they will naturally be more integrated into society.

### 1.5.2 A navigation application

It has to be mentioned, that the POSEIDON project presents itself a suite of framework tools, and that a discussion of these is out of the scope of this example. For simplification purposes, the example presented in this chapter is constrained to an outdoors navigation application, which is bespoke to this particular disability. More specifically, the case study focuses on a mobile application that uses a real-world representation of maps along with location services to support outdoor journeys that might be walking or by bus. Due to space restrictions, this example is further limited to bus displacements happening in London, United Kingdom. There are several differences between this application and other existing navigation applications. The application uses routes with tailored directions, notifications, reminders, and other services which will be triggered depending on the context. The navigation system described in this case study can be found in [41], although it has to be mentioned that not all the features described in this chapter have been implemented into that system. This navigation system is part of a larger project architecture, which has broader purposes. For instance, the architecture enables customisation of services, so that carers of PDS can tailor the navigation application to the particular needs of their protégé [42]. This larger architecture also facilitates the training of PDS in safe environments by using virtual reality [43]. In a virtual environment, PDS associate images and audios are displayed in certain points of the route, so that they can learn and use the same references to better orientate when using the navigational services outdoors.

### 1.5.3 An assistive smart-home environment

The example presented in Section 1.5.2 is mainly focused on mobile environments where the user is expected to be carrying and using small devices such as phones or tablets. Nevertheless, the scope of context-aware computing spans also stationary platforms, enabling other technologies, such as ambient intelligence or ambient assisted living, to happen. In order to illustrate better the creation of context-aware systems in different platforms, another scenario is introduced. In this scenario, the focus is on a stationary platform, installed as a smart-house, that uses different stationary sensors to provide context-awareness. The scenario assumes an individual with Down's syndrome living alone in a smart-house, where the technology provides the user with the means to have a more autonomous life at home. In order to narrow the example, the example will be constrained to the kitchen of the house, particularly to facilitating the person to cook independently.

### 1.5.4 Tool support

This thesis presents an open-source tool supported framework for the development of context-aware systems. The examples available throughout the thesis, including diagrams, have been created with the framework and are an illustration of what can be done with these tools. Appendix A explains how to download and install these tools. This step is assumed to be completed before continuing with the case study.

## 1.6 Document structure

This doctoral thesis is divided into eight main chapters. A review of context conceptualisation is presented in Chapter 2, which proposes updates to existing definitions. This conceptualisation is the base for the remainder of the thesis. Chapter 3 is a literature review in relation to the requirements, model-driven development, and implementation techniques for the creation of context-aware systems. The following chapters of this dissertation introduce the *Context-Aware Systems Engineering Framework* (C-ASEF), an open-source tool supported framework for facilitating the creation of more usable context-aware systems. This framework is divided into three sub-frameworks, according to the main development stages of a software development life cycle: Requirements, Design, and Implementation, Deployment and Maintenance. Chapter 4 introduces the *Requirements for the Context-Aware Systems Engineering Framework* (RC-ASEF), which is a sub-framework of C-ASEF that is focused on non-contextual requirements for the creation of context-aware systems. Chapter 5 presents the *Situational Requirements for the Context-Aware Systems Engineering Framework* (SRC-ASEF), a specialisation of RC-ASEF which focuses on the contextual requirements elicitation for context-aware systems. Chapter 6 describes the *Design for the Context-Aware Systems Engineering Framework* (DC-ASEF), a sub-framework of C-ASEF which is focused on the design of rule-based context-aware systems. Chapter 7 introduces the *Verification for the Context-Aware Systems Engineering Framework* (VC-ASEF), a specialisation of DC-ASEF which is focused on the verification of reasoning rules created with the DC-ASEF framework. Chapter 8 presents the *Implementation, Deployment and Maintenance for the Context-Aware System Engineering Framework* (IDMC-ASEF), a sub-framework of C-ASEF which is specialised into the Implementation, Deployment and Maintenance of rule-based context-aware systems that use radio-based and mobile sensors. Chapter 9 summarises the main conclusions of the thesis and suggests future work and extensions of this work. The thesis is accompanied by a set of Appendices. Appendix A introduces a guide for downloading and installing all the open-source tools that support the C-ASEF. Finally, the publications written during this doctoral thesis are listed in Appendix B.

Chapter 2

# Conceptualisation

## 2.1 Introduction

The notion of context is an important research theme, as its better understanding naturally facilitates the engineering of context-aware systems. But research on this concept is not modern, and it has been approached from the point of view of multiple disciplines throughout history. Currently, and in regard to context-aware computing, many definitions have been presented, and there appears to be a lack of agreement on its meaning. As this is such an important challenge, which underpins the broad theme of this work, this chapter is dedicated to the understanding of context and context-awareness, and the current limitations that it imposes on the engineering of context-aware systems. The first half of this chapter analyses the state-of-the-art specific to the conceptualisation of context and context-awareness, which is used in the second half of the chapter to create a conceptualisation which is aimed towards the engineering of more usable context-aware systems. This first part has been kept separate from the general review of context-aware software engineering because of its underpinning importance to the rest of the work. The conceptualisation introduced in the second part does not constitute a piece of the literature review, but rather a contribution to the conceptualisation of context, context-awareness and context-aware systems, which has been published in [44] and [45]. The remainder of this chapter is as follows. Section 2.2 presents a literature review which is focused on the most important concepts that surround context within the theme of context-aware computing. Section 2.3 introduces a more holistic view of context and context-awareness, which analyses the main issues in its conceptualisation, exploring areas which are out of the scope of computer science, such as philosophy. Section 2.4 introduces a novel conceptualisation of context which is conceived for guiding developers with respect to the creation of more usable context-aware systems. Finally, 2.5 summarises the chapter and introduces the main conclusions that guide the rest of the research work conducted in this dissertation.

## 2.2 Context in context-aware computing

### 2.2.1 Context

The word "context" has evolved from the Latin word "contextus", which is composed by the prefix "con" (together) and the root word "texere" (weave). The meaning of the word is now used to broadly define the set of circumstances that frame an event or an object [46]. Despite the numerous attempts to define this concept, there is still a lack of consensus on what context and context-awareness really mean. The objective of this explanation is not to provide an exhaustive analysis of all the definitions that have appeared in the literature, but to show the Reader a brief history of the term in context-aware computing.

The term of context in context-aware computing was first introduced by Schilit and Theimer [47], which considered it as the *"Location, identities of nearby people and objects and changes to those objects"*. One year after them, Brown [48] defined context as *"The elements of the user's location, the environment, the identity and the time."* Dey initially understood this term as *"the user's emotional state, focus of attention, location, and orientation, date and time, objects and people in the user's environment"* [49]. Similarly, Hull [50] defined the concept as *"the aspects of the current situation"*. Ryan et al. [51] as the *"User's location, the environment, the identity and the time"*. The term context was also defined in the ISO 13407 [52] standard, which considered it as the *"User characteristics, task, as well as the technical, physical, and social environment"*. Later on, the most acknowledged definition of context was introduced by Dey and Abowd [8][18], which considered context as:

> *Any information that can be used to characterise the situation of an entity, where the entity is a person, place, or object that is considered relevant to the interaction between a user and its application, including the user and the application themselves.*

But their definition did not bring an absolute consensus about what context means. Some authors tried to extend the definition of context by providing more operational definitions and include other dimensions of context [53] [54]. Other authors reported some drawbacks of this definition, such as being too broad [55] [35]. More definitions were also introduced after Dey and Abowd's. Yau et al. [56] defined context as *"any* *instantaneous, detectable and relevant condition of the environment or the device, such as time, location, light intensity, noise level and available bandwidth."*. Bazire [46] defined it as a *"set of constraints that influence the behaviour of a system (a user or a computer) embedded in a given task"*. Roto [57] as the *"Circumstances under which the activity takes place."* More recently Ye et al.[58] define context as *"a well-structured concept that describes a property of an environment or a user"*.

#### 2.2.1.1 Categorisation

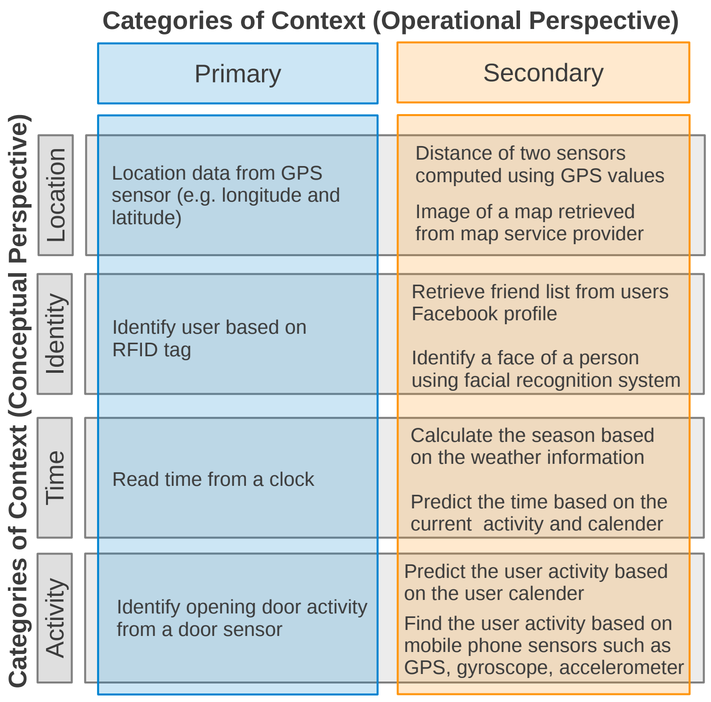

Categories of Context (Operational Perspective)
Primary
Secondary
Categories of Context (Conceptual Perspective)
Location
Identity
Time
Activity
Location data from GPS sensor (e.g. longitude and latitude)
Distance of two sensors computed using GPS values
Image of a map retrieved from map service provider
Identify user based on RFID tag
Retrieve friend list from users Facebook profile
Identify a face of a person using facial recognition system
Read time from a clock
Calculate the season based on the weather information
Predict the time based on the current activity and calender
Identify opening door activity from a door sensor
Predict the user activity based on the user calender
Find the user activity based on mobile phone sensors such as GPS, gyroscope, accelerometer

**Figure 2.1: Context categorisation divided into conceptual and operational perspectives, as presented in [1].**

Not only is it difficult to reach an agreement on what context is, but also it is difficult to reach an agreement on how can it be categorised. Many authors have approached context by dividing it into categories and taxonomies [7] [59] [60] [1]. Perera et al. [1] presented a broad comparison between categories and taxonomies, including their relationship, advantages, and disadvantages. As it can be observed in Figure 2.1, they acknowledge two different types of categorisation schemes: operational and conceptual. While the operational category helps to understand the issues and challenges of data acquisition techniques, the conceptual scheme allows an understanding of the relationships between different contexts. After comparing all the existing categorisation schemes, Perera et al. acknowledge that there is no single categorisation scheme that can accommodate all the demands for context-awareness[1].

#### 2.2.1.2 Context model

Since computers need to be programmed, it is natural in the field of context-aware computing to seek formal or semi-formal descriptions of the context information used by the system. A context model typically tries to provide the interface or behavioural description of the information that the computer handles. In an ideal world, a working definition should express a class of systems that one can then capture as a model, since models can omit all unnecessary detail. Nevertheless, the concept of context entails such complexity that it is very difficult to capture a unique model to fit all systems. Besides, this model would necessarily be evolving to capture new and changing contexts. Henricksen [60] considers that a context model:

*"Identifies a concrete subset of the context that is realistically attainable from sensors, applications, and users and able to be exploited in the execution of a task".*

She mentions that the context model employed by a given context-aware application is usually explicitly specified by the application developer, but may evolve over time. Also, she distinguishes a context-attribute as being:

*An element of the context model describing the context [60].*

Each system should support its own context model, where each of the context-attributes can be mapped to the particular functional requirements that it is related to. The existence of appropriately-designed context information models, which are mappable to requirements, would naturally ease the development and maintenance of future systems.

[1]Although the particular research of Perera et al. [1] is bounded to the internet of things paradigm, the cited conclusion is also applicable for the areas covered by this thesis.

This dissertation acknowledges the definitions of the context model and context-attribute as introduced by Henricksen [60].

#### 2.2.1.3 Situation

The term situation can also be observed in the literature. Its meaning is often ambiguous but strongly related to context. McCarthy [61] introduced the theory of situation calculus. Later on, the same theory was formalised by Reiter et al. [62] into the action theory, considering the situation as *"a finite sequence of actions"*. Not a state or a snapshot, but a history. Yau et al. [56] define a situation as *"a set of past context-attributes and/or actions of individual devices in the application which is relevant to future device actions"*. Anagnostopoulos [63], defines the situation as *"the concurrent activities performed in a specific location for a certain period of time"*. Kim and Kim, consider that a *"situation is a problematic and developing state of a computational element characterised by its context"*. By "*problematic*", they mean that there is a problem of which the system should take care in a given situation, and by "*developing*" they mean that a situation is changeable to another situation or state by system operations for solving the problem. In Ye et al. [58] a situation is defined as *"an external semantic interpretation of sensor data"*. Where "interpretation" means that situations assign meanings to sensor data. The term "*external*" means that the interpretation is from the perspective of applications, rather than from the sensors. Finally, the term "*semantic*" means that the interpretation assigns meaning on sensor data based on structures and relationships within the same type of sensor data and between different types of sensor data. Typically, approaches related to situations are more related to the field of artificial intelligence and have more to do with the logical and mathematical formalisation of the term. Situation awareness is more related to an artificial intelligence approach, where it is important to take into account the perception of the environment by the system with respect to time, the understanding and meaning, and the projection of future states. The approach is useful for critical decision-making and it is applied to areas such as aviation, transport or emergency services. Nevertheless, the scope of this thesis is more focused on areas such as intelligent environments, where the aim is to provide services to humans in daily life situations. Ruiz-Lopez et al. [64] also introduce the notion of context situation, understood as *"a set of context-attributes repres-*

*enting a situation of interest to the system"*. Where a context-attribute is *"any observable property in the system environment that characterises a situation of interest for itself"*.

### 2.2.2 Context-awareness

Dey [8] also defined a system as context-aware if *"it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task"*. Independently from this definition, the phrase "context-aware" is generally used in the literature to describe any type of system that is able to use context. Also, systems that have characteristics which could be considered as "context-aware" use other terms such as "smart", "intelligent" or "automated" or simply do not specifically refer to this aspect. For instance, let us think about the feature of smartphones that changes the orientation of the screen (landscape/portrait mode) depending on the phone position. Although it could be considered as "context-aware", it is typically known as simply auto-rotate or automatic screen orientation. Also, recently created applications such as Google Now [65] use different terms for these kinds of features. On the one hand, the great variety of systems and features to be considered "context-aware" make it very difficult to formulate a definition that is suitable for all of them. On the other hand, since "context-awareness" relies on the definition of context, and since there is no consensus on its definition, it is also very difficult to characterize what contextual-awareness is.

### 2.2.3 Features

Schilit et al. [7] first identified different classes of context-aware applications. Later, Pascoe [66] aimed at identifying the core features of context awareness. Dey and Abowd [18] presented a categorisation for features of context-aware applications, based on the classification of Schilit and Pascoe, namely:

1) Presentation of information and services to the user.
2) Automatic execution of a service.
3) Tagging of context information for later retrieval.

With regard to the first feature, the system decides which information and services are presented to the user, based on context. Nearby located objects might be emphasised, or for instance, a printer command might print to the nearest printer. The second feature refers to the automatic execution of a service. For example, let us consider a smart-home environment. "*When a user starts driving home from their office, a context-aware application is employed in the house, which should switch on the air conditioning system and the coffee machine to be ready to use by the time the user steps into their house*" [1]. Finally, Dey and Abowd present "contextual augmentation", which extends the abilities of sensing, reacting and interacting with the environment by using additional information. This is done by associating digital data with a particular context. For example, a tour guide can augment reality by presenting information about the attractions that surround the tour party, or that are approaching [66].

### 2.2.4 Interaction modalities

Barkhuus and Dey [67] classified the possible interactions into three main categories. The first category is *personalisation*, in which the users are able to set their preferences, likes, and expectations of the system manually. The second category is *passive context-awareness*, where the system is constantly monitoring the environment and offers choices to the users in order to take actions. The last category is *active context-awareness*, where the system is continuously monitoring the environment and acting autonomously.

### 2.2.5 Life-cycle of context-information

Perera et al. [1] review different works on life cycles of context information, concluding with a life cycle structure that consists of four phases. First, context is acquired from different sources (context acquisition). Second, the collected data is modelled and represented meaningfully (context modelling). Third, the modelled data is processed to obtain high-level context information. Finally, low and high level data is distributed to applications.

## 2.3 Reflections on the context conceptualisation

Section 2.2 has analysed the main concepts around context and context-awareness. This section goes beyond the theme of context-aware computing, to review literature on the most relevant conclusions regarding the conceptualisation of context and context-awareness in other disciplines such as philosophy or human-computer interaction. The conclusions presented at the end of this section are used for creating the conceptualisation presented in Section 2.4.

### 2.3.1 On the limitations of context-aware systems

Making computers intelligent, able to think and appear to be conscious, has been a topic of discussion since the very inception of computer science [68] [69] [70] [71] [72] [73] [74]. The latest achievements in the simulation of human intelligence are remarkable. Computers can recognise objects, emotions or even transcribe and understand speech at a professional translation service level [75] [76]. In artificial intelligence for games, one of the major advances of the century has been the creation of an artificial intelligence which is able to defeat top-tier human Go players [77] by using general purpose neural networks that resemble the way in which the human brain uses memory. Expert systems are now able to substitute human workforce in industry, or manage to drive autonomous cars and drones. Even human-scale cortical simulations of the brain [78] have been achieved. Nevertheless, although seems to be no problem in the simulation of intelligence (weak AI) [71], it still remains a topic of debate as to whether or not it is feasible for a computer program to create intentionality in an artificial way (Strong AI), since a computer program is essentially syntactical and human-like consciousness requires semantics [71]. The lack of biological embodiment could also be a limitation for computers in the acquisition of expertise to the same degree, and as extensively as humans [73]. Currently, it is difficult to understand how consciousness and intelligence fit in the physical world, and there is not such a thing as a comprehensive science of consciousness. Neither scientists nor philosophers agree on a universal conceptualisation for the human mind, or its causal interactions with the body. Also, there are still many open questions about how human-like artificial intelligence would behave. Since intelligence is strongly related to unpredictability, it is difficult to ensure that a human-like intelligent system would remain within the constraints of a software system, and would not go beyond its function. It is also hard to know if the copy of a consciousness will have the same mental-life as the original. Further discussion of these topics is out of the scope of this thesis, as this dissertation is not focused on hard artificial intelligence, but rather on the more specialised theme of context-awareness. Nevertheless, it is important to note that although the simulation of intelligence has achieved outstanding advances, the creation of human-like consciousness is still in its infancy. The Reader should bear in mind not only that the contextual awareness exhibited by state-of-the-art intelligent machines is not the same as that of humans [17], but that it is going to take considerable time to reach such an achievement, if it is ever reached. Therefore, the main conclusion that should be drawn is that there is a need to understand the current limitations and strong points of context-aware systems (and computers in general), in order to focus the engineering of these systems into those aspects that strengthen the advantages and mitigate the limitations as much as possible.

### 2.3.2 A philosophical conceptualisation analysis

As can be concluded from Section 2.2, the concept of context is still not fully understood. This section aims to shed some light on the reasons why there is not a consensus on the definition of context. The interest in the notion of context to computer science started around the end of the 20th century, and it was stimulated by previously introduced arguments of social science with regard to traditional interactive system design, which often fails to respond to the settings in which the action occurs [79]. Developers have hoped that by incorporating notions of context into information technologies, computers can be made more sensitive to the details of the specific settings of use, but turning social observation into technical design seems to be problematic. In order to better understand how to engineer more usable C-AS, there is a need for a broader understanding of influencers and ideas that can serve as a source for inspiration, exploration and innovation that refocuses upon the first-person human experience of ubiquitous computing and C-AS [80].

Dourish [81] acknowledged that the drive to represent context is inspired by, and in some cases the direct response to, sociological investigations. Nevertheless, the philosophical tradition behind those investigations (phenomenology[2]) stems from a dif-

[2]A philosophical tradition related to the study of *phenomena*, or things, as they appear in a first-person experience, or consciousness.

ferent tradition than that of computer science (positivism[3]). In the phenomenological perspective, context is understood as a continually evolving and highly situation-dependent construct [35]. Therefore, context is an issue that has a strong bond with the concept of interaction, where:

1. Contextuality is a relational property that holds between objects or activities. It is not a matter of something being or not being context; rather it may or may not be contextually relevant to some particular activity.
2. The scope of contextual features is defined dynamically. Rather than being something that can be delineated and defined in advance.
3. Context is particular to each occasion of activity or action. Context is an occasioned property, relevant to particular: settings, instances of action and parties to that action.
4. Context arises from activity, being actively produced, maintained and enacted.

However, the representational nature of computing systems demands a different approach to the concept of context. After analysing the conceptual work of several definitions, Dourish extracted four assumptions that seem to underlie the notion of context as it operates in the view of computer science, where it is treated as a representational problem rather than an interactional one. The assumptions are:

1. Context is a form of information. Something that can be known, encoded and represented in the same way as other information is in software systems.
2. Context is delineable. For some set of applications, one can define what counts as the context of activities that the application supports, and do so in advance.
3. Context is stable. Although the precise elements of a context representation may vary from application to application, they do not vary from instance to instance of an activity or an event. The determination of any contextual element can be made once and for all.

As Dourish [81] remarks, in context-aware computing, there is an attempt of deriving positivist responses from phenomenological arguments. One one hand, context in social sciences is understood as something dynamic, with a set of nuances that make each situation unique, and particular to the occasion. On the other hand, context in computer science necessarily trends towards the creation of models that are abstracted

[3]A philosophical system that recognises only that which can be scientifically verified.

from the detail of particular occasions, so that they can be programmed in a computer. Even in computer systems that resemble the human brain, such as neural networks, the models need to be programmed and trained in order to obtain the desired output. Regardless of the way in which it is done, a developer will necessarily have to program a model of the situations and the actions to be taken in each of them. Based on Greenberg's [35] reflections, the dual nature of context has the following implications[4]:

- $I_1$ There is a need to conceptually support developers in: A) Enumerating the set of contextual states that may exist; B) Knowing what information could accurately determine a contextual state within that set; C) Stating what appropriate action should be taken in that particular state. Developers require a better understanding of the situations which are relevant to provide services according to the needs, limitations and preferences of the users. Even if frameworks and toolkits provide elegant ways to design and implement context-aware applications, they fall into a design trap if they do not provide any support for informing the developers of what contextual situations are appropriate to the system.
- $I_2$ Even with adequate support, it is very difficult or even impossible to foresee all the situations in order to program them. Even some situations that might seem similar a priori, can greatly differ from the actual instantiation of the situation.
  - $I_{2.1}$ C-AS have a high chance of taking actions that might not be the most appropriate in certain situations. While the engineering of these systems matures, there is a need to mitigate the impact of context misinterpretation.
  - $I_{2.2}$ There is a need to direct the research efforts towards the discovery and analysis techniques of the different situations in which the system can offer services, as they are key to the development of a C-AS.

## 2.4 Perspectives on context for the engineering of more usable context-aware systems

The following section constitutes a significant enhancement to the conceptualisation of context that underpins the related work of this dissertation. The aim is to create

[4]Note that the listed implications will be referenced as [$I_1$], [$I_2$.1], and [$I_2$.2] along the rest of the report. [$I_1$] and [$I_2$] appear in [35], while [$I_2$.1], and [$I_2$.2] are reflections of the authors of this paper.

a conceptualisation of context which can guide the creation of more usable context-aware systems. Although it takes into account the state-of-the-art conceptualisation of context introduced in Section 2.2, and the more holistic analysis of the concept introduced in Section 2.3, this section introduces a revised conceptualisation of context and context-awareness for the engineering of more usable context-aware systems.

### 2.4.1 Interacting with context-aware systems

Context-aware systems were originally conceived to operate without human supervision. Nevertheless, the state-of-the-art in its conceptualisation shows that creating exclusively autonomous systems, in all but simple cases, could significantly increase the chances of these taking inappropriate actions. Section 2.3, explains the need for strengthening the abilities of context-aware systems and reduce the negative implications of their limitations [$I_{2.1}$]. For this purpose, this thesis extends the approach of Barkhuus and Dey [67] as follows. When a system exhibits a context-aware feature, there are always two different dimensions that should be taken into account:

1) Execution: Referring to the actions/behaviours of the system when a specific situation arises;
2) Configuration: Relating to the adjustment of actions that a system will exhibit and which takes place following implementation.

Both dimensions are mutually exclusive but both can be executed in two different modalities:

A) Active, where the system changes its content autonomously;
B) Passive, where the user has explicit involvement in the actions taken by the system.

It is important to emphasise that a system can have many services and each of them can be executed in each of the interaction modalities. This can help the creation of more flexible context-aware systems [82]. The result is a classification into four different interaction types:

1. Active Execution: The systems have autonomy to execute services, and self-adapt them depending on the context. For example, the screen of a smartphone can switch for landscape to portrait automatically, when reaching certain accelerometer values.

2. Passive execution: The users are involved in the action-taking process of the system, where they specify if and how the application should change in a specific situation. The system can present services for that specific situation or ask permission of the user to take an action.
3. Active Configuration: The user is not directly involved in the evolution of the system after it is implemented. The system is able to learn from the user preferences, which are used (autonomously or through non-user human intervention) in order to maintain its rules.
4. The user is involved in the manual personalisation of preferences, likes, and expectations of the system after its implementation. Overall programming complexity is reduced by introducing abstractions that enable users to act like software engineers to directly modify these rules, in order to obtain the desired behaviour.

With these categories, developers take not only into account the modality for the execution of services, but also how these will be configured, evolved and maintained. Each interaction modality has its own advantages and disadvantages (see Table 2.1). Taking them into account, developers can mitigate the negative effects of not having a completely mature technology, in each particular service of the application or system. The determination of the most suitable interaction modality will depend on the particular situation and will have to be decided separately.

### 2.4.2 Features of a context-aware system

Also taking into account the limitations and real abilities of context-aware systems [$I_{2.1}$], an enhancement of Dey and Abowd's context-aware feature classification [8] is introduced. The extended approach accommodates the interaction modalities explained in Section 2.4.1. Additionally, it extends the information presentation to any system stakeholder, rather than limiting it to the users. This change accommodates the usage of this technology to emerging fields such as that of data-science. The proposed extension is as follows:

1) Presentation of information to the stakeholders.
2) Active or passive execution of a service.
3) Active or passive adaptation of a service.

| Name | Pros | Cons | Name | Pros | Cons |
|---|---|---|---|---|---|
| *Active Execution* (Self-adaptivity) | • Little or no effort required by users [83] • No special user knowledge is needed [83] | • Difficult to ensure that the system will take an appropriate action (Difficult to validate and verify the system) • Loss of control over what the system is executing and why [83] • There are still some open issues [84] [85] • Developers have all the burden • Users can be uncomfortable not understanding what happens with the information that the machine gathers from them | *Passive Execution* (Intelligibility & Control) | • Augments the trust of users [86] since they understand better how the system works • Easier to evaluate the system behaviour • The system will take the actions that the user wants | • Requires developers to understand how to generate explanations [87] • The users might not have enough expertise to take decisions on their own • Applications need to convey more information to explain actions to users [86] • May compromise the privacy of users if they are used on social interactions • Users can use their higher context understanding for a better control the system [86] [17] |
| *Active Configuration* (Learning & Adapting) | • Little or no effort required by users • No special user knowledge is needed • Can unearth needs, preferences or habits difficult to see in other ways [88] | • Difficult to determine when rules should be created or deleted • The rules are based on sensors values (inaccuracy and uncertainty) • Loss of control over what the system is executing and why [83] | *Passive Configuration* (End-user programming) | • Offers greater motivation, control, ownership, creativity and quality to end-users [89] • Users are in control; users know their tasks best [83] • Releases developers burden | • Users might be forced to contribute and cooperate in context for which they could lack experience [89] • Meta-design is more complex and abstract than design • Complexity is increased (users need to learn adaptation components); Systems may become incompatible [83] |

**Table 2.1: Comparative analysis on the interaction modalities that context-aware systems can have.**

4) Tagging context to information.

The first feature is very similar to that presented by Dey and Abowd. It keeps the essence of Pascoe's "presenting context" and Schilit's "proximate selection", but "contextual commands" are merged with the second feature. The notion of collaboration among stakeholders is introduced, rather than just the users. There can be different situations of interest according to the category of stakeholders. For example, in an ambient-assisted living system that takes care of a person with disabilities, stakeholders can be roughly divided into protégés and carers. Each group of stakeholders can have different needs and might require being aware of disparate pieces of information. Some context of interest from all these stakeholders will typically intersect. They do not necessarily have to be disjoint. Equally, they do not have to be exactly the same. There are no a priori relations and it all depends on the applications and personal choices. Even when some "situations of interest" may be the same for different stakeholders, they may be interested in them for different reasons and may expect different outcomes. This happens especially in healthcare related scenarios, where primary users are typically the patients, secondary the carers, and tertiary other professionals. The second feature extension includes all the different interaction modalities and clearly differentiates between the execution of a service and its configuration or evolution. A service can automatically be triggered, making the system autonomous in its decision. But it also enables an interaction where permission is requested from the user before executing. Also, a list of different choices could appear on the screen, as in Schilit's "proximate selection". The third feature extension is related to the user-centred perspective, where the possibility of having end-users as part of the configuration and personalisation of the services exists. Finally, the last feature (tagging context to information) is the same as that presented by Dey and Abowd.

### 2.4.3 Situation of interest

**Definition 1.** Situation of interest:
*The circumstance in which developers understand that the system can potentially exhibit features which are relevant for the intention, preferences, and needs of its stakeholders at that particular moment.*

The power of the presented context definition resides in the role that the *situation of interest* (SOI) takes in the development of C-AS. Since the SOI is understood as an observer-dependent phenomenon, it is targeted to represent the interpretation that developers give to it. This implies that developers inherently need to engage in an understanding process for developing a C-AS. Particularly, they first have to understand how users give meaning to the actions they take in a SOI (semantics), and then find the best manner in which the computer can realise that situation (symbols) and to provide useful services that can help them accomplish their actions. Therefore, the SOI acts as a nexus between two key conceptual components in the development of C-AS, facilitating the application of the principles for getting the correct context [$I_1$].

The first conceptual component that is related to the SOI is the provision of useful context-aware features, which are directly related to the needs and preferences of the stakeholders, and are relevant to their intention in that particular situation, as shown in Figure 2.3. The second conceptual component related to the SOI is the representation of the developers' plan to make the system realise that the particular SOI is happening. For the second component, the *context-attribute* concept is used, as shown in Figure 2.2, and further explained in the next subsection. It needs to be mentioned that developers can consider more than one SOI detection plan, and evaluate which one is more suitable to be implemented, taking into account the particular restrictions of the project in which it is being developed. It could also happen that developers deem they do not have enough resources to make the system identify a particular SOI under their current project scope. Making such realisations and decisions at an early stage is fundamental to the creation of a successful C-AS. This conceptual tool enables the stakeholders to have more accurate expectations on the behaviours that the system will exhibit. It is also important to note that this relation between SOIs, context-aware features, and SOI detection plans, facilitates the analysis of C-AS not only during the requirements elicitation stage of the system, but also during maintenance stages, after the system has already been implemented and deployed. Additionally, since the definition of context-aware features considers all the interaction modalities, these are intrinsically included in the conceptualisation of what context and SOI are. Consequently, its own conceptualisation helps developers to foresee and reduce the potential misbehaviours of the system.

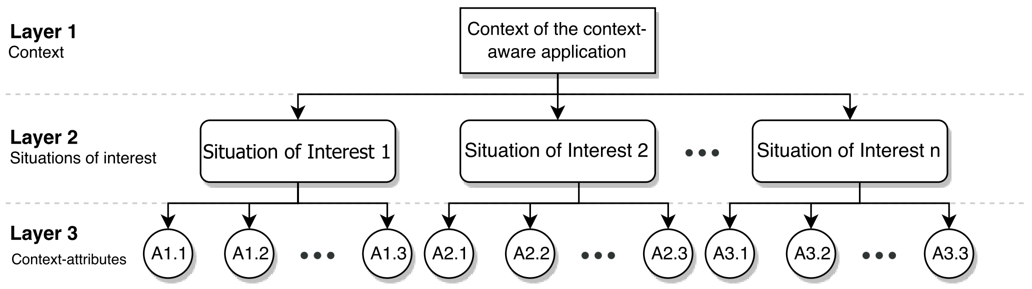


**Figure 2.2: Decomposition of the different concepts related to context.**

### 2.4.4 Context-attribute

**Definition 2.** Context-attribute:
*An observable property of a situation of interest which can be realistically attained from a sensor, application or stakeholder.*

A context-aware system is able to characterise a situation of interest by observing a set of context-attributes. The definition is inspired by that of context model introduced by Henricksen et al. [60], and that of context-attribute provided by Ruiz-Lopez [64]. The context-attribute is not only that which is used by the application to characterise a situation of interest, but also an element of the context model, which describes the whole context of the system. As acknowledged in [64], a context-attribute can have a context value. This represents the particular value that a context-attribute can take, and can be a punctual value, an interval or a set. Finally, a context value can be in a context-value domain, which indicates all the possible values that the context-value can take. Finally, it needs to be mentioned that although typically a sensor refers to a hardware sensor, in this case, it has a broader sense, including physical, virtual and logical sensors [1] [90]. Physical sensors refer to those tangible (hardware) sensors that provide data by themselves. Virtual sensors are those which are not tangible (software) and do not necessarily generate information by themselves. They can gather data from different sources and publish it as sensor data (e.g., twitter status, emails, contacts). Logical sensors combine physical and virtual sensors to provide more meaningful information.

### 2.4.5 Context

**Definition 3.** Context:
*The information which is relevant for a computing system to characterise situations of interest.*

The aim of this conceptualisation is to introduce a perspective that acknowledges the duality of context, that which brings closer the phenomenological (dynamic) and positivist approaches (static). This context definition acknowledges the positivist perspective, demanded by computerised systems, where context is necessarily a form of information. More precisely, context is considered as the sum of all the symbolic representations required by the computer to figure out when different situations of interest are happening in the real world. Notice that this definition of context depends on the concept of *situation of interest*, which, as explained in Section 2.4.3, is recognised as an observer-dependent and ontologically subjective phenomena. Although, in comparison, whilst the definition is similarly broad to that of Dey and Abowd's [8], there are two main differences. The first is that the context information exists without requiring an interaction between the user and an application [55]. Rather, it requires the existence of some added value for the stakeholders, provided in the form of one of the context-aware features presented in Section 2.4.2. The second difference, as further explained in the next subsections, is that it facilitates the application of development principles to identify the correct context [$I_1$].

### 2.4.6 Context-awareness

**Definition 4.** Context-awareness:
*The ability of a system to use context for exhibiting features which are useful to the stakeholders because they directly relate to their preferences and needs.*

This definition is similar to that presented by Dey and Abowd [8], but introduces a more user-centred perspective, connecting the definitions of context to the abovementioned context-aware features and its usability. Therefore, contextual information becomes that which makes the system run better for the stakeholders of the system. Also, it is important to mention the introduction of stakeholders as a concept with a broader scope than users. While "users" is a word more focused on those who have the most direct experience with the final product, stakeholders not only encompass them but also other people who might have an interest or a concern with the project. This includes companies interested in making money from the system, or governments that have interests in its implications. It is practical not only for commercial reasons but for other applications such as healthcare, where the approval of medical staff is essential to certify the safety of end-users. Also, the enhanced definition of context-awareness above is linked to the provision of its features, as defined in Section 2.3, which relate to the different interaction modalities which aim to reduce the impact of misinterpreting the context [$I_{2.1}$].

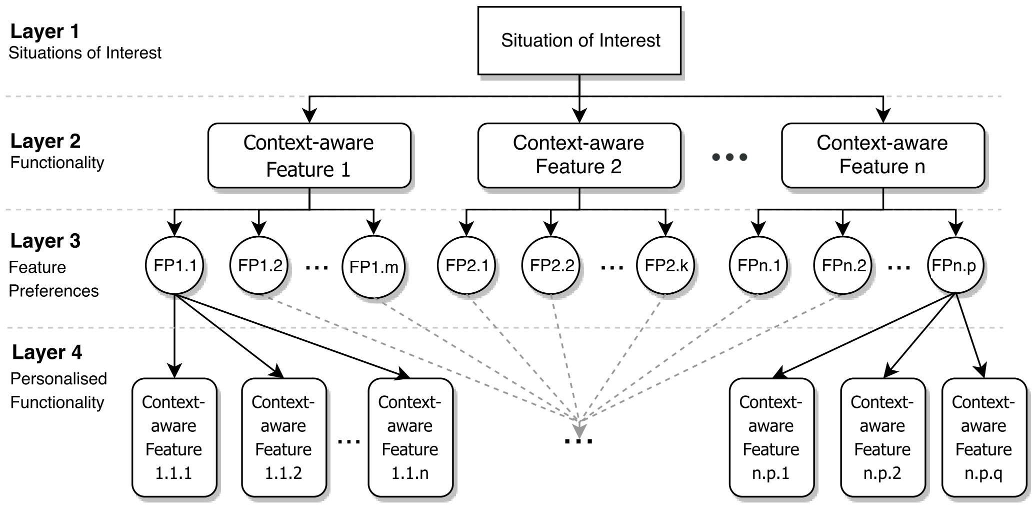


**Figure 2.3: Decomposition of the different concepts related to the functionality of the system and its personalisation. Note that additional layers for feature preferences and personalised functionality could be added, as context-aware features can be decomposed in other context-aware preferences depending on the feature preferences of the stakeholders.**

### 2.4.7 Personalisation management

One important feature of the introduced conceptualisation is the end-user stakeholder centred perspective. For this reason, it is also important to introduce a way in which preferences can be handled using this conceptualisation. Section 2.4.1 has introduced an interaction modality that enables the configuration of the context-aware application, either in an active or a passive way. Particularly, the following subsection focuses on the modelling of subjective knowledge relative to the preferences of the end-user stakeholders, by introducing concepts to abstract the specific values of user preferences. This facilitates the treatment of generic knowledge about preferences, letting the users, after the system is implemented, define their own preferences as they use the application. For the passive configuration modality, the final values of preferences are meant to be determined by their own users. For the active configuration modality, instead, the final values of the preferences are meant to be determined by preference learning algorithms. The abstraction of particular preference values helps to handle subjective knowledge during the requirements, design and implementation stages. The preference configuration of the end-user stakeholders can be categorised in one of two ways, according to the introduced context conceptualisation, as follows:

#### A - Context-preference

**Definition 5.** Context-preference:
*A type of context-attribute whose particular values are to be personalised by a stakeholder or an agent after the system implementation.*

The first configuration type is related to the detection of a situation of interest, via personalisation of certain context-attributes. For example, let the Reader imagine a scenario where it is required to know the temperature, and where a certain threshold temperature will detect a situation of interest where the associated context-aware feature is to automatically turn on the heating. In this hypothetical scenario there will be two context-attributes for detecting the situation of interest: temperature and temperature threshold. It is important to highlight that under the current context definition, both will be considered as being context. Particularly, the temperature threshold will be considered as a context-preference, which can be configured after the system is implemented.

#### B - Feature preference

**Definition 6.** Feature preference:
*A non-contextual software variable which is used to personalise the way in which a context-aware feature is provided.*

On the other hand, the second configuration type consists of personalising the provision of context-aware features according to the user preferences. Under the current context conceptualisation, feature preferences are not used for identifying a situation of interest, but to personalise the way in which a context-aware feature is provided. Therefore, the Reader should note that they can not be considered as context. For instance, let the Reader imagine that in the previous scenario, apart from triggering the heating to turn itself on, there is another context-aware feature which is to prompt the user informing that the heater has been turned on. Some users with visual difficulties might prefer to receive this notification with a big font size. Other users might prefer to receive this notification with a normal font size. In this case, the feature preference associated to the provision of the context-aware feature will be the visual acuity of the user.

## 2.5 Conclusions

This section has reviewed the literature on context conceptualisation for context-aware computing, concluding that there is no consensus on what context means. A further examination of the notion beyond the area of context-aware computing sheds some light on the causes for this disagreement. The main challenge behind the conceptualisation of context is that computing systems have a different philosophical tradition than that of social sciences, which is behind the traditional explanations of context. This chapter further explores the current limitations in the conceptualisation of context for the development of context-aware systems, concluding that, to the possible extent, there is a need to conceptually support developers in: A) Enumerating the set of contextual states that may exist; B) Knowing what information could accurately determine a contextual state within that set; C) Stating what appropriate action should be taken in that particular state. Additionally, it is concluded that even with adequate support, it is very difficult or even impossible to foresee all the situations in order to program them, and that even some situations that might seem similar a priori, can greatly differ from the actual instantiation of the situation. Context-aware computing is a still maturing technology, where context-aware systems have a high chance of taking actions which might not be the most appropriate in certain situations.

The conceptualisation presented in this chapter is not intended to solve all these challenges, but rather to encourage further research in the direction of maximising the usability results of context-aware systems, mitigating the limitations of C-AS and strengthening their real capabilities. The underlying premise for this purpose is that current C-AS can maximise their usability results when:

- Developers adequately identify those situations that, to some extent, are possible to be predicted and represented as computational models.
- Developers understand the intention, meaning of the actions, preferences and needs of the users in a particular situation of interest (end-user stakeholder centred perspective).
- There exists a set of observable properties which can identify a situation of interest with enough accuracy to distinguish it from the maximum number of similar-looking situations where the context-aware feature to be displayed is not appropriate.
- The properties that identify a situation of interest can realistically be attained from sensors.
- The developers can implement the situation of interest detection while it is still meaningful for the users.

The updated conceptualisation of context and context awareness presented in this chapter takes into account three main aspects. The first aspect is the separation of concerns into the mentioned three high-level ideas that allow developers to get the context right. The concept of situation of interest is introduced as a central tool which enables the separation of the information used by the system to identify a situation and the services to be provided under that situation. Besides, it also provides an abstraction to make the developers understand and be guided during the process of developing context-aware systems. The conceptualisation approaches context acknowledging its two philosophical perspectives. The situation of interest enables the management of the notion as an observer-dependent phenomena. This concept is used as a key nexus between the detection of the situation via context-attributes, and the triggering of different services which are associated with the situation of interest. Also, a conceptual tool for managing the preferences of the users is introduced. The second aspect is an end-user stakeholder centred perspective that on one hand considers the preferences and needs of the stakeholders, and on the other hand extends the notion of user to that of stakeholder. Finally, the last aspect is the consideration of different interaction modalities to mitigate the effects of using a still maturing technology. The outcome of this chapter has been used as foundation for the rest of this dissertation, with special influence into the design principles stemming from this definition, introduced in Chapter 5.

# Chapter 3

# State of the art

## 3.1 Introduction

This chapter presents an overview of previous work that is related to the development of context-aware systems, which is partly based on a published state-of-the-art survey conducted as part of the contributions of this thesis [45]. Current research in the development of context-aware systems happens to be typically scattered and disconnected, mainly adapting traditional software development methods to some particular needs of context-aware systems. As previously introduced in Chapter 1, the aim of this dissertation is to help build the foundations of a more holistic approach in the development of context-aware systems. It becomes necessary, then, to conduct a literature review to find the most adequate tool supported frameworks and methodologies, as well as the existing gaps, in order to propose a more holistic framework which can be used as the backbone of an engineering process with regard to these systems. An initial exploration of the literature shows that there is a vast amount of research in the field of context-aware computing, and in similar areas such as that of Ubiquitous Computing, Intelligent Environments or Ambient Intelligence. In order to avoid further review of dated, and perhaps redundant works in some cases, this survey has been strategically divided into three different research themes, according to the challenges explored in Chapter 1, and the insights obtained from the conceptualisation of context in Chapter 2. These themes are as follows[1]:

- Requirements: In Chapter 2, it was concluded that there is a special need during the development of context-aware systems, and this is the need for guiding developers in the design of adequate systems [Chg1]. For this, there is a need for developers to exercise an understanding of the context of the system, to discover different contextual services that can be provided to the end-user stakeholders. This understanding process has a strong connection with the requirements elicitation process, which helps developers to reach a better understanding of the user needs and demands by finding a systematic approach for eliciting, analysing, documenting, validating and managing software requirements from individual stakeholders [91] [92] [93]. If the right requirements are not well-defined

[1]Note that the following explanation includes a citation in the form of [Chg$x$], referring to those challenges mentioned during Chapter 1.

prior to the development of the system, it will be more likely to fail meeting the user and other stakeholder's expectations (stakeholder validation). An early identification of the implementation feasibility of certain situations of interest, or the triggering of different associated services, can save the development team from wasting efforts which can translate into high development costs [Chg4]. Therefore, the creation of a coherent framework for requirements elicitation will occupy a considerable part of the research efforts of this dissertation, as it is considered as a key stage for providing guidance to the main aspects for getting the context right [35]. Consequently, there is a need to research the existing methods for this purpose, in order to find and fill existing research shortcomings.

- Model driven development: Another important aspect of context, as introduced in Chapter 2, is its dynamic nature. It is important to note that not only the context, but the associated services of a context-aware system are expected to be in constant evolution, in order to meet the always changing demands of the end-user stakeholders [Chg6]. Therefore, an important aspect of a development framework is that it speeds the process of implementing those changes. For this reason, the model-driven approach is chosen as another of the three main research themes to be further analysed. Model-driven development leverages the development abstraction level, enabling the development of the system through graphical models and pre-built application components. Such an approach, not only is cost-efficient [Chg4] and fast, but it also facilitates the communication between stakeholders, empowering domain experts, and allowing developers to focus on the technical aspects. Additionally, this approach can help to manage the context information [Chg3], leading to more meaningful validation, as the high-level abstraction can avoid functional errors [Chg5]. Existing model-driven approaches to context-aware systems development will be analysed, in order to find potential applications which can be reused in the context of a more general framework.
- Implementation techniques: Finally, the last research theme focuses on the different techniques which have been used for implementing context-aware systems, so that the model-driven development can be mapped to the code automa-

tion focused on these implementation techniques. Once techniques are identified, further research is constrained to existing tools for giving response to the different techniques. The aim is to find which tools have the more useful aspects for a context-aware engineering methodology, discover the gaps between the different tools, so that they can be further extended into a more holistic tool which can support the implementation and deployment of diverse systems [Chg2].

The remainder of the chapter is as follows. Section 3.2 introduces the state-of-the-art in requirements elicitation for traditional and context-aware systems. With this information, Section 3.3 focuses on the literature review related to the model-driven development of context-aware systems. Section 3.4 analyses the different implementation techniques used for creating context-aware systems. With this information, an analysis of existing tools for this purpose is created. Finally, Section 3.5 concludes with a set of the most adequate methodologies and tools which will be used to support the creation of a framework for the engineering of context-aware systems.

## 3.2 Requirements engineering

This subsection focuses on requirements engineering for both context-aware and traditional systems, with two main purposes. The first is to analyse the suitability of the different approaches to elicit requirements for context-aware systems. The second aim is to identify existing methods and techniques that could be used in partnership to provide a more holistic requirements elicitation methodology. The remainder of the subsection explains goal-oriented, scenario-based, and hybrid techniques for this purpose.

### 3.2.1 Goal-oriented

Goal-based requirement engineering techniques use the concept of a goal as a logical mechanism for identifying, organising and justifying software requirements [94].

#### 3.2.1.1 Framework for Requirements Engineering for Context-Aware Services

Finkelstein and Savigni [95] introduce a goal-based framework for eliciting requirements of context-aware services. They present five main elements to complement the concept of a goal: *Environment*, which is whatever in the world that provides a surrounding in which the machine is meant to operate; *Context*, understood as the reification of the environment; *Requirement*, which represents one of the possible ways of achieving a goal; *Service description*, a meta-level representation of the actual, real-world service; and *Service*, which provides the actual behaviour as perceived by the user. Along with these elements, Finkelstein and Savigni also introduce seven different relationships among them: Context *influences* requirements, while goals *operationalise* them; The requirements of the system *determine* the service descriptions, which *reflect* and are *reified* by services; The environment *monitors* context, and *constrains* the services provided.

#### 3.2.1.2 Requirements Analysis using feedback from context

Oyama et al. [96] present an approach to elicit goals using feedback from context-aware systems based on templates from the CAPIS model [97]. The context goal elicitation is based on Data, Information, Knowledge, and Wisdom (DIKW) templates [98]. First, the structure of DIKW guides developers to infer situations and intentions of a user, conceptualising service problems and issues through goal elicitation. Then, developers describe service problem templates, which use DIKW to help them describe contexts as a set of <*situation, cause, effect*>. Finally, developers use service issue templates, which use DIKW to help them describe corresponding goals as a set of <*alternatives, goal, significance*> that explains the engineer's decision process for identifying the user goal. The overall process has three main steps:

1) Conceptualisation of a service problem based on the contexts. First, a list of contexts and its meaning are extracted from the information layer and then it lists situations for the users;
2) Goal identification through the temporarily desirable results of previous and changed intention;
3) The conceptualisation for a service issue. First, the process in the knowledge

layer describes tools, methods, and environments of services to achieve a goal. Then, the goal meaning is explained and a goal graph is shown as data.

#### 3.2.1.3 FLAGS: Fuzzy Live Adaptive Goals for Self-adaptive systems

Goal relaxation acknowledges the partial or fuzzy satisfaction of goals. While the methods introduced in the previous subsection consider the completion of a goal as something binary, this approach also acknowledges some objectives as being so vague that cannot be objectively measured. The introduction of fuzziness in the satisfaction of goals enables engineers to define objectives without the sufficient definition, giving the team a sense of direction and purpose while leaving them free to follow their intuition.

Baresi et al. [99] introduce an extension of KAOS [100][101][102], another goal-oriented methodology for requirements engineering that is supported by a specification language. It is focused on adaptive systems, which can benefit from the allowance of small violations in the satisfaction of goals. The process they propose is as follows: 1) Authors create a goal model following the traditional KAOS notation. In such models, conflictive relations between goals are identified. Those in conflict are assigned a priority. Finally, Temporal Linear Logic is used for formalising the objectives. 2) Goals in conflict are softened, by using fuzzy logic. There are two types of goals: crisp and fuzzy. The former are binary while the latter map to a greater range. 3) Specify the adaptation at execution time through adaptive goals, which have: a trigger for determining when such goal needs to be satisfied, a condition for its activation and an objective to achieve. Moreover, a set of actions to be taken is specified, including to add or remove goals from the model, modify the preconditions of a goal, add or remove entities, events or agents.

### 3.2.2 Scenario-based

Scenario-based requirements elicitation approaches use descriptions of activity sequences and foreseeable interactions with the different user roles of the system under development to extract and determine its requirements. Scenarios can be used as: descriptors of the unsatisfactory state of affairs that the system under development has to solve;

Visions of how the system might operate; Descriptions of behaviour that represent the users and the system [103].

#### 3.2.2.1 Discovery of requirements for context-aware systems

Seyff et al. [104] [105] present a framework for the scenario-based requirements engineering of context-aware systems. They show that is feasible to develop requirements elicitation approaches which use ubiquitous technologies to identify the current work context. Their approach is divided into three layers. The first level supports conventional approaches for understanding context, such as contextual inquiry that supports comprehending users in their workplace and examining how people work to meet real world requirements. The aim is to have an understanding of the context and no software tool support is required. The second level represents user-driven requirements engineering tools which enable analysts to elicit requirements in the work context. Finally, the third level is supported by context-aware requirements engineering tools that are capable of identifying the current work context. Such tools offer features to inform users about context changes.

### 3.2.3 Hybrid approaches

There are also approaches which combine the strengths of scenarios with the representation of goals.

#### 3.2.3.1 PC-RE: Personal and contextual requirements engineering with some experience

Sutcliffe et al. [106] propose a three layered framework to explore requirements specification for specific individuals. The method focuses on the evolution of individual needs over time and its evolution once the individual learns how to use the system. Each layer contemplates changes on the spatial and temporal dimensions. The layers are described as follows:

- Stakeholder group: This layer collects the most general requirements of the system as well as its evolution over the two dimensions. The changes process (temporal change) and cultural differences or internationalisation (spatial change).

Contextual enquiry and prototype evaluations specify product versions for different cultural markets. Other architectural implications are: Design for customisation, design of monitors and adaptive functions for mobility; customisable or adaptive user interfaces for changes depending on the evolution of user skills, and a flexible adaptable architecture to evolve as business processes change.

- User characteristics: In this layer, the system-domain independent needs of the users are captured in a model. Abilities and needs are gathered from interviews and psychology-based questionnaires and captured in a user profile. The spatial dimension is affected by changes in the physical and social context, while the temporal dimension is affected by the changes in the abilities of the users over time.
- Personal goals: This layer analyses the requirements from an individual point of view. Changes in the temporal dimension depend on the stability of user desires, while the spatial change depends on how the location affects the goals.

The personal-level consideration of requirements entails the appearance of many alternatives to satisfy similar goals. To help prioritise and discard such alternatives for a later design and implementation, authors propose a cost-benefit analysis technique, consisting of: (a) Estimated benefits of achieving the desired goal; (b) Cost of each design alternative; (c) Cost penalties if the solution alternative does not achieve the goal; (d) Costs associated to infringements of non-functional requirements.

#### 3.2.3.2 RE-CAWAR: Requirements engineering for context-aware systems

Sitou and Spanfelner [107] propose the use of an integrated model of usage context in which three different dimensions that affect the context are considered: Changing participants, in regard to the location and orientation of the users, as well as their personal, mental and psychological states, expectations and social dependencies; Changing activities, referring to the tasks and goals of participants influenced by events in the environment and; Changing operational environment, such as the location of the application, network conditions, devices and communications quality, etc. In order to help the systematic handling of context, the authors propose the following models:

- User, representing the aspects of users and user groups.

- Task, responsible for identifying which task and which interactions are needed to perform a task.
- Domain, modelling the operational environment which consists of any user visible, accessible and manipulable objects in the system domain.
- Platform, which shows the physical infrastructure and the relationship between involved devices.
- Dialog, representing the interaction between the user and system.
- Presentation, that includes the visual, haptic and audio elements needed for the interaction.

The core of the methodology is based on approved methods from traditional and model-based requirements engineering such as scenario and goal-based approaches. It is based on two main parts. The first part is the stability check, that consists of eliciting, analysing and specifying general requirements for the system core and the user interface. For this purpose, use-cases based on previous models are employed and context dependent needs are identified. Then, this data is used as input to the next part, identification checking. In this part, the aim is to identify needs that could be automatically recognised by the system at runtime and at converting them into adaptation requirements.

#### 3.2.3.3 REUBI: Requirements engineering method in ubiquitous systems

Ruiz-Lopez et al. [32] present a hybrid method between scenario and goal-based requirements elicitation that provides guidance to discover the top-level goals of the system and its non-functional requirements. It has the following characteristics: (1) Has support for obstacle analysis. (2) Models the contributions of potential architectural and design decisions to satisfy different objectives. (3) Collects rationale to support the decision-making process during the requirements analysis phase. (4) Shows the impact of the context on the objectives that need to be satisfied. (5) Allows representing variable prioritisation with regard to the satisfaction of objectives related to the influence of context in priority changes. (6) Provides an evaluation procedure to determine which decisions are the most suitable for the satisfaction of objectives in different contexts. The method is as follows:

1. Value Model: Represents the main interests of the stakeholders (actors) and

defines the relevant goods, services or information (value) that will exchange in the system under development. It also represents the quality properties that enhance a certain value or the process of acquiring it, relating them to the different values.

2. Goal and soft-goal refinement: From the value model, an extension of the soft goal interdependency graph is used to decompose goals and soft-goals hierarchically. Scenarios can also be considered during this stage.
3. Obstacle Analysis: Apply a sequential procedure to deal with adverse and undesirable situations which are identified and related to the goals or soft-goals they may hinder.
4. Resource Exchange: Relationships between objectives and resources are identified. A resource is a set of data that is originated after the realisation of an objective or that is necessary for its satisfaction.
5. Operationalisation of goals and soft-goals: Alternatives that help to achieve goals and soft-goals are proposed, which can be decisions that refer to the way in which software is structured (architectural), or can refer to the detailed definition of software components and connectors (design). Sets of operationalisations with the same purpose can also be grouped. Finally, the relation towards the different goals is analysed, whether or not they have a positive or negative impact on them, or whether or not they make its achievement or deny its associated group.
6. Argumentation: Maintains a record of the decisions made during the previous steps by modelling the claims, or reasons that justify the decisions taken.
7. Context modelling: The impact of context situations on other elements of the interdependency graph is modelled, rather than the actual representation of the context of the system under development. They include three main components: (1) Context situation: A set of attributes representing a situation of interest in the system which has impact on the interdependency graph; (2) Context attribute: Any observable property in the environment which characterises a situation of interest to the system; (3) Context value: A possible value for a context attribute. They use a context-dependency relationship to trace the context situations to the element that needs to be addressed when the situation

arises.

8. Prioritisation: Objectives are classified into three different levels of priority: normal, important and critical. Context situations are related to the relationships of decomposition or operationalisation they affect by means of a context-dependency.
9. Evaluation procedure: Based on the NFR framework and the use of rules, an evaluation is conducted to check the satisfaction of the goals and soft-goals contained in the model.

#### 3.2.3.4 R4IE (AmI): Requirements for Ambient Intelligence

Evans et al. [108] analyse the state-of-the-art in requirements engineering for context-aware systems, and acknowledge the following prominent themes: a) A consideration, adoption and possible enhancement of a context taxonomy; b) General assumption that systems need to be adaptable to be context-aware; c) In HCI focused works, elicitation techniques for capturing end-user cognitive tasks require enhancement to account for context-awareness; d) Identification of target groups in contrast of individual user customisation and an acknowledgement that contextual requirements for different profiles may evolve over time; e) Requirements may be context-driven and change dynamically, unlike high order operational goals; f) Consideration of cultural context; g) The adoption of goal-based requirements engineering and support for the adoption of scenario-based modelling. They also map the context-awareness domain to that of ambient assisted living, acknowledging that: 1) Goal oriented tasks are usually evident; 2) Context can vary significantly, and there can be associated prioritisation of design and implementation activities in terms of the services associated with those contexts; 3) Depending on the nature of the assistance, there can be a demarcation between individual requirements with a degree of customisation and distinct user groups that support an individual who also present different set of requirements; 4) When distinct target groups and individual stakeholders are identified, a prioritisation of the highest stakeholder value is demanded; 5) There can be important ethical issues, including the matter of privacy relating to context; 6) Strong association between requirements for interaction design and the design of appropriate user training support due to differences in aspects such as modality, physical and cognitive skills and experience; 7) Need for harmonisation in terms of coordination, planning, and management of different specialist knowledge required for ambient assisted living. Taking into account these themes, they present a requirements engineering process for Intelligent Environments which has been instantiated for ambient assisted living. As it can be observed in Figure 3.1, the process follows five main steps which are carried out iteratively. The central step consists of establishing the scope in terms of the boundaries of the project. This activity is complemented by the following four phases: establishing the high-level goals, identifying the tasks through scenario-based techniques, identifying system performance qualities and identifying stakeholders.

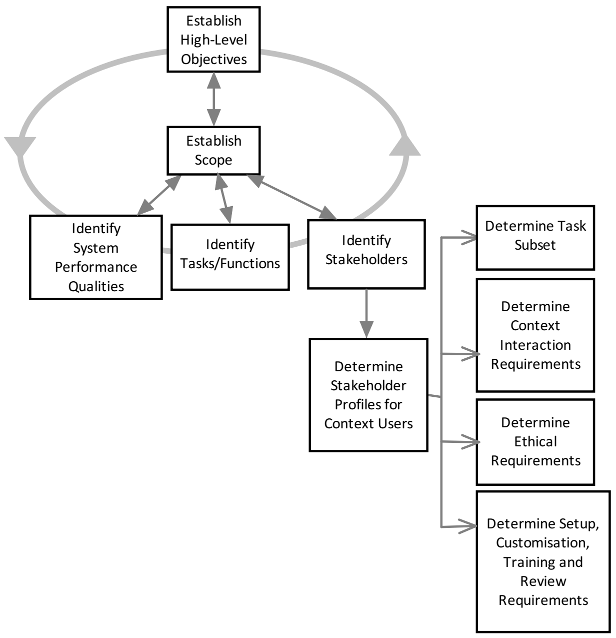

Establish High-Level Objectives
Establish Scope
Identify System Performance Qualities
Identify Tasks/Functions
Identify Stakeholders
Determine Task Subset
Determine Context Interaction Requirements
Determine Stakeholder Profiles for Context Users
Determine Ethical Requirements
Determine Setup, Customisation, Training and Review Requirements

**Figure 3.1: Core Activities in the R4IE (AAL) Framework.**

### 3.2.4 Analysis

An analysis of the previously described requirements methodologies is presented in Table 3.1. In order to determine the compatibility between the different requirements engineering methodologies, the table covers the following aspects:

- Scope: It analyses the coverage of the methodologies for the typical elicitation activities. Columns 4, 5 and 6 represent the support of the methodologies for acquisition, elaboration and modelling activities respectively. It can be observed that most of the methodologies give support for these three main activities.

- Approach: Columns 8 and 10 represent whether or not the methodology is based on goals, scenarios or a hybrid approach. Column 9 describes if the methodology has support for the partial satisfaction of goals rather than just being binary. Column 11 indicates whether or not the methods provide specific and systematic treatment of non-functional requirements. The theme is to have a goal-oriented approach and then offer support either to scenario-based techniques or the partial satisfaction of goals. The most complete approach is that of REUBI [109], which has the potential to cover all these approaches.
- Context-awareness: Column 12 indicates whether or not the methodology takes into account the needs, preferences and limitations of the end-users, or in its absence, they support personalisation to a certain degree. Only three methodologies support this feature, from which PC-RE and R4IE are highlighted. Column 13 shows whether or not the methodologies take into account the influence of contextual aspects. Many methodologies support this, but each has its own particular way to manage this. Column 14 indicates if the methodology has specific support guiding developers into: (a) enumerating the set of contextual states that may exist, (b) knowing what information could accurately determine a contextual state within that set, and (c) stating what appropriate action should be taken from a particular state [35]. Oyama et al. [96] present a series of templates for this purpose which could be reused for other methodologies.
- Tool support: Column 15 and 16 show whether or not the approaches have tool support and if such tool is freely and easily available for other researchers to be extended.

The analysis of the methodologies has been conducted for a number of aspects that are considered relevant for the topic of this dissertation. From the point of view of the analysis of those aspects, REUBI [109] is the most complete methodology, as it can be observed in Table 3.1. Nevertheless, there are three main aspects that this method does not complete. Namely, the explicit lack of a user-centred perspective, and a lack of a tool which is publicly available. Also, it does not provide guidance for de-

| No. | Name | Year | Reference | Acquisition | Elaboration | Modelling | Goal-oriented | Softgoals | Scenario based | NFR treatment | User-centred | Context-awareness | Developer Guidance for context | Tool support | Open source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) | (6) | (8) | (9) | (10) | (11) | (12) | (13) | (14) | (15) | (16) |
| (a) | - | 2001 | [95] | - | ✓ | ✓ | ✓ | - | - | - | - | ✓ | - | - | - |
| (b) | PC-RE | 2006 | [106] | ✓ | ✓ | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | ~ | - | - |
| (c) | RE-CAWAR | 2007 | [107] | - | ✓ | ✓ | ✓ | - | ✓ | - | - | ✓ | ~ | - | - |
| (d) | - | 2008 | [96] | ✓ | - | - | ✓ | - | - | - | ✓ | ✓ | ✓ | - | - |
| (e) | - | 2008 | [104] | ✓ | ✓ | ~ | - | - | ✓ | - | ~ | ✓ | ~ | ✓ | - |
| (f) | FLAGS | 2010 | [99] | - | ✓ | ✓ | ✓ | ✓ | - | - | - | - | - | - | - |
| (g) | REUBI | 2013 | [32] | ~ | ✓ | ✓ | ✓ | ✓ | ~ | ✓ | ~ | ✓ | ~ | ✓ | - |
| (h) | R4IE | 2014 | [108] | ✓ | ✓ | - | ~ | - | ~ | ~ | ✓ | ✓ | - | - | - |

✓ = The property is completely satisfied. ~ = The property is partially satisfied. "-" = The property is not satisfied at all.

**Table 3.1: Comparison of current methodologies for requirements engineering in traditional and context-aware systems.**

velopers to discover context, according to the three main principles to get it right [35]. From the point of view of guiding the developers towards the discovery of situations and context, the data, information, knowledge, wisdom model of Oyama et al. [96] could also be employed for this purpose. Nevertheless, Oyama's model lacks mechanisms for elaborating and modelling requirements. Compared to REUBI [109], it also lacks mechanisms for handling soft goals and non-functional requirements. Additionally, there is no tool support for this approach. For the purpose of this dissertation, the REUBI methodology [109] is the most relevant reference point. Therefore, it is concluded that this tool will be used as the foundation from which the requirements framework for engineering context-aware systems will be built. Although this approach has partial support for scenario based techniques, which can be used to understand and gather the context of the system, this is not a necessary requirement for this thesis, and it can be further complemented. A necessary aspect that needs to be covered for this thesis, is that of the user-centred perspective. This gap can be addressed by combining other existing methodologies. The R4IE [108] and PC-RE [106] approaches have some synergies that can be used to complement this characteristic. Additional techniques for analysing stakeholders and their needs can also be useful for this purpose. In order to address the shortcomings related to guiding developers into context discovery, a set of guidelines which are based on the conclusions of Chapter 2 will be included as part of the methodology. Finally, and significantly, whilst the REUBI approach has no explicit open-source tool support, the work described here has a specific goal of developing an open-source tool to support the proposed software development framework, which includes specific support for requirements engineering.

## 3.3 Model-driven engineering

Section 3.2 has discovered a set of different methodologies which will be reused to create a requirements elicitation process which is specialised in the engineering of context-aware systems. As part of the objectives of this dissertation, an open-source tool to support this methodology is required. As previously mentioned in Section 3.1, a model-driven approach is acknowledged as being effective for managing the adaptation of context-aware systems [Chg6], speed-up and reduce costs in the development process

[Chg4], handle the modelling of context information [Chg3], and create more reliable systems [Chg5]. Therefore, this section will be constrained to the search of tools which can aid the modelling of requirements and design elements, empowering these mentioned aspects.

The model-driven paradigm raises the level of abstraction in the specification of the systems to models, which are not treated just as documentation artefacts but also enable the automation of its development. Modelling helps developers to see and solve the most important problems at early development stages, also preventing developers from getting distracted by swarms of detail that are better to suppress until later [110]. Models can be used to communicate ideas between stakeholders with expertise in different areas, avoiding the ambiguities of spoken language. This particular approach is useful to the objectives of this thesis, as models might be used throughout all the engineering process, in any discipline, and in any application domain. Also, it needs to be mentioned that existing meta-models are naturally compatible with new meta-models, domain specific languages, or profiles, facilitating the re-utilisation of different already existing pieces in order to form a bigger picture. Moreover, since models raise the level of abstraction, they can be cheaply and easily reused to maintain and evolve the code, which is especially useful for the dynamic nature of context. Model-driven engineering is a broad discipline, which comprises the following three high-level approaches: Model-driven development, model-based testing, and model-driven architecture [111].

- Model-Driven Development (MDD): Is mainly focused on the requirements, analysis and design, and implementation disciplines. Provide modelling languages to specify the system under study, creating model to model (M2M) and model to text (M2T) transformations in order to improve the productivity and quality of the process [111].
- Model-Driven Architecture (MDA): Is a standard model-driven approach proposed by the Object Management Group (OMG) [112], which is primarily focused on the definition of models and their transformations.
- Model-Based Testing (MBT): Is primarily focused on the automation of the testing discipline. The models in this approach represent the desired behaviour of the system under test, the testing strategies, and the testing environment. Test

cases derived from such models are functional tests on the same level of abstraction as the model, and might then be mapped into executable tests with specific testing tools and frameworks.

A modelling language is the set of all possible models that are consistent with the modelling language's abstract syntax, represented by one or more concrete syntaxes and that satisfy a given semantics [111]. The purpose of models ranges from assisting different stakeholders' communication to testing case generation or the automatic implementation of the developed system. There are many modelling languages, from behaviour trees [113] to Petri nets [114], the CK theory [115] or the Object-Role Modelling [116].

### 3.3.1 UML: Universal Modelling Language

A prominent example of a modelling language is the OMG [117] Unified Modelling Language (UML) [118]. It is a standardised general-purpose notation that provides a way to visualize the design of a system. It was conceived to promote communication and productivity, avoiding the risks that emerge from miscommunication between stakeholders when resolving complex problems. Although it was aimed to support the development of object-oriented software, it has made inroads into almost every type of system and software development [110]. The UML language is methodology independent and has a generic extension mechanism for customising models for particular domains and platforms, also known as profiles [119]. This characteristic of UML has enabled developers/designers to extend and customise it to for different purposes, including several Domain Specific Languages (DSLs).

### 3.3.2 SysML: Systems Modelling Language

The OMG [117] Systems Modelling Language (SysML) [2], is a general-purpose modelling language for systems engineering applications. It supports the specification, analysis, design, verification, and validation of a broad range of complex systems that may include hardware, software, information, processes, personnel, and facilities. It reuses a subset of UML 2, providing additional extensions needed to address the requirements. This standard has been used in industry for helping engineers describe the system to be developed, including hardware and software aspects simultaneously. A SysML specification is more precise than its natural language equivalent [120]. Since UML is part of SysML, systems engineers modelling with SysML and software engineers modelling with UML 2 will be able to collaborate on models of software-intensive systems. It can also be used along with other standards such as the UML Testing Profile (UTP), or other UML-based requirements profiles such as UML-AT [121]. Besides, it also has the potential to be used along with other requirements capturing approaches and languages such as KAOS [122] [102] and RELAX [123]. On top of these advantages, SysML enables the traceability of requirements through all the stages of the development process, which is also a key goal with respect of the context-aware engineering approach of this research. Although there are other languages [124] [100] and frameworks [125] for modelling requirements, SysML and UML have more potential to cover all the stages of the development process [126], which is key for the research described here.

### 3.3.3 UTP: UML Testing Profile

UTP is a standardised language based on the OMG Unified Modeling Language (UML) for designing, visualising, specifying, analysing, constructing, and documenting the artefacts commonly used in and required for various testing approaches, in particular model-based testing (MBT) approaches [127]. It is the only standardised language for Model-based approaches to help in the validation and verification of software-intensive systems. The standard can be used along with UML for the following purposes [127]:

- Build the model-based test plans on top of already existing system models.
- Model test cases in order to assess the quality of the test item and verify if it complies with its specification.
- Model test environments, including hardware, software, instrumentation, simulators, software tools and other support elements.
- Model deployment specifications of test-specific artefacts.
- Model the data, including the values used as stimuli into the test system, as well as for responses expected from the test item such as the test oracle.
- Provide the information for test scheduling optimisation.

- Document test-case execution results, to associate test cases with the actual outcome of their execution within the very same model in order to perform tasks such as a more extensive analysis, and the calculus of more specific metrics amongst others.
- Document traceability to requirements and other UML model artefacts.

### 3.3.4 Other approaches for model-driven context-aware systems development

Sheng and Benatallah [128] presented ContextUML, a modelling language based on the Unified Modelling Language (UML) [129] for the model-driven development of context-aware web services. Serral et al. [130] [131] introduce a model-driven development method for context-aware pervasive systems. It applies the Model-Driven Architecture (MDA) [132] and Software Factories (SF), along with the PervML modelling language and the SOUPA ontology. Tesoriero et al. [133] presented CAUCE, a methodology based on MDA [132], to provide a model-driven development of applications for Ubiquitous Computing environments. It is also worthy to be mentioned that there are some Domain Specific Languages (DSL) for the development of context-aware software systems [134] [135]. Recently, a domain specific language called Trigger-action programming [136] [137] [138] is gaining popularity. This programming model is based on the End-User Development paradigm [139], where average users can manually customize a service according to their preferences, likes and expectations [138]. By reducing the complexity of programming, expressing the system behaviour becomes accessible to end-users. These, only need to handle simplified if-then programming rules that match a trigger with an action. Is starting to emerge in areas such as smart-homes/buildings [140] or smartphones [141]. Services and applications such as IFTTT [2] or Tasker [3] let end-users create rules with sensors/devices that

[2]IFTTT (If This Then That): Is a web-based service that allows users to create chains of simple conditional statements, triggered based on changes to other devices or web services (Facebook,Gmail,Calendar). https://ifttt.com/

[3]Tasker: An android application for performing tasks based on contexts (application,time,date,location,event,gesture) defined in user profiles or in click-able or timer home screen widgets. http://tasker.dinglisch.net/

they already have and use in their daily life. Although there is a considerable corpus of research in regard to the model-driven engineering for the creation of context-aware systems [142] [143] [144] [28] [22] [24] [30] [109] [145] [146] [32] [25] [31] [147] [3], the further analysis of these approaches is out of the scope of this thesis, since the most common approach is the creation of UML profiles that facilitate the automation of implementation code and these approaches are specific to their corresponding implementation frameworks. A decision is taken for the framework presented in this thesis to use a domain-specific modelling language, based on the UML profiling extension mechanism.

### 3.3.5 Tool support

There are different existing tools for the creation of SysML-based diagrams. As mentioned in the objectives of the thesis (Section 1.4), it was an important consideration for this thesis to provide an open-source tool which can be extended by other researchers. There is less open-source support for modelling SysML-based constructs. At the time when this research started, only Modelio [148] and Papyrus [149] provided SysML support to open-source free tools. At this point Modelio was a more mature alternative, in its third version, while Papyrus still remained in its first version. Both tools are based on the Eclipse IDE [150], Modelio as an RCP standalone application, and Papyrus as a plugin. The main difference is that Modelio enables module creation, which can be flexibly integrated as a plug-in of the development environment, also enabling the creation of new profiles. Compared to Papyrus, Modelio offers wider support for developers, in terms of community and development tools. Modelio also offered support for other standards in the form of external modules, as it is illustrated in Table 3.2. It is also worth noting that Modelio offers the *link edition view*, where interesting relationships among elements can be observed. This enables for example, checking of all the elements that would be affected after the removal or modification of a situation of interest. For these reasons, Modelio was chosen as the main software to be used as the backbone application for the tools created for this work, as it is an open-source tool which delivers a broad-focused range of standards-based functionalities for software developers, analysts, designers, business architects and system architects.

| Module | Description | Standard | Cite |
|---|---|---|---|
| MARTE Designer | Embedded software systems modelling using the OMG MARTE standard. | ✓ | [151] |
| SoaML Designer | Model SOA Architectures using SoaML standard (UML Profile). | ✓ | [152] |
| SysML Architect | Complex system modelling, complete SysML support. | ✓ | [2] |
| TOGAF Architect | Enterprise Architecture modelling with the TOGAF Standard. SOA Architecture and BPM support. | ✓ | [153] |
| UTP | Implements the UML Testing Profile (U2TP) standard, for modelling software tests. | ✓ | [154] [127] |
| Web Model Publisher | Generates documentation in HTML for web publishing. | - | |
| WSDL Designer | Web services modelling using diagrams and definition of the exchange message types. | - | [155] |
| XSD Designer | Generation and reverse of XSD schemas from/to UML models. | - | - |
| Java Designer | UML to Java code Generation, Java to UML reverse engineering. | - | - |
| JUnit | Creation of the test model from a Java model. | - | - |
| Excel Exchange | Excel spreadsheet generation and reverse. | - | - |
| Hibernate Resource | Model and automate Hibernate mapping and Java classes generation. | - | - |
| Teamwork Manager | Collaborative modelling environment that allows different team members to work on the same repository-shared project. | - | - |

✓ = Is a standard "-" = Is not a standard

**Table 3.2: Open-source and non-commercial modules for Modelio [5].**

Although the Modelio platform provides support for most of the SysML diagrams, the original SysML Requirements Diagram cannot be used in the non-commercial version. As part of the contribution of this dissertation, the missing tools will be complemented in order to make a free and open-source version of SysML, that includes SysML *traceability matrices* and *requirements tables*.

## 3.4 Implementation support

One of the most important themes for the implementation of context awareness is that of context information management as, typically, context-aware applications require a whole engine for this purpose. There has been some research with regard to context information management techniques [156] [157] [158] [90] [159]. Perera et al. [1] provides an analysis of context information management, proposing the context-information life-cycle, as further explained in Section 2.2.5. Based on this life-cycle, they provide an exhaustive analysis of the different techniques for implementing context-aware systems, as it is further summarised in the remainder of the section. More information on the advantages and disadvantages of each technique can be found in their original work.

### 3.4.1 Implementation techniques

The first step of the life-cycle is related to the acquisition of information from different sources. The following is a description of the five factors to be taken into account:

1. Responsibility: It is related to the way in which the data is provided to the software component in charge of acquiring it. There are two types: pull and push. In the pull or query based responsibility, the software component responsible for acquiring the data makes a periodical request to the sensor. On the other hand, in the push or publish/subscribe responsibility the sensor sends the data to the software component that is responsible for obtaining it.
2. Frequency: It is related to the frequency in which the context-information events happen, and there are two types: instant and interval. With the first type, the

events occur when a threshold is trespassed (*e.g.*, Switching on a light or opening a door). With the interval frequency, the events span a certain period.

3. Source: There are three different sources from which the context could come from. The first source is sensor hardware, where context is directly acquired from the sensor by its hardware and related APIs. In the middleware source sensor data is acquired by a middleware infrastructure. The last source type are context servers, where data is acquired from several other context storages such as databases, RSS feeds or web services.
4. Sensor Types: Based on the classification of Indulska and Sutton [90], Perera et al., explain three different types of sensors. Physical sensors are tangible hardware devices which measure a determined physical property. Virtual sensors are software-based sensors which do not necessarily generate data by themselves, but they retrieve it from many sources and publish it as sensor data (*e.g.*, calendar, contacts, emails, etc.). Logical sensors are a combination of physical and virtual sensors which produce more meaningful information.
5. Acquisition process: There are three main acquisition processes. In the sense process, the data is directly sensed from sensors. In the derive process, the information is generated by performing computational operations over sensor data. Additionally, users can manually provide context information via predefined settings options such as preferences.

In order to implement models related to context, there is a need for platforms and techniques with the power to support the expression and handling needs of context information. Below is a brief introduction to the most commonly used techniques for context representation and modelling [156]:

I) *Key-Value*: The simplest form of context models, involving a name and context value pair. Used to model limited amount of data such as user preferences and application configurations. Contain mostly independent and non-related pieces of information, which are suitable for limited data transfer and any other less complex temporary modelling requirements.

II) *Markup Scheme*: Hierarchical data structures are formed using these models, consisting of mark-up tags, attributes and content. It can be the intermediate data organisation format as well as mode of data transfer over network. It can be used to decouple data structures used by two components in a system.

III) *Graphical*: Modelling of context using graphical notation as UML, Object-Role Modelling, and other DSLs. Ideal for long term and large volume of permanent data archival. Historic context can be stored in databases.

IV) *Object Oriented*: Takes advantage of object-oriented concepts and techniques as encapsulation and inheritance. To represent context in programming code level. It allows context runtime manipulation.They work on a very short term, temporary and mostly stored in computer memory. Also, support data transfer over network.

V) *Logic Based*: Use facts, expressions and rules to define formal models. Different facts can be inferred separately and then used in existing rules to derive higher context knowledge. It is used for generating high-level context using low-level context, generating new knowledge. It is also used for modelling events and actions as well as for defining constraints and restrictions.

VI) *Ontology Based*: Can be used to describe taxonomies of concepts, including relationships. Besides, they allow different context reasoning techniques and inference rules. Rather than storing data in ontologies, data can be stored in appropriate data sources, while structure is provided by ontologies.

All techniques have their strong points and drawbacks. Although ontologies are the most widely adopted approaches, they still have some deficiencies that could be mitigated in hybrid approaches [158]. Although the representation and information retrieval in ontologies can be complex, they support semantic reasoning, expressive representations of context, have strong validation, are application independent, allow sharing, have strong support by standardisations and have fairly sophisticated tools available.

Once the context is modelled, there is a need to create new knowledge and have a better understanding based on the currently sensed context. Techniques for this purpose can be divided into [1]:

I) *Supervised Learning*: Training examples are collected to label them according to the expected results. Finally, a function can generate the expected results using the training data. Techniques such as decision trees, Bayesian Networks, Artificial Neural Networks and Support Vector Machines are considered in this group.

II) *Unsupervised Learning*: Techniques that can find hidden structures in unlabelled data. Such as K-nearest neighbour, Kohonen Self Organising Map (KSOM), Noise and outlier detection and Support Vector Machines.

III) *Rules*: One of the simplest, straightforward and popular reasoning methods. They usually have an IF-THEN-ELSE structure, but they can be based on simple mapping associations of IDs to entities (RFID) [87]. From the works reviewed in [1], it is recognised that the majority use rule-based reasoning.

IV) *Fuzzy Logic*: Allows approximate reasoning instead of fixed reasoning, extending the Boolean values 0 or 1 to expressions that simulate closeness to a natural language. The confidence values represent degrees of membership rather than probability.

V) *Ontological*: Based on description logic, ontological reasoning is supported by OWL and RDF, rules as SWRL, are increasingly popular.

VI) *Probabilistic*: It allows decisions to be made based on probabilities attached to the facts related to the problem. These include techniques such as Dempster-Shafer, Hidden Markov Models and Naive Bayes.

Finally, the context needs to be disseminated, for which similar techniques to those presented for the acquisition of context can be used.

### 3.4.2 Research prototypes and systems

A common approach to reduce development costs is to have a framework or middleware to increase the development speed. There is a also a great amount of research and prototypes on middleware aimed at facilitating the creation of context-aware systems. In order to fence in the analysis, the work of Perera et al. [1] is used to analyse existing systems and approaches that go along with the objectives of this thesis. This work surveys a total of 50 research prototypes, systems and approaches intended for easing the creation of context-aware systems.These approaches, have been filtered by their capabilities of using a combination of learning and reasoning techniques. There are two main approaches which meet the criteria: Cosar [160] and Intelligibility [87]. Cosar [160] combines ontological reasoning with statistical reasoning to achieve more accurate results in the field of activity recognition. Although it combines learning with reasoning, this is with the aim of discovering the context of the user, and the reasoning is not intended for triggering corresponding services, which puts the approach out of the scope of this research. In the intelligibility toolkit [87], the authors present a system which combines supervised learning techniques with rule and probabilistic reasoning. Nevertheless, the intention of their toolkit is to provide intelligibility to the users about why is the system taking decisions. Although the approach is quite interesting for reducing the rejection of the system by the end-users, it is also out of the scope of this dissertation. Other relevant cases such as CoReAmI [161], also present a combination of learning and reasoning, but are mainly focused on addressing three main dimensions of Ambient Intelligence: Activity recognition, human energy expenditure and fall detection. These approaches are more related to a specific area, while the purpose of this dissertation is to be more generic. In [162], the authors present a development environment which compatibilises learning tools for behavioural pattern discovery and reasoning capabilities. The M IDE facilitates the creation of context-aware reasoning components specified in a formal language, M, and the deployment of Z-Wave sensors connected to these components. The reasoning language M is also compatible with the output of a tool for discovering frequent patterns of user behaviour [88].

#### 3.4.2.1 M Language

M [162], is a temporal extension of a reasoning language [163] conceived to reason about deterministic devices and protocols. This temporal extension has been designed in order to be compatible with a tool for learning frequent patterns of user behaviour (LFPUBS) [88]. The LFPUBS environment learns patterns of the user, gaining knowledge about the preferences, needs and habits of the users, in order to facilitate the assistance provided. Knowing the frequent behaviours of users enables the system to act in a more intelligent and proactive way. The M system comes with a graphical interface to automate the translation of the outcome patterns of the LFPUBS system, into M compatible rules [164] [165]. Additionally, the graphical interface also enables the creation of system specifications in the M language, and deploy/maintain them into

Z-Wave based radio sensors, which are ideal for a smart-house or smart-office environment. Their language is open-tool supported and is compatible with a learning engine for learning frequent patterns of user behaviour [88]. For space restrictions, the full explanation about the internal functioning of the M approach can be found in Section 7.2 and in [162].

#### 3.4.2.2 Android Context Reasoner

The M IDE only supports the deployment of reasoning components in stationary platforms. As part of the POSEIDON project [14] [37], there has been an effort to create open-source tools [166] [167] that support the deployment of reasoning and learning components for context-awareness in mobile platform [3]. The approach presented as part of the framework for development in POSEIDON is based on stream reasoning using languages including C-SPARQL [168], where continuous streams of raw data in RDF can be reasoned over. C-SPARQL queries consider windows, i.e., the most recent triples of such streams, observed while data is continuously flowing. Supporting streams in RDF format facilitates interoperability and opens up important applications, in which reasoners can deal with knowledge evolving over time [169]. This approach has been created to be compatible with the temporal operators presented in [162].

### 3.4.3 Reliability

The system complexity, and hence the likely number of design errors, grows exponentially with the number of interacting system components. Although program testing can be a very effective way to show the presence of bugs, it is inadequate for showing their absence [170]. In these cases, verification techniques are used to explore some general properties about the behaviour of a program. Most of the verification done in C-AS is in the form of model checking, an approach to formal verification that proves whether or not a model meets a given specification. Along with verification techniques, C-AS are usually evaluated by using simulations. In these experiments, the behaviour of a system is imitated in order to provide a preliminary understanding of its performance. The rest of the section discusses some representative samples of the state-of-the-art for evaluating the design of a C-AS.

Formal verification techniques provide a safer development of systems in intelligent environments, aiding an increase their reliability [171]. Augusto et al. [172] show techniques as well as tools that can be used to model processes and interactions, detecting problems through simulation and verification in early stages of the development. In other work, Augusto and Hornos [34] present a methodological guide which provides strategies and suggestions on how to model, simulate and verify these types of systems. The methodology is divided in four stages: A) Informal modelling; B) Structural modelling; C) Behavioural modelling; D) Simulation and verification. The methodology is centred on a refinement strategy which starts identifying the core components of Intelligent Environments (sensors/actuators, actors, interfaces and communication mechanisms) and then works on successive models of increasing complexity. Although their methodology is tool-independent, they illustrate it using SPIN [173], a generic and open verification system that supports the design and verification of asynchronous process systems. Preuveneers and Berbers [174] also support a model checking approach in order to verify the many possible configurations and contextual situation that a C-AS can be in. They discuss the major benefits and weaknesses of the SPIN tool. D'Errico and Loreti [175] present a set of formal tools that allows specifying systems along with a model-checking algorithm to verify whether or not a considered specification satisfies the expected properties. They introduce $\mu$KLAIM, based on a simplified version of a Kernel language for agent interaction and mobility [176], which is based on an assume-guarantee approach: A system is not considered as isolated, but in conjunction with assumptions on the environment behaviour where it is executed. The system can be specified in: I) Process, accurately defined; II) Environment, more abstract and formalised by logical formulae. To specify properties of $\mu$KLAIM systems they use modal logic (MoMo) that allows describing interactions that the enclosing environments can have. Liu et al. [177] present AFChecker, a publicly available tool to improve user's fault detection and inspection experiences. It has three major components: 1) *Model checker* based on a technique for fault patterns and their automated identification [178]. Which derives a state transition model from a set of user-configured adaptation rules and verifies the model to detect five[4] common types of adaptation faults; 2) *Constraint inference engine*, that infers both deterministic and probabilistic constraints based on CHOCO[5] by analysing the propositional atoms in the user-configured adaptation rules; 3) *Fault Report Processor*, that processes the fault reports generated by its underlying model checker. The ranking of fault reports for user's inspection can be dynamic or static, depending on the interaction mode.

Park et al. [179] present CASS, a simulation tool for smart-homes that is able to generate virtual people in order to perceive its movements and actions through sensors. The tool is programmed in Java, and it allows modifying and deleting sensors/devices according to the developers preferences. After, it can perceive simulated movements of virtual people, generating proper values for each sensor type. They also describe the system architecture and hierarchical rule structure model for smart-homes. Wang et al. [180], provide an approach for automating the generation of tests for context-aware pervasive applications. They provide an integrated solution to identify when context changes may be relevant, and a control mechanism to guide the execution of tests into potentially interesting contextual scenarios as defined by a coverage criterion that is context-cognisant. Their solution can be used to enhance other test suites of context-aware applications.

Bertran et al. [25] introduced DiaSuite, a tool suite for the development of sense-compute-control applications. Within their suite of tools, they present DiaSim [181], a parametrised simulator to ease the acquisition, testing and interfacing of a variety of software and hardware components. The simulator is parametrised to a high-level description of the target environment, written in their own specification language (DiaSpec). This description is used to generate both a programming framework to develop the simulation logic and an emulation layer to execute applications. Furthermore, the simulation can be rendered, allowing visual monitoring and debugging of the system. Their tool can be found as an Eclipse[6] plugin. Yu et al. [182] apply a bi-graphical reaction system to model the environment that interacts with the middleware and domain

[4]Non-deterministic adaptations, dead rule predicates, dead states (meaning that no rules can be satisfied in these states), adaptation traces and unreachable states.

[5]Open source Java library. http://choco-solver.org/

[6] Eclipse is an integrated development environment (IDE) from the open source community of tools, projects and collaborative working groups Eclipse. https://eclipse.org/

services in the development of C-AS. To model the data entities in the environment, they extend the bi-graphical sorting predicate logic and build a meta-model. Then, they create a model of the middleware using an extended finite state machine. By synchronising the bi-graphical reaction system with the state machine, they can generate test cases to verify the interactions between the environment and the middleware. Finally, they show the reductions of the number of test cases by using a bi-graphical pattern-flow based testing on an airport example. Their tool is also in the form of an Eclipse 6 plugin.

Generally, authors recognise three main issues when simulating C-AS[25][182]: Modelling, source simulation and performance. First, it is difficult to determine what to model and in what granularity. Likewise, the model needs to be accurate enough to match such granularity. Second, some issues about the correctness of the stimulus producers may arise when either the logged data are replayed from actual sensors or a domain-specific modelling function is introduced. Emulated sensors must be programmed in such way, that for a given input, they produce the same output as its equivalent real sensor. Besides, merging the different intensities of simulated sensors requires domain-specific knowledge. Finally, physical spaces may involve lots of services, accurate simulation models and rich simulation logics which can be resource consuming.

**Analysis** As explained in Section 3.4.2.1, a specific tool has been chosen to be extended as part of the framework presented in this thesis. Further analysis has been done in the specific requirements related to those models representing the behaviour of reasoning rules specified in M, as shown in Table 3.3. The approach proposed in this thesis is that of presenting a proof-of-concept, based on the NuSMV [183] model-checker, but other discrete-time model checkers could be used, if found to be more efficient.

## 3.5 Conclusions

This chapter has presented an analysis of the literature review, which was focused on three main research themes. This exercise was used to identify a set of existing tools

| Aspect | Dimension | Suitable |
|---|---|---|
| **Model** | Linear | YES |
| | Branching | YES |
| | Discrete | YES |
| | Dense/Continuous | NO |
| | State based | YES |
| | Event based | YES |
| **State spaces representation** | Event based | YES |
| | Difference Bound Matrix | NO |
| | Binary Decision Diagram | YES |
| | Set of constraints | * |

Table 3.3: **A suitability analysis of the verification aspects for the M reasoning language.**

and methodologies which can be further combined and extended in order to create a context-aware systems engineering framework according to the objectives of this dissertation, introduced in Chapter 1. The remainder of the thesis presents the Context-Aware Systems Engineering Framework (C-ASEF), a framework to facilitate the development of context-aware systems. This framework is based on a set of existing methodologies and tools, which have been selected through an analysis of each of these research themes. The C-ASEF is divided into three main sub-frameworks, dedicated to different stages of the development process of a context-aware system. Figure 3.2 shows the relation between the tools selected during the analysis of this chapter for being adequate for the objectives of this thesis, and the resulting frameworks and sub-frameworks of this dissertation work.

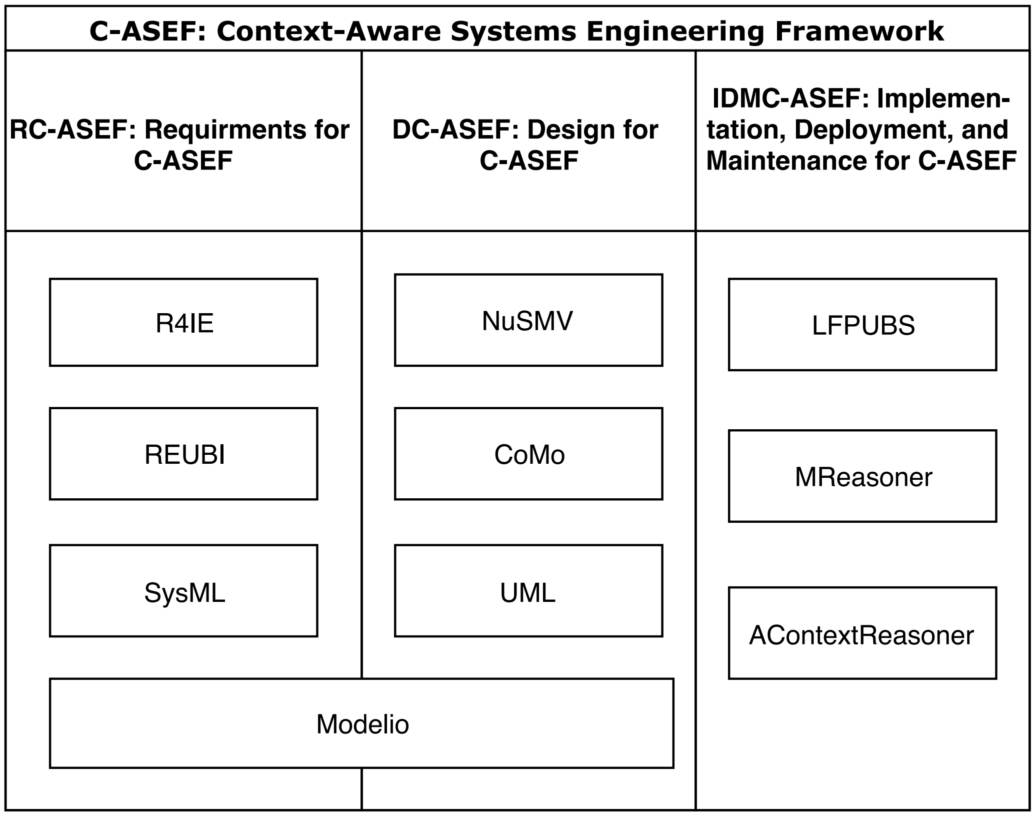

C-ASEF: Context-Aware Systems Engineering Framework
RC-ASEF: Requirments for C-ASEF
DC-ASEF: Design for C-ASEF
IDMC-ASEF: Implementation, Deployment, and Maintenance for C-ASEF
R4IE
REUBI
SysML
NuSMV
CoMo
UML
Modelio
LFPUBS
MReasoner
AContextReasoner

**Figure 3.2: Illustration of the selected methodologies and tools which will serve as a foundation for the C-ASEF.**

# Part II

# Requirements stage

Chapter 4

# RC-ASEF: Requirements for the context-aware systems engineering framework

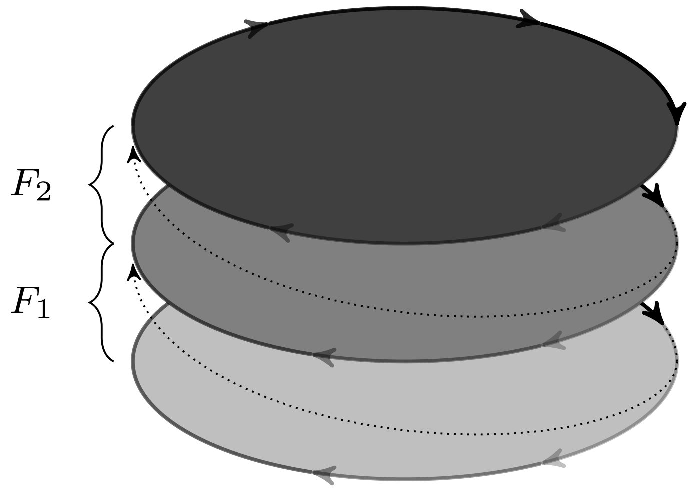
## 4.1 Introduction

Chapter 3, Section 3.2 analyses different methods and tools for requirements engineering in context-aware systems, where the methods presented in [109] [106] [108] are highlighted. The support provided for the requirements elicitation stage in the Context-Aware Systems Engineering Framework is divided into two main foci, as represented in Figure 4.1. During early stages of the requirements elicitation process, the methodology is focused on the generic or non-contextual aspects of the system ($F_1$), to then iteratively advance towards the requirements which are more related to the identification of situations, the way in which they are planned to be detected by the system and their associated context-aware features ($F_2$). This section presents the Requirements for Contex-Aware Systems Engineering Framework (RC-ASEF), a framework for requirements engineering which is specialised for the requirements elicitation of the non-contextual aspects of context-aware systems, corresponding to $F_1$, and which has been developed with reference to previous work [184] [109] [106] [108]. In particular, the framework is based on a collection of models, presented as a combination of dynamic and static diagrams which collectively define this new requirements eliciation framework, for which in addition, new, open-source tools have been developed. These constructs have been strategically chosen to be based on UML profiles, as this facilitates its use along with other existing standards such as UML [118], SysML [2], and U2TP [127], or other UML-based requirements profiles such as UML-AT [121]. A specialisation of the requirements elicitation methodology into the contextual aspects of context-aware system, corresponding to $F_2$, is introduced in Chapter 5. RC-ASEF has been created following the objectives explained in Section 1.4, and supports:

- An end-user, stakeholder centred vision, which guides the analysis of stakeholders towards the discovery of specific stakeholder profiles and their particular needs, preferences, and limitations. It also enables the representation of stakeholders and stakeholder profiles, as well as their interests, value requests and provision, source of power, support towards the project, influences, ethical aspects, and the ways in which values are exchanged between them. The methodology also supports the capture of these aspects into a *Stakeholder Diagram* model.

- Guidance for discovering objectives (goals and soft-goals), enabling their hierarchical decomposition into sub-objectives, obstacles and resource analysis, as well as their representation into models, using an *Objective Diagram*.
- The guidance for operationalising objectives into requirements and refining them into lower-level requirements. This allows not only their modelling, but also the modelling of development decision contributions towards objective satisfaction, as well as the cause for making the decision. It also enables the engineers to model the way in which requirements will be tested after their implementation. All this can be modelled in a *Requirements Diagram*.
- An evaluation procedure for determining which objective operationalisation decisions are most optimal towards the satisfaction of particular objectives, using a set of evaluation procedures based on existing rules and heuristics.
- A model-based representation of the relevant elements presented as part of a UML profile. The representation of requirements and related elements during the development process is supported by an open-source tool, which includes: SySML Diagram Representation, including traceability matrices, requirements tables and a link editor view, automatic generation of certain documentation artifacts, and compatibility with other open-source and licensed modelling tools, such as the UML testing profile, UML, SySML and other tools for completing the remaining development stages of this methodology.

Figure 4.2 presents the six main activities of a coherent methodology out of the most relevant approaches identified in the state-of-the-art review for the purpose of creating a framework for supporting the non-contextual aspects of the requirements elicitation, influenced by R4IE [108]. The main enhancement is that the identification of system performance qualities, used for gathering non-functional requirements, is now part of the objective establishment. A new activity group, corresponding to the evaluation of the objectives and requirements, which is partially based on the harmonisation



F1 = Generic aspects of the system (Figure 4.2); F2 = Situations of interest, the plan for making the system detect them, and their associated context-aware features (Figure 5.1).

**Figure 4.1: Different foci of the engineers as the methodology iterates.**

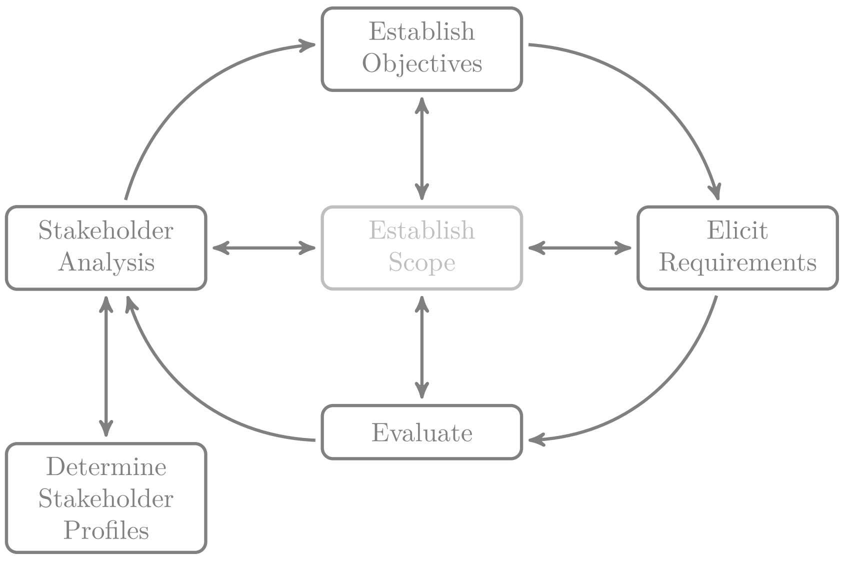


**Figure 4.2: Core activities in the early requirements elicitation stage, $F_1$.**

activity from R4IE, is introduced. The activities in Figure 4.2 are divided into different sub-activities, as shown in Figure 4.3, which are mainly influenced by the works presented in [184] [109] [108]. The method gives great importance to the exhaustive analysis on the stakeholders of the systems, as part of the identification of their needs

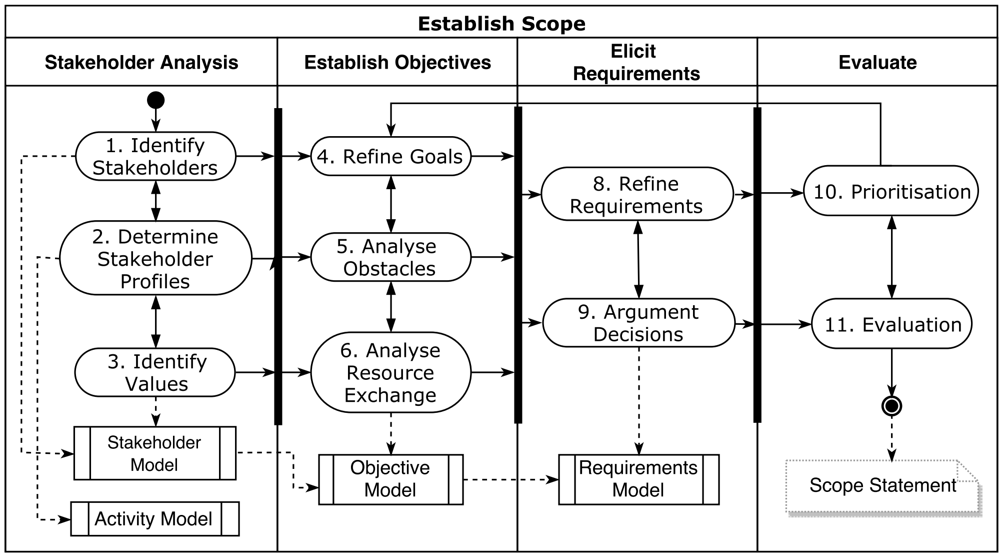


**Figure 4.3: Activity model representing the core sub-activities in the early requirements elicitation stage, corresponding to F1.**

and preferences in further stages. The remainder of the chapter is directly related to the framework shown in Figure 4.2. The sub-activities constitute an enhancement of the R4IE methodology, where the first sub-activity of the stakeholder analysis is inspired by [185] and [184], and the second sub-activity is impacted by the profiling of users [106] [108], the ethical analysis recommendation in [185], and the e-FRIEND ethical framework [186]. It is also influenced by the conceptualisation perspectives presented in Chapter 2, as it adopts a standpoint of understanding stakeholders and then to analyse their activities/behaviour in order to give developers a better understanding on the meaning behind their action, and help them to create services that are tailored to the needs and preferences of the stakeholders. The last activity in the stakeholder analysis, and the sub-activities related to the establishment of objectives, have been adopted from [109]. Finally, those sub-activities corresponding to the identification of functional requirements and the application of the evaluation procedure are inspired by those activities in [109], and influenced by the heuristics and rules from the NFR Framework [187] as well as the SysML [2] standard. The methodology introduced in this chapter reuses a particular vocabulary and definitions, which are adopted from that used in previous work [109][64] [188] [106] [185] [184] [189] [2] [187] [1]. The remainder of the chapter is as follows. Section 4.2 is related to the establishment of a project scope. Section 4.3 corresponds to the stakeholder analysis of the methodology. Section 4.4 is related to the objective establishment activity. Section 4.5 corresponds to the identification of functional requirements. Section 4.6 corresponds to the evaluation activities of the methodology. Finally, Section 4.7 summarises the chapter.

## 4.2 Establish scope

The central activity of the methodology during $F_1$ is to establish the scope of the system in terms of the system boundaries (*i.e.*, what is inside the system and what is immediately external to it) . As it can be observed, this activity is influenced by the remaining core activities in $F_1$, which help to determine the objectives, resources, budget and schedule to be included within the scope statement.

## 4.3 Stakeholder analysis

The initial step consists of a stakeholder analysis, which allows documenting and modelling the outcome from the array of techniques proposed in [185], using a UML profile for the creation of Stakeholder Diagrams. The stakeholders are identified, and their different relevant relationships to the project are analysed. The outcome of this activity is used as part of the scope statement and part of the models. Finally, the aim is to identify different user profiles, in order to pave the way for discovering useful Situations of Interest in $F_2$. Using the information gathered during the stakeholder analysis, it focuses on the identification of activities.

### 4.3.1 Identify stakeholders

The first step consists of identifying the different stakeholders who are interested in the system under development. This activity is initiated by a small group, and later reviewed with a larger group of stakeholders. After the review with a larger group of stakeholders, the participants should think about those stakeholders who are still not included. If there are more interested parties, a bigger group should be assembled to review the stakeholders [185]. This process iterates until a consensus has been arrived at such that it is considered that all relevant stakeholders have been accounted for. A set of techniques are recommended to guide this process, which have been adopted from [185] [184]. Each of which can build on the previous technique, and some of which are explained below in this list:

- Listing Stakeholders: Consists of brainstorming a list of potential stakeholder groups or individuals. This is the initial step on which subsequent stages build.
- Basic Stakeholder Analysis: This technique can be used to identify stakeholders and their interest, as well as the stakeholders' view of a focal organisation, facilitating the later identification of coalitions of support and opposition. It consists of, for each stakeholder identified in the brainstorming session, creating a list with the criteria that this stakeholder would use to judge the organisation's performance or the expectations that this stakeholder could have about the organisation. Finally, quick actions which can be taken to satisfy the stakeholder are identified and recorded, as well as long term issues. Other additional steps might also include the specification of how each stakeholder influences the organisation, what the organisation needs from the stakeholder, and a ranking of stakeholders according to their importance to the organisation.
- Power Versus Interest Grids: Stakeholders are arranged on a two-by-two matrix in which one dimension reflects the stakeholder's interest in the project development and the other dimension maps to the stakeholder's power to affect the project development. This classifies the stakeholders into four main categories (*i.e.,* Subjects, Players, Crowd and Context Setters). This technique helps in determining which stakeholder's interests must be taken into account in order to address a particular problem. Also, the technique provides information on how to convince stakeholders to change their views.
- Stakeholder Influence Diagrams: Indicate how stakeholders on a power versus interest grid influence one another. For this, lines with arrows are drawn into the Power Versus Interest Grid, in order to indicate the direction of the influence. Two-way influences are possible, but the primary direction in which the influence flows between stakeholders should be clear.

The stakeholder identification can also be complemented with a stakeholder analysis, as further explained in [185] [184].

### 4.3.2 Determine stakeholder profiles

The aim of this activity is to identify stakeholder profiles, by establishing personal goals and setting different levels of achievement. The user profiling is attained by setting certain achievement levels and monitoring progress towards those personal goals [106]. In order to set the achievement levels, three main dimensions are analysed during $F_1$, which include the cultural aspects of the stakeholders, their quotidian activity, and their relevant ethical aspects. Finally, the information obtained from this analysis is used to customise the requirements, as well as the system set-up and training. In activities related to stakeholder profiling corresponding to $F_2$, other dimensions are analysed, namely, the interaction modalities, and the mechanisms for monitoring the achievement of personal user goals. The user profiling activity is mainly based on the activity with the same name in R4IE [108], but it also includes the cultural analysis and profiling guidelines from PC-RE [106] and the ethical analysis mechanisms from [185] and [186]. The main enhancement is that the *task subset* and *context-interaction requirements* sub-activities of R4IE [108], and the *monitoring mechanism specification* related activities of PC-RE [106] have been moved to the context-aware specialised stage, $F_2$. Also, a new sub-activity has been proposed, to analyse the activity of stakeholders in order to prepare the situation of interest identification in $F_2$.

#### 4.3.2.1 Cultural analysis

The first sub-activity of stakeholder profiling deals with the system from an international point of view, where the different effects of culture are analysed in order to influence the definition of requirements for localising systems and specifying how it will be tailored for its different cultural profiles. During this activity, scenarios are sourced from users who belong to the cultures, nationalities and linguistic groups inside the intended market. The four[1] main steps of the guide proposed in [106] can be applied for this purpose:

[1]Note that the fifth step has been moved to the user profiling activity in $F_2$.

- Consider the cultural impact on the social context of the user: This includes an analysis of five main aspects: cultural *uncertainty avoidance*, which can cause the users to prefer more precise instructions and fewer options; *power-distance* which is related to the way in which users react to authority, initiative and responsibility; *Individualism or collectivism* in societies, which can result in users having different attitudes towards personal or collective goals; *Context* representations in a more visual or symbolic way, against representations in hard facts, detail, and statistical evidence; and *Time*, which is related to the preference of doing one task at a time against multi-tasking.
- Beware of the impact of authority relationships on user goals: Take into account the effect of different levels of authority and responsibility as obstacles for the acceptance of goals on users, where goals are owned by stakeholders that are managers.
- Assess the impact of culture on the users' work patterns: Consider the effect of culture on how work is organised by the system.
- Assess the literacy of the local user population: Consider the possibility of users not having the knowledge or literacy skills to operate complicated functions.

#### 4.3.2.2 Ethical analysis

An ethical analysis can contribute to ensure the ethical appropriateness of actions are ultimately taken in a project. For this purpose the use of Ethical Analysis Grids is recommended [185]. This grid can aid the satisfaction of both deontological (duty-based) and teleological (results-oriented) obligations. It consists of classifying some characteristics of each stakeholder into: High, Medium, Low and None. The characteristics are the vulnerability and gravity of the stakeholder, her/his dependency on the government, likelihood remedy, risk to fundamental value and policy impact. Although the ethical analysis proposed in [185] is useful for general purpose systems, it is not focused on context-aware systems. Context-awareness is the essence of different areas[2] that typically raise some ethical concerns which are different to those of traditional systems. For this reason, this sub-activity also adopts the eFRIEND ethical

[2]Particularly referring, in this case, to Ubiquitous & Pervasive Computing, Intelligent Environments, Ambient Intelligence and Ambient Assisted Living.

framework [186]. In order to apply it, it is recommended to carefully evaluate and discuss with the end-user stakeholders the different ethical concerns that might arise, until there is an agreement between all parties (*e.g.*, increasing user safety at the expense of giving up some privacy). The discussions can be complemented by questionnaires or interviews. The outcome of those discussions at a conceptual level can be used to modify or create different objectives and requirements.

#### 4.3.2.3 Activity analysis

This stage consists of analysing the activity of end-user stakeholders, and is especially focused on that activity of end-user stakeholders. The purpose is to facilitate (for the benefit of developers) the identification of the meaning behind the behaviour of the end-user stakeholders. Particularly, by analysing how they usually behave in their quotidian tasks, and by thinking about how the stakeholders could use the proposed system to improve the way in which they achieve these tasks. This gives more opportunities to identify services that can be provided to them according to their particular needs, preferences, and limitations. Techniques such as observation, prototyping, scenarios or *wizard of oz* [190] can be used. Other approaches such as ethnomethodology can be adopted to understand the meaning of the actions of the end-user stakeholders. On the other hand, data analysis techniques such as classification or pattern-recognition could also help in revealing unexpected relations in the behaviour of the stakeholders. Other workflow techniques such as UML Activity Diagrams can also facilitate the capture of the relevant end-user stakeholder activities into workflows that can be later on associated to situations of interest.

#### 4.3.2.4 Determine customisation, set-up, and training

The method proposed in 4.2 is iterative. Once developers have defined some requirements, it is time to use the information gathered during this activity to customise existing requirements. The main dilemma is to specify context-aware systems that suit the requirements of individual users, while delivering a general system that can be used by many (individually different) users [106]. Not only this, but requirements can also evolve for the same user. For instance, as users become more experienced using the system, they require less help and supportive dialogues, and can access more sophisticated features. Also, the requirements engineering process should take into account aspects of maintenance and bespoke tailoring (to different stakeholders) after the system is deployed [108]. In order to help the identification of different stakeholder profiles, there is a need to think about how the system will be set-up by/for the different stakeholders, trying to distinguish the different common needs of stakeholders that can be classified into profiles. As well as how the different stakeholders will want to customise the system, what type of training will they receive, and how will they receive it. In order to enable the customisation of the system, an individual user profile is defined first. The requirements are frequently set by another expert stakeholder (*e.g.,* a teacher sets certain requirements for a student's learning abilities). Individuals directly elicit (and own) personal goals. For both personal goals and user profiles, attainment targets can be set which become benchmarks for monitoring processes. A trade-off analysis might help to identify any conflicting user profile goals set by the expert stakeholders with the personal ambitions of the end-user stakeholder.

### 4.3.3 Identify values

During this sub-activity the aim is to identify the different values to be produced by the system and consumed by the stakeholders. This model has been adopted from the value model introduced in REUBI [109]. It takes existing stakeholders, humans or agents, and identifies the different values that are expected to be exchanged. Particularly, stakeholders which produce, consume a value, or are interested in a value or in its acquisition quality. Also, developers analyse what values are interchanged by the system and other stakeholders. Then developers reflect on how these values can be enhanced. Aspects such as how to improve the value, what is the expected quality of the value, what time restrictions exist in the provision of the value are taken into account. Other enhancement aspects apply, such as the flexibility in the value acquisition, precision or reliability restrictions, as well as cost or security restrictions applicable to the value.

## 4.3.4 Stakeholder diagram

Inspired by the techniques of the sub-activities explained in this section, the Stakeholder Diagram is introduced, which can model relevant stakeholder related information, as it is shown in the parts I and II of the meta-model of the Stakeholder Diagram (Figures 4.4 and 4.5). Note that the Stakeholder Diagram also supports the modelling of the different grids and elements introduced in Section 4.3.

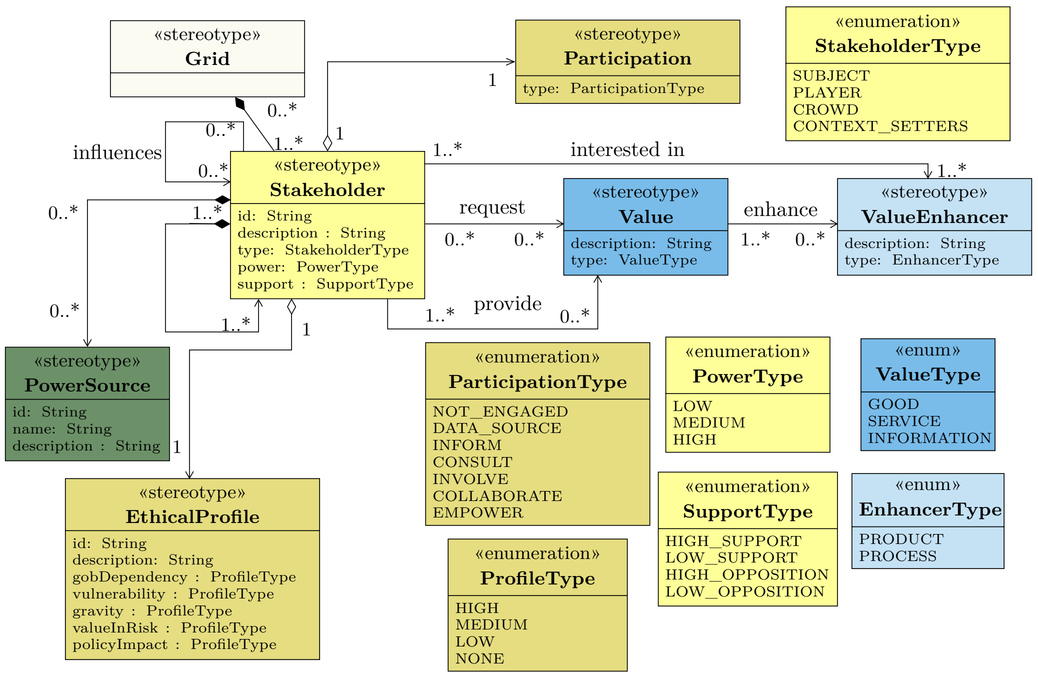


**Figure 4.4: Part I, meta-model for the Stakeholder Diagram.**

## 4.3.5 Example

The stakeholder identification activity presents a set of techniques that build on the previous activity. Following the case-study introduced in Section 1.5.2, the following list of stakeholders are identified: 1) Primary Users (PU), people with Down's Syndrome; 2) Secondary Users (SU), parents or carers of people with Down's Syndrome; 3) POSEIDON Managers, the management team of the POSEIDON project; 4) POSEIDON Development Partners, POSEIDON project partners which work in creating code or libraries that are to be reused by this application. 5) Developers, the developers

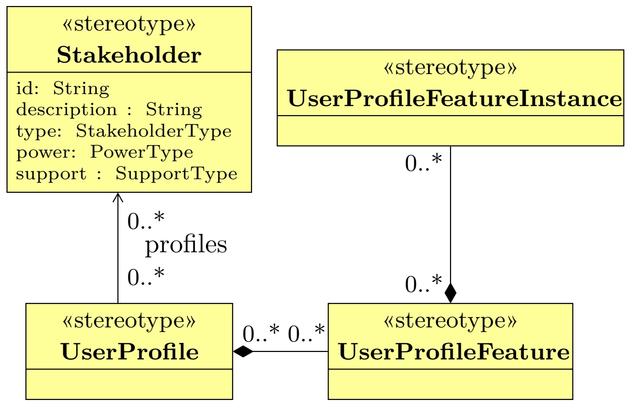


**Figure 4.5: Part II, meta-model for the Stakeholder Diagram.**

of the navigational system; 6) Bus driver, the person(s) that drive(s) the bus in which the PU will get on; 7) Bus company, the company in charge of the bus line; 8) Calls and Internet provider, referring to the company that provides phone calls, SMS and internet to the mobile device; 9) Device Manufacturer, company that manufactures the device; 10) Operating System Developers, group involved in the development of the operating system of the device; 11) Maps Library Developers, group involved in the development of the maps libraries. The list is further refined into the power versus interest grid, which evolves through iterations into the stakeholder influence grid, as it is shown in Figure 4.6. The stakeholder diagrams introduced give better insights about who are the stakeholders of the system and their relevant aspects to the project. The analysis of the stakeholders and their profiles can provide relevant information about the stakeholders which can be later reused for identifying their needs and preferences in the context related requirements. The stakeholder profiling activity follows. The POSEIDON project involved a total of three different countries, namely, United Kingdom, Germany and Norway. These three cultures are similar in the sense of avoiding uncertainty, having similar work patterns, and responding similarly to authority, initiative and responsibility. Additionally, the United Kingdom has a different currency, representation of metrics, and driving direction than Germany and Norway. This might affect the payments of users for public transport, the location of bus stops,

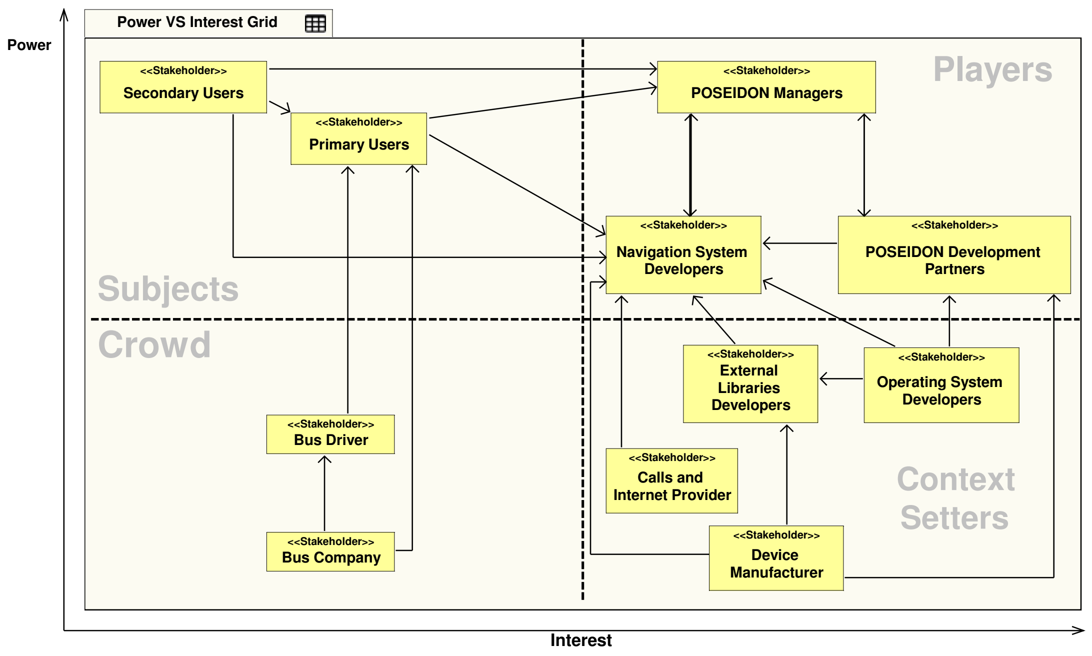

Power VS Interest Grid
Power
Interest
Players
Subjects
Crowd
Context Setters
<<Stakeholder>> Secondary Users
<<Stakeholder>> Primary Users
<<Stakeholder>> POSEIDON Managers
<<Stakeholder>> Navigation System Developers
<<Stakeholder>> POSEIDON Development Partners
<<Stakeholder>> External Libraries Developers
<<Stakeholder>> Operating System Developers
<<Stakeholder>> Bus Driver
<<Stakeholder>> Calls and Internet Provider
<<Stakeholder>> Bus Company
<<Stakeholder>> Device Manufacturer

**Figure 4.6: Power vs Interest Grid representation, created with the Stakeholder Diagram from the RC-ASE Tool.**

as well as the distance representation in the maps. The discovery of personas[3] revealed that some particular users have visual or auditive impairments, and the questionnaires revealed that different skill levels using information technologies [39]. The different profiles identified can be observed in Figure 4.8. There are five different user profile features for the primary user stakeholder: Culture, visual impairment, skills with technology, independence degree, and auditive impairment. Each of the profile features is divided into its corresponding user profile feature instances. Following, the activity of the end-user stakeholders when displacing is analysed, as it is represented in Figure 4.7. This information will be used to identify situational interests in $F_2$. The ethical analysis, customisation, set-up and training example sub-activities examples from the user profiling activity can be found in Section 4.5.4.

The last activity in the stakeholder analysis consists of creating a value model, as

[3]The persona is a virtual character that substitutes for the human users [93]. These hypothetical characters distil the characteristics of the majority of the system users. The usage of personas is useful when real users are not available or are too numerous to interview all of them. Personas can be built by surveying the user community and deriving an archetypical character to represent that community.

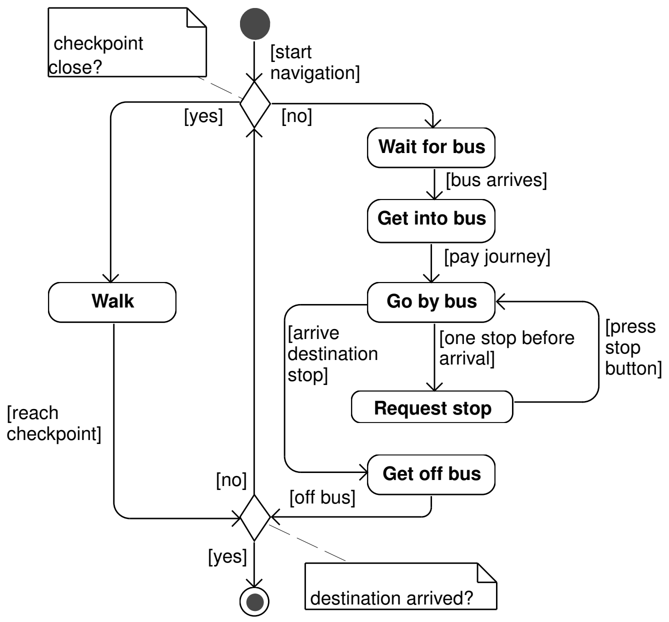


Figure 4.7: Activity diagram of primary user displacements, UML Activity Diagram, Modelio Tool.

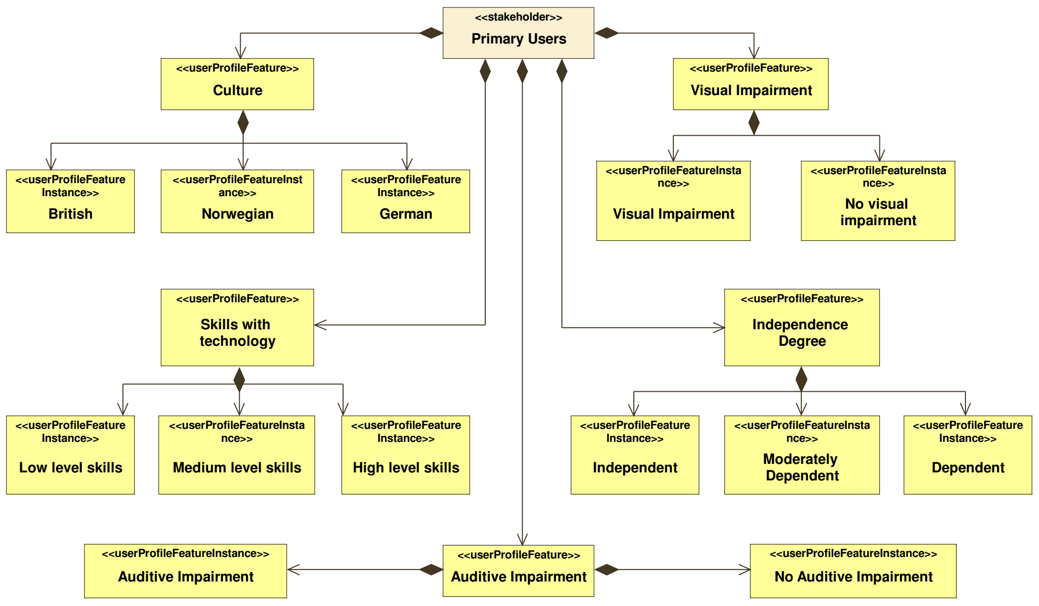

<<stakeholder>> Primary Users
<<userProfileFeature>> Culture
<<userProfileFeature>> Visual Impairment
<<userProfileFeatureInstance>> British
<<userProfileFeatureInstance>> Norwegian
<<userProfileFeatureInstance>> German
<<userProfileFeatureInstance>> Visual Impairment
<<userProfileFeatureInstance>> No visual impairment
<<userProfileFeature>> Skills with technology
<<userProfileFeature>> Independence Degree
<<userProfileFeatureInstance>> Low level skills
<<userProfileFeatureInstance>> Medium level skills
<<userProfileFeatureInstance>> High level skills
<<userProfileFeatureInstance>> Independent
<<userProfileFeatureInstance>> Moderately Dependent
<<userProfileFeatureInstance>> Dependent
<<userProfileFeatureInstance>> Auditive Impairment
<<userProfileFeature>> Auditive Impairment
<<userProfileFeatureInstance>> No Auditive Impairment

Figure 4.8: Different profile features and profile feature instances of primary users.

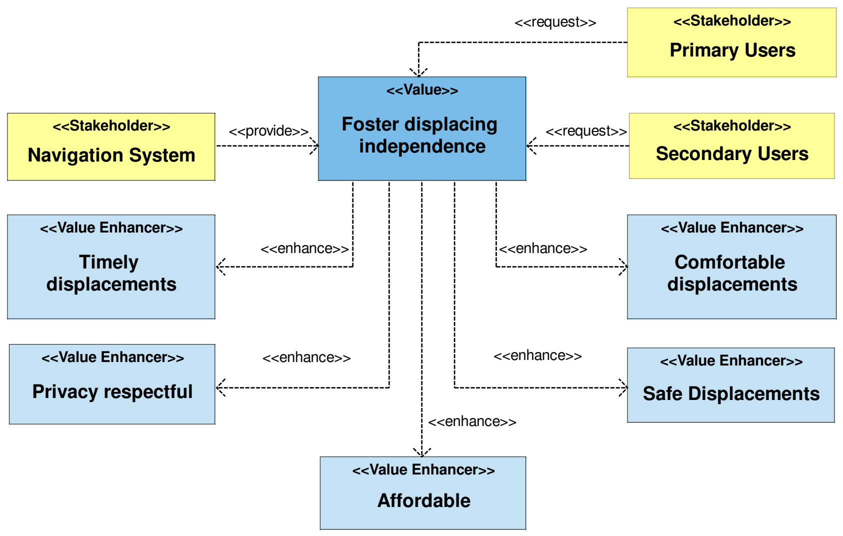


**Figure 4.9: Value model representation, Stakeholder Diagram, RC-ASE Tool.**

shown in Figure 4.9. The first actor to consider is the navigational system itself, which offers the value of fostering the independence of both primary and secondary users. That value is a service, which is requested by the primary and secondary users. Five main aspects enhance the value provided by the navigation system. These are: that primary and secondary users can afford the system, that the system can preserve the privacy of the users, that the primary users can displace safely when using the system, that the primary users can reach their destination on time, and that the instructions given by the navigation system are understandable by primary users.

## 4.4 Establish objectives

Taking into account the value analysis, the objectives of the system are declared. Then, from the higher order objectives, a refinement process is applied in order to obtain and decompose them into sub-objectives. This step is followed by an analysis of the adverse conditions that may impede the satisfaction of a goal. Following this, the analysis focuses on the resources required for the satisfaction of goals.

### 4.4.1 Refine goals

Once the system boundaries and the higher-order objectives are identified, in the form of *values* and *value enhancers*, it is necessary to derive more specific objectives, and progressively refine them in order to obtain more knowledge about the system under development. Objectives act as a bridge between the system values and the final requirements of the system, providing specific guidance during the requirements elicitation process. Objectives are divided into *goals* and *soft-goals*, according to the identified *values* and *value enhancers*. Goals have a clear criteria of satisfaction. Soft-goals do not have a clear criteria of satisfaction, which means that they can be used for identifying and modelling non-functional requirements. The objectives can also be progressively decomposed using *inclusive* (AND), *alternative* (OR), or *exclusive* (XOR) relationships between them. Such relationships can help in determining if their corresponding parent objectives are satisfied or not, as further explained in the definitions for the *refine* relationship introduced in [64]. The introduction of objectives not only guides the process of requirements elicitation, but also enables an early evaluation of the requirements satisfying the goals, as further explained in Section 4.6.2.

### 4.4.2 Analyse obstacles

The dynamic nature of context-aware systems is closely related to the existence of multiple adverse conditions which can make it difficult for system objectives to be met. This sub-activity consists of identifying obstacles which may affect meeting a specific goal, in the same way as described in [109]. The main objective is to determine those situations which are likely to be inconvenient for meeting the objectives, even if obtaining a complete set of adverse conditions can be a difficult achievement.

### 4.4.3 Analyse resource exchange

Sometimes, there exist restrictions on the way in which sub-objectives need to be satisfied in order to satisfy the parent objective; such as not satisfying an objective until other objectives are satisfied, mainly because these require access to certain resources which are generated as a result of satisfying other objectives. The objectives relate to the resources through two different relationships: *provision* and *demand*, as further explained in [109].

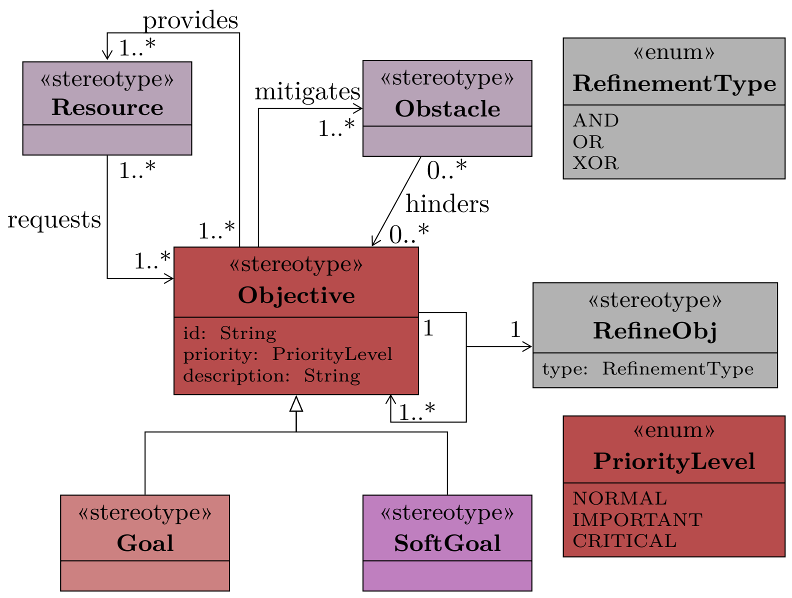


**Figure 4.10: Objective Diagram meta-model.**

### 4.4.4 Objective diagram

With the purpose of facilitating the objective, obstacle and resource exchange analysis is explained in this section, the *Objective Diagram* is introduced, which has been adopted from the Interdependency Graph in [109]. The meta-model of this diagram can be observed in Figure 4.10. More information about the meaning of each stereotype, relationship and enumeration can be found in [64].

### 4.4.5 Example

The main goals and soft-goals of the system are derived from the value model shown in Figure 4.9. In this way, the goal *Guide displacements*, is related to the *Foster displacing independence* value, as shown in Figure 4.11. Since this value is still too generic, it needs to be refined. The goal can be decomposed into two sub-goals: *Walking displacement guidance* and *Bus displacement guidance*. Note that for satisfying the high level objective, both lower level objectives must be satisfied. It equally happens with the value enhancers. *Walking displacement guidance* is refined into the objective "*Time-based guidance*", which proposes that the guidance received by the stakeholders will take into consideration time constraints, as shown in Figure 4.12. This goal is divided into another two lower level goals, which are to provide guidance about when to start the displacement, and to provide guidance according to the walking speed. The value enhancer *Affordable*, is distilled into the *Low-cost* soft-goal, which at the same time is divided into *Low-cost hardware* and *Low-cost software* soft-goals, as it can be observed in Figure 4.13. The value enhancer *Privacy respectful* is also refined into the soft-goal *User privacy*, that is divided into the two soft-goals *Anonymity/pseudonimity* and *Confidentiality*, as shown in Figure 4.14. The value enhancer *Safe displacement* is refined into the soft-goals *Displace through safe environments*, and *Support lost users*, as it appears in Figure 4.15. Finally, the value enhancer *Comfortable displacement*, is distilled into the goal *Guide on required objects*, that supports the user with a list of objects that can make more comfortable the displacement or the activity to do where the user is displacing. Also, this value enhancer is refined into the soft-goal *Provide understandable guidance*.

Following, an obstacle analysis over the objectives proceeds, as shown in Figure 4.17. The main obstacle found is due to the interruption of the service, caused by a lack of power. The battery may run off, and the user is left without instructions to follow. In order to mitigate, the soft-goal *Availability* is added. Next, is the Resource analysis, shown in Figures 4.18 and 4.19. The first of these two figures shows how the goal of guiding displacements refines from the *Start instructions* resource, generated from the time-based guidance goal. The second of these two figures indicates how the goal for guiding displacements also requires from the *Personal object list* resource, generated from the*Guidance on object list* goal.

## 4.5 Elicit requirements

Once the objectives of the system are defined, they need to be operationalised into requirements. Then, an analysis of the contribution that the requirements have to objectives should be performed. This stage is inspired by the *task/function* and *sys-*

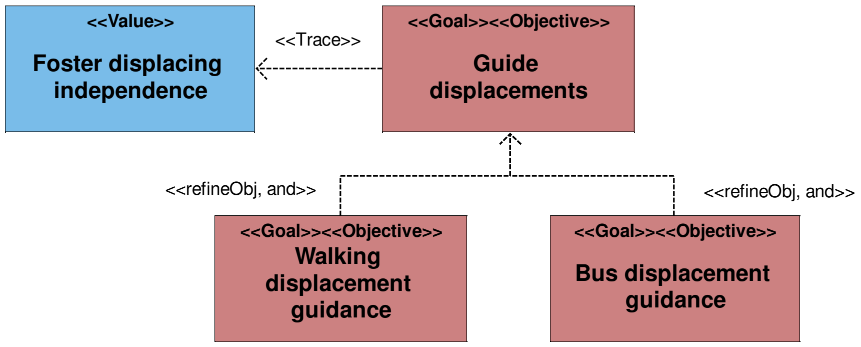


Figure 4.11: Goal Decomposition I, Objective Diagram, RC-ASE Tool.

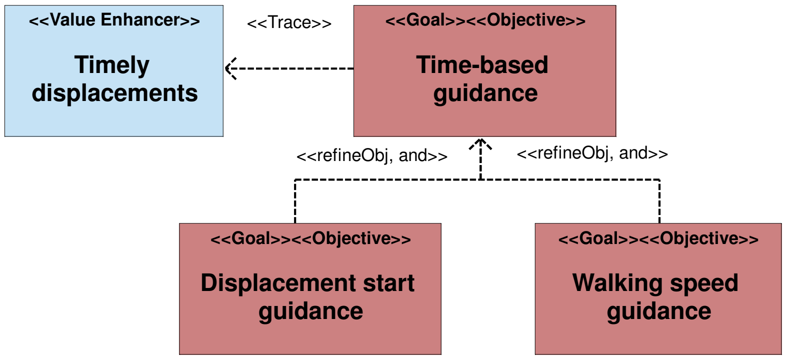


Figure 4.12: Goal Decomposition II, Objective Diagram, RC-ASE Tool.

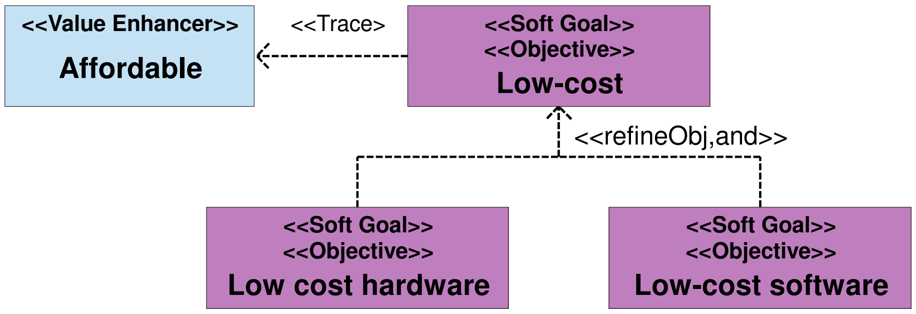


Figure 4.13: Goal Decomposition III, Objective Diagram, RC-ASE Tool.

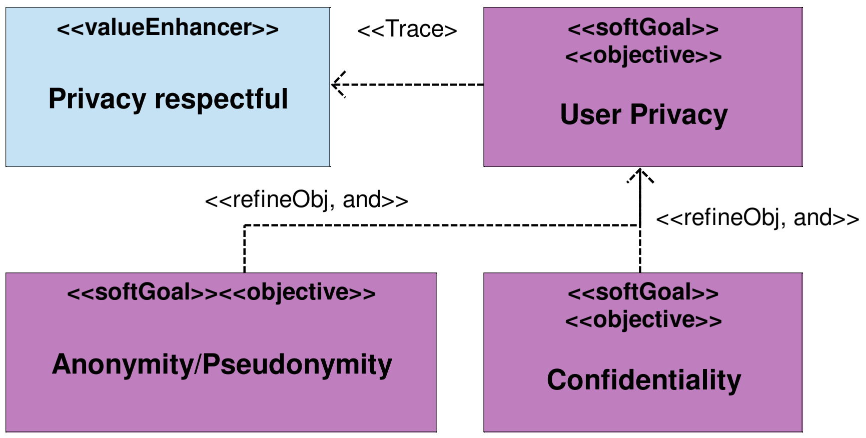


Figure 4.14: Goal Decomposition IV, Objective Diagram, RC-ASE Tool.

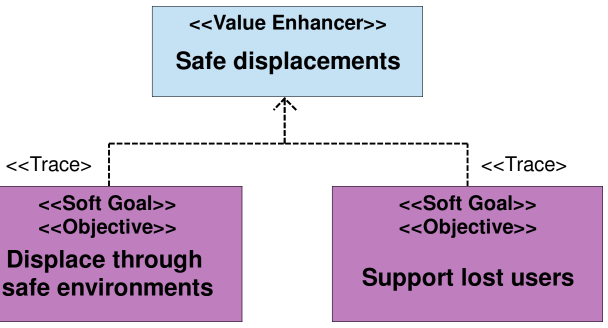


Figure 4.15: Goal Decomposition V, Objective Diagram, RC-ASE Tool.

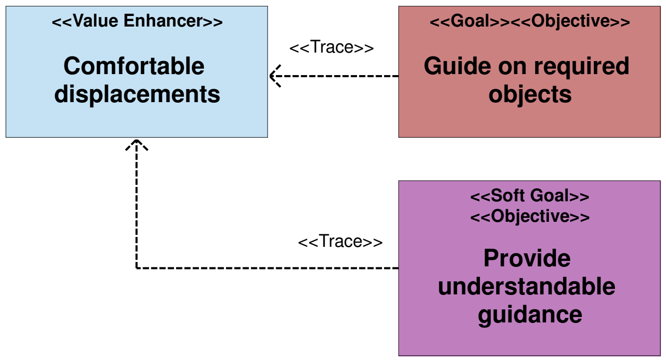


Figure 4.16: Goal Decomposition VI, Objective Diagram, RC-ASE Tool.

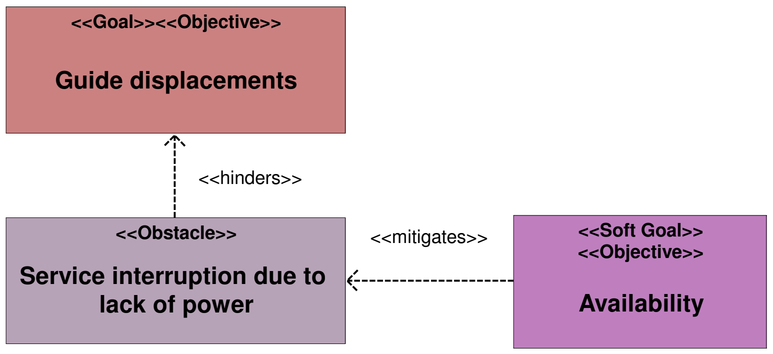


Figure 4.17: Obstacle analysis, Objective Diagram, RC-ASE Tool.

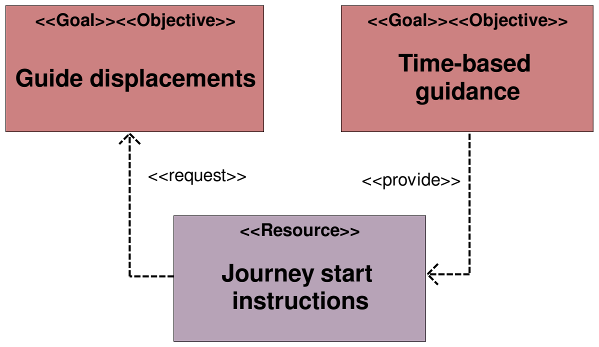


Figure 4.18: Resource analysis I, Objective Diagram, RC-ASE Tool.

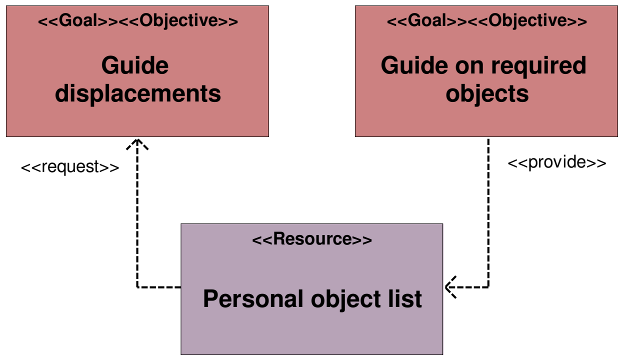


Figure 4.19: Resource analysis II, Objective Diagram, RC-ASE Tool.

*tem performance qualities* identification activities of R4IE [108]. Following this, requirements are refined, decomposed into sub-objectives, and related to other model elements. All the decisions taken need to be documented as rationales, in order to facilitate requirements tracing, by modelling the reasons which developers are following to make decisions.

### 4.5.1 Refine requirements

Once the engineers agree upon the representation of values, objectives, their decomposition, obstacles and resources; the next step is to discover alternatives which can satisfy the objectives, finding their possible operationalisations, in the form of requirements. In the previous sub-activity, higher-order requirements are identified, as well as their contribution to the objectives. In this sub-activity, those requirements are refined into more precise requirements, and are related to other elements of the system. In addition to those relationships (RefineObj and Contribute) introduced by the Interdependency Graph in REUBI [109], the Requirements Diagram inherits five different types of relationships from SysML, as can be observed in Figure 4.20. This sub-activity mainly consists of applying the Refine, DeriveRqt, and Copy relationships. More information about the elements in the diagram can be found in [64].

### 4.5.2 Argument decisions

Once the requirements are operationalised and refined, the aim is to model the decisions taken during the previous activities. The Requirements Diagram enables this through the use of the Argumentation stereotype, which is related to other elements in the Requirements Diagram as illustrated in Figure 4.22. SysML already provides a means to argument relationships via the Rationale stereotype. Nevertheless, the Argumentation stereotype provided in the Interdependency Diagram of REUBI [109], facilitates the specialisation of the rationale into support and rejection arguments. More information about the elements in the diagram can be found in [64].

### 4.5.3 Requirements diagram

For the purpose explained in this section, the Requirements Diagram is introduced, which inherits the stereotypes of the OMG SysML Requirements Diagram [2] shown

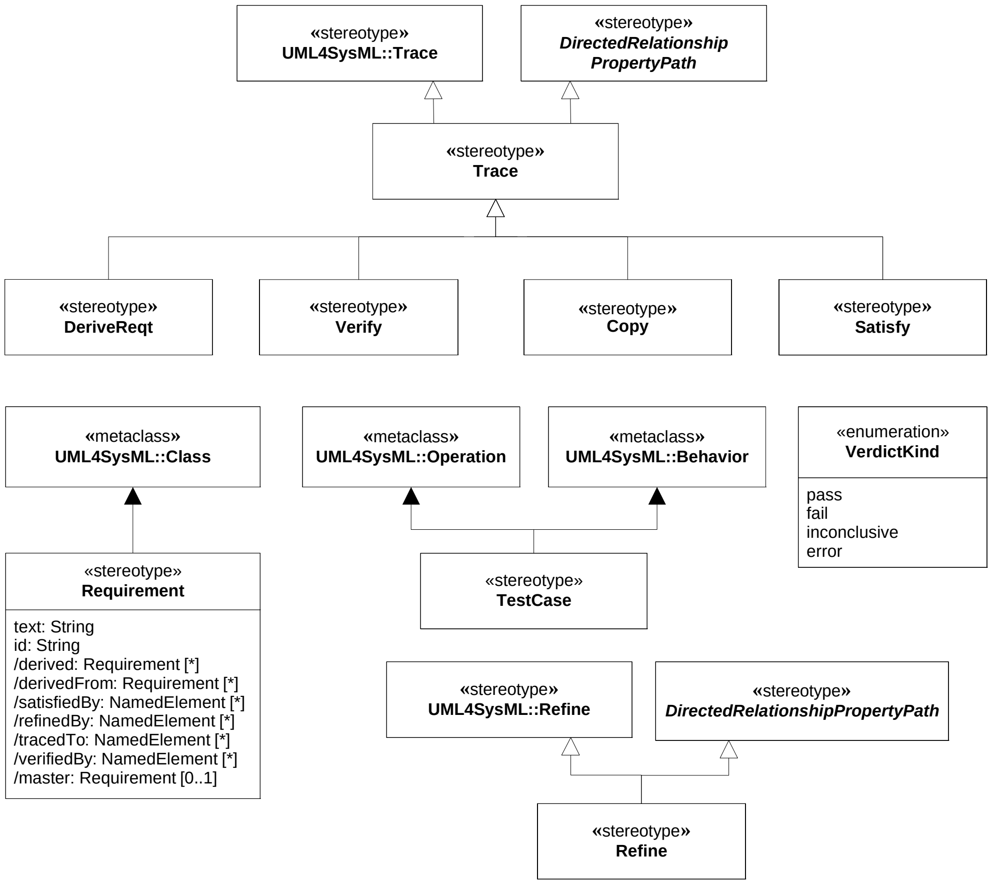


**Figure 4.20: Stereotypes for the OMG SysML Requirements Diagram [2].**

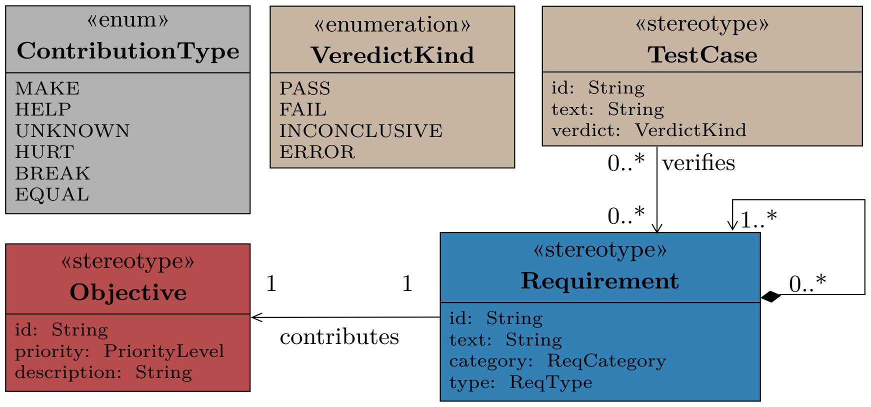


**Figure 4.21: Requirements Diagram meta-model, part I.**

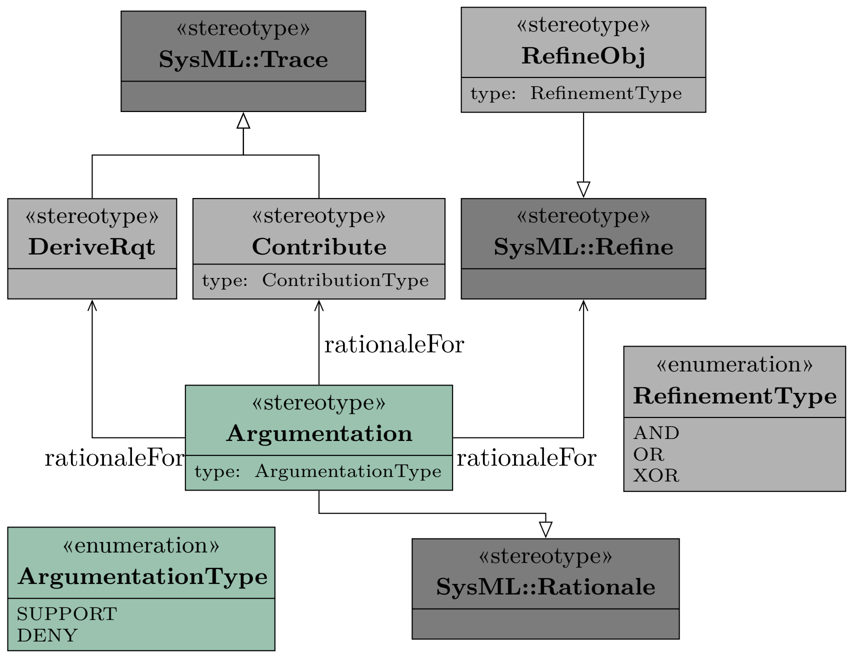


**Figure 4.22: Requirements Diagram meta-model, part II.**

in Figure 4.20, and the object and justification meta-models from the REUBI Interdependency Graph [109]. Also, the requirement categorisation has been adopted from the ISO 25010 standard[4] [189] [191]. The Operationalisation stereotype from REUBI, has been substituted by the SysML Requirement, which gives the following advantages:

- Requirements Traceability [192]: Keeping track of what happens to a Requirement during system modelling and specification by identifying the sources, destinations and links between requirements and models created during system development. This can give engineers the possibility of ensuring that all requirements are fulfilled by the system and sub-system components.
- Requirements Evaluation Constructs: Specifically, SysML provides a way of documenting how the Requirements will be tested through the stereotype of TestCase. Such modelling constructs can also be used with other UML-based

[4]Although the introduced Requirements Diagram only supports the requirement categorisations of the ISO 25010 standard, those requirements categorisations in FURPS+ or the ISO/IEC 9126 could also be used for the classification of requirements in this methodology [191].

standards such as the UML 2.0 Testing Profile [127], to facilitate the design and automation of test runs [193].

- Requirements Visualisation Approaches [192]: Requirements Tables provide means to identify, prioritise and improve requirements traceability by directly showing a table with the id, name of the requirement, and its description. Requirements Traceability Matrices capture all the proposed requirements and their corresponding traceability in a single table.
- Use-Case Diagram Compatibility: Although the exclusive use of Use-case diagrams might be limited for the requirements engineering process, the use of SysML requirements to complement them represents an advantage and improves standardisation [192]. Use-case Diagrams can be used to describe the early system requirements, facilitating the comprehension of the system and its features by non-technical stakeholders, as they can present different scenarios which can be detailed through informal descriptions. SysML complements the use of use-case diagrams by enabling its traceability[5] through the model.
- Objectives are used as a bridge between the stakeholder analysis and the requirements. They act as an intermediate step to guide the discovery of requirements. Specifically, the objectives from REUBI have been created to facilitate the discovery of non-functional requirements.
- The operationalisation of objectives can be evaluated using the evaluation procedures from REUBI, as further explained in [64].

Figure 4.21 shows part of the meta-model for the Requirements Diagram. More information about the elements in the diagram, including the different requirements types (*ReqType*) and categories (*ReqCategory*), can be found in [64].

### 4.5.4 Example

At this stage of the method, the different goals of the system are refined into requirements, that represent a condition or capability that the system needs, and which contribute to the satisfaction of objectives. These design decisions, as well as the positive or negative contributions of the decisions are studied. For this, the lowest-level goals

[5]Specifically, with the *refine* relationship

are considered (*i.e.,* those goals which do not have any sub-goal). Note that in order to facilitate the readability of the diagrams, the argumentation is omitted from then, and it is explained in the text. Also, note that the requirements are specified with short names, but that Modelio enables a longer description of the requirement. This longer description of each requirement can be found in the main example of [194]. In the previously introduced goal models, there are 5 low-level goals, which are used to define the functional requirements of the system, and 7 low-level soft-goals, which are used to define the non-functional requirements. For simplicity, the Requirements Diagrams of this example have been divided into three parts: requirements related to navigation, as shown in Figure 4.23; requirements related to reminders of the system, as shown in Figure 4.24; and non-functional requirements, shown in Figure 4.25.

Navigational requirements are based on a main requirement, *Navigation Map*, that specifies that the user will be able to observe a map that represents the real-world surroundings. Note that this requirement is a positive contribution towards two low-level goals: *Walking displacement guide* and *Bus displacement guide*. However, since

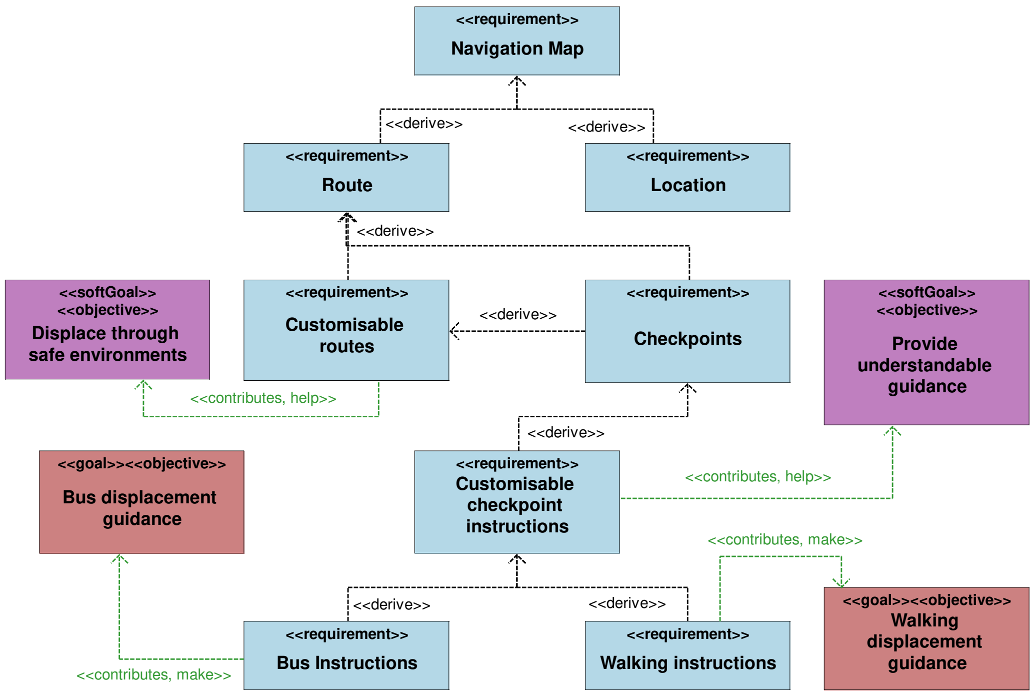

<<requirement>> Navigation Map
<<derive>>
<<derive>>
<<requirement>> Route
<<requirement>> Location
<<derive>>
<<softGoal>> <<objective>> Displace through safe environments
<<requirement>> Customisable routes
<<derive>>
<<requirement>> Checkpoints
<<softGoal>> <<objective>> Provide understandable guidance
<<contributes, help>>
<<derive>>
<<goal>><<objective>> Bus displacement guidance
<<requirement>> Customisable checkpoint instructions
<<contributes, help>>
<<contributes, make>>
<<derive>>
<<derive>>
<<goal>><<objective>> Walking displacement guidance
<<requirement>> Bus Instructions
<<requirement>> Walking instructions
<<contributes, make>>

**Figure 4.23: Requirements model I, Requirements Diagram, RC-ASE Tool.**

just showing a map can not be considered as providing enough guidance, the contribution relationship can not be considered as a *Make* contribution. To keep the diagram simple, the *help* relationships between requirements and these two goals have been omitted in Figure 4.23. Since the *Navigation Map* requirement is not enough proof for satisfying the two previously mentioned goals, this main requirement is divided into another two additional requirements which are to show specific instructions on the next movements that users need to do in order to ultimately arrive at their destination. The navigation map will have a *route*, indicating the path that the user has to follow in order to arrive at her/his destination. Additionally, the navigation map will display the *location* of the user in the map in real-time. Although these two new requirements also provide a positive contribution towards the satisfaction of the two main guidance goals of this diagram, they are still not enough proof for providing adequate guidance to the users when walking and displacing by bus. Taking into account the low level objective of *Displacing through safe environments*, the requirement *Customisable routes* is included, where it is specified that the secondary users will be able to create their own routes for the primary users. The difference between the application under development and other navigation applications, is that this option increases the security of the primary users, as parents are expected to send them through safer and easier routes, instead of the most complicated routes. This requirement also satisfies the needs of users with different skill levels. As it can be observed in Figure 4.23, this new requirement is considered as a positive contribution towards the soft-goal for safe environments. Routes will also have *checkpoints* that divide the route into more manageable smaller parts. Although this requirement by itself does not provide any contribution to the objectives, it is necessary to understand the next requirement that derives from it: *Customisable checkpoint instructions*. The checkpoints of the routes, will not only be located by the secondary users, but they will include a set of personalised instructions about the next movement. For example, it could be " *When you see the blue house with a white door, turn left, using the crosswalk*". An additional picture of the blue house can be included for making the instruction more clear. This requirement positively contributes to the soft-goal of *Provide understandable guidance*. These customisable checkpoint instructions can map at *walking* or for *bus displacements*. These last two instruction types *make* the main guidance goals. Therefore, the requirements engineers can consider this diagram as finished, and continue with the following diagram.

Reminder related requirements have a main requirement, which describes that the system is going to be able to prompt the user with reminders, as shown in Figure 4.24. The *Reminders* requirement positively contributes, but is not enough to satisfy, the following goals: *Displacement start guidance*, *Guide on required objects*, and *Walking speed guidance*. Again, in this diagram, *help* type of relationships have been omitted. This requirement is divided into three different reminders, according to each of the goals treated in this diagram: *Displacement reminder*, *Walking speed reminder*, and *Object list reminder*. These three requirements *make* the previously mentioned three goals. The *Displacement reminder* requirement is divided into a reminder for the previous day displacement, previous half an hour and current displacement reminders. From *Object list reminder* requirement, a reminder to take a charger and an extra battery is added, in order to contribute to the *Availability* soft-goal.

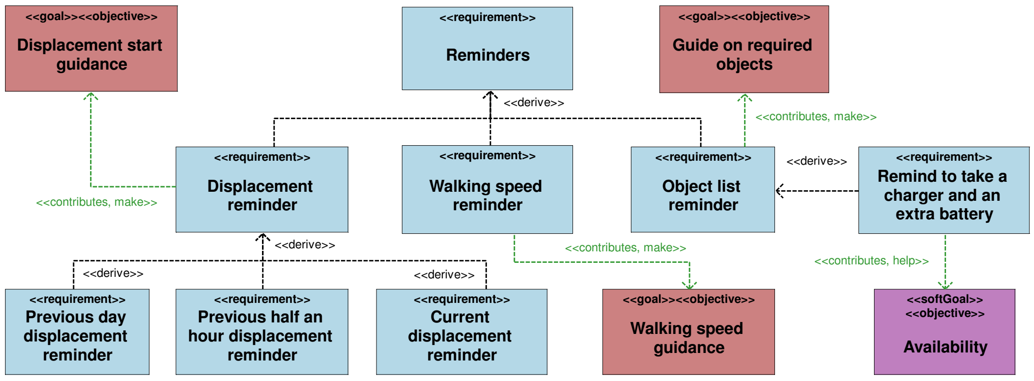
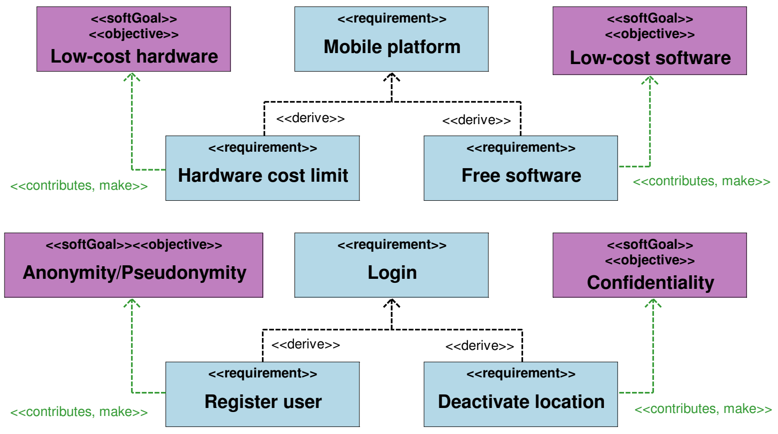


**Figure 4.24: Requirements model II, Requirements Diagram, RC-ASE Tool.**

The remainder of the non-functional requirements are shown in Figure 4.25. This diagram is focused on two main aspects, maintaining the low-cost of the product and respecting the user privacy. Regarding the low-cost aspect, the product should be able to be used on a *mobile platform*. The hardware of this platform has to be *limited* to less than £300. This requirement *makes* the soft-goal *Low-cost hardware*. The software of the application will use a free operating system (Android), and it will be released for free in its market. This other requirement *makes* the soft-goal low-cost software. Regarding the privacy of the users, they will be required to login to the application before being able to navigate. No personal data will be gathered from the users, and they will be able to *register* using a pseudonymous name. This requirement *makes* the *Anonymity/pseudonymity* soft-goal. Additionally, the users will be able to *deactivate* their location to the application at will.



<<softGoal>> <<objective>> Low-cost hardware
<<requirement>> Mobile platform
<<softGoal>> <<objective>> Low-cost software
<<derive>>
<<derive>>
<<requirement>> Hardware cost limit
<<requirement>> Free software
<<contributes, make>>
<<contributes, make>>
<<softGoal>><<objective>> Anonymity/Pseudonymity
<<requirement>> Login
<<softGoal>> <<objective>> Confidentiality
<<derive>>
<<derive>>
<<requirement>> Register user
<<requirement>> Deactivate location
<<contributes, make>>
<<contributes, make>>

**Figure 4.25: Requirements model III, Requirements Diagram, RC-ASE Tool.**

After completing the operationalisation of objectives into requirements, the next step is to personalise or create new requirements according to the different user profiles, as shown in Figures 4.26 and 4.27. As it can be observed in the first figure, the cultural profile affects the existing communication with the users. Therefore, this figure illustrates the different requirements that are created to satisfy the demands of a British profile. The project supports English, German and Norwegian languages, British pounds (BGP) and Euros (EUR), as well as the Imperial and Metric systems. On the other hand, the second figure enables a different communication with the users. For those users with visual impairments, audio based communications will be present, and for those with auditive impairments visual communications will be enabled. The sub-activity for personalisation introduced in Section 4.3.2.4 also includes a specification of the set-up and training. The users of the navigation application, will have to their disposition a training tool for letting them acquire navigation skills in a virtual environment, without exposing themselves to unnecessary risks. For space reasons, further explanation on the training framework is out of the scope of this example, but the Reader is referred to [43] for more information about how users can train using this system.

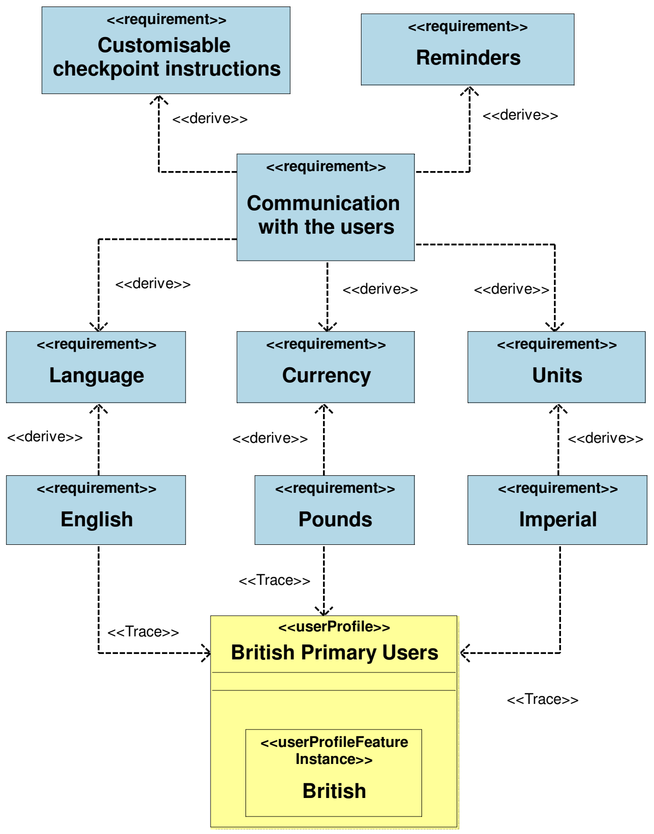


Figure 4.26: Personalised Requirements model I, Requirements Diagram, RC-ASE Tool.

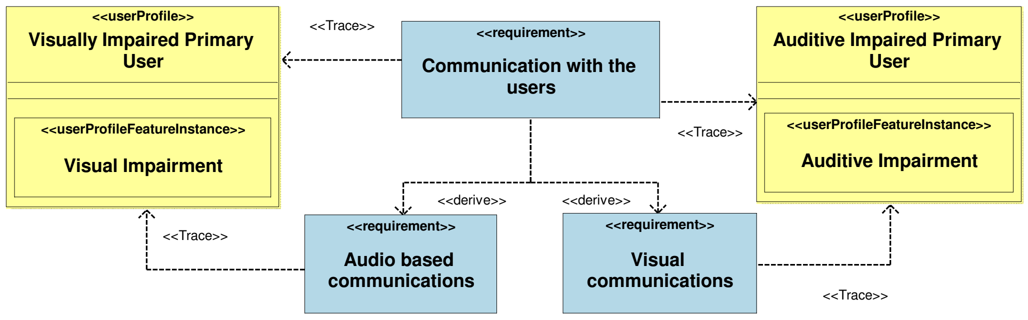


Figure 4.27: Personalised Requirements model II, Requirements Diagram, RC-ASE Tool.

Finally, an ethical analysis of the stakeholders is done. Figure 4.28 shows an example of an ethical analysis of the primary user stakeholder against the *Login*, *Mobile* *platform*, *Reminders*, *Navigation map*, and *Communication with the users* requirements. Note that the figure does not represent the values of the analysis, which are contained in the properties of the *Ethical profile* and can be modified using the Modelio tool. For this profile, there is no *dependency* of the stakeholder on the government regarding the mentioned requirements. Also, there is a medium level of *vulnerability* from the users, in case they can get lost by misinterpreting indications. Nevertheless, the *gravity* of this stake is low. There is a high *likelyhood* that there will be a remedy for this which will be addressed when creating the context-awareness specialisation of the requirements methodology. There is a medium *risk* to the integrity of the stakeholders, and the *policy impact* is high.

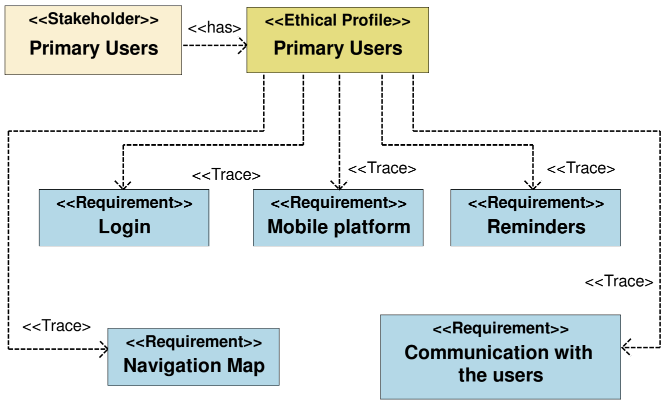


**Figure 4.28: Ethical analysis, Stakeholder Diagram, RC-ASE Tool.**

## 4.6 Evaluate

Finally, an evaluation of the objectives and requirements is conducted, which is guided by a set of heuristics, that has been adapted to their application to the framework presented in this chapter from [187] [109]. Then, a plan for evaluating the requirements is created, setting the criteria for how each requirement will be evaluated once the system is implemented.

### 4.6.1 Prioritisation

There are some objectives that are more important than others. In order to focus the development efforts on the most important objectives, requirements engineers need to classify and prioritise the objectives first. For this, three main priority levels exist, as inherited from [109]: NORMAL, IMPORTANT, and CRITICAL. Objectives, as introduced in the Objective Diagram (Figure 4.10), are defined with a priority attribute, which enables the selection of a value in between these three priority levels. The requirements engineers can classify an objective using the following criteria [64]. An objective is considered as having a CRITICAL priority if its insatisfaction results in a failure for the system (its satisfaction is vital for the success of the system). An objective has IMPORTANT priority if its satisfaction gives an added value to the system. Finally, an obective has NORMAL priority if its partial satisfaction does not severely affect the loss of quality of the system. As a main difference with the activity presented in [109], is that the prioritisation activity presented here does not take into account any context or situations during $F_1$. Instead, a different prioritisation of situations of interest is introduced as further explained in Section 5.3.

### 4.6.2 Evaluation

The next sub-activity helps engineers to determine if the current modelled operationalisation of objectives into Requirements satisfies the objectives. For this, the evaluation procedure for the NFR framework is adopted, as presented in Page 70 [187] and Page 146 [64]. The evaluation procedure uses a set of 11 rules, available which determine the satisfaction result of the proposed objectives, and its propagation to higher level objectives. These rules are further explained in [64]. An algorithm inspired in the work of [64] is introduced, which has been informally applied, and which helps the evaluation of the main requirements of the system:

1) From a given model, for all those Objectives that are not the source of an objRefine relation, apply the satisfaction function rules from [64].
2) Propagate the satisfaction function results to higher level objectives following the rules from [64].
3) If there exists an Objective whose satisfaction function is not considered as SAT-

ISFIED, and the Objective is CRITICAL, then the verdict is considered as failed.

4) If there exists an objective whose satisfaction function is not considered as SATISFIED, and the objective is IMPORTANT, then the verdict is considered as passed with warnings.
5) For other cases, the verdict is considered as passed.

The purpose of this evaluation is to show the potential of the approach for future extensions, that can also include other types of formal evaluation. Note that this evaluation has still not been fully formalised and is not proposed as a fully mature evaluation, rather as something which it is still under assessment.

### 4.6.3 Example

First of all, if it has not been done already, all the objectives defined in 4.4.5 need to be prioritised. In this case, the objectives have been given a priority as shown in Table 4.1.

Once the priority of all the objectives is established, the R-CASE module will automatically give a verdict on the satisfaction of the objectives of the system, according to the algorithm and rules explained in Section 4.6. For this example, the result of this evaluation can be observed in Figure 4.29. The current verdict of the example is *WARNING*, as the *Support lost users* objective is *DISSATISFIED*, and the *Availability* objective is *PARTIALLY_SATISFIED*. This means that in order to improve the verdict to *PASS*, special attention should be paid to the completion of these objectives when eliciting requirements related to the context-awareness of the system.

## 4.7 Conclusions

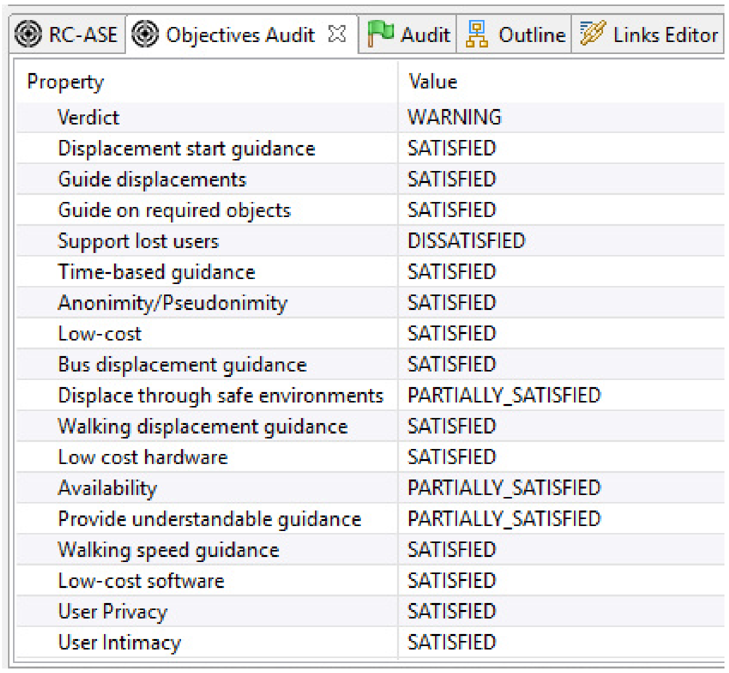
Chapter 2 has presented a novel conceptualisation of context which is aimed at maximising the usability results of developed context-aware systems, mitigate the limitations of C-AS and strengthen their real capabilities. Chapter 3 analyses existing methods and techniques for eliciting requirements for context-aware systems. The conclusion from this analysis is to follow an approach that assembles existing methodologies for requirements elicitation, and then embedding in them the conceptualisation of context

| Objective | Priority |
|---|---|
| Guide displacements | Critical |
| Walking displacement guidance | Critical |
| Bus displacement guidance | Critical |
| Time-based guidance | Important |
| Displacement start guidance | Important |
| Walking speed guidance | Important |
| Low-cost | Normal |
| Low-cost hardware | Normal |
| Low-cost software | Normal |
| User privacy | Normal |
| Anonimity/Pseudonimity | Normal |
| Confidentiality | Normal |
| Displace through safe environments | Important |
| Support lost users | Important |
| Guide on required objects | Important |
| Provide understandable guidance | Important |

**Table 4.1: Prioritisation values for the different objectives presented in section 4.4.5.**

introduced in Chapter 2. Accordingly, the description of the requirements methodology is divided in two chapters. This chapter creates the foundations of a requirements elicitation framework (RC-ASEF), by assembling different requirements elicitation methodologies and tools. Chapter 5 extends the framework introduced in this chapter to embed the conceptualisation of context presented in Chapter 2.

The framework presented in this chapter provides a coherent guide for developers for the requirements elicitation process which is specialised for gathering the non-contextual aspects of context-aware systems. Developers are guided from the identification of stakeholders, to the identification of objectives and its corresponding operationalisation into requirements. This guidance process facilitates the identification of key information which can be used in the specialisation of the framework (SRC-ASEF), introduced in Chapter 5. Chapters 1 and 2 highlight the need to have an end-



| Property | Value |
|---|---|
| Verdict | WARNING |
| Displacement start guidance | SATISFIED |
| Guide displacements | SATISFIED |
| Guide on required objects | SATISFIED |
| Support lost users | DISSATISFIED |
| Time-based guidance | SATISFIED |
| Anonimity/Pseudonimity | SATISFIED |
| Low-cost | SATISFIED |
| Bus displacement guidance | SATISFIED |
| Displace through safe environments | PARTIALLY_SATISFIED |
| Walking displacement guidance | SATISFIED |
| Low cost hardware | SATISFIED |
| Availability | PARTIALLY_SATISFIED |
| Provide understandable guidance | PARTIALLY_SATISFIED |
| Walking speed guidance | SATISFIED |
| Low-cost software | SATISFIED |
| User Privacy | SATISFIED |
| User Intimacy | SATISFIED |

**Figure 4.29: Screenshot of the evaluation of objectives using the RC-ASE module in Modelio.**

user stakeholder centred perspective. In order to support this, the framework supports an initial stakeholder analysis, from which more individualised stakeholder profiles are derived, enabling to match functional requirements to particular capabilities of users. Another salient feature of the profiling activity includes a core ethical model.

In order to facilitate the management of this information, a model-based approach is taken. The framework introduces an enhancement of some existing UML/SysML profiles, introducing the: Stakeholder, Goal, and Requirements Diagrams. As part of the contribution of this chapter, a new module for Modelio has been created, namely Requirements for Context-Aware Systems Engineering (RC-ASE) [194]. This module implements not only the Diagrams introduced during this section, but also the missing SysML features that the free version has, including *traceability matrices* and *requirements tables*, as well as other relevant functionality such as partial documentation generation. This tool has been extended in Chapter 5. More information on how to download, install, use or develop the tool can be found in Appendix A.

Additionally, the framework inherits the means to analyse and guide the software design, exploiting the benefits of design techniques that adequately satisfy the objectives of the system, through the study of its contributions. This analysis includes a study of the possible obstacles that might hinder the satisfaction of objectives and shows the impact of the proposed requirements in the system objectives, as well as the decision taken towards the satisfaction of objectives. This evaluation process has also been implemented in the open-source tools. The usage of the framework is illustrated using the case-study introduced in Section 1.5.2. The resulting models obtained with the application of the proposed method contain reusable information that can be used as an input for the situational requirements and for the design stage, introduced in Chapters 5 and 6.

Chapter 5

# SRC-ASEF: Situational requirements for the context-aware systems engineering framework

## 5.1 Introduction

Chapter 2 analyses the problems behind the conceptualisation of context and context-awareness, introducing new definitions of the concept with the aim of directing them towards the engineering of more usable context-aware systems. Chapter 4 has introduced RC-ASEF, a framework for gathering requirements of context-aware systems, which is divided into two main foci: $F_1$, for non-context related requirements and $F_2$ for context related requirements. This section presents the Situational Requirements for the Context-Aware Systems Engineering Framework (SRC-ASEF), a specialisation of RC-ASEF focused on the contextual aspects of the requirements elicitation in context-aware systems ($F_2$), which is based on the conclusions presented in Chapter 2. The framework presented in this chapter has a situation-based approach to guide developers with regard to the identification of situations, situation detection mechanisms and associated *context-aware features*[1]. For this, it relates to the outcome of the framework introduced in Chapter 4, RC-ASEF. The previously introduced RC-ASEF framework gives great importance to the exhaustive analysis of the system stakeholders. This information is used in SRC-ASEF for the discovery of *situations of interest* (SOI[2]), as well as for choosing associated services according to the needs and preferences of the stakeholders. While the artefacts produced by RC-ASEF are more likely to be static, those artefacts produced in SRC-ASEF aim to dynamically model the different aspects of the system. In this way, requirements engineers can focus on the analysis of the requirements according to the different possible variations of the models created with SRC-ASEF. The SRC-CASEF framework has been created following the objectives explained in Section 1.4, and it supports:

- A set of guidelines which are based on the perspectives and definitions introduced in Chapter 2, which take into account the current strengths and limitations of the state-of-the-art in context-aware systems.
- The discovery of different SOIs, as well as the contextual objectives of the systems, and the needs of the stakeholders in those situations.
- The identification and establishment of situational needs and objectives, in order to guide requirements engineers towards the adequate context-aware fea-

[1]The concept of context-aware feature is further explained in 2.4.2.

[2]The concept of SOI is explained in Definition 1, Section 2.4.3.

tures to be triggered in the situations analysed, supported by the *Situation of Interest Diagram*. This includes an evaluation of the objective operationalisation in the same fashion as that evaluation included in Section 4.6. As it also enables the association of the context-aware features displayed in a certain SOI to static objectives, it facilitates the analysis of the effects on dynamic changes to the different SOIs, not only by determining which context-aware features are most adequate towards the satisfaction of situational objectives, but also facilitating design and implementation of related decisions, by analysing the completion of higher order objectives when adding/removing/editing SOIs.

- Guidance for developers to analyse different approaches to the detection of a SOI once the system is implemented. Including the automated recommendation evaluation procedure to facilitate the analysis of the proposed detection plans and associated context-aware features of a certain SOI, supported by the *Situation Detection Plan Diagram*.
- A model-based representation of the relevant elements presented as part of a UML profile. The representation of requirements and related elements during the development process is supported by an open-source tool, which includes: SysML Diagram Representation, including traceability matrices, requirements tables and a link editor view, automatic generation of certain documentation artefacts, and compatibility with other open-source and licensed modelling tools, such as the UML testing profile, UML, SysML and other tools for completing the remaining development stages of this methodology. The tool is also open-source, and it constitutes an extension of the open-source tool presented in Chapter 4, which includes the model-based representation of two novel diagrams and their corresponding elements, presented as part of the SRC-ASEF UML profile. Also, it includes the algorithms for evaluating the completion of objectives with the proposed SOIs, associated context-aware features and associated situation detection plans.

The remainder of the chapter is as follows. Section 5.2 explains the main activities of SRC-ASEF. Section 5.3 explains the different evaluation procedures for determining the suitability of the different approaches. Section 5.4 illustrates the usage of the framework with an example based on the case-study presented in Section 1.5.2. Finally, Section 5.5 summarises the chapter.

## 5.2 Main activities

This section explains the main activities of the framework presented in this chapter, as shown in Figure 5.1. The central activity of the framework uses the output from the end-user stakeholder activity analysis in $F_1$ to identify SOIs. For each SOI identified, the rest of the peripheral activities of the methodology are applied. Once a situation of interest is selected for analysis, and although there is iterative flexibility on the steps, the natural activity that follows is that of analysis of the needs and objectives of the stakeholders. The stakeholder analysis conducted during $F_1$ can contain useful

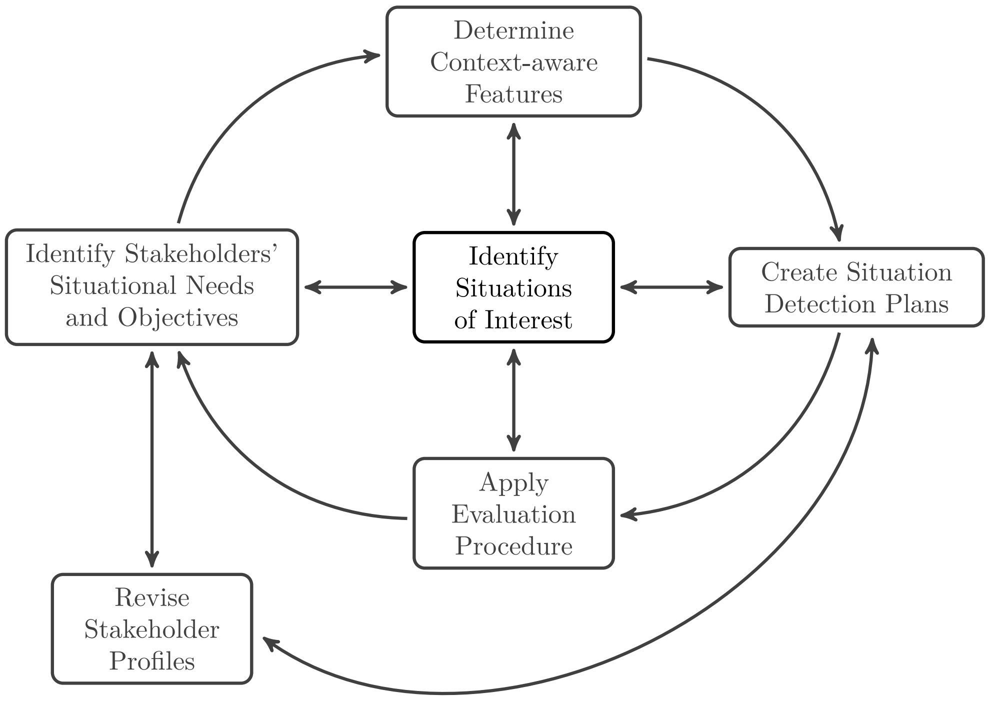

Determine Context-aware Features
Identify Stakeholders' Situational Needs and Objectives
Identify Situations of Interest
Create Situation Detection Plans
Apply Evaluation Procedure
Revise Stakeholder Profiles

**Figure 5.1: Context-related activities in the late requirements elicitation stage, corresponding to F2.**

information to identify the particular needs of each end-user stakeholders in the different situations of interest. Additional interviews and questionnaires can be conducted to have a better understanding of the needs of the end-user stakeholders for this purpose. The aim of this activity is that of understanding the meaning of the action of the end-user stakeholders in order to identify the objectives and needs of the different stakeholders in the situation under analysis. Once the needs of the stakeholders are analysed, these are reflected as situational objectives. The main distinction between an objective and a situational objective is that the objective ultimately stems from a value or a value enhancement of the system, while the situational objective ultimately stems from, and only exists, in the domain of a situation of interest. As it is done with regular objectives, these need to be refined, analysed for obstacles and resource exchanges in the same way as explained in Sections 4.4.1, 4.4.2, and 4.4.3. Additionally, situational objectives can be related to other existing objectives of the system. Another activity is the determination of the adequate context-aware features, which are relevant to the intention of the users and help them achieve their goals according to their preferences and needs. In the same way that requirements are operationalised from objectives, as further explained in Section 4.4, the situational objectives are operationalised into context-aware features. Note that there could be more than one context-aware feature triggered by a particular SOI. Introduced context-aware features relate not only to situational objectives, but they might also relate to regular objectives.

Situation of interest detection plans also need to be proposed. These consist of identifying the different *context-attributes*[3] that can be used to detect a particular situation of interest. Note that there could be more than one plan for implementing the same situation of interest. This step can have as much detail as developers wish to consider, and it can be an informal definition or it can include the assignation of values or value domains to context-attributes, or even the definition of particular rules or ontologies that will be used for inferring higher level context-attributes from lower-level context-attributes, or to trigger the associated services.

User profiles identified as part of $F_1$ have to be taken into consideration, *i.e.*, that which can help discover new, or personalise existing context-aware features. Note that, as a consequence of applying the different activities of SRC-ASEF, it might happen that some existing user profiles might change. For this reason, the stakeholder pro-

[3]The concept of context-attribute is defined in Definition 2, Section 2.

files need to be revised. The following sub-activities of the revise stakeholder profiles activity have been adopted from R4IE [108], and PC-RE [106]:

- Task subset: There might be different levels of users that should be reflected in a stakeholder profile, these different levels of users might have different situational needs, objectives and required context-aware features.
- Context interaction requirements: The most appropriate interaction modality for the proposed service is determined. For each of the context-aware features proposed, requirements engineers determine the most adequate *interaction modality*[4] for the different user profiles.
- Monitoring mechanisms: Using the sensors of the system, it might be beneficial for the system, or the business model of the application, to monitor certain end-user stakeholder activities. During this stage, the activities to be monitored are determined, as well as the mechanisms to be used for that purpose (e.g., supervised or unsupervised learning mechanisms). Taking into account the goals of particular users, certain achievement levels can be set and monitored. Goals of particular user profiles can be set-up for being monitored, to be aware of their achievement levels.

The analysis of the SOI ends with its evaluation, in order to give the developer more information about its implementation feasibility. This process is further explained in Section 5.3.

## 5.3 Apply evaluation procedure

During this activity, developers evaluate different aspects which can help them determine if the situation of interest under analysis should be implemented or not. The evaluation process consists of the following steps, as illustrated in Figure 5.2:

- Prioritisation: Similar to what it is explained in Section 4.6.1, situational objectives are prioritised. But also, not all situations of interest have to be approached with the same degree of effort. Some situations of interest are more important than others, and this should be accounted for. As well as with objectives, situ-

[4]The different ways of interacting with a context-aware feature, are explained in Section 2.4.1.

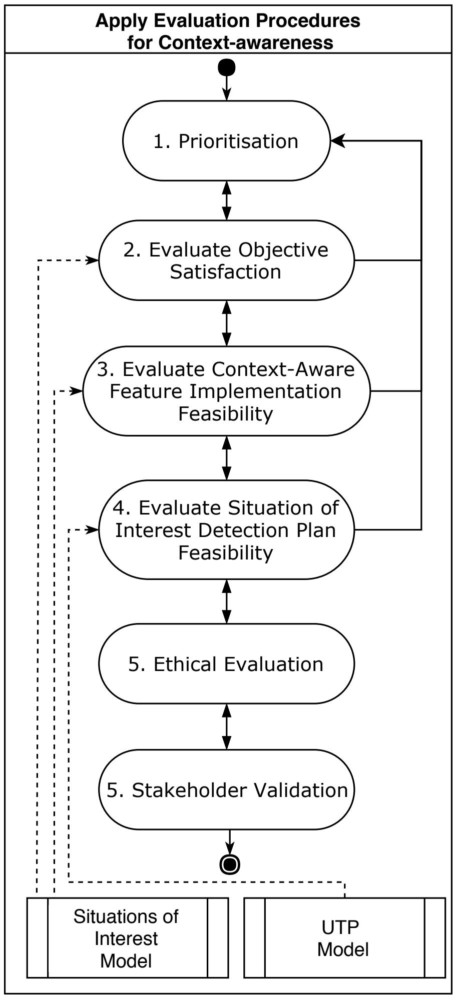

Apply Evaluation Procedures for Context-awareness
1. Prioritisation
2. Evaluate Objective Satisfaction
3. Evaluate Context-Aware Feature Implementation Feasibility
4. Evaluate Situation of Interest Detection Plan Feasibility
5. Ethical Evaluation
5. Stakeholder Validation
Situations of Interest Model
UTP Model

**Figure 5.2: Core sub-activities in the early requirements elicitation stage, corresponding to F2.**

ations of interest are prioritised with one of the three priority levels: Critical, Important and Normal [64].

- Objective satisfaction evaluation: During the activity for identifying situational objectives, several situational objectives are identified, and these, along with the proposed context-aware features, are related to previously existing objectives. During the goal operationalisation, the objectives of the system after including the SOI are re-evaluated, according to the evaluation procedure introduced in Section 4.6.2.
- Evaluation of context-aware feature implementation feasibility: There are three main aspects that should be taken into account to conduct this cost-benefit analysis. The first aspect is the cost estimation, where developers gauge the cost of implementing that particular context-aware feature. The second aspect is the frequency with which the situation of interest related to the context-aware feature under analysis is expected to occur. Finally, the last aspect is the detection plan feasibility of the situation of interest related to the context aware feature under analysis. This analysis is explained in the previous bullet-point. This analysis can help to avoid implementing features that have a high cost and are not going to occur frequently, taking also into account the detection plan feasibility. The specific procedure to conduct this analysis is further explained in Section 5.3.2.
- Situation of interest detection plan feasibility: Each proposed situation detection plan is analysed in more depth. For this analysis, two main features are taken into account. The first feature focuses on determining how likely it is for the current plan to misunderstand the occurrence of a particular situation of interest. For this, developers can check the accuracy of each of the proposed context-attributes. The second feature has to do with the impact, in terms of objectives of the system, that a failure in detecting a particular situation of interest can cause. A cost-benefit analysis using these features can help developers to determine the feasibility of implementing the detection plan under analysis. This can help to avoid the implementation of situation detection plans that have a high failure likelihood and a considerable situation detection failure impact. The specific procedure to conduct this analysis is further explained in Section

5.3.1.

- Ethical evaluation: Engineers evaluate if the implementation of the detection of the situation of interest or its associated context-aware features has a conflict with other stakeholders, or if it implies ethical or privacy concerns which are against those of the development team values. An ethical framework is recommended for this purpose [186]. It is recommended to carefully evaluate and discuss with the end-user stakeholders the different ethical concerns that might arise, until there is an agreement between all parties. The discussions can be complemented by questionnaires and/or interviews.
- Validation: The situation of interest should pass through a selection process that helps to determine if the situation of interest is important. For this, a priority is assigned to the situation of interest. Engineers check if the situation is within the system scope or budget, and ask themselves if its associated context-aware features will truly help the user or not according to their preferences and needs. The proposed context-aware features can be validated with the end-user stakeholders in order to check whether or not the proposed features and their interaction modality are adequate for them. For this purpose, interviews or questionnaires can be used. During this stage, it is not the intention to run a questionnaire or interview for each of the identified situations of interest, but rather to add the corresponding questions to a general questionnaire or interview.

The proposed evaluation procedures are not just envisaged for complementing the requirements elicitation stages of the system, but can also be applied at later stages of the life-cycle of a C-AS, such as maintenance. This conceptualisation enables the developer to have better control over the context-attributes and services associated to a particular SOI, facilitating the maintenance of systems. The dynamic nature of context demands that developers constantly exercise an understanding of the interaction between the user and the system in different situations. Such conceptual tools empower developers, as they guide them exactly to those elements that they need to alter in an already developed system. Developers can decide to remove or modify a particular SOI, controlling which context-attributes and services will be affected. Also, they can have a better control over the system design when adding/removing associated services.

## 5.3.1 Situations of Interest Diagram

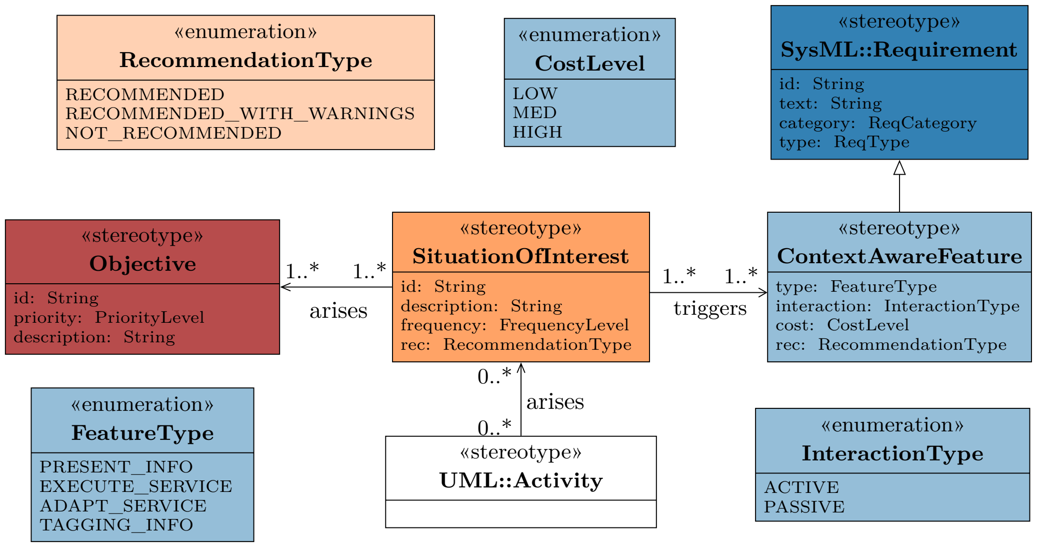


**Figure 5.3: Metamodel for the Situation of Interest Diagram.**

In order to facilitate the capture of the relevant information resulting from the application of the framework, the Situation of Interest Diagram is introduced, whose metamodel is illustrated in Figure 5.3. This diagram introduces two main stereotypes: the SituationOfInterest and the ContextAwareFeature. The SituationOfInterest has an attribute, *frequency*, which determines an estimation of the occurrence frequency for that situation of interest, which can have a value in between: LOW, MEDIUM and HIGH. The ContextAwareFeature has an attribute, *interaction*, which determines the interaction type of the feature. The interaction type can be one of the four interaction types introduced in Section 2.4.2. Finally, the ContextAwareFeature stereotype also has another attribute, *cost*, which estimated the implementation cost of the feature in a level between: LOW, MEDIUM and HIGH.

Both SituationOfInterest and ContextAwareFeature stereotypes have an attribute, *rec*, which determines a recommendation level for implementing the element represented by the stereotype, in a scale between: RECOMMENDED, RECOMMENDED_WITH_WARNINGS, and NOT_RECOMMENDED. In order to automatically estimate the implementation feasibility value of this attribute, two evaluation procedures are introduced, one for each stereotype.

The evaluation of the *rec* attribute of the ContextAwareFeature stereotype is introduced in Table 5.1, where three main factors are taken into account:

- Occurrence Frequency: This factor focuses on analysing the expected frequency that a SOI is expected to occur, as determined by the stakeholders in the Situations of Interest Diagram.
- Cost: It analyses the estimated cost that the engineers give to the particular ContextAwareFeature in the SOI Diagram.
- SOI Detection Feasibility: This last evaluation factor directly depends on the evaluation of the *rec* attribute of the SituationOfInterest stereotype. Particularly, it depends on the recommendation of its proposed DetectionPlans, from which the SOI detection feasibility is calculated, as shown in Table 5.2. which is conducted as shown in Table 5.2, it is determined whether or not it is feasible to detect a given SOI, as it is further explained in Section 5.3.2.

## 5.3.2 Situation Detection Plan Diagram

The Situation Detection Plan Diagram, whose metamodel is illustrated in Figure 5.4, facilitates requirements engineers to design different ways in which the system under development can detect a situation of interest. This diagram introduces three main stereotypes: DetectionPlan, ContextAttribute, and ContextPreference. The detection plan represents set of observable properties (ContextAttributes and ContextPreferences, as defined in Sections 2.4.4 and 2.4.7) of a Situation of Interest which are specifically combined for detecting the occurrence of that Situation of Interest. It helps modelling the way in which the system is going to be aware of a SOI by decomposing it into its observable properties. The DetectionPlan stereotype has an attribute, *rec*, which is related to the recommendation for each DetectionPlan to be implemented. The value of this attribute can be calculated using the evaluation procedure shown in Table 5.3, which is based on analysing the following dimensions:

- Failure Likelihood: The failure likelihood of a DetectionPlan determines the tendency of a situation of interest DetectionPlan to fail detecting the occurrence of a particular situation of interest. A DetectionPlan is conformed by a set of primary (low-level) ContextAttributes that relate through *derive* relationships to other secondary (high-level) ContextAttributes. Typically, this follows

| Frequency SOI | Cost CAF | SOI Recommendation SOI | Recommendation CAF |
|---|---|---|---|
| LOW | LOW | NOT_REC. | NOT_REC. |
| LOW | LOW | REC_WITH_WARNINGS | NOT_REC. |
| LOW | LOW | RECOMMENDED | REC_WITH_WARNINGS |
| LOW | MED | NOT_REC. | NOT_REC. |
| LOW | MED | REC_WITH_WARNINGS | REC_WITH_WARNINGS |
| LOW | MED | RECOMMENDED | REC_WITH_WARNINGS |
| LOW | HIGH | NOT_REC. | NOT_REC. |
| LOW | HIGH | REC_WITH_WARNINGS | NOT_REC. |
| LOW | HIGH | RECOMMENDED | REC_WITH_WARNINGS |
| MED | LOW | NOT_REC. | NOT_REC. |
| MED | LOW | REC_WITH_WARNINGS | REC_WITH_WARNINGS |
| MED | LOW | RECOMMENDED | RECOMMENDED |
| MED | MED | NOT_REC. | NOT_REC. |
| MED | MED | REC_WITH_WARNINGS | REC_WITH_WARNINGS |
| MED | MED | RECOMMENDED | REC_WITH_WARNINGS |
| MED | HIGH | NOT_REC. | NOT_REC. |
| MED | HIGH | REC_WITH_WARNINGS | NOT_REC. |
| MED | HIGH | RECOMMENDED | REC_WITH_WARNINGS |
| HIGH | LOW | NOT_REC. | REC_WITH_WARNINGS |
| HIGH | LOW | REC_WITH_WARNINGS | REC_WITH_WARNINGS |
| HIGH | LOW | RECOMMENDED | RECOMMENDED |
| HIGH | MED | NOT_REC. | NOT_REC. |
| HIGH | MED | REC_WITH_WARNINGS | REC_WITH_WARNINGS |
| HIGH | MED | RECOMMENDED | REC_WITH_WARNINGS |
| HIGH | HIGH | NOT_REC. | NOT_REC. |
| HIGH | HIGH | REC_WITH_WARNINGS | NOT_REC. |
| HIGH | HIGH | RECOMMENDED | REC_WITH_WARNINGS |

SOI = SituationOfInterest

CAF = ContextAwareFeature

REC_WITH_WARNINGS = RECOMMENDED_WITH_WARNINGS

NOT_REC.= NOT_RECOMMENDED

**Table 5.1: Recommendation calculation table for a context-aware feature.**

| **Group of recommendations of all the Detection Plans associated to a situation of interest with a *detects* relationship** | **SOI Det. Feas.** |
|---|---|
| Exists a plan in the group that is RECOMMENDED | FEASIBLE |
| Does not exist a plan in the group that is RECOMMENDED and exists a plan in the group that is REC_WITH_WARNINGS | UNDET. |
| Does not exist a plan in the group that is RECOMMENDED and does not exist a plan in the group that is REC_WITH_WARNINGS and exists a plan in the group that is NOT_RECOMMENDED | NOT_FEASIBLE |

SOI Det. Feas. = Situation of Interest detection feasibility.
REC_WITH_WARNINGS = RECOMMENDED_WITH_WARNINGS
UNDET. = UNDETERMINED

**Table 5.2: Situation of Interest Detection Feasibility according to the Recommendation of the DetectionPlans which have a *detects* relationship with the Situation of Interest under evaluation.**

a pyramidal structure that finishes in a single ContextAttribute that is not the source of any *derive* relationship with other ContextAttribute in the same DetectionPlan, as shown in Figure 5.7. The failure likelihood is calculated by inverting the *accuracy* attribute of this highest level ContextAttribute, as shown in Table 5.4. If there is not a specific reason to consider the opposite, the *accuracy* attribute of the secondary ContextAttributes is obtained from propagating the *accuracy* attributes of lower level ContextAttribute stereotyped elements. This means that there is a need to specifically determine the values of the *accuracy* attributes of the lowest level (primary) ContextAttributes, and then propagate these values until the *accuracy* attribute result of the highest order level ContextAttribute is obtained. The accuracy between context attributes related with the *derives* relationship can be propagated taking into account the lowest accuracy of the sources pointing to the particular target. Having a set of context-attributes related with the derives relationship to a particular context-attribute. Then, the accuracy can be propagated using the minimum accuracy from a set

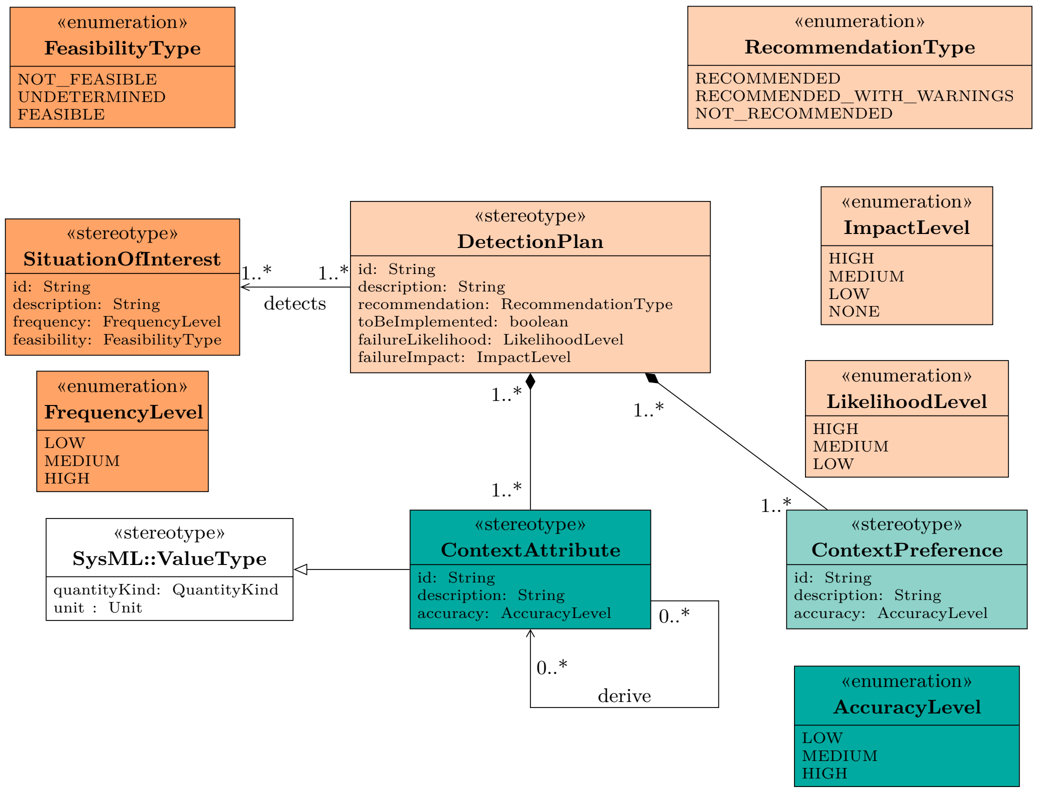


**Figure 5.4: Metamodel for the Situation Detection Diagram.**

of accuracy values where $(LOW < MED < HIGH)$, as shown in Table 5.6.

- Detection Failure Impact: This factor is centred on analysing the impact, in terms of Objectives, that a failure in detecting a particular SOI might cause. For this, the SituationOfInterest stereotyped elements associated to the detection plan with a *detects* relationship is identified. Then, the corresponding ContextAwareFeature stereotyped elements which are related[5] to the identified SituationOfInterest are analysed. Finally, the RefineObj relationships between these ContextAwareFeatures and the Objectives of the system is studied. This result indicates the effect in the Objectives that a failure in detecting a particular SituationOfInterest, following a given DetectionPlan would cause. The detection failure impact value is in between HIGH, MED, LOW and NONE. The failure

[5]Note that ContextAwareFeatures can be derived into other ContextAwareFeatures which may be refined from other situational objectives which also need to be taken into account.

impact is determined by the result of this function, and the type of Objective, as it is shown in Table 5.5. Note that when there are more than one objectives in the analysis, the impact result is considered as the maximum result from all the objectives affecting the detection plan.

| Failure Likelihood Level | Failure Impact Level | Recommendation |
|---|---|---|
| LOW | LOW | RECOMMENDED |
| LOW | MED | RECOMMENDED_WITH_WARNINGS |
| LOW | HIGH | RECOMMENDED_WITH_WARNINGS |
| LOW | NONE | RECOMMENDED |
| MED | LOW | RECOMMENDED_WITH_WARNINGS |
| MED | MED | RECOMMENDED_WITH_WARNINGS |
| MED | HIGH | NOT_RECOMMENDED |
| MED | NONE | RECOMMENDED_WITH_WARNINGS |
| HIGH | LOW | NOT_RECOMMENDED |
| HIGH | MED | NOT_RECOMMENDED |
| HIGH | HIGH | NOT_RECOMMENDED |
| HIGH | NONE | RECOMMENDED_WITH_WARNINGS |

**Table 5.3: Table for calculating the DetectionPlan implementation recommendation attending to the two main factors: a) Failure Likelihood Level of the detection plan; and b) Failure Impact Level of the detection plan.**

| Accuracy Level | Failure Likelihood |
|---|---|
| LOW | HIGH |
| MED | MED |
| HIGH | LOW |

**Table 5.4: Calculation of Failure Likelihood, given the accuracy of those ContextAttributes in a given DetectionPlan which are not the source of any other ContextAttribute in that DetectionPlan.**

| Contribution of context-aware feature related to detection plan | Objective Priority | Impact |
|---|---|---|
| MAKE | CRITICAL | HIGH |
| MAKE | IMPORTANT | MED |
| MAKE | NORMAL | LOW |
| HELP | CRITICAL | MED |
| HELP | IMPORTANT | MED |
| HELP | NORMAL | LOW |
| UNKNOWN | CRITICAL | MED |
| UNKNOWN | IMPORTANT | MED |
| UNKNOWN | NORMAL | MED |
| HURT | CRITICAL | LOW |
| HURT | IMPORTANT | MED |
| HURT | NORMAL | LOW |
| BREAK | CRITICAL | LOW |
| BREAK | IMPORTANT | MED |
| BREAK | NORMAL | MED |
| No objectives | * | NONE |

**Table 5.5: Failure impact for objectives traced to the DetectionPlan with a contributes relationship. It is reviewed in order of positive contribution, *e.g.,* if there is one MAKE relationship and the objective priority is CRITICAL, the impact will be HIGH, regardless of the rest of the contributions.**

## 5.4 Example

### 5.4.1 Identify situations of interest

The metamodels introduced in this chapter will be illustrated using the case study introduced in Section 1.5.2. In order to keep the example simple, it is constrained to the analysis of the most common activities that the primary users perform on being displaced by bus, analysed in Figure 4.7. An additional constraint is set as part of this case study, which is that of focusing on bus displacements happening in London, United Kingdom. The activity for situations of interest identification consists of requirements

| Source Context Attribute 1 | Source Context Attribute 2 | Target Context Attribute |
|---|---|---|
| LOW | LOW | LOW |
| LOW | MED | LOW |
| LOW | HIGH | LOW |
| MED | MED | MED |
| MED | HIGH | MED |
| HIGH | HIGH | HIGH |

**Table 5.6: Example of accuracy propagation for two source context attributes related with a derives relationship to a target context attribute.**

engineers thinking about meaningful actions that the system can accomplish during the different activities that the primary users perform, for which the activity diagram shown in Figure 4.7 is used. The first activity is that of *waiting for a bus*. During this activity two main situations of interest are identified, when *the primary user arrives* at the bus stop, and when *a bus arrives* at the bus stop. These situations are relevant, as the context-aware system can provide services to let the user know how much time is remaining until the bus arrives, or if the bus that has just arrived is the one which the user should get on. The next activity is to *get on the bus*, in which the situation of interest *paying for the journey* is identified. Following, for the *going by bus* activity, two main situations of interest arise, when *primary user has to press the stop button*, and when *the bus arrives at the destination stop*. When *getting off the bus*, it is important to know if the user *failed to get off at the right stop*, especially if there are no more buses coming after that. When the users are *walking*, it might happen that *the user gets lost*. Additionally, when generally using the application it might occur that *the device loses battery power while navigating*, as the primary user will end up without any instructions to continue. Another important situation of interest is when *the user starts a journey with the device having not enough battery to support it without switching off*. The listed situations of interest are just an example of some situations that might be relevant to the case study. Note that these situations can be further nuanced and analysed. Descriptions of situations of interest can vary from requirements engineer to requirements engineer, as they are a reflection of their subjective interpretation of what is required by the end-user stakeholders. Also, this analysis can be applied to different observations of the end-user stakeholders' behaviours, patterns or activities they might perform. The main benefit of SRC-ASEF is the ability to handle the dynamic nature of context, and the requirements engineers understanding of it, as it enables the addition/edition/removal of situations of interest, and analysis of the consequences of the changes in terms of requirements satisfaction.

Table 5.7 summarises the result of applying RC-ASEF to some of the most relevant SOIs of this example, and the main results of their corresponding analysis. Table 5.7 specifically presents data that was a result of a questionnaire used as part of the requirement elicitation of the POSEIDON project, which involved potential uses of an application developed as part of that project [195]. For this questionnaire, 130 British families were contacted, and these were composed of at least one family member with Down's syndrome. The respondent population was divided into people with Down's syndrome (primary users) and carers of people with Down's syndrome (secondary users). A total of 52 responses from potential secondary users and 29 from potential primary users were obtained. Each group had a different format of questionnaire. That prepared for primary users was an "easy-to-read" version, in which they were asked only about the different services. Also, this group was helped by their carers during the process. The questionnaire for the group of secondary users included questions about the situations and the services. This example was a specific scenario used in the requirements engineering phase of the POSEIDON project [14]. During this stage, several meetings, discussions, interviews, and questionnaires were conducted.

Table 5.7 is divided into three main blocks. In the first block (A), the description for each SOI is included. Also, the mean (A-III) and standard deviation (A-IV) of the responses to the questionnaires regarding the perception of the primary users on the usability of the SOIs is provided. In the second block (B), each proposed context-aware feature is illustrated (B-III), for each of the end-user stakeholders of the system (B-I), as well as the proposed execution modality (B-IV). Aditionally, mean (B-V) and standard deviation (B-VI) of the results of the questionnaire regarding the perceived usefulness of the features is included. Also, how much the primary users liked the feature (B-VII). Finally, the last block (C) represents the proposed operationalisation plan in which each context-attribute required is illustrated. It should be noted that these only represent a small sample of all the possible SOIs that could arise during the development of such a system. Some other situations include when the primary user gets lost, someone tries to steal his/her phone, or the primary user falls asleep when travelling by bus. The secondary users that participated in the discussion provided information, using a scale from one to five, on the usefulness of particular SOIs (Table 5.7-A-III/IV) and its associated context-aware features (Table 5.7-B-V/VI). Primary users were asked, in a binary scale, if they would like to have the context-aware feature proposed (Table 5.7-B-VII). The interviews also prompted them about new SOIs and the best interaction modalities for the different features. The questionnaires related to this table can be found in [195].

## 5.4.2 Context-aware features

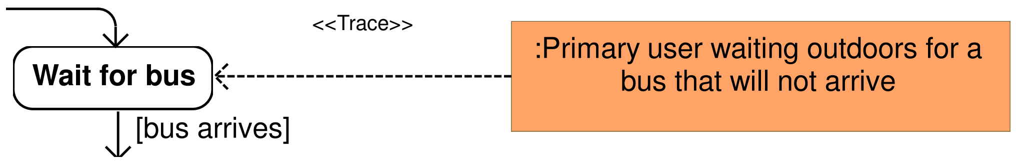
Due to space reasons, only a single SOI will be analysed for the remainder of the example, and this is when the primary user is waiting for a bus that, due to unforeseen circumstances, will not arrive on time, or at all. In this particular SOI there are no more buses available within a given time-frame that will take the user to the destination. Another thing that needs to be taken into account for this SOI is whether or not the user is waiting under certain comfort standards. Some people with Down's syndrome might not be able to realise by themselves that a particular bus is taking more time than it should to arrive. So, particularly, if the user is standing outdoors for a long time, under bad whether conditions, it could imply certain risks for the user's health. Developers can also analyse other alternative variations or extension points on the target SOI. There could be other SOIs where more buses for the same line are available after the the one that is missing, which would arrive in a given time-frame, and would take the user to the destination. There could also be buses from another line available for an alternative route to the destination. The SOI analysed in this particular example is as follows:

*The primary user is waiting outdoors, under bad weather conditions, for a bus that will not arrive due to unforeseen circumstances, and there are no more buses available in a given time-frame to the user's destination.*

| A | | | | B | | | | | | | C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | II | III | IV | I | II | III | IV | V | VI | VII | I | II |
| Activity | Situation of interest | Mean | STD | Stakeholder | Situational Needs and Preferences | Context-aware Feature | Execution | Mean (PU) | STD (PU) | Likes C-A Feature (SU) | Operationalisation | Context-attributes |
| Waiting for bus | PU arrives to the origin stop | 3.9/5 | 1.1 | PU | • Know waiting time<br>• Remember bus line | Notify primary user the bus line and the time remaining | A | 4.9/5 | 0.9 | 28/28 | Navigating, the PU location is that of the bus stop | • Device location<br>• Origin bus stop location |
| | A bus arrives at the bus stop | 4.1/5 | 1.0 | PU | • Identify whether is the bus they need to get on or not | Notify PU to get (or not) on the bus that has arrived | A | 4.3/5 | 0.9 | 23/24 | Current time is (approx.) the scheduled for bus arrival | • Current time<br>• Scheduled bus arrival time<br>• Margin time |
| Going by bus | The bus arrives to the destination stop | 4.3/5 | 0.9 | PU | • Remember that they need to get off the bus in this stop | Notification reminding that they need to get off the bus in this stop | A | 4.5/5 | 0.7 | 22/26 | The primary user's device location is in the destination bus stop location | • Device location<br>• Destination bus stop location |
| | The primary user has to press the stop button | 4.3/5 | 0.9 | PU | • Remember that they need to press the stop button | Notification reminding that they need to press the stop button | A | 4.5/5 | 0.7 | 23/26 | The primary user's device location is one stop before the destination stop | • Device location<br>• Location of the previous bus stop to the destination one |
| Getting off the bus | The primary user fails to get off at the correct stop | 4.5/5 | 0.8 | PU | • Realise situation<br>• Know how to correct it | Notify situation and give instructions | A | 4.5/5 | 0.9 | 17/29 | PU speed is close to walking speed and is earlier than arrival time | • Device speed<br>• Walking speed value<br>• Current time |
| | | | | | | Call secondary user | P | 4.6/5 | 0.7 | 12/29 | | |
| | | | | SU | • Realise situation<br>• Safety and comfort of the PU<br>• Know if the PU can resolve the siuation | Notify situation | A | 4.8/5 | 0.4 | 16/29 | The PU is in the bus route and the movement speed is (approx.) higher than walking speed | • Bus arrival time<br>• Device movement speed<br>• Walking speed<br>• Device location<br>• Bus route |
| | | | | | | Request the location of the primary user | P | 4.8/5 | 0.5 | - | | |

**Table 5.7: Situations of interest (A), context-aware features (B) and context-attributes (C) of the illustration example for a navigation application to support people with Down's syndrome integrate in society. (PU = Primary User, SU = Secondary User, A = Active, P = Passive, STD = Standard Deviation)**

Developers begin to consider meaningful actions that the user can take in that SOI, as well as the needs of the stakeholders. In this case, the requirements analysis puts the potential user into the situation (hypothetically) and asks them for possible actions. Ideally, this process should also observe the users in such scenarios, to help get a better understanding of their actions. It is also worth remembering that the process applies to all the stakeholders, and has to be applied to each of the identified SOIs. The SOI is related to its corresponding element in the model, in this case to the *waiting for bus* activity, as shown in Figure 5.5. In order to select the priority of the situations of interest, the previous evaluation conducted during RC-ASEF is taken into account. As further explained in Section 4.6.3, the objective *support lost users* is not satisfied. Although the context-aware features proposed are not enough to satisfy the objective, they present a positive contribution towards its satisfaction. Additional context-aware features, related to the specific situation *the user gets lost while navigating* will address the satisfaction of this objective. For this reason, the SOI under analysis is categorised as *CRITICAL*. Similarly, other CRITICAL situations of interest are when *the device loses battery power while navigating* and *the user starts a journey with the device having not enough battery to support it without switching off*, as these are related to the *availability* objective. The process of applying SRC-ASEF finishes once the verdict for the objectives and objective analysis is recommended (or recommended with warnings).



<<Trace>>
Wait for bus
[bus arrives]
:Primary user waiting outdoors for a bus that will not arrive

**Figure 5.5: Illustration of the traceability link between the SOI under analysis and a UML activity.**

***Primary Users:*** In this situation, the primary user needs to know that the bus she/he is waiting for will not arrive, and what to do next in order to safely arrive at their destination. During this process, they need to wait in comfortable conditions. Based on these needs, following services are proposed:

- The primary user receives a notification communicating that the bus will not arrive, and a set of instructions to follow.
- The primary user is prompted to call a secondary user, and a list of carers appears in the screen as possible options.

Looking at the interaction modalities of the services, the notification service is chosen to be purely active at execution time. Nevertheless, the instructions received are passively configured by the secondary users, as further explained in the next paragraph. On the other hand, the call to the secondary user will be merely a prompt (passive execution). This second context-aware feature can be passively configured, enabling the secondary users to set a list of preferred secondary users to call.

***Secondary Users:*** In the case of secondary users, they need to ensure the comfort and safety of the primary users during the process of finding another alternative and reaching their destination. The proposed service consists of sending a notification to the secondary users, prior to the notification for primary users, letting them know their current situation. Then, a set of options would be displayed, which include:

- Giving a call to the primary user.
- Requesting the location of the primary user. If the location is available:
  - Suggesting alternative places where the primary user can wait, according to her/his current location. After selecting them, the route to these places will be automatically added to the primary user instructions.
  - Automatically ordering a taxi. Once the user has arrived at the suggested waiting destination, secondary users can select to order a taxi, by automatically sending the location to the taxi service, that will automatically charge them through their bank account.

Although most of the services, when selecting them, are automated, the secondary user is in control of the critical decisions the system will take (passive execution) while the system provides as much relevant information as possible to ease the decision-making. They can evaluate better than a machine whether or not it is the best option for the primary user to go to the nearby sheltered places in particular SOI instances. It might be that the place is inappropriate or that there is no information about opening/closing times, and it is risky to send the primary user there whilst it is raining.

***Independent Primary Users:*** More useful services can be provided to the primary users, but they require the use of the current location of the user. Looking at the ethical aspects, the primary users deserve privacy and intimacy, and should be able to prevent the secondary users for accessing their location. However, in potentially dangerous situations like this, the secondary users are allowed to have access to this information. Figure 4.8 illustrates the main profiles of the Primary Users. In this case, there is a profile regarding the dependency level of the Primary Users. Only in cases where the Primary User is considered as being Independent will they be able to have the following features:

- The device asks permission to reveal the location to the secondary user. In case the permission is given, the Secondary User will manage the presentation of the instructions containing alternative places where to wait and ordering a taxi. If the Independent Primary User does not give permission to the Secondary User, the following instructions will be manually configured.
- Suggesting alternative places where the primary user can wait, according to her-/his current location. After selecting them, the route to these places will be automatically added to the primary user instructions.
- Automatically ordering a taxi. Once the user has arrived at the suggested waiting destination, they can select to order a taxi, by automatically sending the location to the taxi service, that will automatically charge them through their bank account.

In this case, all the context-aware features will be passive at execution and configuration time. This subsection reflects on the result of the feature proposal process. Note that different developers may derive different services or execution modalities, according to their own criteria. The evaluation of the services may depend on different factors. Figure 5.6 represents the Situation of Interest Diagram for the Primary Users and the Independent Primary User profiles. Note that for visualisation purposes, some of the the context-aware features for Secondary Users have been omitted, as well as some relations between elements. Each of the context-aware features has an *arises* relationship with the situation of interest appearing in the diagram. The *Call secondary user* and *Order a taxi* context-aware features *make* the objective *support lost users* soft-goal, while the *Communicate that the bus will not arrive* context-aware feature *helps* to the same soft-goal. The three situational objectives *Know that no more buses are arriving*, *Know what to do next* and *Wait in comfortable conditions* are traced to the Primary User stakeholder. The situational objective *Know that no more buses are arriving*is also traced to the secondary user.

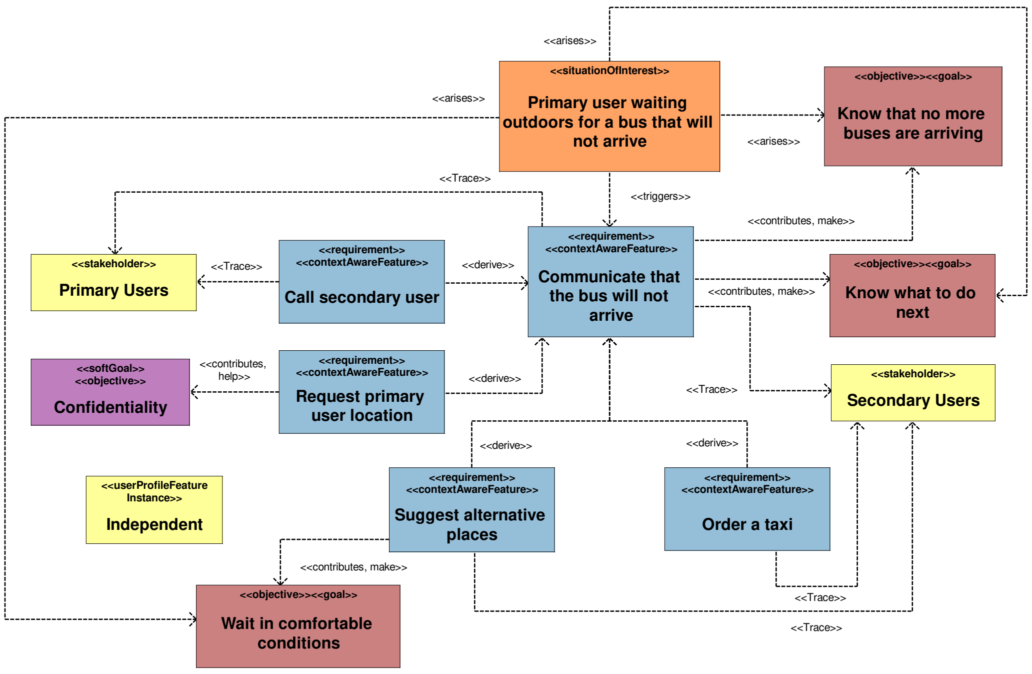


**Figure 5.6: Representation of the situation of interest presented in 5.4.2, created with the Situations of Interest Diagram from the RC-ASE Tool.**

### 5.4.3 Situation Detection Plan

This particular SOI, can be split into: 1) The user waiting outdoors; 2) The user is standing still for a long time; 2) Bad weather conditions; 3) The bus will not arrive in a specified time-frame; 4) No more buses will arrive in a specified time-frame. For 1, the position can be operationalised just by using the signal to noise ratio from the phone of the primary user, and deducing if the person is indoors or outdoors. For 2, it can be estimated with the location history of the user device. In the case of 3, the weather conditions can be obtained via http, through the API of openweathermaps[6] and the location of the user. For 3 and 4, as the users will be located in London, the unified Transport for London (TfL) API[7] can be used. Also, for 3, the user location could be used for estimating how much time has elapsed since the user has been stationary. Note that this is just an example of one possible operationalisation, but developers can consider more than one approach for this purpose, to then compare them using the evaluation procedure presented in Section 5.3. This particular operationalisation has the following main context attributes: *inout*, *standingStillForTooLong*, *weather*, *busNotArriving*, and *noMoreBusesArriving*. In the case of *inout*, it will indicate that the user is indoors when the signal noise-to-ratio value[8] is below 26, and outdoors otherwise. The value indicating if it is day or night will be taken from the date of the calendar. Note that at this stage, it is not important to describe the exact values of the sensors, as this will happen when designing reasoning rules[9]. The context attributes can be considered as variables, as they only describe an observable property, and not its final value. For the *busNotArriving* context-attribute, first, the location and time that the user has been standing in the stop will be taken into account (*standingStillForTooLong*). When the time waiting is higher than a maximum amount of minutes, personalisable by the user, the TfL API will be checked to see if the bus is delayed or not. The data for the origin and destination bus stops and current time will be needed for querying the TfL API. The current bus stop can be deduced from the user location, the bus line from the route, and the time from that of the phone. For *noMoreBusesArriving*, other alternatives to reach the destination can be checked in the TfL API. Note that developers can analyse in greater depth and even start creating the corresponding rules or ontologies.

Finally, the context attributes for *standingStillForTooLong* and *weather* will be enabled for personalisation as follows. The *standingStillForTooLong* attribute, can be derived from two context-attributes, the *location* of the user, and a *timer* that counts how

[6] https://openweathermap.org/api

[7] https://api.tfl.gov.uk

[8] The suggested signal noise-to-ratio value is known to be accurate for this purpose.

[9] More information on how the reasoning rules are created can be found in Section 6.4.2.

much time the user has been in the same position. Additionally, a context-preference will be used, which will let the users decide on how much time the application should consider her or him to be standing still. On the other hand, the *weather* context-attribute can be divided into *precipitation* and *temperature* context attributes. If the weather is cold or there are precipitations, the weather will be considered as bad weather. In order to calculate whether if the temperature is cold or not, a context-preference, *coldTemperature*, will be introduced. With this preference users can introduce to the system at what temperature they feel cold. The representation of the whole plan is shown in Figure 5.7.

### 5.4.4 Evaluation

Finally, the four main aspects of the operationalisation plan are evaluated to determine if the situation of interest and its associated services should be implemented. For this purpose, the aspects introduced in 5.3 are analysed. Particularly, the first two aspects and their sub-aspects are evaluated using the following metric: LOW, MEDIUM and HIGH.

1. Situation of interest detection plan feasibility: For estimating the detection plan feasibility, the failure likelihood and the failure impact are evaluated. For the failure likelihood, all the proposed low level context attributes are evaluated individually, as exemplified below.

   a) *inout*: This main context attribute is based on the *deviceSignal*. With an adequate implementation, the device signal can be used to detect open outdoors, semi-outdoors, light indoors and deep indoors environments with 100% accuracy [196]. Therefore the precision of the main and low level context-attributes can be estimated as HIGH.

   b) *standingStillForTooLong*: This context-attribute directly depends on the accuracy of the *deviceLocation*, depends on the chipset of the phone of the user, and the number of satellites available in the zone. It is reasonable to estimate that in London, the accuracy of the user location will be around 10 meters. This can be used to detect if the primary user is standing still for a long time, in an area of 10 square meters. For this reason, the

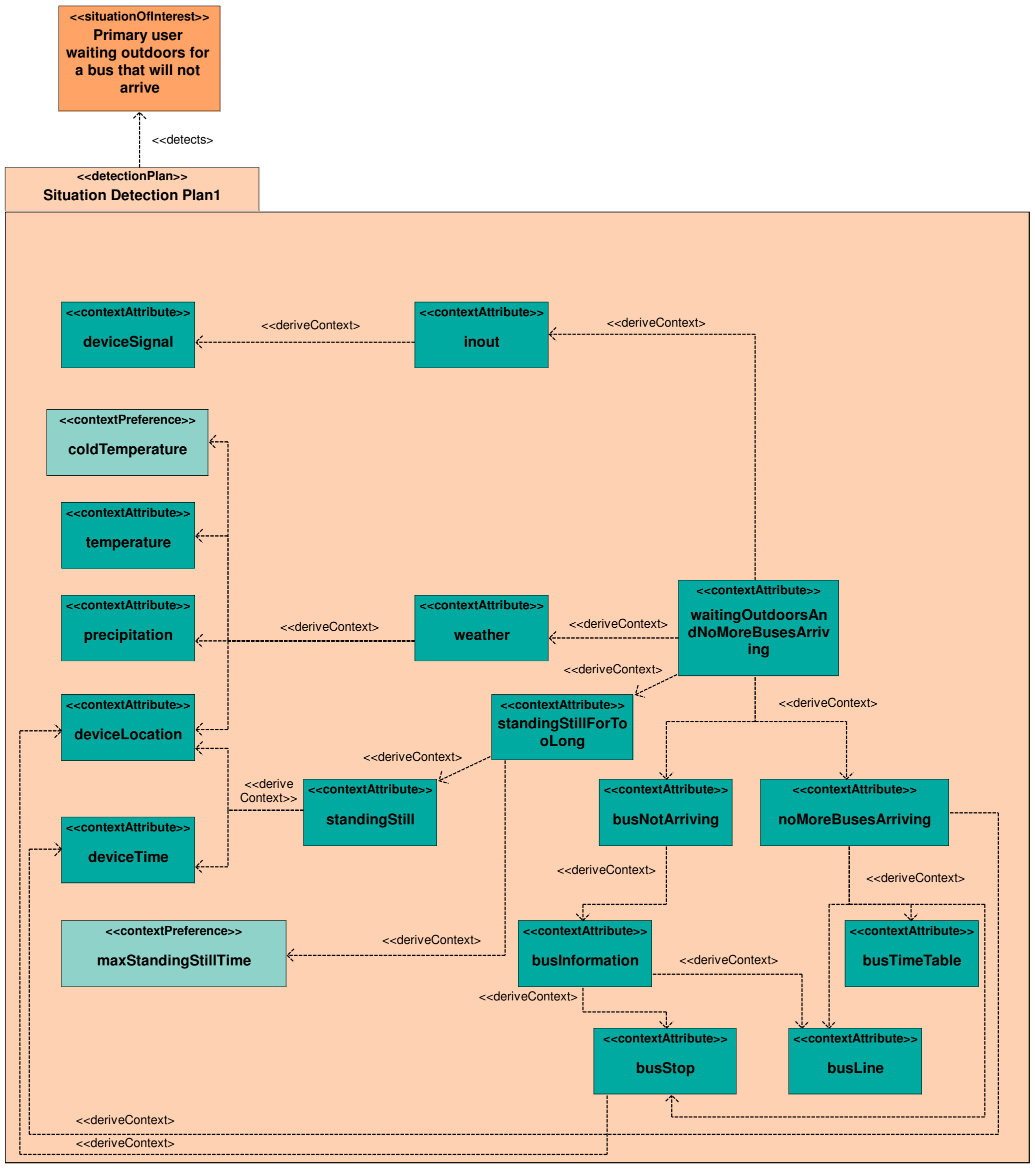


Figure 5.7: Representation of a detection plan for the situation of interest presented in 5.4.2, created with the Situation Detection Plan Diagram from the RC-ASE Tool.

*deviceLocation* context-attribute is considered as having HIGH accuracy. Regarding the *deviceTime*, typically mobile devices have an accurate representation of time passing. Therefore, this low level context attribute can also be considered as having a HIGH accuracy. According to the accuracy propagation rules explained in Section 5.3.2, the *standingStillForTooLong* attribute will have HIGH accuracy.

c) *weather*: As mentioned before, the *deviceLocation* has a HIGH accuracy. As explained in section 5.4.3, the use of open weather maps has been proposed. This service retrieves raw data from airport weather stations, radar stations, and other official weather stations. Additionally, it also involves weather station owners that can help increasing the weather data accuracy. Although this information could not be perfect, it is estimated that for the purpose of this application the *precipitation* and *temperature* accuracy is HIGH. Applying the accuracy propagation rules, the *weather* attribute is considered as having HIGH accuracy.

d) *busNotArriving*: As previously mentioned, the accuracy of the *deviceTime* attribute is HIGH. The real time traffic information about the bus, *busInformation*, can be obtained from London's TfL API. It is known that many transport applications like Citymapper [197] use real time information which is directly retrieved from the official London´s transport system, via the TfL API. Initial qualitative analysis (as part of the POSEIDON project) has concluded that the accuracy provided by the TfL API is sufficiently HIGH. The *busStop* can be retrieved from the *deviceLocation*, having a HIGH accuracy, and the *busLine* can be retrieved from the navigation application. The application of propagation rules, makes the accuracy of the *busNotArriving* context attribute HIGH.

e) *noMoreBusesArriving*: This attribute shares the *busLine* and *busStop* attributes with the *busNotArriving* attribute. It also requires the *deviceTime* to estimate if more buses are scheduled to the location. All these attributes have a HIGH accuracy. Similarly to the *deviceInformation* attribute, the *busTimeTable* attribute can be obtained form the TfL API, with HIGH accuracy. Therefore, the *noMoreBusesArriving* attribute can be considered as having HIGH accuracy.

Overall, the estimation for the failure likelihood is LOW, as the accuracy of all the analysed context-attributes has been estimated as HIGH. On the other hand, for the failure impact detection aspect, two cases can occur. The first case is when the system interprets the occurrence of the situation of interest, but this situation is not really occurring in the real world. In this case, primary users can simply ignore any prompts, and continue with what they were doing. It could happen that the secondary users get worried, but a phone call would take their doubts away. For this reason, the failure impact in this case can be deemed as LOW. The second case is when the situation of interest is occurring, but the system fails to detect it. In this case, it might happen that the user waits for an undetermined time period, for a bus that will never arrive, without the secondary users knowing about it. This situation can entail some risks, but this risk can be estimated as MEDIUM, as the user life would not necessarily be in danger. Although this situation is undesirable, the failure likelihood has been estimated as LOW. For this reason, the result of the situation of interest detection feasibility is HIGH.

2. Feature Implementation Feasibility: For estimating the feasibility of implementing the proposed context-aware features three main dimensions are analysed: cost, frequency of occurrence and detection plan feasibility. For calculating the cost, each of the proposed context-aware features for the situation of interest are analysed individually against these criteria.

    a) The primary user receives a notification: As it can be observed in Table 5.7, the system under development will use notifications in many occasions. Therefore, once a basic system for prompting notifications is created, the cost of implementation of this particular feature will be considered as LOW.

    b) Give the primary user a choice to call secondary user: This feature is quite similar to the previous feature, in terms of implementation costs. It will be simply a prompt with different options. Since Android is an operating system that is focused on mobile devices, it also provides an easy framework from which programming a phone call is straightforward. Therefore, the implementation cost of this particular feature can also be considered as LOW.

c) Secondary users can call the primary users: As mentioned in the previous point, programming this feature in Android is straightforward. Therefore, the cost of implementing this feature can also be estimated as LOW.

d) Request location of the primary users: Although retrieving the location of the user can be readily achieved with the Android platform, this information needs to be transmitted to the phone of the secondary users. For this, an additional secure connection needs to the POSEIDON server, where devices of both users would need to connect. For this reason, the implementation cost for this feature is estimated as MEDIUM, as it requires the reliability of an additional server to work. The implementation cost is not estimated as HIGH, as the non-contextual requirements contemplate the creation of a server for storing the location of the primary users when required.

e) Suggest nearby places: In order to retrieve nearby places using the location, the Google Places API can be used. This does not have any additional difficulty other than learning how to use the API, therefore it can be considered as having an estimated LOW cost.

f) Order a taxi: For ordering a taxi, the Uber API [198] for Android can be used. This does not have any additional difficulty other than learning how to use the API. Therefore, the cost can also be considered as LOW.

It is difficult to predict the exact frequecy of occurrence of the situation of interest, as this would require a further analysis in the field, which can demand the observation of the users, and/or having some interviews with them about the matter. For the purpose of this example, a MEDIUM frequency of occurrence for the situation of interest will be assumed. As introduced in the previous point, the detection plan feasibility result has been HIGH. The feature implementation feasibility will be considered HIGH for all the features with a LOW COST, and MEDIUM for the location request.

3. Ethical concerns: For evaluating the ethical aspect of the situation of interest, the eFRIEND [186] ethical framework can be used. This enables the analysis

of the situation of interest and its associated context-aware features, particularly with regard to the following principles:

   a) Privacy: It is typically difficult to keep a balance between the privacy concerns and the design of context-aware systems. In different interviews and meetings [199], the primary users expressed their concern for the secondary users knowing their location at all times. For this reason, and in order to maintain a balance between the design with the privacy concerns, an option to allow the primary users control when to share their location with the secondary users is provided. Secondary users can request the location of the primary users, but it is the primary users who determine whether or not their location is shared.
   b) User Autonomy: Analysing the proposed features, both primary and secondary users have control over the actions of the system. In the case of primary users, they receive indications and they decide if they want to call the secondary users or resolve the situation by themselves. In the case of secondary users, they can also select from a list of different services available.

   Other ethical aspects such as non-maleficence, user-centred perspective, security, transparency, equality, dignity and inclusiveness, are inherently included in the eFriend framework. In regards to data protection, the information about the users will be stored under the required conditions in the POSEIDON server.

4. Validation: Finally, the situation of interest and their corresponding services were brought up for discussion with the different stakeholders. The secondary users gave 4.5 out of 5 in usefulness to this situation of interest, with a 0.6 standard variance. The context-aware features proposed for the primary users were both graded by the primary users in usefulness with a 4.5 out of 5 (0.5 standard deviation). Those services proposed for the secondary users were similarly graded with 4.7 out of 5 (0.3 standard deviation). Overall the validation gave positive results for the proposed situation of interest and associated services.

In this particular case, the whole analysis gives a positive result, and the situation is considered as implementable under the proposed situation detection plan. In this particular example, the situation of interest and its corresponding context-aware features are considered for implementation. Note that during the evaluation process, the situation of interest under analysis, or some of the associated context-aware features, could be stopped to be considered as implementable for the system under development. In order to calculate the priority, the security of the user will be taken into account. Since its implementation might directly affect the health of the users, the priority of this situation of interest is considered as HIGH.

Note that the whole process is supported by the open-source tool developed for this chapter, which is an extension of the open-source tool presented in 4. This tool can be used not only to facilitate the modelling of the different elements in this section, but to trace them to other elements produced by RC-ASEF. This tool helps to automatically evaluate the proposed situation detection plans and context-aware features, as well as the satisfaction of objectives with new requirements and context-aware features. Additionally, it can also be used to trace the different elements between themselves. This takes special importance when displaying situations of interest, as this concept helps to separate the operationalisation of its detection from the provision of goals, as shown in Figure 5.8. The level of relationships that should be displayed can be customised, as well as the stereotypes displayed. This can also be useful for other aspects, such as the visualisation of those context-aware features and requirements related to a user profile or stakeholder, as well as their relation to other user profiles or stakeholders, as shown in Figure 5.9. This link view could also be used for identifying, among many other things, how many times is a context-attribute reused, and how many elements it affects, as shown in Figure 5.10. This can facilitate the analysis on which elements should be modified/added/removed.

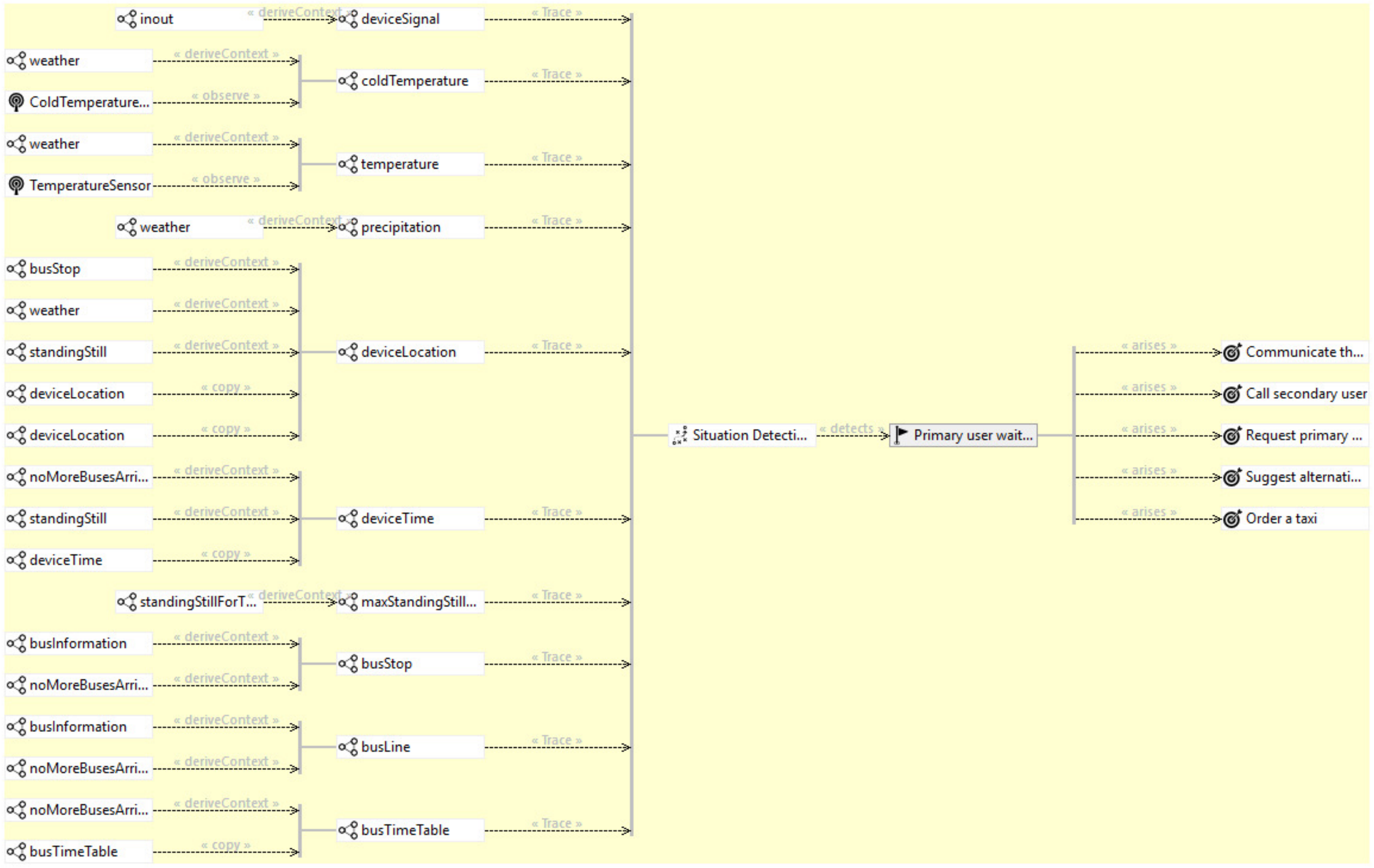


Figure 5.8: Example of the Link view provided by Modelio with the RC-ASE tool for the situation of interest introduced in the example of this chapter.

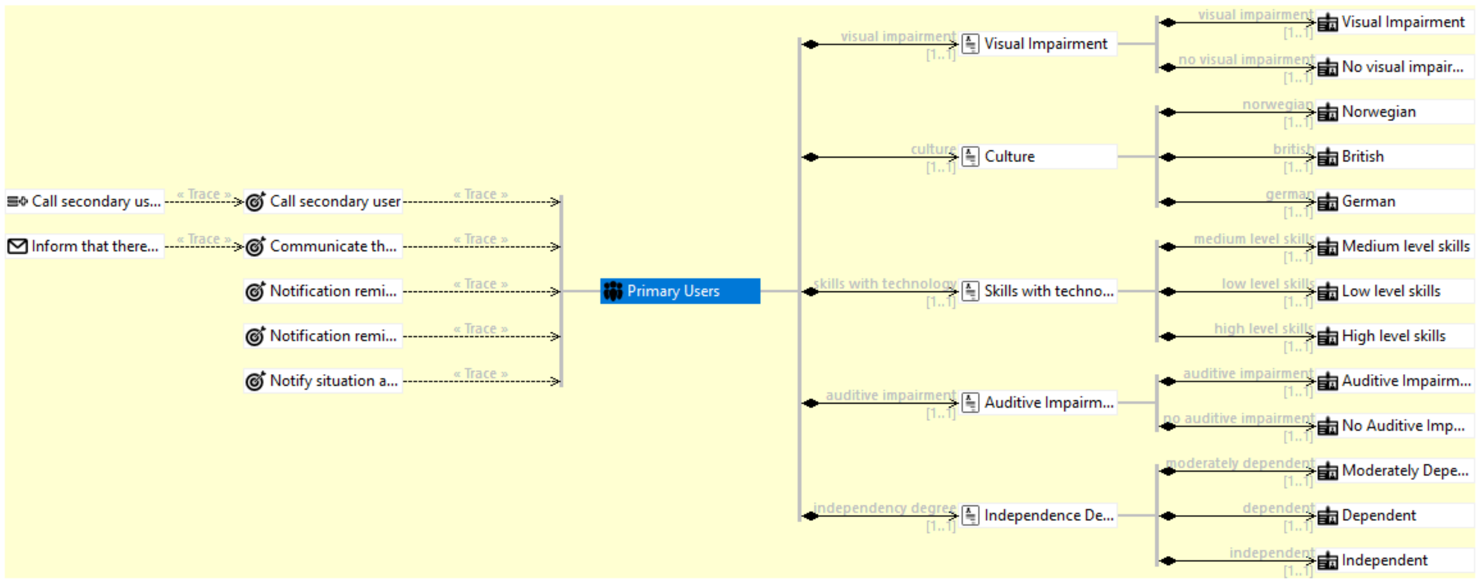


**Figure 5.9: Example of the Link view provided by Modelio with the RC-ASE tool for the primary user stakeholder element.**

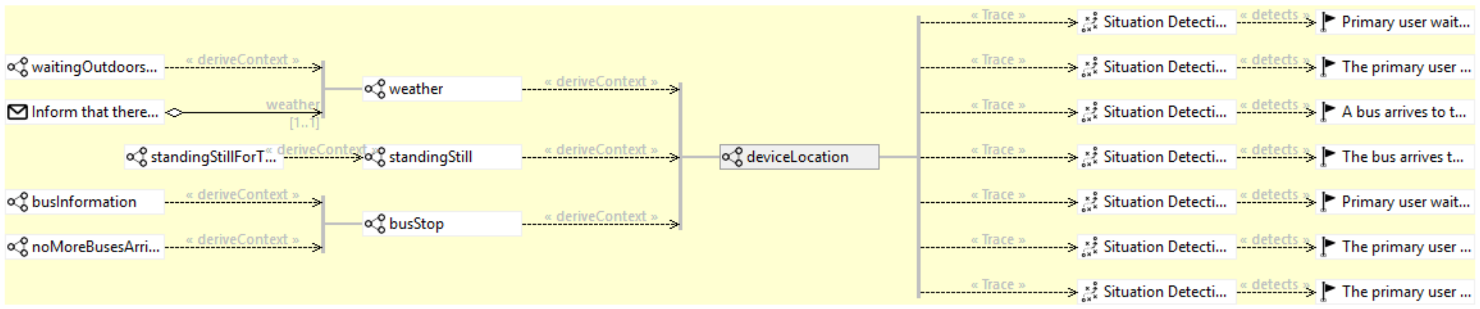


**Figure 5.10: Example of the Link view provided by Modelio with the RC-ASE tool for the deviceLocation context-attribute element.**

## 5.5 Conclusions

According to the conclusions for the analysis of requirements elicitation methodologies for context-aware systems presented in Chapter 3, Chapter 4 follows an approach of assembling existing methodologies for requirements elicitation of the non-contextual aspects of context-aware systems (RC-ASEF). The framework presented in this chapter introduces an extension of the framework for requirements elicitation introduced in Chapter 4 that is more focused on the contextual aspects of context-aware systems. For approaching the contextual aspects of context-aware systems, it accommodates the perspectives on the conceptualisation of context introduced in Chapter 2, introducing a guide for the discovery, documentation, and modelling of context based on the three main principles to get the context right [35]: A) Enumerating the set of contextual states that may exist; B) Knowing what information could accurately determine a contextual state within that set; C) Stating what appropriate action should be taken in that particular state. It inherits and extends the most important aspects from the framework presented in Chapter 4. The extension of the framework introduced in Chapter 4 facilitates the discovery of situations of interest with an end-user stakeholder centred perspective, where the specific user profiles are handled in more detail, and used for the identification of adequate objectives and corresponding functional requirements in the form of context-aware features. As well as RC-ASEF, the framework specialisation introduced in this chapter provides means to analyse and guide the software design exploiting the benefits of design techniques that adequately satisfy the objectives of the system that stem from the different situations of interest. This analysis includes a study of the possible obstacles that might hinder the satisfaction of situational objectives and showing the impact of the proposed requirements in the system objectives, and the decision taken towards the satisfaction of situational objectives. Also, it includes an ethical evaluation. It is important to note that this framework is not aimed at discovering all possible situations that might occur, as this could be a very difficult or even impossible task. It is acknowledged that the number and quality of the situations of interest identified mostly depends on the ability of the requirements engineers to identify them, and their subjective way to understand the context of the end-users.

It may occur that some of the situations of interest proposed, and their associated context-aware features, rise the expectations of some project stakeholders [17]. The framework introduced in this chapter, facilitates the evaluation of different situation detection mechanisms and associated context-aware features to be implemented, during early stages of the development life-cycle. Such an analysis enables to gauge the real abilities that the context-aware system under development is going to exhibit. This can be particularly useful for enabling some project stakeholders with less expertise in computer science to have more realistic expectations. In order to document and handle the information required to analyse these aspects, a model-based approach is taken. The evaluation procedure method proposed for this framework is based on two main diagrams: Situations of Interest Diagram and Situation Detection Plan diagram. The first model represents the functional requirements related specification of context-aware systems, helping developers to state what appropriate actions should be taken in a particular situation of interest. The second model is more related with facilitating developers to define the mechanisms to define what information could accurately determine a situation of interest.

Finally, the novel diagrams and the evaluation mechanisms introduced in this chapter have been implemented and released as open-source, as an extension of the RC-ASE tool [194], introduced in Chapter 4. The section also illustrates the usage of the framework with an example, where several situations of interest are identified, and the whole analysis is conducted for one of them. This illustration example includes screenshots of the models obtained using the corresponding modelling tool of the framework.

The resulting models obtained with the application of the proposed method, contain reusable information that can be used as an input for the design stage, introduced in Chapter 6. Also, the SOI based approach enables maintenance of the system after its implementation, as further explained in Chapter 8. As part of the contribution of this chapter, an extension of the Modelio module introduced in Chapter 4 has been created, namely Requirements for Context-Aware Systems Engineering (RC-ASE) [194].

# Part III

# Design stage

Chapter 6

# DC-ASEF: Design for the context-aware systems engineering framework

## 6.1 Introduction

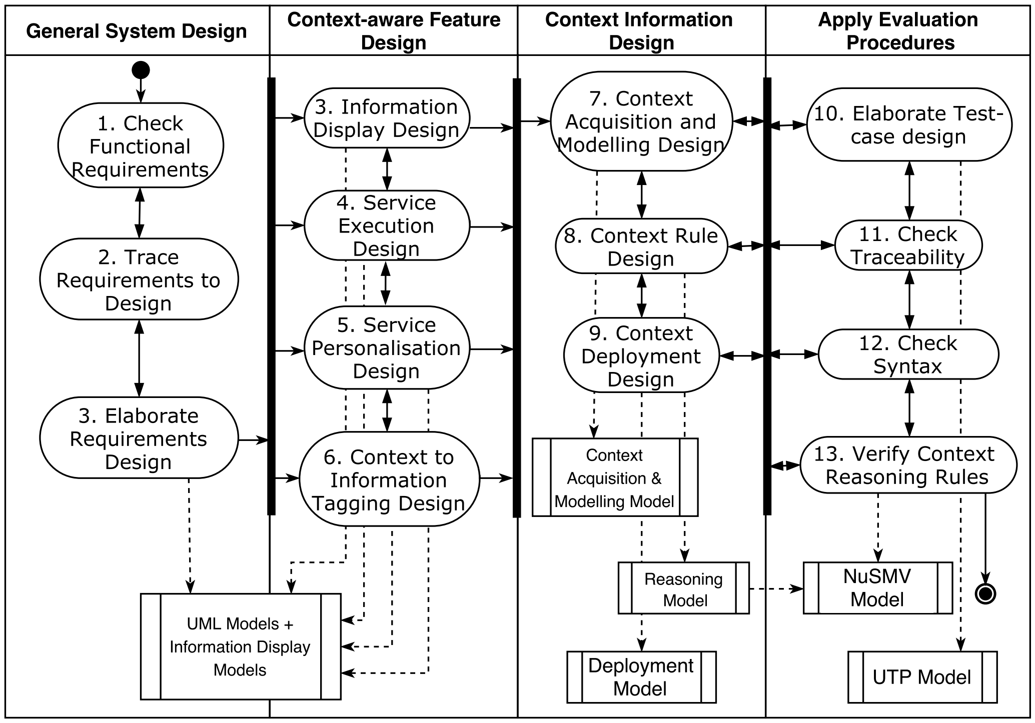
Chapters 4 and 5 present a framework to specifically guide the requirements elicitation of context-aware systems, taking into account the needs of the end-user stakeholders. This chapter introduces the Design for the Context-Aware Systems Engineering Framework (DC-ASEF), a complementary framework of RC-ASEF and SRC-ASEF for elaborating the identified requirements into the design of context-aware systems. In order to narrow the scope of the approach, the framework supports exclusively the most common type of reasoning in context-aware systems [1]: rule-based reasoning. In order to have a broader approach to the heterogeneous implementation of context-awareness, the approach supports the use of both mobile and stationary platforms. Particularly, the approach presented in this chapter is constrained to Android mobile devices and Java, supporting systems for stationary platforms that can plug-in Z-Wave radio protocol based sensors. It is also worth mentioning that during this chapter, UML specification based profiles are introduced, as the process model of the remainder of this work is constrained to the object-oriented paradigm. Note that the requirements methodology presented in Chapters 4 and 5 could also be extended to support the design of other types of reasoning, mobile platforms and radio protocols if required. The design framework has been created following the objectives explained in Section 1.4, and it supports:

- The guidance of developers with regard to the elaboration of requirements into design constructs, aiding the process with traditional UML Diagrams for this purpose.
  - Requirements can be traced to other elements of the design, including the test-case stereotyped elements, which are part of the SysML specification, facilitating more control over elements when adding, editing or removing them.
  - Test-case stereotyped elements can be further designed using the UML Testing Profile (U2TP) [127].
- The guidance of developers with regard to the elaboration of context-aware features into design constructs, enabling the design of each feature corresponding to its different modality, as conceptualised in Chapter 2. Depending on the

modality type, a set of UML diagrams is recommended, including a novel diagram presented as part of the design profile, the *Information Display Diagram*.

- The guidance of developers with regard to the elaboration of context-information into design constructs.
  - A domain-specific UML profile to facilitate the design of context information throughout its life-cycle. Including the design of the acquisition, modelling, reasoning, and dissemination stages of the information [1]. This includes a set of new diagrams: *Context Acquisition and Modelling Diagram*, *Reasoning Diagram* and *Context Deployment Diagram*.
  - The creation of more reliable context reasoning rules, by enabling their model-checking verification. Model-checking is a technique for automatically verifying correctness of a given model. The design models describing the context reasoning of the system are automatically translated into models of a formal model-checking tool that can be used to verify whether or not the rules comply with a set of properties specified by the developers. The specifics of this approach are further explained in Chapter 7.
- The following models are all supported by an open-source Modelio implementation which is available to the general public [200]. This tool includes support for: The UML profile introduced in this section, as well as traditional UML Diagram Representations, compatibility with other open-source and licensed modelling tools such as the UML testing profile, and compatibility with the previously introduced tool for requirements.

The remainder of the chapter is as follows. Figure 6.1 shows the main activities and subactivities of this specialised methodology. Section 6.2 introduces the framework activity related to the design of the non context-aware parts of the system, which traces the design to previous requirements models. Section 6.3 explains the framework activity related to the design the context-aware feature related design. Section 6.4 describes the activity consisting of designing the context-information with respect to its life-cycle [1] (acquisition, modelling, reasoning and dissemination). Then, Section 6.5 presents the activity to evaluate the satisfaction of the requirements after implementation, designing test-cases and verifying the design for inconsistencies. Section 6.6 summarises the chapter.



General System Design
Context-aware Feature Design
Context Information Design
Apply Evaluation Procedures
1. Check Functional Requirements
2. Trace Requirements to Design
3. Elaborate Requirements Design
3. Information Display Design
4. Service Execution Design
5. Service Personalisation Design
6. Context to Information Tagging Design
UML Models + Information Display Models
7. Context Acquisition and Modelling Design
8. Context Rule Design
9. Context Deployment Design
Context Acquisition & Modelling Model
Reasoning Model
Deployment Model
10. Elaborate Test-case design
11. Check Traceability
12. Check Syntax
13. Verify Context Reasoning Rules
NuSMV Model
UTP Model

**Figure 6.1: Core sub-activities for the design methodology.**

## 6.2 General system design

### 6.2.1 Check functional requirements

The first activity of the methodology is a nexus between the requirements elicitation and design stages. Since it is related to the requirements elicitation stage[1], it is recommended to apply the requirements elicitation process first, as it produces a set of functional requirements which can be used as input to inform the design decisions. During this sub-activity, the designers analyse the previously identified requirements to start creating the correspondent design.

[1]More information about the requirements eliciation stage can be found in Chapters 4 and 5.

| Requirement Category | Requirement Type | UML Diagram |
|---|---|---|
| Functional Suitability | All | Class Diagram and/or |
| | | Sequence/Communication Diagram |
| | | State Machine/Activity Diagram |
| Reliability | Fault tolerance | Component Diagram |
| | Recoverability | Sequence Diagram |
| Performance Efficiency | Time Behaviour | Timing/Sequence Diagram |
| | Resource Utilisation | Class Diagram or NA |
| | Capacity | Class Diagram or NA |
| Security | All | Class Diagram and/or |
| | | Sequence/Communication Diagram |
| | | State Machine/Activity Diagram |
| Compatibility | Coexistence | Component Diagram |
| | Interoperability | Class Diagram |
| Maintainability | Modularity | Class/Package Diagram |
| | Reusability | Class/Package Diagram |
| | Modifiability | Class/Package Diagram |
| Portability | Adaptability | Class Diagram |
| | Replaceability | Class/Package Diagram |
| Context-aware Feature | All | Class diagram and/or |
| | | Sequence/Communication Diagram |
| | | State Machine/Activity Diagram |
| | Information Presentation | Information Diagram |
| | Passive Service Execution | Information Diagram |
| | Active Service Configuration | |
| | Passive Service Configuration | |
| | Tag Information to Context | Information Diagram |

NA = Not Applicable    Info. = Information

**Table 6.1: Mapping between requirements types and UML Diagrams.**

### 6.2.2 Trace requirements to design and elaborate requirements design

Once the designers have a clear idea about what the functional requirements of the system are, the aim is to trace them to the system design. Table 6.1 can be used to aid such decisions, based on the ISO 25010 [189] classification of requirements types, and their relation with a set of UML diagrams that will be used to design those functional requirements. The table has been adopted from [191], and enhanced with the corresponding diagrams for the context-aware features.

During the next sub-activity, engineers elaborate the design of non context-aware related parts of the system, as per traditional software design.

## 6.3 Context-aware feature design

The context-aware feature design activity is divided into four main activities, focused on the design of the features for context-aware systems introduced in Section 2.4.2: Information display, service execution, service personalisation, and the tagging of context to information.

### 6.3.1 Information display design

For the information display design sub-activity, the context-aware features that consist of displaying information are designed. When designing the information that is going to be displayed with respect to a context-aware feature, the definition of context provided in Section 2.4.5 is critical, as it helps to distinguish between context and other information of the system. This definition considers context only as the information which is used to characterise a situation of interest. Therefore, although the information which is displayed in a context-aware feature can be reused, or even derive from a sensor, it does not necessarily have to be related to context-information. For facilitating the design of these types of services, a novel Diagram for Information handling is proposed, as it is illustrated in Figure 6.2.

The *message* stereotyped entities are the central unit in the *Information Display Diagrams*. Messages are formed of a string, and can additionally have information which is gathered from a sensor, but is not considered context (*SensorInformation*), or information from a context state (*ContextState*). The *msg* attribute in the message stereotype can reference *SensorInformation* or *ContextState* stereotyped elements. The message stereotype is intended for a stakeholder or a stakeholder profile, that will re-

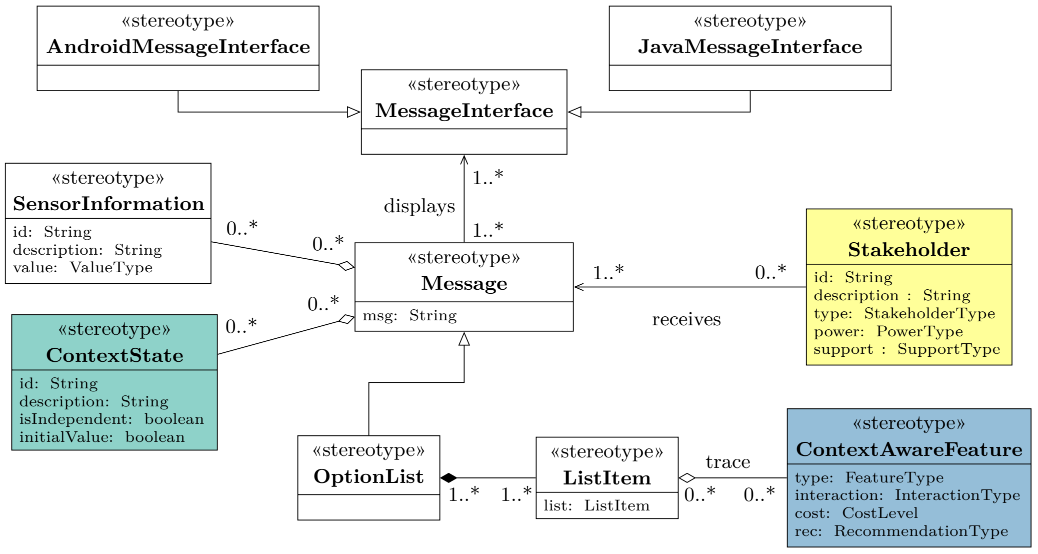
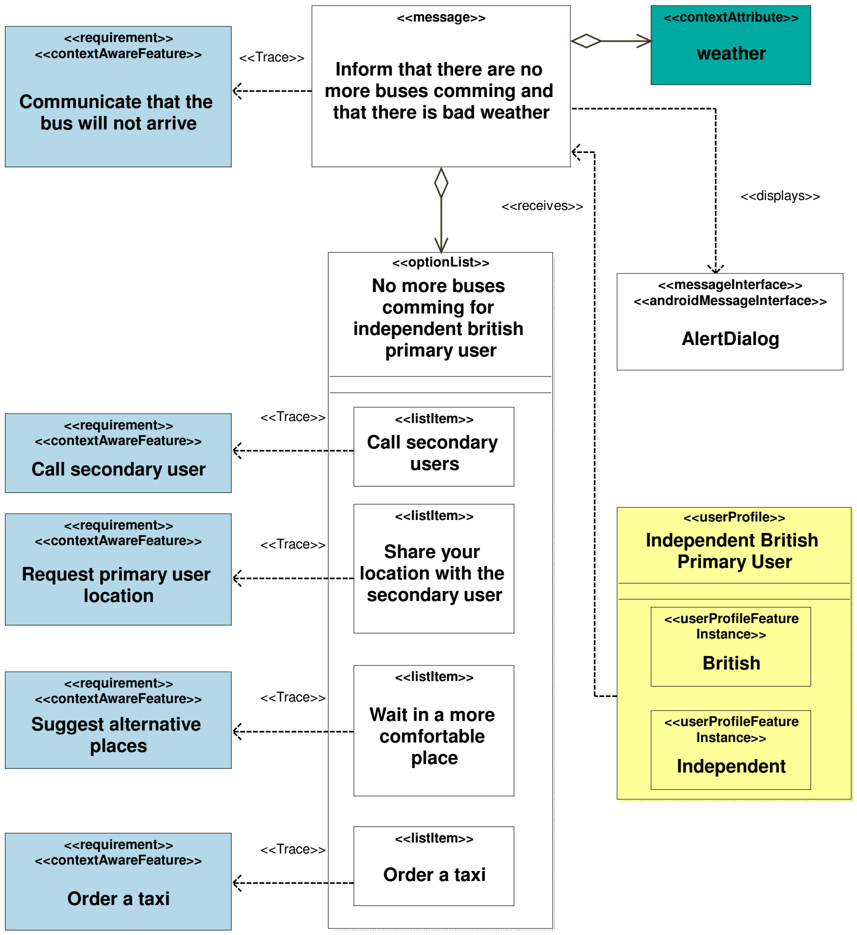


**Figure 6.2: Metamodel for the Information Display Diagram.**

ceive that message. Messages can be in the form of an option list, displaying the list of possible context-aware features to be triggered. Finally, messages are also displayed in a certain label or text-based widget within a user interface, which can be Android or Java based. For this, traditional Java syntax for Strings can be used.

The purpose at this stage is to create the messages that will be displayed to the end-user stakeholders. Each existing user profile might prompt a different message or service in the same situation of interest. For this, the Information Display Diagram helps developers to design how the different messages and services will be presented to end-user stakeholders. For instance, let us review the example introduced in Figure 5.6, Section 5.4.2, and particularly let us design the *Communicate that the bus will not arrive* context-aware feature. For spatial reasons, let the display of the context-aware feature be constrained to a user profile of Primary Users, particularly to independent British primary users. The different characteristics for user profiles introduced for this example can be found in Figure 4.8. Figure 6.3 represents the main message that the independent British Primary User will receive. Note that the attribute *msg* of the message stereotype is not represented in this figure. Let this *msg* attribute be: *The next bus you are waiting for is not going to arrive. The current weather is "+badWeather+", perhaps is better to order a taxi and wait for it somewhere else. Choose one of the following* *options:* . Note that "+*badWeather*+" references the context state that will represent the current value at that time. In the same way, *SensorInformation* or *ContextPreference* stereotyped elements can also be referenced in the same way. Similarly, the option list to be displayed for this message is also represented, and traced to the corresponding context-aware features. Finally, the message is associated to the Android *AlertDialog*[2], as this is the way in which this message will be represented.



<<requirement>> <<contextAwareFeature>> Communicate that the bus will not arrive
<<Trace>>
<<message>> Inform that there are no more buses comming and that there is bad weather
<<contextAttribute>> weather
<<receives>>
<<displays>>
<<optionList>> No more buses comming for independent british primary user
<<messageInterface>> <<androidMessageInterface>> AlertDialog
<<requirement>> <<contextAwareFeature>> Call secondary user
<<Trace>>
<<listItem>> Call secondary users
<<requirement>> <<contextAwareFeature>> Request primary user location
<<Trace>>
<<listItem>> Share your location with the secondary user
<<userProfile>> Independent British Primary User
<<userProfileFeature Instance>> British
<<userProfileFeature Instance>> Independent
<<requirement>> <<contextAwareFeature>> Suggest alternative places
<<Trace>>
<<listItem>> Wait in a more comfortable place
<<requirement>> <<contextAwareFeature>> Order a taxi
<<Trace>>
<<listItem>> Order a taxi

**Figure 6.3: Example of an Information Display Diagram for representing how messages are displayed for the context-aware feature introduced in Figure 5.6, Section 5.4.2.**

[2]https://developer.android.com/reference/android/app/AlertDialog

### 6.3.2 Service execution, personalisation and tagging context to information design

The sub-activity for service execution design is more centred on the design of services that are going to be triggered at execution time. As further explained in Section 2.4.2, services can be executed in a passive or active modality. The design of the active execution can be done using traditional UML Class, Sequence/Communication, State Machine/Activity diagrams. The design of the passive execution can also benefit from the Information Display Diagram, as it facilitates the way in which prompts or lists can be presented to the users.

For the service configuration and personalisation design sub-activity, there is a need to identify the different variables required for configuring or personalising the services. It does not really matter for which configuration modality (active or passive) these variables are intended for, as they are expected to change dynamically with the changing preferences or needs that end-user stakeholders may have after interacting with the system.

However, it is important to distinguish two types of service configuration, as this can help to distinguish what is context from what is not. The first type represents the configuration of those context-attributes that can be used for triggering services. For example, in a scenario where there is a threshold temperature for turning on the heating, one user can set it to 17°C and another user might prefer it to be triggered when having an ambient temperature of 14°C. Note that this type of service configuration design includes both the active and passive modalities introduced in Section 2.4.1. The remaining type of service configuration design consists of triggering the same service in different ways, according to the configuration values. Note that since these configuration variables are not used for identifying a situation of interest, but to configure the way in which the service is provided, the variables identified for the service adaptation can not be not considered as context. As it happens with the previous service configuration design type, this type can also be designed for active or passive modalities.

Once the variables are identified, the design has to be centred around the ways in which the information is going to be retrieved from the stakeholders, or is going to be modified by intelligent agents. For cases in which the stakeholders have to input this information, this sub-activity provides a means for designing user interfaces that can be personalised to display information according to the different needs and preferences of the users. For the design purposes of this sub-activity, traditional UML Class, Sequence/Communication, State Machine/Activity Diagrams can be used.

Finally, for tagging the context to information, the *Information Display Diagram* can be used, as well as traditional UML Class, Sequence/Communication, State Machine/Activity diagrams.

## 6.4 Context information design

### 6.4.1 Context acquisition & modelling design

The sub-activity is related to the implementation stage, which is further explained in Chapter 8. It consists of identifying the sources that will produce the context information, and how this information is going to be represented in the system. For this, a novel diagram for designing Context Acquisition & Modelling is introduced, as illustrated in Figures 6.4 and 6.5.

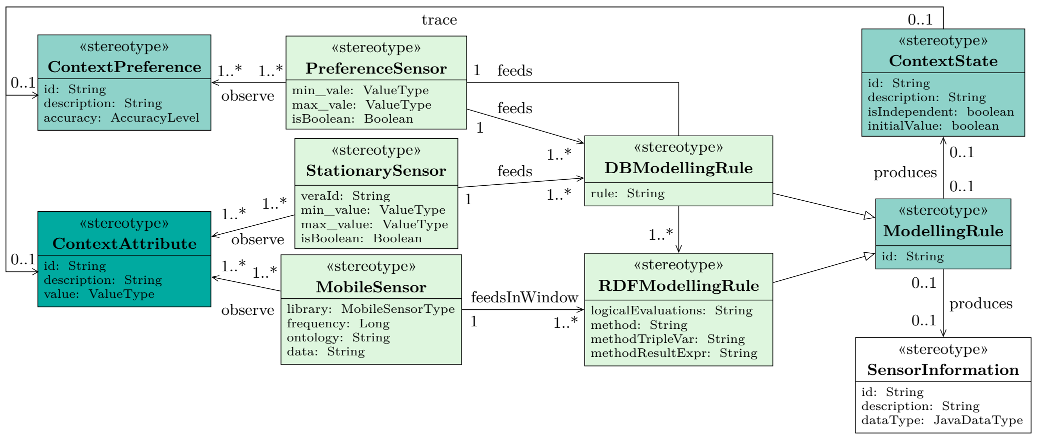


**Figure 6.4: Part I, Metamodel for the Context Acquisition and Modelling Diagram.**

First, the existing primary context-attributes or context-preferences are mapped to the sensors of the system so that each observable property is translated into a value that is tangible for the computer. The observable information can be measured using a great variety of sources, which are commonly known as Sensors. Nevertheless, the information that comes from sensors is typically raw, and needs to be processed and modelled in order to make it ready for its consumption by applications. The *Sensor* stereotype represents a sensor, and the *ModellingRule* stereotype a rule that makes the information available for its consumption. When the context information is considered ready for its consumption by a context-aware application, it is indicated by the use of the *ContextState* stereotype. For the purpose of these diagrams, that are bounded with the implementation of the system, the *ContextState* stereotype represents a Boolean variable, which is obtained by applying a modelling rule to a sensor.

In the proposed Context Acquisition and Modelling diagram, there exist two different ways of obtaining a context state, depending on if they are planned to be executed within a stationary or a mobile application. The modelling of rules consists of denoting different inferences which can be used for creating more meaningful atomic context information. For example, when considering remaining battery life in a mobile device, it might be more useful to know if the battery is low or not, rather than knowing that it is x% charged [3]. Modelling rules can contain a mix of simple or more complex logical evaluations, enabling the creation of rules on particular logical conditions such as “if battery level is greater than 25%”. This thesis acknowledges the context information life-cycle introduced in [1], and it adopts the Responsibility, Regularity and Sensor types classifications for acquiring data presented there, which are further explained in [1]. A modelling rule can not only create context states, but it can also create sensor information, which can represent information in any supported Java type. This information will not be considered as part of the context-model, but it can be used to complement the information displayed as part of certain context-aware features or system functionalities. As it can be observed in Figures 6.4 and 6.5, there are three types of stereotypes inherited from the *Sensor* stereotype: *StationarySensor*, *MobileSensor* and *PreferenceSensor*. The first two sensor stereotypes are related to the stationary and mobile platforms respectively. The preference sensor can be used in both platforms in order to represent the preferences of the stakeholders. The information obtained from this source can be introduced in the context model and be used in the context model.

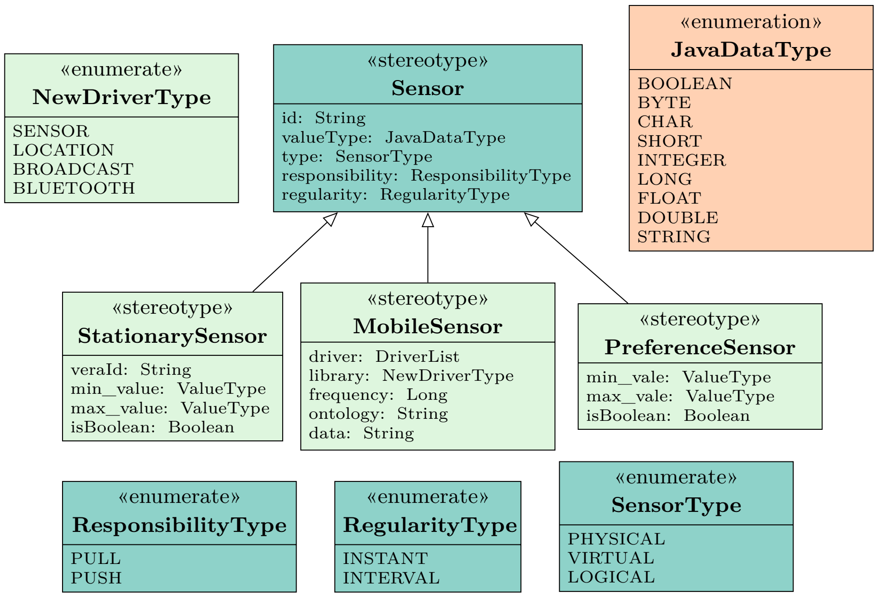


**Figure 6.5: Part II, Metamodel for the Context Acquisition and Modelling Diagram.**

#### 6.4.1.1 Modelling context for mobile platforms

This sub-section explains how the Context Acquisition and Modelling diagram, illustrated in Figure 6.4, facilitates modelling the system according to its future implementation on a mobile platform. Particularly, the following stereotypes are introduced for this purpose.

**MobileSensor:** Corresponds to those sensors that are intended to acquire information from mobile platforms once the system is implemented. The stereotype has been enhanced from the *ContextSource* stereotype in CoMo [3], therefore representing the capture of some sort of raw source data, expected to be received in RDF triples. Also, it adopts two properties from the CoMo modelling language. The first adopted property, *ontology*, defines the information source ontology string, in the form of a C-SPARQL[3] ontological prefix. The second adopted property describes the *data* string which relates to the RDF triples that MobileSensor provides. The variable names in the data are expressed using the question mark symbol ,?, as a prefix, in the same way

[3]More information about C-SPARQL is introduced in Section 3.4.2.2.

as in C-SPARQL. Since it inherits from the Sensor stereotype it also has its properties. As it is based on the Android Reasoner Library [166], it can either implement one of the existing drivers or facilitate the creation of a new one. The driver list includes: Weather based on location, battery, compass, current location, distance travelled, external storage space, GPS for indoors, light, device plugged in sensor, pressure, relative humidity, step counter, telephony, temperature, wifi, heart rate, and mood. For those drivers which are not covered by those in the list, there is an option to create new drivers based on the following libraries:

- *Location sensor:* Senses the location of the mobile device it will be deployed in;
- *Broadcast sensor:* Transmits information to all the listening Receivers;
- *Bluetooth sensor:* Senses information coming from Bluetooth wireless technology standard, in the form of low-power radio waves;
- It will be considered as simply a *sensor* for any other type.

For those cases in which the responsibility of the sensor is *pull*, the frequency in which the information is pulled can be determined by the *frequency* attribute, in form of a *Long Integer* value.

**RDFModellingRule:** The stereotype has been adopted from the *inferenceRule* stereotype in CoMo [3]. It supports in modelling raw data from sensors into data that can be used by applications, using logical expressions. For creating these logical expressions, the variables declared in the *data* attribute of the MobileSensor stereotypes are used, where only variables related with the *models* relationship can be used. In the case of multiple expressions, these are formed using regular expressions separated with a coma, that expresses the conjuncture of different expressions. The function usage is related to the functions that are built-in to C-SPARQL. The attributes *method, methodTripleVar*, *methodResultExpr,* and *logicalEvaluations* are used for this purpose. The *method* attribute of the stereotype allows the system to call more complex methods such as $COUNT()$, to count the number of times that a condition or variable is met. If the developers want a particular set of RDF triples being used with the functions, these can be stated in the *methodTripleVar* attribute. Lastly, the developer can also apply logical evaluations, expressed in the *logicalEvaluations* attribute, to the returned result of those functions, expressed in the *methodResultExpr* attribute.

**FeedsInWindow Relationship:** The stereotype has been adopted from the relationship stereotype in [3]. It helps to specify what raw data the inference rule queries over. This relationship enables the expression of temporal operators in regard to the raw RDF. The stereotype enables the expression of two different types of temporal operators. The first temporal operator is related to the data window size, that expresses how much RDF data can be included within the rule, and it is specified in the *For* attribute of the relation stereotype. As in C-SPARQL, the data window can be either *physical* (a given number of RDF triples) or *logical* (a triples occurring with a given time interval) [3]. The second temporal operator expresses the execution frequency, and it relates to how often the inference rule is run in the reasoner, as it is specified in the *Every* attribute of the relation stereotype. The aforementioned frequency is related to the possibility of RDF triples being present in more than a single logical window, as in the sliding of logical windows [201] in C-SPARQL.

**Example:** The example introduced in Section 5.4.3, shows the Reader how the Situation Detection Plan Diagram can help to evaluate the feasibility of implementing a situation of interest in which the Primary User was waiting for a bus that will not arrive, under bad weather conditions. But this is not the only purpose of the Situation Detection Plan Diagram. Each of the low-level context attributes and context preferences represented in the Situation Detection Plan Diagrams can be used for guiding the acquisition of information from the sensors of the system. Following the introduced example, Figure 6.6 illustrates how the context attribute *temperature* is modelled using the Context Acquisition and Modelling Diagram. As it is shown in this figure, the *temperatureSensor*, stereotyped as *mobileSensor*, *observes* the context-attribute *temperature*. Although it is not detailed in the figure, the *temperatureSensor* element has a set of attributes, which have been filled as follows. The data type of the sensor is Integer, as these are the values that will come from the open weather API. The sensor type is categorised as virtual, as the information about the temperature comes from an external server. The responsibility of this sensor has been set to *pull* as it will be this sensor in charge of getting the information from the weather API, and it will do so in intervals of 2000 milliseconds. The image shows the RDF ontology used and the two different variables used for the RDF query. This sensor will create a CSPARQL stream, that will feed information each four seconds in a window of ten seconds. As it can also be observed in Figure 6.6, the use of context-preferences is allowed for generating C-SPARQL queries. This diagram can automatically generate C-SPARQL queries. The particular query generated from the diagram shown in Figure 6.6 can be found in Figure 8.11. Finally, the *coldTemperature* context state is produced, which will be used as part of the context model.

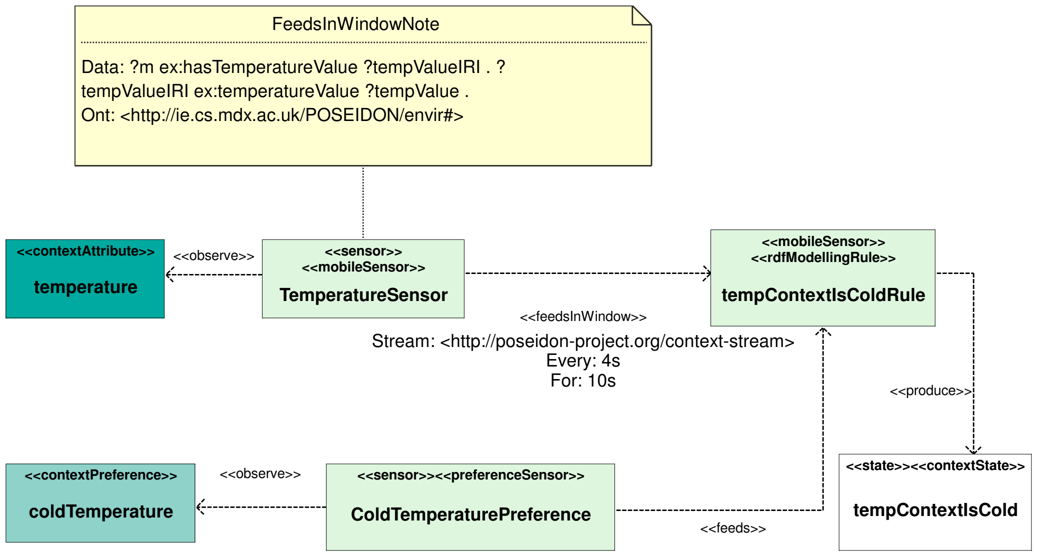


**Figure 6.6: Example of the usage of the Acquisition and Modelling Diagram for mobile platforms. Particularly related to the temperature context-attribute, introduced in the example from Section 5.4.3, Figure 5.7.**

#### 6.4.1.2 Modelling context for stationary platforms

This sub-section explains how the Context Acquisition and Modelling diagram, illustrated in Figure 6.4, also facilitates the context modelling for stationary platforms. Two related stereotypes have been introduced for this purpose. Note that the design of components for the stationary platform is closely related to the database of the M Reasoner, which has been enhanced from that presented in [162]. More information on the modifications introduced to this database can be found in Appendix A. The two stereotypes mentioned are:

**StationarySensor:** This stereotype corresponds to those sensors that are intended to acquire information from stationary platforms once the system is implemented. The following stereotype is used to declare the type of value for which the system will be feeding information to the database. Both variables are ValueType to enable the declaration of more complex data structures than those introduced by UML. Note that in this case, the attribute *veraId* reflects the Id by which the sensor is specified in the Vera Router of the system.

**DBModellingRule:** These rules help to store the data gathered from the sensors into a database. The name of the Stationary Sensor is used for this purpose, along with the minimum and maximum values supported. Any rule in SQL language[4] which can be applied to the value of the sensor is also added here.

**Example:** Section 1.5.3 introduces a kitchen scenario in a smart-house that it is conceived to foster the independence of the people with Down's syndrome at home. Let us assume that there has been a previous analysis in which the following situation of interest has been identified: *The cooker is unattended*. Also, that the proposed context-aware service is switching off the cooker when the cooker is unattended in order to avoid accidents. This situation can be detected by the computerised system using a presence sensor in the kitchen and a sensor for knowing that the cooker is on. The user can set, as a preference, the duration after which the cooker should switch off. Figure 6.7 illustrates the use of the Acquisition and Modelling Diagram for mobile platforms in this scenario. The *PresenceSensor* has a Boolean data type, as it only detects if there is presence or not. The sensor type is physical, and it has a push responsibility, with an instant regularity. The Vera id of the sensor is obtained after pairing the Z-Wave sensor to the Vera router, using its operating system. In this example it is assumed that the id of this sensor is 03. In this case, as it just represents boolean values, the minimum value will be false, and the maximum value will be true. The *rule* attribute of the database

[4]It currently supports MySQL and PostgreSQL query types.

modelling rule stereotype, uses the variable name "sensorValue" for enabling the use of standard Java operators against it. For this example, the value of the *rule* attribute is "sensorValue == true". In more complex examples, the rule attribute could be in the form of, for instance, "sensorValue >= 50". The *CookerSwitchSensor* has the same configuration as the *PreferenceSensor* except the Vera id, that it is assumed to be 04. Two main context states are created from this diagram, as part of the context model: *atKitchen* and *cookerOn*.

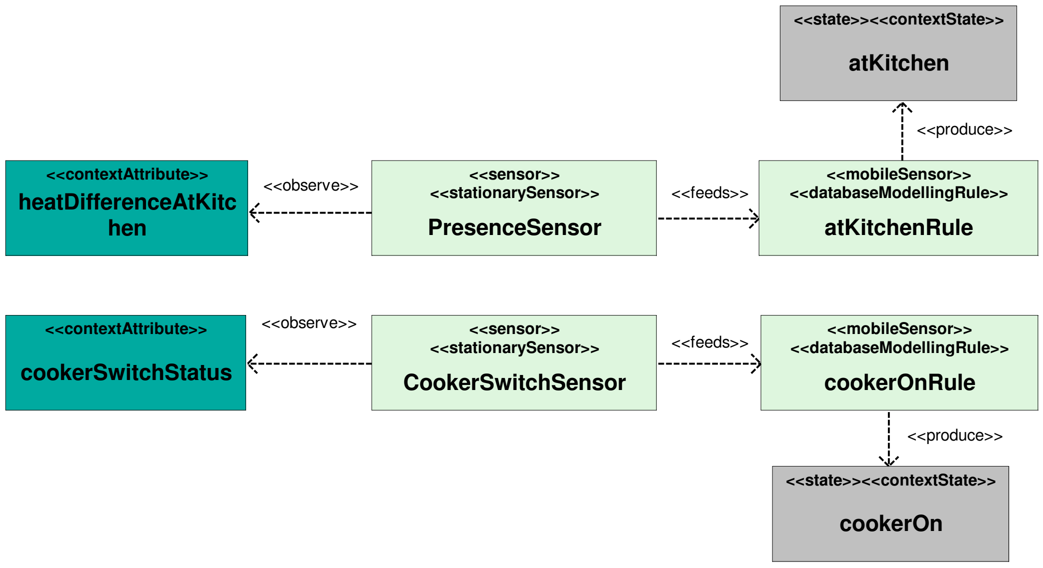


**Figure 6.7: Example of the usage of the Acquisition and Modelling Diagram for stationary platforms. Particularly related to the smart-kitchen scenario from Section 1.5.3.**

### 6.4.2 Context rule design

During the previous sub-activity the aim is to design the acquisition and modelling of the context information. In this sub-activity, particular instances with particular values of context states will be used to design reasoning rules. These rules can imply an alteration on the instance of a particular context state. Also, rules can produce new context states. Once the raw information is modelled into atomic context, it can be used to produce higher level context that can help to understand better the situation. This process is typically known as reasoning, where rule based reasoning is the most common approach for reasoning in context-aware systems [1]. During this sub-activity, those rules that will be used to make high-level context inferences will be designed. For this purpose, a novel diagram for designing Context Reasoning is introduced, whose metamodel can be observed in Figure 6.8. The context rules available in the models are inspired by those appearing in the M reasoning system [162]. Particularly, those stereotypes which can be found in Figure 6.9. The State stereotype can be a ContextState or an InternalTimeState. There are two different ways of creating a ContextState, by applying a modelling rule to raw sensor information (Acquisition and Modelling Diagram), or by applying a reasoning rule to an existing ContextState (Reasoning Diagram). InternalTimeStates can be in an instant or an interval, and can represent clock, week or calendar values. More information about the M system can be found in [162]. The following stereotypes of the metamodel are as follows:

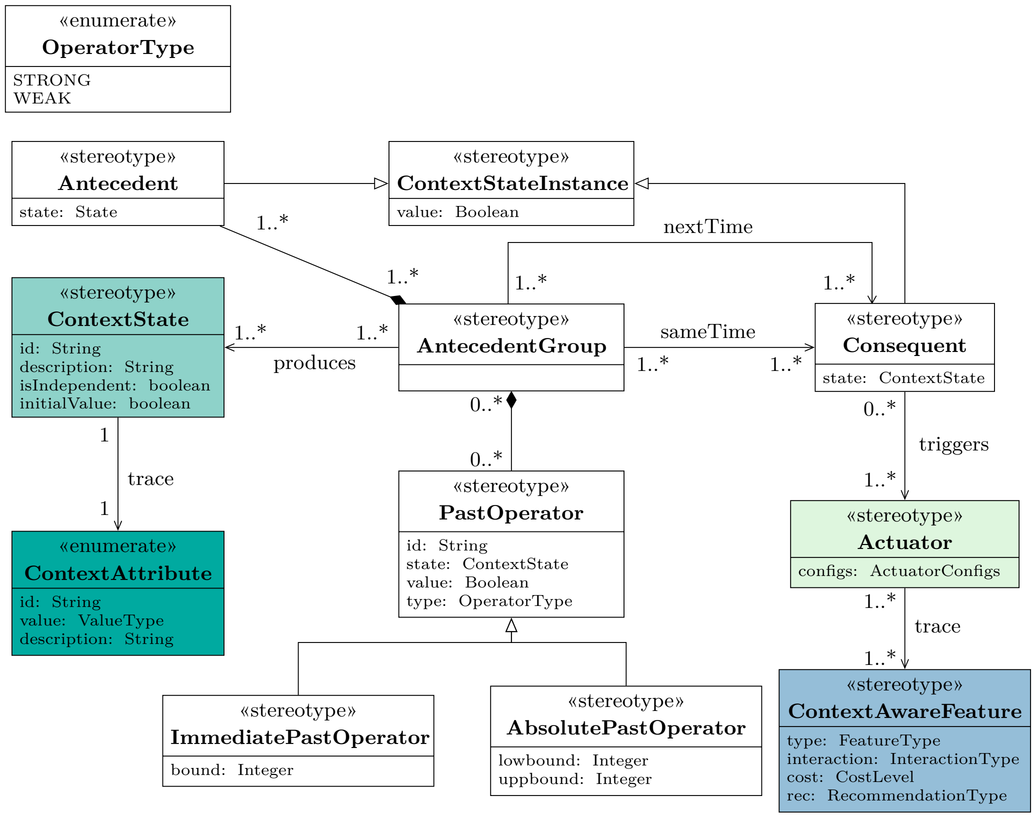


**Figure 6.8: Metamodel for the Reasoning Diagram.**

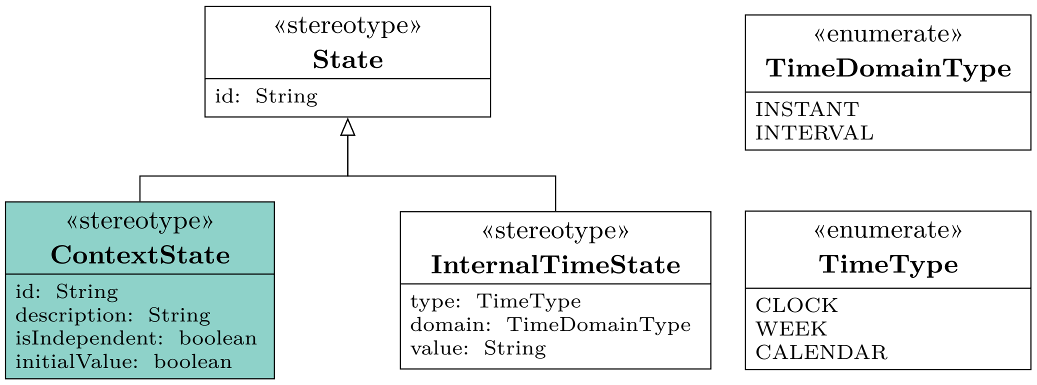


**Figure 6.9: Metamodel for the States of the model in the Reasoning Diagram.**

**ContextStateInstance:** As introduced in the previous sub-activity, *ContextStates* are Boolean variables generated by the application of a modelling rule to the outcome of the raw information produced by a sensor. This stereotype represents a particular Boolean value of a context state, and is the minimal unit which can be used for creating a rule. There are two types of *ContextStateInstance* stereotype types. The first type is the antecedent, which is used in the first part of a logical proposition, and it can represent the particular values of *ContextState* and *InternalTimeState*. The second type is the consequent, which is used in the second half of a logical proposition, and it can only represent values of the *ContextState* stereotype. The consequent will occur when the antecedent(s) of a proposition are met.

**AntecedentGroup:** It contains a set of antecedents that will be related between each other with the truth-functional operator of logical conjunction "and". Therefore, in order for the *AntecedentGroup* to be considered as True, all its contained Antecedents need to be true. An antecedent group can be used to create a *ContextState*, which can be related to a *ContextAttribute*.

**PastOperator:** As the rule models are related to the M reasoning system [162], they also enable the use of temporal operators to refer to what had happened at specific times in the immediate past or to specific times of the day. There are two types of PastOperator, which are ImmediatePastOperator and AbsolutePastOperator, as further explained in [162].

**Imply relationship:** This relationship can be of two different types, according to the types of rules introduced in the M reasoning system. These types are *same-time* for those rules whose consequent is applied on the same iteration, and *next-time* for those rules whose consequent is applied on the next iteration of the M Reasoner. More information about the same and next-time rules can be found in [162].

**Example:** Following the example from 6.4.1.2, Figure 6.10 illustrates the usage of the context reasoner. Context reasoning rules are modelled in the same way for mobile and stationary platforms. The immediate past operator *[-][120] ¬atKitchen* will be considered as true, only when the *atKitchen* context state has been false for 120 consecutive time units. The antecedent *cookerOn* will be considered as true, only when the context state cookerOn is true. When this immediate past operator and this antecedent are true at the same time, the *cookerUnattended* context state will be considered as true. The remaining three antecedent groups work in the same way.

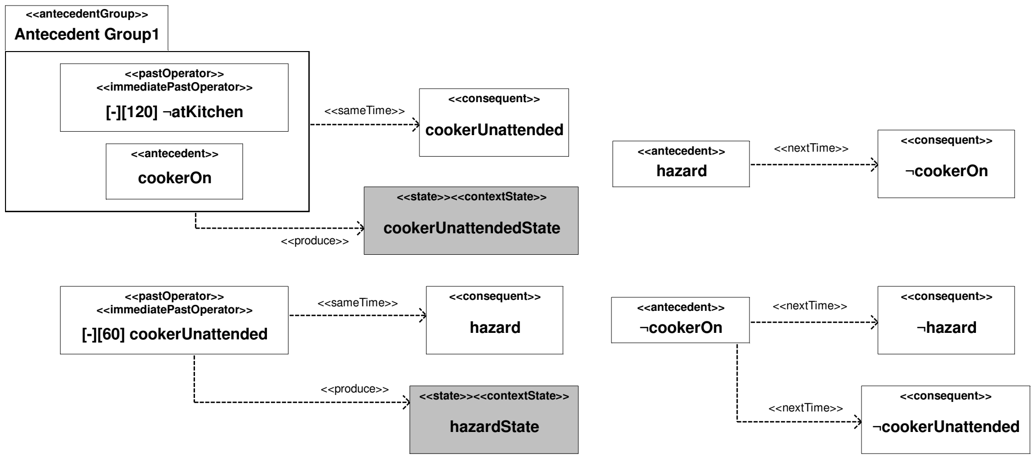


**Figure 6.10: Example of the usage of the Context Reasoning Diagram. Particularly related to the smart-kitchen scenario from Section 1.5.3.**

### 6.4.3 Context information deployment design

The previous sub-activities explain how to acquire, model, and reason over context information. During this sub-activity, the aim is to design how the context information is going to be disseminated, from the sensors, to the reasoning modules, to the different actuators that the system might have. For this purpose, a novel diagram for Context Deployment is introduced, as it is shown in Figure 6.11.

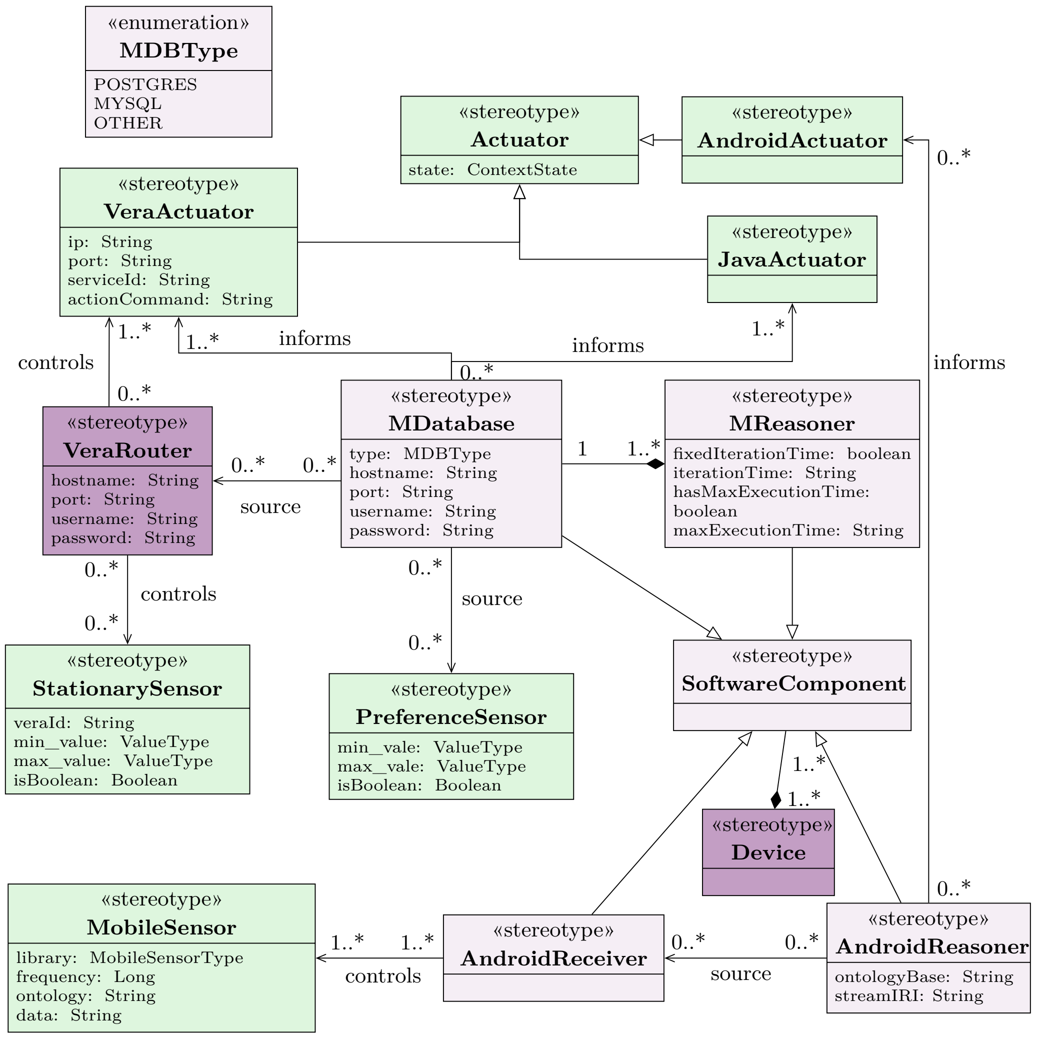


**Figure 6.11: Part of the metamodel for the Context Deployment Diagram.**

**Device:** The device stereotype facilitates the representation of different device types in which a software component can be executed. It can be used to allocate the different software components in a specific computerised system. Similarly, the Vera Router represents a special type of device, which is used as a common source to communicate with ZWave radio based sensors.

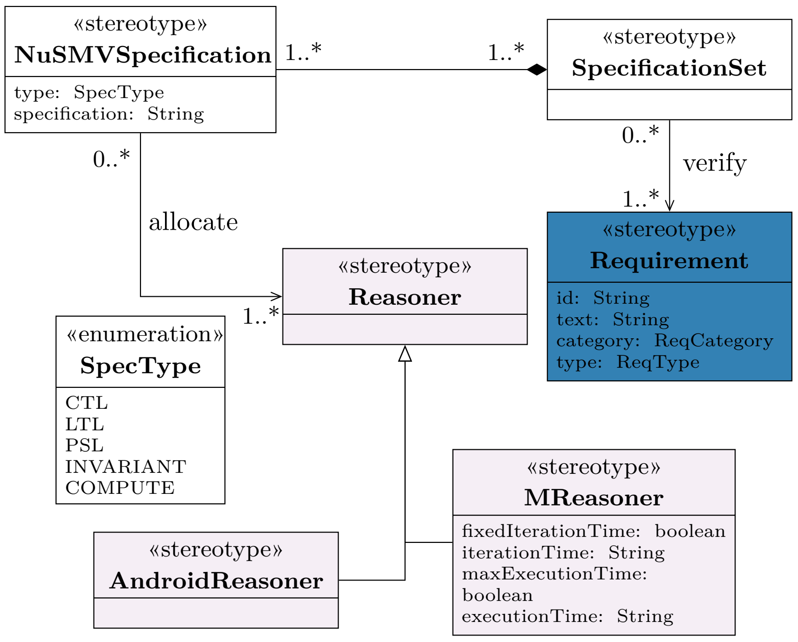


**Figure 6.12: Part of the metamodel for the Context Deployment Diagram.**

**SoftwareComponent:** A software component represents a unit of software that can be executed in a device. There are four main types of software components. The reasoning components are in charge of executing the reasoning rules. This can happen on a mobile platform (AndroidReasoner) or on a stationary platform (MReasoner). A single android reasoner can be used for different applications that run on the same mobile device. For each application, an android receiver is required, which will handle all the sensors corresponding to that application. On the other hand, the M reasoner requires a database, where different applications can update the latest status of different sensors. The database can be implemented as PostgreSQL or MySQL. If developers prefer to implement another database, they can also choose the OTHER option. Note that the selection of this last option for the database implementation implies that Developers will have to implement this database manually.

**Sensor:** Additional to the previously introduced sensors (ZWaveSensor and AndroidSensor), this diagram also introduces the PreferenceSensor, which can be used to facilitate the acquisition of user preferences that are treated as context.

**Actuator:** An actuator represents a software artefact that will be triggered when certain context values are reached, as designed in the Context Reasoning Diagram. There are three main types of actuators: AndroidActuator, VeraActuator, JavaActuator. The first type represents software artefacts intended to be executed on an android platform. The actuators of the second type are intended to trigger a change over a ZWaveSensor which is connected to a Vera Router. For this, the service id and the action command of the Vera Router are determined. Finally, the JavaActuator is intended to execute a functionality in Java code, for any platform that supports it.

**NuSMV Specification:** As it can be observed in Figure 6.12, this stereotype enables one to use a NuSMV specification to check the rules to be deployed in a certain reasoner. The NuSMV specifications can be grouped into specification sets. Specification sets can be traced to requirements using the SysML verify relationship. The developers can specify in a string any of the specifications, using the same format as specifications in [202].

**Example:** Following the previously introduced example from Section 6.4.1.2, the different sensors participating in the acquisition of information that is relative to the presence in the kitchen and the cooker being turned on are related to a Vera router. The *cookerSwitchSensor* element is stereotyped as both *stationarySensor* and *veraActuator*. This means that it can inform of when it is switched on and off, but it also can be used as an actuator, to remotely control the switch status. The veraActuator stereotype of the cookerSwitchSensor requires it to configure the serviceId attribute with the service id of the sensor in the Vera router. The Vera router is configured with its corresponding ip, hostname, and username and password for a user to connect via ssh. Both the database and the reasoner are executed in Laptop A. The database is a Postgres database, and its corresponding ip, port, hostname, and password are configured too. The M reasoner is configured for not having a fixed iteration time, and is set to a maximum execution time of 25775 time units. Finally, let the example assume an already existing requirement that forces the maximum execution time for the cooker to be no more than 61 time units. A specification is created in order to facilitate the developers knowing the maximum number of computational time units that the reasoning algorithm can take to execute.

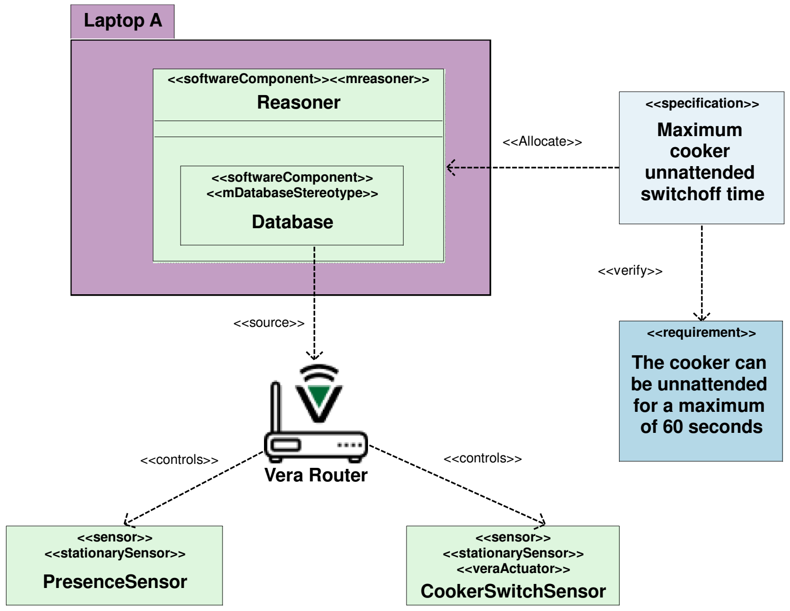


**Figure 6.13: Example of the usage of the Deployment Diagram for stationary platforms. Particularly related to the smart-kitchen scenario from Section 1.5.3.**

## 6.5 Apply evaluation procedure

### 6.5.1 Elaborate test-case design

The evaluation mechanisms for requirements introduced in Section 4.6.2, provide a means to help developers specify requirements that satisfy the objectives of the system. During the design stage, different mechanisms for evaluating the requirements after the implementation of the system are created. This activity also covers any test design to be applied in real scenarios aimed at acceptance testing, after the system is already implemented. The Requirements Diagram adopted the *TestCase* stereotype from SysML, that can also enable engineers to model and document this process. The *Verify* relationship stereotype from SysML connects a Requirement to a Test Case, indicating the process that will be used for this purpose. The test case, for example, can be further modelled as a state-machine diagram containing all the steps necessary for its verification. This process can include standard verification methods for inspection, analysis, demonstration or test [203]. Although the methodology introduced in this chapter is not attached to a particular mechanism to elaborate the test-cases design, the use of the UML Testing profile (UTP) [127] is recommended, as it is already supported by a Modelio module [154] [204].

### 6.5.2 Check traceability

Another prominent feature of the models introduced in this, and the previous chapters, is the ability of tracing the different elements of the design to those in requirements, aiding the decision-making process of the developers. The next sub-activity of the methodology consists of checking the traceability of requirements and context information throughout its life-cycle. For facilitating this purpose, a novel evaluation procedure is introduced, which helps the traceability evaluation of the system design:

1) For all the Requirements in the models, check their traceability using the reqTraceCheck function.
2) For all the situations of interest:
    - 2.1) For all DetectionPlans which detect a SituationOfInterest and are to be implemented (*DetectionPlan.toBeImplemented = true*):
        - 2.1.1) For all the ContextAttributes contained in the DetectionPlan:
            - I) If the ContextAttribute is Primary, for all the context attributes that have this primary ContextAttribute as a source, apply the *contextModellingTraceCheck* evaluation function:
            - II) While the ContextAttribute is not the source of any derive relationship, apply, for all the target ContextAttributes of derive relationships of the context reasoning traceability checking function: *contextReasoningTraceCheck*.
            - III) For all the ContextAttributes which are the target, and are not the source, of at least one derives relationship, apply the context-aware feature traceability checking function: *contextAwareFeatureTraceCheck*.
            - IV) Apply the secondary context state creation traceability check: *contextStateTraceCheck*.

3) Forall sensors in the system, apply the following traceability checking function:
   I) If the sensor is ZWaveSensor, apply the *stationaryDeploymentTraceCheck* function.
   II) If the sensor is PreferenceSensor, apply the *stationaryPreferenceDeploymentTraceCheck* function.
   III) If the sensor is AndroidSensor, apply the *mobileDeploymentTraceCheck* function.
4) Check that every software component is allocated into a device using the *softwareComponentTraceCheck* function.
5) If, in any of the presented traceability functions, the result is DISSATISFIED, the result of the final verdict is DISSATISFIED. If all the traceability functions applied give as a result SATISFIED, the result of the verdict is SATISFIED.

## 6.5.3 Check syntax

Certain elements of the design, use the syntax of an external tool. Namely:

- Message stereotype: The attribute *msg* of the message stereotype, uses a String which is formatted as a Java string.
- RDFModellingRule: The attributes of this stereotype use RDF formatted syntax.
- DBModellingRule: The rule attribute of this stereotype uses MySQL or PostgreSQL syntax, depending on what is indicated in the DBType, as it is shown in the Context Information Deployment diagram (Figure 6.11).
- NuSMVSpecficiation: It uses the NuSMV specification syntax for enabling the verification of context reasoning rules.

Each of these elements need to be checked to see if they comply with their corresponding syntax.

## 6.5.4 Verify context reasoning rules

A key feature of the framework is that it facilitates the verification of the context reasoning rules against properties specified by the developers. The model-checking subactivity of the methodology also includes the verification of context reasoning rules.

In order to check the design of the context reasoning, the approach is to automatically generate verifiable models from the existing Context Reasoning diagrams. The approach proposes to translate the M theory [162] components into NuSMV [202] components. The details of this translation are further explained in Section 7.3.

## 6.6 Conclusions

This chapter has presented a framework to guide and help to document the design stage of rule-based context-aware systems, taking into account the limitations in its creation, as it adopts the concepts introduced in Chapter 2. The framework supports, with models, the application of the methodology, and provides guidance for developers. It uses already existing models, as those presented in the UML and UTP standards. Additionally, a range of new diagrams have been proposed: Information Display, Context Acquisition and Modelling, Context Reasoning, and Context Deployment. As part of the software process framework presented in this thesis, significant effort has been dedicated to the production of an open-source tool to support the creation and management of the range of design diagrams that have been proposed. The set of tools presented in this chapter are compatible with the tool introduced in Chapter 4, as it is also a Modelio module, programmed in the Java language. Another important feature of the framework presented in this chapter is the ability to evaluate the design and trace it to the requirements. Since it is compatible with both the previously introduced Requirements Modelio module, and the UTP Modelio module [154], the tool enables the design of *model based testing*, tracing it to the requirements. Additionally, the theory that enables the verification of context reasoning rules has been integrated within the tool. The Context Reasoning Diagrams generated by the developers can be translated to NuSMV models, in order to verify their correctness [205]. Additionally, it also supports the evaluation of the rules introduced in Section 6.5.2. The main contributions of this chapter are:

- A guide for the requirements elicitation process from the identification of stakeholders, to the identification of Objectives and its operationalisation of goals, using a methodology which is based on existing approaches, and introducing novel diagrams for supporting the documentation and modelling of the process.

- A guide for designing each of the different context-aware features that the system can exhibit.
- A guide for designing the context information in each of the different stages of its life-cycle. From acquisition and modelling to reasoning and dissemination.
- Tool-supported mechanisms for evaluating the design aspects shown in Section 6.5.
- Implementation of an open-source tool which supports the Diagrams introduced in this section, as well as the theory for checking the models.

The theory for doing the verification of context-aware rules is further explained in Chapter 7. The resulting models obtained with the application of the proposed method contain reusable information that can be used for generating code that can facilitate the implementation of the context-aware system, as it is further explained in Chapter 8.

Chapter 7

# VC-ASEF: Verification for the context-aware systems engineering framework

## 7.1 Introduction

Chapter 6 has introduced DC-ASEF, a framework for designing rule-based context-aware systems in both stationary and mobile platforms. This chapter presents the Verification of Context-Aware Systems Engineering Framework (VC-ASEF), a specialisation of DC-ASEF that presents an approach for verifying context-aware rule specifications designed with the same framework. The approach presented in this chapter explains in more detail the verification sub-activity of the evaluation activity of DC-ASEF, introduced in Section 6.5.4. The framework has been created following the objectives explained in Section 1.4, and it supports:

- A proof-of-concept theory for automating the translations based on M language specifications [162] into NuSMV models.
- The implementation of the theory as an open-source Java library [205], which can be reused both in the DCase Modelio module [200], and in the M Integrated Developing Environment (IDE) [206].
- The automatic translation from DC-ASEF reasoning diagrams to the NuSMV model-checking tool models, enabling the verification of NuSMV formatted specifications which include Computation Tree Logic, Invariant, Linear Time Logic (LTL), real-time LTL, and probabilistic soft logic specifications [183].
- The automatic translation from DC-ASEF reasoning diagrams into M specifications [162], which can be automatically deployed using a version of the M IDE [206]. The IDE has been extended from that presented in [162], mavenised[1], refactored, and released as an open-source integrated development environment for M.
- The automatic translation from DC-ASEF reasoning diagrams into reasoning rules compatible with the Android Context Reasoning tool [167], which can be directly implemented into the library for reasoning in mobile platforms, which has been extended from its original as part of the contribution of this work.

The rest of this chapter is organised as follows. Section 7.2 explains the language M for context-aware reasoning, which can be coupled with a learning system. Section 7.3 explains how the M reasoning language can be mapped to the NuSMV model checker.

[1]https://maven.apache.org/

Section 7.4 introduces an illustrative example using a real-world scenario. Section 7.5 evaluates the performance of the model translation. Finally, Section 7.6 presents the summary of the chapter.

## 7.2 The M language

The building blocks of the M language are a set $S$ of atomic states, a set of rules $R$ and a set of events $E$. The left column in Table 7.2 shows the top-level description of an M specification, as used in the IDE.

### 7.2.1 States

The set of all atomic states $S$ is divided into three main subsets, such that: $S = S_I \cup S_D \cup S_T$. $S_I$ represents *independent states*, $S_D$ *dependent states* and $S_T$ *internal-time handling states*. An independent state does not depend causally on other states. $S_T$ states express a boolean value encoding whether or not the time of the system is *at* a specific *time-expression* ($S_{T_{At}}$), or *between* a *time expression* interval ($S_{T_{Btw}}$), where: $S_T = S_{T_{At}} \cup S_{T_{Btw}}$. The set of *time expressions* $T_{S_T}$ is partitioned into three subsets, such that: $T_{S_T} = T_C \cup T_D \cup T_W$. $T_C$ represents *clock* times of internal-time handling states expressed in *hh:mm:ss*[2] format. $T_w$ represents *week* times of internal-time handling states expressed as $Monday, Tuesday, \ldots, Saturday$ or $Sunday$. $T_D$ represents the *date* times of internal-time handling states, expressed as *dd/mm/yyyy*[3]. All possible internal time-handling state types can be represented as: $clockAt(\tau_c)$, $calendarAt(\tau_d)$, $clockBetween(\tau_c, \tau_c^{'})$, $weekDayBetween(\tau_w, \tau_w^{'})$, $weekDayAt(\tau_w)$, or $calendarBetween(\tau_d, \tau_d^{'})$, where $\tau_c < \tau_c^{'}$ and $\tau_c, \tau_c^{'} \in T_C$; $\tau_w < \tau_w^{'}$ and $\tau_w, \tau_w^{'} \in T_W$; $\tau_d < \tau_d^{'}$ and $\tau_d, \tau_d^{'} \in T_D$. Each state in $S$ can either be true or false, expressed with the notation $[\neg]s$. In this notation, $s$ is the state, $\neg$ is the boolean operator for negation, and the square brackets indicate optionality. The absence of the boolean operator indicates a state that is true. When specifying a model in the M IDE, all states but $S_T$ type states are declared as shown in Row 1 of Table 7.1. Independent states are declared as shown in Row 2 of the same table.

[2]Hours, minutes, and seconds respectively in the 24-hour time notation.

[3]Day of the month, the month and the year respectively.

### 7.2.2 Events, Rules and Past Operators

Events in the M model represent actions performed on the system by external entities, and require the explicit representation of time. In M, the time $t$ is represented as a discrete series of atomic instants, where $t \in \mathbb{N}$. $HoldsAt([\neg]s, t)$ means that the state $[\neg]s$ holds at an atomic time instant $t$, where $s \in S_I \cup S_D$. In the same fashion, an event occurring in the system is represented as follows: $Occurs([\neg]s, t)$, where $s \in S_I$. The definition[4] of the initial state for $S_I$ and $S_D$ states can be found in Row 3 of Table 7.1. The occurrence of an external event is defined in Row 4 of the same table.

The rule set $R$ is partitioned into two subsets, $R = R_S \cup R_N$, where $R_S$ refers to *same-time rules* and $R_N$ to *next-time rules*. Same-time rules apply their consequent in the same iteration in which the antecedents are satisfied. Next-time rules apply their consequent in the next iteration to that in which the antecedents are satisfied. A rule is specified as shown in Row 5 of Table 7.1. Note that, typically, a system has more than one rule, and that they are specified one after the other. In particular, $R_S$ rules are specified first, and need to be in stratification order, and $R_N$ rules are specified last. More information on stratification procedures can be found in [163]. Note that $R_S$ are modelled using 'ssr' in front of the rule, and $R_N$ using 'sEr'.

The set of all bounded past operators $BP_{Op}$ can be divided into two main groups: *immediate past* ($I$) and *absolute reference to the past* ($A$) where $BP_{Op} = I \cup A$. Both sets can be respectively divided into *strong* and *weak* expressions: $I = I_S \cup I_W$ and $A = A_S \cup A_W$. The syntax of $BP_{Op}$ is shown in Rows 7, 8, 9, and 10 of Table 7.1. Full explanation on the different $BP_{Op}$, and further examples of their usage can be found in [162].

## 7.3 Mapping M to NuSMV

The inner workings of a context-aware system can be very complex. Computerised systems have lots of data available from many types of resources, that span from hardware sensors (GPS, light, temperature, etc.) to software entities (Emails, social net-

[4]Note that more than one event can be defined one after the other.

| | Reference | Corresponding Code |
|---|---|---|
| 1 | $[STATE\ DECL.]$ | `'states(' s1',' s2',' ...',' sn ');'` |
| 2 | $[IND.\ STATE\ DECL.]$ | `'is(' [SIGN] s1',' [SIGN] s2',' ...',' [SIGN] sn ');'` |
| 3 | $[STATE\ INIT.]$ | `'holdsAt(' [SIGN] s',0);'` |
| 4 | $[EVENTS]$ | `'occurs(' [SIGN] s',' t ');'` |
| 5 | $[RULES]$ | `'ssr'\|'sEr' '(' [EXPR] '^'... '^' [EXPR] '->' [SIGN] sc ');'` |
| 6 | $[EXPR]$ | `([SIP] \| [WIP] \| [SAP] \| [WAP]) \| ( [SIGN] s )` |
| 7 | $[SIP]$ | `'[-] [' mu ']' [SIGN] s` |
| 8 | $[WIP]$ | `'<-> [' mu ']' [SIGN] s` |
| 9 | $[SAP]$ | `'[-] [' alpha',' beta ']' [SIGN] s` |
| 10 | $[WAP]$ | `'<-> [' alpha',' beta ']' [SIGN] s` |
| 11 | $[SIGN]$ | `'#'\|' '` |

**Table 7.1: Representation of the corresponding code to the references expressed in the left column of Table 7.2. Where $s1$, $s2$, $sn$ and $sc$ are example states, and $mu$, $alpha$ and $beta$ are example time expressions.**

```
[STATE DECLARATION]
[INDEPENDENT STATE DECLARATION]
[STATE INITIALISATION]
[EVENTS]
[RULES]
```

```
'MODULE main'
'VAR'
'time : 0 ..' [MAXIMUM ITERATION]';'
[STATE DECLARATION]
[AUXILIARY STATE DECLARATION]
[BPOP DECLARATION]
'ASSIGN'
'init(time) := 0;'
[STATE INITIALISATION]
[AUXILIARY STATE INITIALISATION]
[SAME TIME RULES]
[NEXT TIME RULES]
'next(time) := case'
'(time <' [MAXIMUM ITERATION]') : time + 1;'
'TRUE :' [MAXIMUM ITERATION]';'
'esac;'
[PROPERTY SPECIFICATIONS]
```

**Table 7.2: Left column: Syntax for a model specified in M, denoted as $M[MODEL]$; Right column: Syntax for a specification model in NuSMV, denoted as $NuSMV([MODEL])$. Where $M[MODEL]$ is bisimilar to $NuSMV([MODEL])$. Red text between simple quotes is mandatory. Text between square brackets references code shown in Tables 7.1 and 7.3. Note that in order to declare independent states in NuSMV it suffices with not initialising a state. The *[MAXIMUM ITERATION]* value needs to be manually specified.**

```
MODULE strong_immediate_past(state,bound)
VAR
  counter : 0 .. bound;
  live   : boolean;
ASSIGN
  init(counter) := 0;
  live :=   case
      (counter = bound): TRUE;
      TRUE: FALSE;
      esac;
  next(counter) :=  case
      (state=TRUE & counter < bound) : counter+1;
      (state=TRUE & counter = bound) : bound;
            TRUE: 0;
                   esac;
```

Listing 7.1: Syntax for the Strong Immediate Past Operator module in NuSMV language.

```
MODULE strong_absolute_past(state, low_bound,
  upp_bound, t)
VAR
  veredict      : boolean;
  veredict_aux  : boolean;
  live      : boolean;
ASSIGN
  init(veredict_aux) := TRUE;
  init(live) := FALSE;
  veredict := case
        ((state=FALSE) & (t >= low_bound)
          & ( t <= upp_bound))  : FALSE;
        TRUE: veredict_aux;
        esac;
  next(veredict_aux) := veredict;
  next(live) :=   case
            (t >= upp_bound) : veredict;
                      TRUE: FALSE;
                   esac;
```

Listing 7.2: Syntax for Strong Absolute Past Operator module in NuSMV language.

```
MODULE weak_immediate_past(state,bound)
VAR
  counter  : 0..bound;
  live     : boolean;
  live_aux : boolean;
ASSIGN
  init(counter) := 0;
  init(live_aux) := FALSE;
  live := case
```

```
        (state=TRUE)  : TRUE;
        (state=FALSE) & (counter = bound) : FALSE;
        TRUE: live_aux;
        esac;
  next(live_aux) := live;
  next(counter) :=  case
        (state = TRUE) : 0;
        (live_aux=TRUE) &
          (counter < bound) : counter + 1;
                TRUE: 0;
          esac;
```

Listing 7.3: Syntax for Weak Immediate Past Operator module in NuSMV language.

```
MODULE weak_absolute_past(state, low_bound,
upp_bound, t)
VAR
  veredict      : boolean;
  veredict_aux  : boolean;
  live      : boolean;
ASSIGN
  init(veredict_aux) := FALSE;
  init(live) := FALSE;
  veredict := case
        (state=TRUE) & (t >= low_bound)
          & ( t <= upp_bound)  : TRUE;
        TRUE: veredict_aux;
        esac;
  next(veredict_aux) := veredict;
  next(live) :=   case
            (t >= upp_bound) : veredict;
                      TRUE: FALSE;
                   esac;
```

Listing 7.4: Syntax for Weak Absolute Past Operator module in NuSMV language.

works, calls, etc.). The more complex the situations that the system needs to handle and the more actions it needs to trigger, the more complicated it will be to ensure that it will behave adequately and that it will not exhibit any undesired behaviour. One of the prominent features of the framework put forward by this thesis is that it enables the model checking of the reasoning rules created for managing the context-aware features to be triggered. This is especially important in those cases where the system is also learning from the behaviours of the users, to semi-automatically update context-aware rules in order to provide more useful services [45, 88, 207]. The following section focuses on automating the validation of the logical language M [162], by allowing the indirect verification of a system modelled in M against some formalized requirements. The approach consists of a model to model translation, which transforms M models to NuSMV models [202].

The right column in Table 7.2 shows the NuSMV model template resulting from the M to NuSMV model transformation. The resulting model can require up to four additional modules, which correspond to each of the bounded past operators. If a certain bounded past operator is not used in the specific model, then the corresponding module for that bounded past operator can be disregarded. The following subsections explain how the particular elements of the left column in Table 7.2 are mapped to NuSMV elements.

### 7.3.1 Time

As it can be observed when comparing columns in Table 7.2, even if the declaration of time is not explicitly required in M models, the translations to NuSMV demand a discrete clock for the algorithm iterations and to handle the M bounded past operators. Let this clock be $\Gamma$. The bound for the time variable in NuSMV can be computed as follows: let $T$ represent the set of all the time expressions used in the rules of a particular model, which can be in form of bounded past operators or internal-time handling states: $T = T_{BP_{Op}} \cup T_{S_T}.T_{S_T}$ is the set of time expression[5] states $S_T$. $T_{BP_{Op}}$ is the set of time expression for $BP_{Op}$, which can be divided into three subsets, such that: $T_{BP_{Op}} = T_\mu \cup T_\alpha \cup T_\beta$. $T_\mu$ is the set of all time bounds in the immediate bounded past operators. $T_\alpha$ is the set of all lower time bounds in the absolute bounded past operators. $T_\beta$ is the set of all upper time bounds in the absolute bounded past operators[6].

The NuSMV models can accommodate the requirement of having a discrete clock, as NuSMV provides an integer structure which supports $\Gamma \in \mathbb{N}$. Taking into account the NuSMV implementation constraints for the integer type [183], and that the negative values of the integer structure will not be used, the maximum iteration variable can only be a number from zero to $\Gamma_{max} = 2^{31} - 1$. Since time is represented in seconds, this would give the developer the representation power of almost seven decades, which is enough for the intended verification purposes.

[5] $S_T$ internal time-handling states are explained in Section 7.2.1.

[6] Time bounds are further explained in [162].

Notice that in order to have models where the consequent of the highest *stratification level*[7] is reachable, there is also a recommended minimum iteration time, $\Gamma_{min}$, which can be calculated with the following formula[8]:

$$\Gamma_{min} = (\lambda_{max} + 1) + \sum_{i=\phi}^{\omega} (\delta_{max} + 1)$$

$\Gamma_{min}$ lets the model checker have enough iterations for reaching the path where the highest stratification level rule consequents can be met. $R_N$ rules require additional iterations for considering the application of the antecedents, and $R_N$ rules require an additional iteration because past operators are non-reflexive. An example can be found at the end of Section 7.4.1. The maximum iteration, referenced in rows 3, 14, and 15 of the right column in Table 7.2, corresponds to an integer number between $\Gamma_{min}$ and $\Gamma_{max}$. This value specifies the maximum iteration to which the model will be checked.

### 7.3.2 State and Event Declaration

Row 1 in Table 7.3 shows how a state is declared in NuSMV. Note that states are declared one after the other, and that there is no need to declare states in $S_T$. In order to maintain the persistence of states in each iteration, an additional set of auxiliary states are required by the NuSMV model, as it is shown in row 2 from Table 7.3. For each $s \in S_C$ an auxiliary boolean variable needs to be declared, where $S_C$ is the set of all consequents in all the same-time rules of a given model. Each bounded past operator is implemented referencing a different NuSMV template, depending on its operator type. The declaration corresponds to row 3 in Table 7.3. Independent states are not required to be initialised in the NuSMV model, as this lets the model checking engine check for all the different combinations of their values. Note that the dependent states which are in the consequent of a next-time rule have to be initialised, as shown in row 8 from Table 7.3. Each state $s \in S_C$ requires an auxiliary variable, which is declared as

[7]The stratification level is part of the forward reasoning algorithm and it is further explained in [162, 163].

[8]Where $\lambda_{max}$ is the highest time expression from all the rules in $R_N$, $\omega$ is the maximum stratification level stage from the rule set $R_S$, $\phi$ is the stratification level of the rule in the rule set $R_S$, which has the highest time expression. For comparing values in $T_C$, $T_D$ and $T_W$, these will be translated to seconds, as indicated in Section 7.3.3; $\delta_{max}$ is the highest time expression from a set $\Delta_i$, where $\delta_{max} \in T$. $\Delta_i$ is the set of all same-time rules from a particular stratification stage level $i$.

shown in row 9 from the same table. Given that the focus of the chapter is on the verification of models, the translation of external events into NuSMV is not considered, and therefore, there is no mapping for external events.

| | Reference | Corresponding Code |
|---|---|---|
| 1 | *[STATE DECL.]* | `s':= boolean;'` |
| 2 | *[AUX. STATE DECL.]* | `s'_aux := boolean;'` |
| 3 | *[BOP DECL.]* | `[SIP DECL.] \| [WIP DECL.] \|` `[SAP DECL.] \| [WAP DECL.]` |
| 4 | *[SIP DECL.]* | `s'_sip_'mu': strong_immediate_past('` `[SIGN] s',' mu');'` |
| 5 | *[WIP DECL.]* | `s'_wip_'mu': weak_immediate_past('` `[SIGN] s',' mu');'` |
| 6 | *[SAP DECL.]* | `s'_sap_'alpha'_'beta': strong_absolute_past('` `[SIGN] s',' alpha',' beta',' time');'` |
| 7 | *[WAP DECL.]* | `s'_wap_'alpha'_'beta': weak_absolute_past('` `[SIGN] s',' alpha',' beta',' time ');'` |
| 8 | *[STATE INIT.]* | `'init(' s ') :=' [VALUE]';'` |
| 9 | *[AUX. STATE INIT.]* | `'init(' s'_aux' ') :=' [VALUE] ';'` |
| 10 | *[SAME TIME RULE]* | `sc ':= case' [RULE EXPRESSION] ...` `[RULE EXPRESSION] 'TRUE :' sc'_aux;' 'esac;'` `'next('sc'_aux):=' sc';'` |
| 11 | *[NEXT TIME RULE]* | `'next(' sc ') := case' [RULE EXPRESSION] ...` `[RULE EXPRESSION] 'TRUE : 'sc';' 'esac;'` |
| 12 | *[RULE EXPR.]* | `'(('[EXPRESSION]')&('...')&('[EXPRESSION]')):` `'[VALUE]';'` |
| 13 | *[EXPRESSION]* | `([BPOP] '=' [SIGN]) \| (s '=' [SIGN]) \| [TIME]` |
| 14 | *[BPOP]* | `s'_sip_'mu' \| s'_wip_'mu \|` `s'_sap_'alpha'_'beta' \| s'_wap_'alpha'_'beta` |
| 15 | *[SIGN]* | `'!'\|' '` |
| 16 | *[VALUE]* | `'TRUE'\|'FALSE'` |

**Table 7.3: Representation of the corresponding code to the references expressed in the left column of Table 7.2. Where $s1$, $s2$, $sn$ and $sc$ are example states; $mu$, $alpha$ and $beta$ are example time expressions; and $time$ is the iteration number.**

### 7.3.3 Rules

Same time rules are translated as shown in row 10 of Table 7.3, while next-time rules are translated as shown in row 11 of the same table. Time, referenced in row 14 of Table 7.3, corresponds to $S_T$ states, which are modelled as follows: For $\tau_c \in T_C$ the values are translated directly into seconds. For $\tau_w \in T_W$ each day of the week is assigned a number according to its position[9]. This number is multiplied for the number of seconds in a day. For $\tau_d \in T_D$ the current calendar date in seconds is subtracted for the specified date, in seconds. Notice that this happens at model translation time. All the same-time rules in the system need to be declared in stratification order. In NuSMV, rules affecting the same consequent are modelled as different rule expressions inside the same rule, as it is shown in rows 10 and 11 from Table 7.3. This means that rules in different stratification levels, which imply a change in the same state of their consequent, are not supported in these M to NuSMV model translations. To avoid such cases, next-time rules can be used as illustrated in rows 34 and 35 from Listing 7.5. Note that in such cases, the corresponding code for line 3, row 10 in Table 7.3 and the modelling of initialisation events will be omitted, and the state $s_c$ in line 1, row 11, from Table 7.3 will have the suffix '$\_aux$' attached.

### 7.3.4 Property Specifications

The specification languages supported by NuSMV are Computation Tree Logic (CTL), Linear Temporal Logic (LTL), Real-time Computation Tree Logic (RT-CTL) and Probabilistic Soft Logic. The NuSMV engine also allows the checking of the model for deadlocks. The Reader is referred to [183] for additional details. This chapter focuses on CTL; some examples of how property specifications are used are reported in Section 7.4.1. This section provides the sketch of proof of the main theoretical result of the chapter, in which the notation $NuSVM[*]$ is used, where $*$ is an element of M, to denote the NuSVM translation of the element according to the rules presented in Table 7.2 and Table 7.3.

**Theorem 1. .** *Given a model $MODEL$ specified in M (denoted with $M[MODEL]$) and a CTL formula $\phi$, the formula $\phi$ is true in $M[MODEL]$ (denoted with $M[MOD$*

[9] Monday = 1, Tuesday = 2, ..., Saturday = 6, Sunday = 7

*$EL] \models \phi$) if and only if $\phi$ is true in the corresponding NuSMV model, denoted with $NuSMV[MODEL] \models \phi$.*

*Proof.* This section shows that $NuSMV[MODEL]$ is bisimilar to $M[MODEL]$. In particular, it illustrates that, for every pair of states $s$ in $M[MODEL]$ and $NuSMV[s]$ in $NuSMV[MODEL]$,

1. If there exists a rule $R_{s,t}$ such that $R_{s,t}(s) = t$, then there is a NuSMV state $NuSMV[t]$ such that $NuSMV[R_{s,t}](NuSMV[s]) = NuSMV[t]$.
2. If there is a NuSVM temporal transition $(NuSMV[s], t')$, then there is a rule $R_{s,t}$ in M such that $R_{s,t}(s) = t$ and $t' = NuSMV[t]$;

For (1): the rule translation defined in Section 7.3.3 uniquely identifies a NuSVM temporal transition $NuSMV[R_{s,t}]$ that, in turn, results in a new reachable NuSMV state $NuSMV[R_{s,t}](NuSMV[s]) = NuSMV[t]$. For (2): assume that there is a temporal transition $TT = (NuSMV[s], t')$ in $NuSMV[MODEL]$ that associates the NuSMV state $(NuSMV[s]$ to a new state $t'$. The only transitions enabled in $NuSVM[MODEL]$ are those obtained following the translation rules in Section 7.3.3; hence, there must be a rule in M $R_{TT}$ such that $NuSVM[R_{TT}] = TT$. Similarly, as all the reachable states are reachable through a NuSMV temporal transition corresponding to a rule in M, and given that rules are deterministic: $R_{TT}(s') = t$, $NuSMV[t] = t'$, and $NuSMV[R_{TT}]\ (NuSMV(s)) = t'$. Table 7.2 refers to the mapping of all the supporting functions, in particular for the constraints imposed on $NuSMV[R_{s,t}]$ by the bounded-past operators. The claim follows from the fact that bisimilar transition systems satisfy the same CTL formulas. □

## 7.4 Usage illustration

This section is particularly focused on illustrating the case study inspired by a real world scenario to show the potential of the approach with data science in order to provide useful insights. In these types of systems, information from the users can be gathered in an unobtrusive and transparent way. Certain scenarios in which these types of systems provide services are likely to operate in situations in which an inappropriate decision can cause harm to the users or even put their life at risk. Also, a

```
states(cookerOn, atKitchen, cabinet, kettleOn, waterTapOn, cookerUnattended, hazard,
pattern_0, pattern_1, pattern_2, pattern_3, pattern_4, pattern_5, time_context);

is(cookerOn); is(#cookerOn); is(cabinet); is(#cabinet);
is(atKitchen); is(#atKitchen); is(waterTapOn); is(#waterTapOn);

holdsAt(#cookerOn,0); holdsAt(#atKitchen,0); holdsAt(#cabinet,0);
holdsAt(#kettleOn,0); holdsAt(#cookerUnattended,0); holdsAt(#hazard,0);
holdsAt(#pattern_0,0); holdsAt(#pattern_1,0); holdsAt(#pattern_2,0);
holdsAt(#pattern_3,0); holdsAt(#pattern_4,0); holdsAt(#pattern_5,0);
holdsAt(#time_context,0);

//Stratification Level k = 1 Rules
ssr((clockBetween(06:18:06-07:03:10))  -> time_context);
ssr((#clockBetween(06:18:06-07:03:10)) -> #time_context);
ssr(([-][120]#atKitchen ^ cookerOn)     -> cookerUnattended);

//Stratification Level k = 2 Rules
ssr((#atKitchen ^ time_context) -> pattern_0);
ssr(([-][60]cookerUnattended)   -> hazard);

//Stratification Level k = 3 Rules
ssr((cabinet ^ [-][12]pattern_0 ^ time_context)  -> pattern_1);

//Stratification Level k = 4 Rules
ssr((#cabinet ^ [-][22]pattern_1 ^ time_context) -> pattern_2);

//Stratification Level k = 5 Rules
ssr((waterTapOn ^ [-][18]pattern_2 ^ time_context)  -> pattern_3);

//Stratification Level k = 6 Rules
ssr((#waterTapOn ^ [-][9]pattern_3 ^ time_context) -> pattern_4);

//Stratification Level k = 7 Rules
ssr((cookerOn ^ [-][11]pattern_4 ^ time_context) -> pattern_5);

//Stratification Level k = 8 Rules
ssr((atKitchen ^ [-][123]pattern_5 ^ time_context) -> kettleOn);

//Stratification Level k = 9 Rules
ssr(( [-][120]kettleOn ) -> #kettleOn);

//Next-Time Rules
sEr((hazard) -> #cookerOn);
sEr((#cookerOn) -> #hazard);
sEr((#cookerOn) -> #cookerUnattended);
```

Listing 7.5: Example of an M model for the cooker example.

tool which directly translates the outcome of behavioural patterns might foster bad habits instead of correcting them. In order to increase the reliability of the systems created with this IDE, this chapter introduces a theory for translating M specifications into models that can be used as an input of a model checking tool. In this way, systems which automatically evolve according to the preferences of the users can be built in a more reliable way. By understanding the information obtained through sensors, context-aware systems can learn how to adapt their services to the preferences, needs and limitations of the users. In order to illustrate these synergies, let the scenario review the example presented in Sections 1.5.3 and 6.4.2. As introduced in Section 3.4.2.1, the M IDE [206] enables the automatic generation [208] of rules behavioural patterns learnt from the users using the LFPUBS [207] tool. Let the hypothetical scenario assume the generation of rules not only for switching on and off the cooker (Section 6.4.2), but also that the LFPUBS system has discovered that the user generally wakes up around 6:30 a.m., and prepares tea and scrambled eggs for breakfast. The set of ordered actions is as follows: Enters the kitchen, takes some eggs, tea bags, and a pan from the cabinet, fills a kettle with water, and then starts cooking the eggs. Around two minutes later, the user turns on the kettle in order to have the water boiling for the tea by the time the eggs are ready[10]. Suppose that the user is a person in with Down's syndrome trying to be more independent at home through smart-technology. To increase the reliability of the system two main aspects will be considered: the first is to have the cooker switched on for a long time while unattended, and the second is to leave the water tap on without supervision. This system is represented in the IDE as shown in Listing 7.5. Lines 1 and 2 declare all the states existing in the system, line 3 identifies which of the states are independent, and lines 4 to 6 initialise the values of the states. Lines 10, 13, and 29 to 31 declare those rules related to the cooker being unattended. For illustration purposes, we assume that the remainder of the rules, regarding the water tap, describe patterns discovered by the LFPBUS tool which have been automatically translated to M.

[10]In terms of sensors, the result is: $atKitchen \rightarrow (12s.) \rightarrow cabinet \rightarrow (22s.) \rightarrow \neg cabinet \rightarrow (18s.) \rightarrow waterTap \rightarrow (9s.) \rightarrow \neg waterTap \rightarrow (11s.) \rightarrow CookerOn \rightarrow (123s.) \rightarrow KettleOn \rightarrow (120s.) \rightarrow \neg KettleOn$.

### 7.4.1 Evaluating properties

After translating it to NuSMV, the model can be checked against undesired behaviours, as illustrated in this section. First, the recommended minimum execution time[11] is applied, as indicated in Section 7.3.1, which is 25775. Then, the desired properties are added. For example, not having the cooker on while there is nobody in the kitchen (there is a hazard), as shown in Line 1 Listing A.9. Also, the model can be checked to ensure the time it will take from the cooker to be unattended until it is switched off (Line 2). Regarding the more reliable deployment of automatically generated rules of learned user behavioural patterns, it is feasible to make the automation safer by specifying in advance some property specifications about the expected system behaviour. In this scenario, it is undesirable to have the tap on while there is no one in the kitchen. This can be specified in the properties as shown in Line 3 Listing A.9. Checking the model resulting from translating the M specification in Listing 7.5 to NuSMV, against the properties from Listing A.9, takes 98 time units using the minimum execution time $\Gamma_{min}$ from Section 7.3.1, and the same computer as that indicated in Section 7.5. The property of Line 1 is met. The property of Line 2 indicates that the maximum time it will take from the cooker being unattended until it is switched off is 61 seconds. However, the property of Line 3 is not satisfied, as shown in the counter-

```
1 SPEC AG((hazard) -> AX(cookerOn=FALSE));
2 COMPUTE MAX[(cookerUnattended = TRUE),(cookerOn = FALSE)];
3 SPEC AG!( (atKitchen=FALSE) & (waterTapOn=TRUE));
```

Listing 7.6: Example of a specification in NuSMV.

example from Appendix A.5. In this case, the system could prevent such a model from being automatically deployed. In order to make this model satisfiable, the developers

[11]There are no time expressions in the next-time rules, so $\lambda_{max} = 0$. The maximum stratification level from the rule set $R$ is $\omega = 9$. The set of all time expressions is $T = \{22686, 25390, 120, 60, 12, 22, 18, 9, 11, 123, 120\}$. The value of the highest time expression is $t =$ 07:03:10 $= 25390$, $t \in T_C$, which is in the stratification stage level $k = 1$, making $\phi = 1$. The highest time expression for all the rules in each stratification stage level, from $\phi = 1$ to $\omega = 9$ is: $\delta_{max} \in \Delta_1 = 25390$, $\delta_{max} \in \Delta_2 = 60$, $\delta_{max} \in \Delta_3 = 12$, $\delta_{max} \in \Delta_4 = 22$, $\delta_{max} \in \Delta_5 = 18$, $\delta_{max} \in \Delta_6 = 9$, $\delta_{max} \in \Delta_7 = 11$, $\delta_{max} \in \Delta_8 = 123$, $\delta_{max} \in \Delta_9 = 120$. The formula can be applied as follows: $\Gamma_{min} = (0+1) + (25390+1) + (60+1) + (12+1) + (22+1) + (18+1) + (9+1) + (11+1) + (123+1) + (120+1) = 25775$.

could add a stratification level 9 same-time rule to meet the specification property: $\neg atKitchen \land waterTapOn \rightarrow \neg waterTapOn$. Note that if the updated rule models containing new learned patterns do not contradict any of the safety property specifications given in advance, these could be automatically deployed.

| **Experiment** | **Metric** | **Proportion** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| (A) | Mean | 0.03s | 0.05s | 1.01s | 2.14s | 1m 40s | 26m 24s | 1h 15m 03s | * |
| | STD | 0.04s | 0.01s | 0.02s | 0.05s | 11.24s | 1m 48s | 6m 19s | * |
| (B) | Mean | 15m 12s | 17m 27s | 17m 31s | 17m 9s | 21m 14s | 55m 9s | * | - |
| | STD | 32s | 2m 12s | 1m 1s | 1m 48s | 01m 43s | 09m 38s | * | - |
| (C) | Mean | 15m 56s | 20m 41s | 37m 43s | * | - | - | - | - |
| | STD | 1m 21s | 4m 35s | 6m 15s | * | - | - | - | - |
| (D) | Mean | 14m 27s | * | - | - | - | - | - | - |
| | STD | 46s | * | - | - | - | - | - | - |
| (E) | Mean | 15m 21s | 15m 02s | 15m 54s | 15m 00s | 15m 29s | 15m 52s | 14m 55s | 15m 12s |
| | STD | 1m 09s | 0m 54s | 1m 38s | 0m 53s | 1m 09s | 0m 55s | 1m 10s | 1m 46s |
| (F) | Mean | 14m 03s | 14m 13s | 13m 33s | 13m 47s | 13m 53s | 14m 27s | 13m 49s | 15m 36s |
| | STD | 0m 15s | 0m 24s | 1m 06s | 0m 11s | 0m 33s | 0m 39s | 0m 27s | 0m 39s |

Experiment A = ($I_{max} = 10$, $\mu$ = 3, $\alpha$ = 5, $\beta$ = 10) Experiment B = ($I_{max} = 86400$, $\mu$ = 3, $\alpha$ = 5, $\beta$ = 10)
Experiment C = ($I_{max} = 86400$, $\mu$ = 300, $\alpha$ = 500, $\beta$ = 1000)
Experiment D = ($I_{max} = 86400$, $\mu$ = 10000, $\alpha$ = 43200, $\beta$ = 63200)
Experiment E = ($I_{max} = 86400$, no $Bp_{Op}$)
Experiment F = ($I_{max} = 86400$, no $Bp_{Op}$, $S_I no. = 4$, $S_D no. = 5 * p + (S_C no.)$)
STD = Standard deviation "*" = Tests take more than one day "-" = There is no information available.

**Table 7.4: Execution time for six models, scaling the NuSMV size.**

## 7.5 Evaluation

The aim of this section is to provide an early evaluation of the performance of NuSMV to check properties of M specifications. For this, six different types of experiments have been created (from A to F). Experiment A evaluates a system with low time values, specifically in the maximum iteration and the time of the temporal operators. Experiment B is the same as A, but it increases the maximum iteration time to a whole day. Experiment C is the same as B but increases the time value of the temporal operators. Experiment D is the same as C but increases even more the time values of the temporal operators. Experiment E maintains the day length for the maximum time iteration, but removes all temporal operators. Experiment F is the same as E but it reduces the number of independent states. It is important to note that NuSMV models directly depend on how the M models are formulated, as indicated in Table 7.2. For this reason, each experiment type has been proportionally increased up to eight times, using a Java program[12], to generate the code. Each of the experiments for each of the proportions has been conducted ten times, on a Lenovo T440p with Windows 10 Enterprise, 8 GB of RAM memory, processor Intel Core i5-4200M @ 2.5GHz, hard disk with SATA3 (6GB/s). Table 7.4 shows the results of the experiments. Figure 7.1 shows an ex-

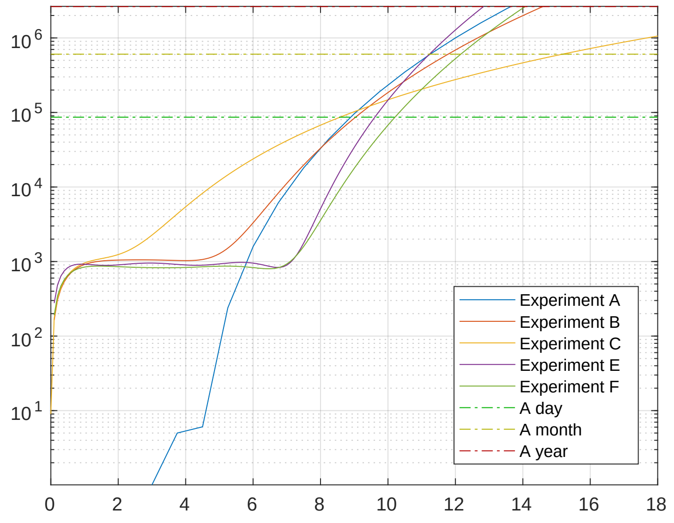


**Figure 7.1: Estimated performance of the experiments by applying Lagrange's interpolation to time results in Table 7.4.**

| M Element | $t$ | $s$ | $S_C$ | $S_T$ | $I_S$ | $A_S$ | $I_W$ | $A_W$ |
|---|---|---|---|---|---|---|---|---|
| NuSMV Variable No. | 0 | 1 | 2 | 0 | 1 | 3 | 2 | 3 |
| NuSMV Counter No. | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

**Figure 7.2: NuSMV variable number required for each M element.**

trapolation of the results to larger model sizes. Experiment D has been omitted from Figure 7.1 as there is insufficient data for obtaining an accurate function. Most of the experiments show a performance of around a day when reaching a factor of 9. This

[12] https://github.com/ualegre/m2nusmv-evaluation

means around 53 states, 44 of which are independent (sensors), 18 rules (9 same-time and 9 next-time), and 7 specification properties. The experiments help to give insights on how regular sized systems can behave. The reader should take into account that the different combination of elements will give different time results. Overall, the results show that it is feasible to check systems that have some balance between the execution time, size and number of temporal operators and independent variables. Better time results could be achieved with more powerful computers with more RAM and faster processors. In some cases, smaller rule sets which are independent from each other could also be verified separately, in order to optimise the performance results. Experiments show good performance for model checking regular-sized context-aware systems, with running times proportional to the size of the model M. The negative effect of model size on performance could be reduced by separating rule chains that are independent from each other.

## 7.6 Conclusions

One of the major disadvantages of rule-based context-aware systems is the lack of verification techniques [1]. This can help developers to reduce development costs by enabling them to develop and deploy context-aware reasoning components in a fast and more reliable way. This chapter has introduced an approach for making more reliable the context rule reasoning in DC-ASEF. Particularly, an approach to model check its reasoning specifications, in order to increase the reliability of automatically translated rules from user behavioural patterns. For this, a proof-of-concept tool to translate specifications from the M language [162] to NuSMV [183] models has been presented, which can check the model against temporal properties. This approach has been supported with an open-source tool integrated as part of the DCase Modelio module [200] from DC-ASEF, and as part of the M IDE [206]. This library is reused as part of the DCase tool [200] for DC-ASEF, and it has also been implemented as an extension of the M IDE for the implementation and deployment of rule-based context-aware systems, which is aimed at stationary platforms. The main advantage of the approach is that the same specifications that can be translated into NuSMV models, can also be translated into reasoning rules for both M IDE [206] (stationary platform) and Android Context Reasoner [167] (mobile platform). The approach enables the merging of reasoning and learning capabilities for both mobile and stationary platforms. In the stationary platform, the output of LFPUBS can be automatically translated into M compatible specifications, which facilitates the anticipation of user needs and preferences.

# Part IV

# Implementation, deployment and maintenance stages

Chapter 8

# IDMC-ASEF: Implementation, deployment and maintenance for the context-aware engineering framework

## 8.1 Introduction

Chapters 4, 5, 7, and 6, introduce a framework to specifically guide the creation of context-aware systems, supporting the requirements elicitation and design stages of the development process, focusing on giving response to the needs of the end-user stakeholders. This chapter focuses on the Implementation, Deployment and Maintenance for the Context-Aware Software Engineering Framework, a framework to support the implementation, deployment and maintenance stages of a context-aware system life-cycle, taking into account the previous requirements and design models. Particularly, the framework has been created following the objectives explained in Section 1.4, and it supports:

- An open-source set of tools to automate the code generation from requirements and Design Diagrams.
- Guidance for the code implementation of the system, as well as for its deployment and maintenance, reusing the concept of situation of interest introduced in Section 2.4.3.
- An open-source set of tools to facilitate the implementation of context-aware systems in stationary platforms such as those in a smart-house or a smart-office.
- An open-source set of tools to facilitate the implementation of context-aware systems in android-based mobile platforms.

As it can be observed in Figure 8.1, there are four main activities for this methodology. The remainder of the chapter explains these activities, as well as the tool support and model to text transformations. Section 8.2.1 introduces the activity of the framework related to the code generation. Section 8.3 explains the framework activity related to the implementation of the system. Section 8.4 describes the activity that relates to deploying and running the system under development. Section 8.5 describes maintenance related activities of the framework. Section 8.6 explains the details of the different tools developed for the aforementioned framework. Section 8.7 describes the different model to text transformations required to generate code from the different diagrams. Section 8.8 summarises the chapter. Finally, Appendix A has been included in order to explain how to download, install and use the different tools presented in this dissertation. Additionally, this appendix includes some model to text transformations which were too long to be included in this chapter.

The first activity consists of generating the code from previous diagrams (*i.e.,* Context Acquisition & Modelling, Reasoning, Context Deployment and Situation Detection Diagrams). Then, the second activity relates to implementing the design of the system. The third activity relates to deploying and running the system. Finally, the last activity occurs after the system is already implemented and running, and consists of giving maintenance to the system. Although maintenance is expected for the whole system, the framework specifically supports the management of changes occasioned by changes in the situations of interest. It is worth mentioning that during this process there is an initial stage of generating a code using a model-driven approach. Although most of the code is automatically generated, there are also some cases in which only templates are generated, which need to be realised by the developers of the system during the implementation stage. Also, note that the automated generation of code is focused on the context-aware aspects of the system, those which are intended to handle the context information. The non-contextual aspects of the system are also to be developed during the implementation stage.

## 8.2 Code generation

### 8.2.1 Generate code

The first sub-activity consists of generating the code from the different models. For space restrictions, this automated process is further explained in Sections 8.7.1.7 and 8.7.2.8. Section 8.7 describes those files that are automatically generated, and how they are used. Section 8.6 gives more information about the tools developed for this

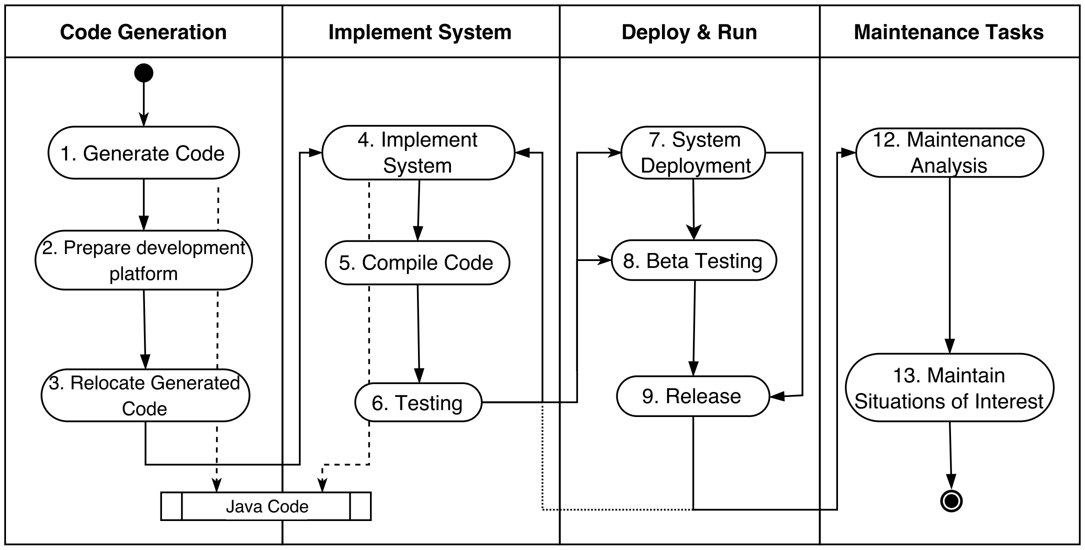


**Figure 8.1: Core sub-activities for the Implementation, Deployment and Maintenance methodology.**

framework.

### 8.2.2 Prepare development platform

Once the code has been generated, it is necessary to deploy it to the relevant platform, and a Java IDE is recommended for this purpose. For this thesis, Eclipse IDE for Java Developers[1] has been used for the stationary platform code [209], and Android Studio[2]for the mobile platform code. More information about how to download, install and use the different tools of the framework can be found in Appendices A.2 and A.3.

### 8.2.3 Relocate generated code

There are certain files which are automatically generated from the different models created using the Modelio tool. These files are automatically generated in a specific folder, which is selected by the developer. Created files need to be merged with the existing development project. It is recommended to respect a certain folder structure hierarchy,

[1]Version Neon.3 Release (4.6.3), Build-id 20170314-1500, JRE 1.8.0, Git and Maven integration.

[2]Version 3.0.1, Build-id # AI-171.4443003 JRE 1.8.0, OpenJDK 64 bit

which depends on the library used for the implementation (mobile or stationary). An explanation and illustration example on this process can be found in Section 8.7.

## 8.3 Implement system

### 8.3.1 Implement system

Not all the code is generated automatically, in isolated cases[3] only some skeleton code is generated and it is required that the developers finish implementing some parts of the system. Also, note that the framework automates the creation of context-related components, but other non-contextual artefacts need to be implemented manually by the developers of the system. For the stationary platform seven different software artefact types can be automatically generated, from which only one is a skeleton that requires realisation by developers. For the mobile platform, five different software artefact types can be automatically generated, from which only one is a skeleton that requires realisation by developers.

### 8.3.2 Compile code

After the code is ready, it naturally[4] requires compilation in order to generate a runnable file that integrates the different libraries that it is using. If developers are using Eclipse, they can compile the code using the instructions from Appendix A. If, instead, developers are using Android Studio, they can compile the corresponding APK using the instruction from Appendix A.3.

### 8.3.3 Testing

Verification tests should have been conducted during the application of the Design methodology, which ensure that the automatically generated code will function as intended. This activity complements the verification process for the design components,

[3]Particularly the Java actuators, the Context Receiver, or the Context Observer, introduced in Section 8.7.

[4]Since the code generated by this platform is written in Java, which is a compiled programming language, it requires compilation. Note that any additional compilation to that of Java is not required.

as it verifies those parts of the system which have not been automatically generated, or which have been generated as templates which have been manually completed by developers. Verification tests include whether or not the system meets the requirements and specifications it was designed and developed for, and if it achieves the results expected by the relevant majority of the stakeholders. The following activity consists of a traditional system evaluation, which comprises the following stages:

- Unit Testing: First, individual units of code are tested independently. For this purpose, Modelio offers a JUnit [210] [211] module which generates test models and code from a given Java model. Any other technique for unit testing can also be used.
- Integration Testing: Then, the individual units of code are combined and tested as larger aggregates.
- System Testing: The original requirements specification is tested against the system, which is now integrated as a whole.
- Alpha Testing: Finally, the latest testing is an early acceptance test, which consists of identifying all possible bugs or issues before releasing the product to the general public. Through the use of blackbox and whitebox techniques, the developers simulate real users, carrying out the tasks that that a typical user would perform. Developers can guide themselves using the acceptance test design created during the test-case design activity, introduced in Section 6.5.1.

## 8.4 Deploy & run

### 8.4.1 System Deployment

Initial deployment requires first-time installation. There are two different aspects that are covered during the installation. Android mobile platforms require execution of a set of commands as described in Appendix A.3. Stationary platforms also require physical installation of the different hardware components, which includes the Vera Router and the different Z-Wave based sensors. These can be distributed across different rooms or spaces. More information on the installation of the stationary platform can be found in Appendix A.2.

Since the methodology is intended to be incremental, when this activity happens for a second time, there is a need to update earlier system versions which are replaced with a newer release. This release might include not only new software components, but also hardware components, and it can affect either the whole system, or simply part of it. Sometimes, the updating process might also require manual deactivation and uninstallation of the system in advance. For this, any existing legacy applications are stopped, uninstalled and removed if necessary. Executing component instances are shut down, as well as any version of the system that might exist simultaneously. Those parts of the system that are no longer required are removed, including dependencies or unused hardware components such as Z-Wave sensors.

### 8.4.2 Beta Testing

This activity is the final test before the product is shipped to the consumers, in which the end-user stakeholders use the deployed system in a real environment. A Beta version of the system is released to a limited number of users, who in return give feedback of their experience and report any existing bugs or issues with the system.

### 8.4.3 Release

Once the development process is completed, there is a release sub-activity. During this process, the resources required for the system to operate are defined, and documentation about the latest changes is prepared.

## 8.5 Maintenance tasks

### 8.5.1 Maintenance analysis

An analysis is conducted to determine what types of modifications are required for the system. Then, a maintenance plan is prepared. This particular activity focuses on the evaluation of the different situations of interest detection plans that are intended to be modified, improved or introduced from scratch.

### 8.5.2 Maintain situations of interest

The next sub-activity consists of maintaining each of the modified, added or deleted situation of interest detection plans. For this, the main activities of the methodology are applied again. During the reapplication of the methodology, particularly during the requirements stage, and for context-maintenance purposes, developers mainly focus on the context-related activities. At this stage, an additional impact analysis is conducted for each modified, deleted or new situation of interest detection plan. This evaluation will determine the impact of the introduced changes, and can be applied using the Modelio Link view tool, which can be used to visualise and trace the different elements that are going to be affected by the changes. After these activities are applied, the design, implementation and deployment related activities are repeated. Note that when the methodology has been applied once for each of the methodology stages, the maintenance process becomes less arduous, as some steps might be omitted for not being necessary. Also, note that when conducting the maintenance of situations of interest, developers can decide to accept or reject certain changes, depending on the results of the situation of interest and corresponding context-aware feature analysis.

## 8.6 Tool support

This chapter has presented a framework to guide and help to document the implementation, deployment and maintenance of context-aware systems. The framework is supported by three different tools that have been developed as part of this thesis, introduced below.

### 8.6.1 Model to text transformation

The first of these tools is part of the DCase module for the design of context-aware systems, introduced in Chapter 6, and it is aimed at generating the code corresponding to the different design and requirements models, as further explained in Section 8.7. As it is a Modelio module, it is fully compatible with the previously introduced modules, as well as with those explained in Section 3.3.5. However, it is also possible to generate code using other tools for code transformation such as Acceleo.

### 8.6.2 Reasoning tool for stationary platforms

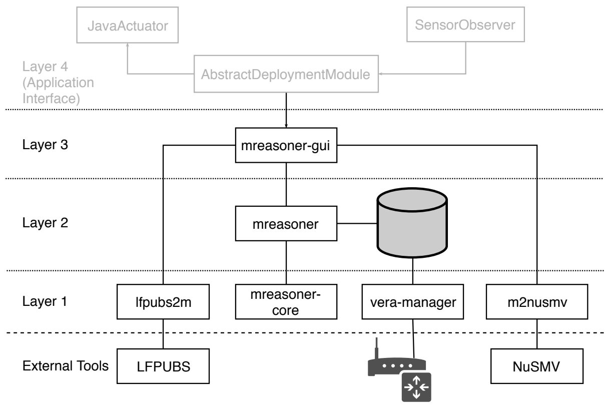


**Figure 8.2: Libraries created and extended during this thesis for the reasoner in stationary platforms and its connectivity to external elements.**

The second tool facilitates the implementation of reasoning mechanisms in stationary platforms. For this purpose, an existing tool [164] has been extended. The contributions to this tool include its publication in an open-source repository [206], its mavenisation, and major refactorisation of the code, as well as its division into different reusable libraries as shown in Figure 8.2. The following libraries have been created or extended for this thesis:

- Graphical User Interface: The mreasoner-gui [206] is an additional layer to the mreasoner library which allows one to graphically configure, create and modify the different elements required for the reasoner to run. It includes the creation of rule specification files and its verification using the m2nusmv library. Also, it includes a graphical tool for translating the output file of the LFPUBS using the lfpubs2m library.
- LFBPUS2M: The lfpubs2m [208] library automatically translates patterns detected using the pattern learning LFPUBS tool [88] [207] into rules that have the M Language format. This library was originally created as part of [164], and was later extended in [165]. It has been improved again to make it more compatible with the framework of this thesis.
- MReasoner Core: The mreasoner-core [212] contains the minimal required elements for reasoning according to the algorithm presented in [164]. It is data-

storage independent, and it can be used as a lightweight independent library.

- Vera Manager: In order to interconnect different types of Z-Wave radio based sensors, the M reasoning system uses Vera Routers [213]. These types of routers provide their own operating system to manage the inclusion and removal of different Z-Wave based sensors. The Vera manager [214] is a reasoner-independent library that facilitates reading the events occurring in the log of a Vera Router using an Observer pattern structure.
- M2NuSMV: This library [205] translates specifications written in the M language to specifications in the NuSMV model checker.
- LFPUBS: Is a system that learns frequent patterns of user behavior, taking into consideration the specific features of Intelligent Environents. It has a learning layer that is independent of the particular environment in which the system is being applied. It also includes a language that allows the representation of discovered behaviours in a clear and unambiguous way. Coupled with the learning language, it provides an algorithm that discovers frequent behaviors using association, workflow mining, clustering, and classification techniques [88]. The work has been released as open-source and it can be found at [207].
- NuSMV: A symbolic model checker tool that reimplements and extends SMV, the first model checker based on Binary Decision Diagrams using the CUDD library [215]. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs, and as a test-bed for formal verification techniques [216] [183].

### 8.6.3 Reasoning tool for mobile platforms

The third software tool developed as part of this thesis work facilitates the implementation of reasoning mechanisms in mobile platforms. It is based on an already existing tool [167], which has been refactored for facilitating the automation of code generation, as part of the contribution to the work of this dissertation. Additionally, another contribution is the influence on the design structure of observers in [217] to include the responsibility of sensors, as introduced by [1]. The reasoner is divided into two main libraries: Android Context Library (aContextLib) and Android Context Reasoner (aContextReasoner). Figure 8.3 illustrates the inner workings of these

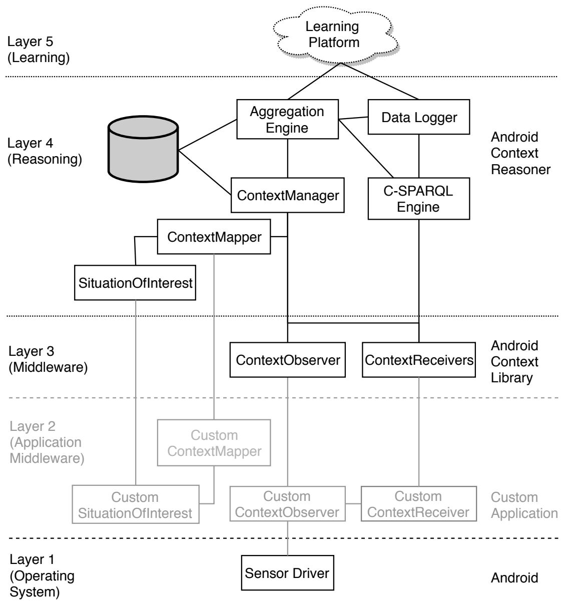


**Figure 8.3: Main elements in the android reasoner and it interacts with the context library, the newly developed applications the operating system and the learning platform.**

two main libraries. The different components of these libraries are explained below[5]:

- Query Platform: The aim of this component is to facilitate the analysis of relevant behavioural traits of users. For this, it uses historical context data produced by the reasoner. With this data, it generates graphical visualisations over a given time window.
- Aggregation Engine: This particular component reasons over primary context in order to produce higher level context information or secondary contexts [1].
- Data Logger: Logs the different changes that occur in the context.
- Context Manager: It is a core component than handles the operations between the database, the receivers (data from sensors), and the reasoner.
- CSPARQL Engine: Handles the stream reasoning engine for handling primary contexts in the form of ontologies.

[5]Note that specifics of the application are explained in Section 8.7.1, as well as the SituationOfInterest, ContextObservers and ContextReceivers.

- Context Mapper: This component is responsible for registering and de-registering both context and context modelling rules.
- Sensor Framework: Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions [218]. For the purpose of letting the developer access the raw data from sensors, Google provides the android sensor framework.

## 8.7 Model to text transformations

It is anticipated that the framework developed and described in this thesis report will improve the efficiency of the software developer, and, to some extent, speed up the implementation stage. The key for making the implementation faster is the feature of the model driven development to generate code text from diagram models. Previous sections have merely introduced the generation of code as part of the methodology. This section focuses on explaining the specifics of code generation aspects that have been implemented for the automatic code generating tools of this framework. The different transformation rules are divided into two main groups: those applied to the stationary platform related models and those applied to the mobile platform related models.

### 8.7.1 Process for mobile platforms

This subsection explains in detail how the code is automatically generated, compiled, executed and deployed, for mobile platforms, as it is illustrated in Figure 8.4. The figure maps the different Design models to the different software artefacts of the Android Context Reasoner, as well as the steps that the developers need to follow for this purpose.

#### 8.7.1.1 ContextReceiver Class

The purpose of the ContextReceiver class is to have a unified callback for an application to send raw data. The Android Context Library [166] can be used as a common software module for more than one context-aware application simultaneously. Each

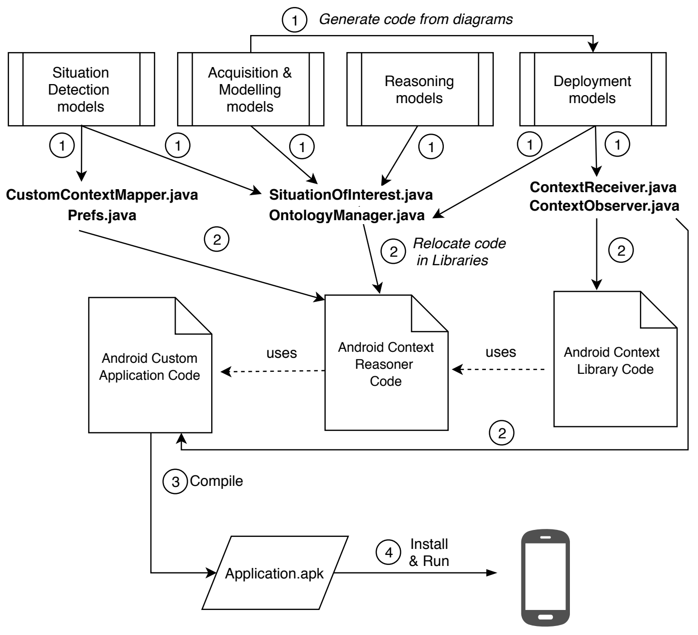


Figure 8.4: Map of the software artefacts produced by the Acquisition & Modelling, Reasoning and Deployment Diagrams, and their relation to the Android libraries and the mobile system under development.

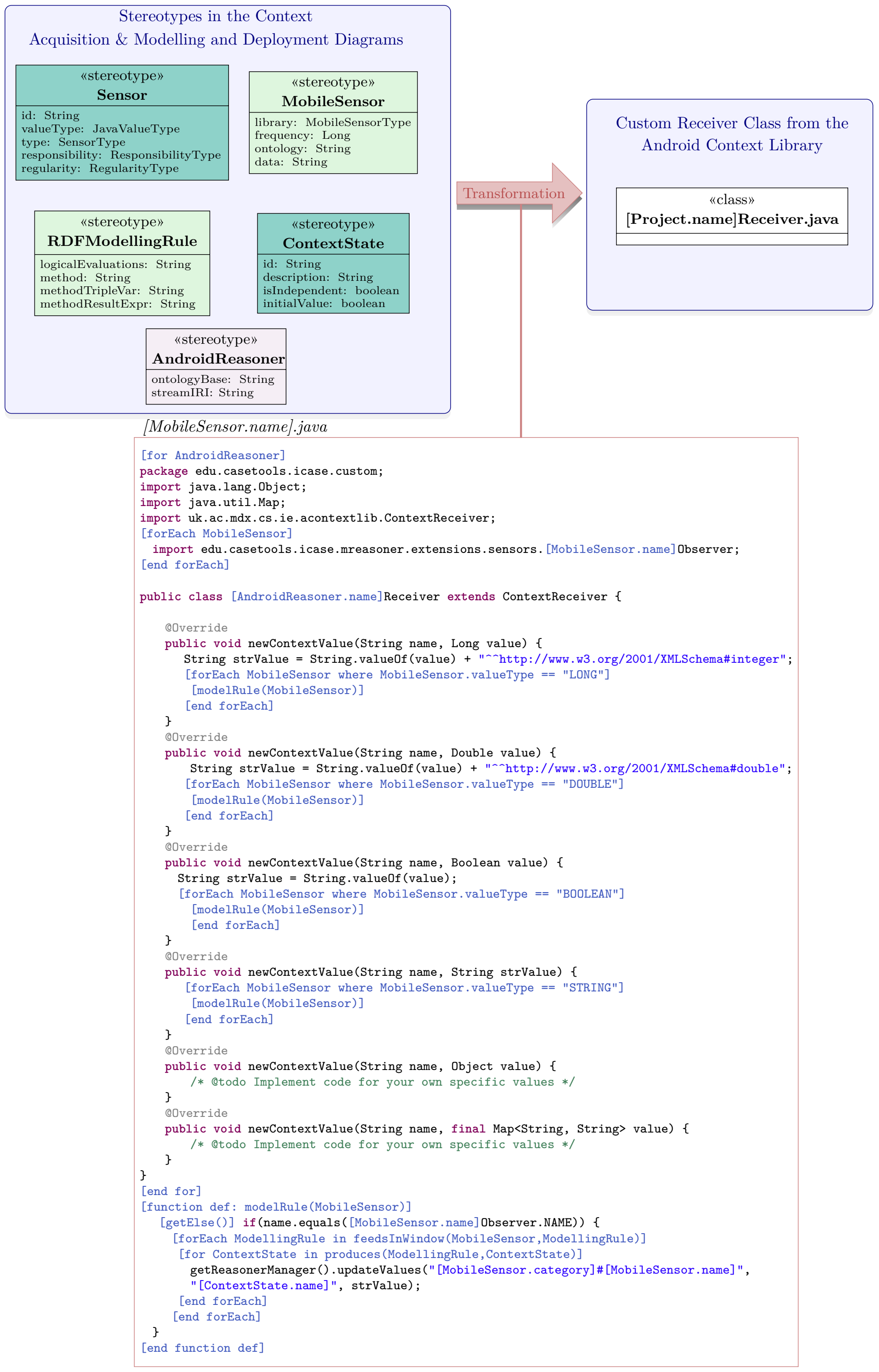
application requires a context receiver, where all the raw data from the sensors will be sent. There are two different flavours: The IContextReceiver is an interface which can be used as callback for data when using observers singularly without the reasoner. The ContextReceiver is an abstract class which helps in collecting data and feeding it into the reasoner, and generating RDF on the fly. Figure 8.5 shows the pseudo-code rules for creating the context receiver of the project. Note that depending on the project, functions that receive Object types or string maps need to be manually implemented.

#### 8.7.1.2 ContextObserver Class

ContextObservers are the primary components of the library for the collection of raw data coming from the Android sensor framework or other datasources. Receivers are registered to observers and receive callbacks with the raw information produced by the sensor. All context observers contain four main methods for managing their life-cycle (*i.e.* start, pause, resume, stop). As a contribution to this thesis, a general schema for classifying observers has been designed [6], which is based on the previous design of the context library, and the sensor type classification introduced by [1]. According to this classification, each observer can extend one of the following abstract classes:

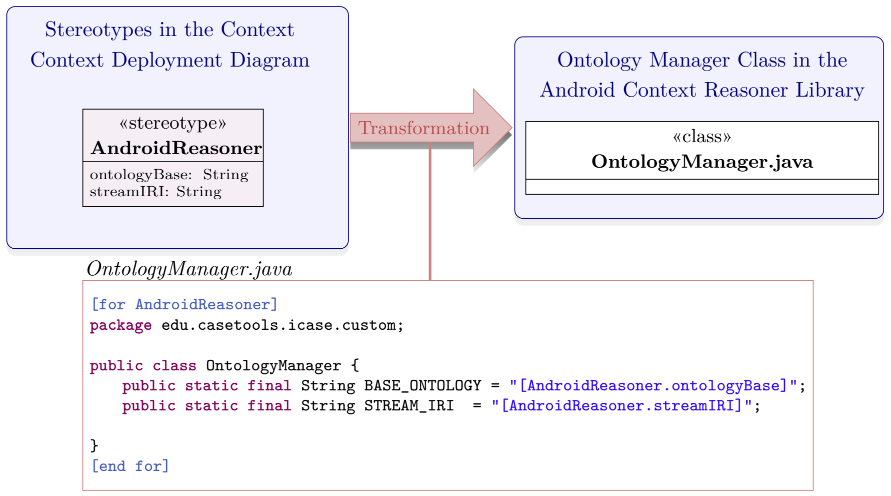
- ContextObserver: The core abstract class that each observer must at least extend.
- PullObserver: In order to obtain the desired data, the observer is responsible for making a request from the Android Sensor Framework or from other datasources.
- PushObserver: The sensor is responsible for pushing the raw data to the observer. Although a custom sensor can directly extend from the PushObserver class, there are currently four different types of PushObservers for this library:
  - (Android) SensorObserver: It is used for getting raw data directly from the Android Sensor Framework.
  - BroadcastObserver: Which gets data from Android device intent broadcasts.
  - BluetoothLEObserver: Which gets data from Bluetooth Low Energy devices.

[6]Note that although the classification schema was produced as a contribution to this thesis, it was implemented by Dean Kramer, a researcher funded by the POSEIDON project.



```
[for AndroidReasoner]
package edu.casetools.icase.custom;
import java.lang.Object;
import java.util.Map;
import uk.ac.mdx.cs.ie.acontextlib.ContextReceiver;
[forEach MobileSensor]
  import edu.casetools.icase.mreasoner.extensions.sensors.[MobileSensor.name]Observer;
[end forEach]

public class [AndroidReasoner.name]Receiver extends ContextReceiver {

    @Override
    public void newContextValue(String name, Long value) {
       String strValue = String.valueOf(value) + "^^http://www.w3.org/2001/XMLSchema#integer";
       [forEach MobileSensor where MobileSensor.valueType == "LONG"]
        [modelRule(MobileSensor)]
       [end forEach]
    }
    @Override
    public void newContextValue(String name, Double value) {
        String strValue = String.valueOf(value) + "^^http://www.w3.org/2001/XMLSchema#double";
       [forEach MobileSensor where MobileSensor.valueType == "DOUBLE"]
        [modelRule(MobileSensor)]
       [end forEach]
    }
    @Override
    public void newContextValue(String name, Boolean value) {
      String strValue = String.valueOf(value);
      [forEach MobileSensor where MobileSensor.valueType == "BOOLEAN"]
        [modelRule(MobileSensor)]
        [end forEach]
    }
    @Override
    public void newContextValue(String name, String strValue) {
       [forEach MobileSensor where MobileSensor.valueType == "STRING"]
        [modelRule(MobileSensor)]
       [end forEach]
    }
    @Override
    public void newContextValue(String name, Object value) {
        /* @todo Implement code for your own specific values */
    }
    @Override
    public void newContextValue(String name, final Map<String, String> value) {
        /* @todo Implement code for your own specific values */
    }
}
[end for]
[function def: modelRule(MobileSensor)]
   [getElse()] if(name.equals([MobileSensor.name]Observer.NAME)) {
     [forEach ModellingRule in feedsInWindow(MobileSensor,ModellingRule)]
      [for ContextState in produces(ModellingRule,ContextState)]
        getReasonerManager().updateValues("[MobileSensor.category]#[MobileSensor.name]",
        "[ContextState.name]", strValue);
      [end forEach]
     [end forEach]
  }
[end function def]
```

**Figure 8.5: Illustration of the pseudo-code rules for transforming stereotypes in the Context Acquisition & Modelling Diagram, to the Java class for the context receiver in the Android Context Library. Where $getElse()$ is a function that writes an else if there has been a previous if, and text between square brackets represents an attribute from the stereotype in the form of *Stereotype.name*.**



```
[for AndroidReasoner]
package edu.casetools.icase.custom;

public class OntologyManager {
    public static final String BASE_ONTOLOGY = "[AndroidReasoner.ontologyBase]";
    public static final String STREAM_IRI  = "[AndroidReasoner.streamIRI]";

}
[end for]
```

**Figure 8.6: Illustration of the pseudo-code rules for transforming stereotypes in the Context Deployment Diagram, to the Java class for the ontology manager in the Android Context Reasoner.**

– LocationObserver: Which gets data from the Android Location services.

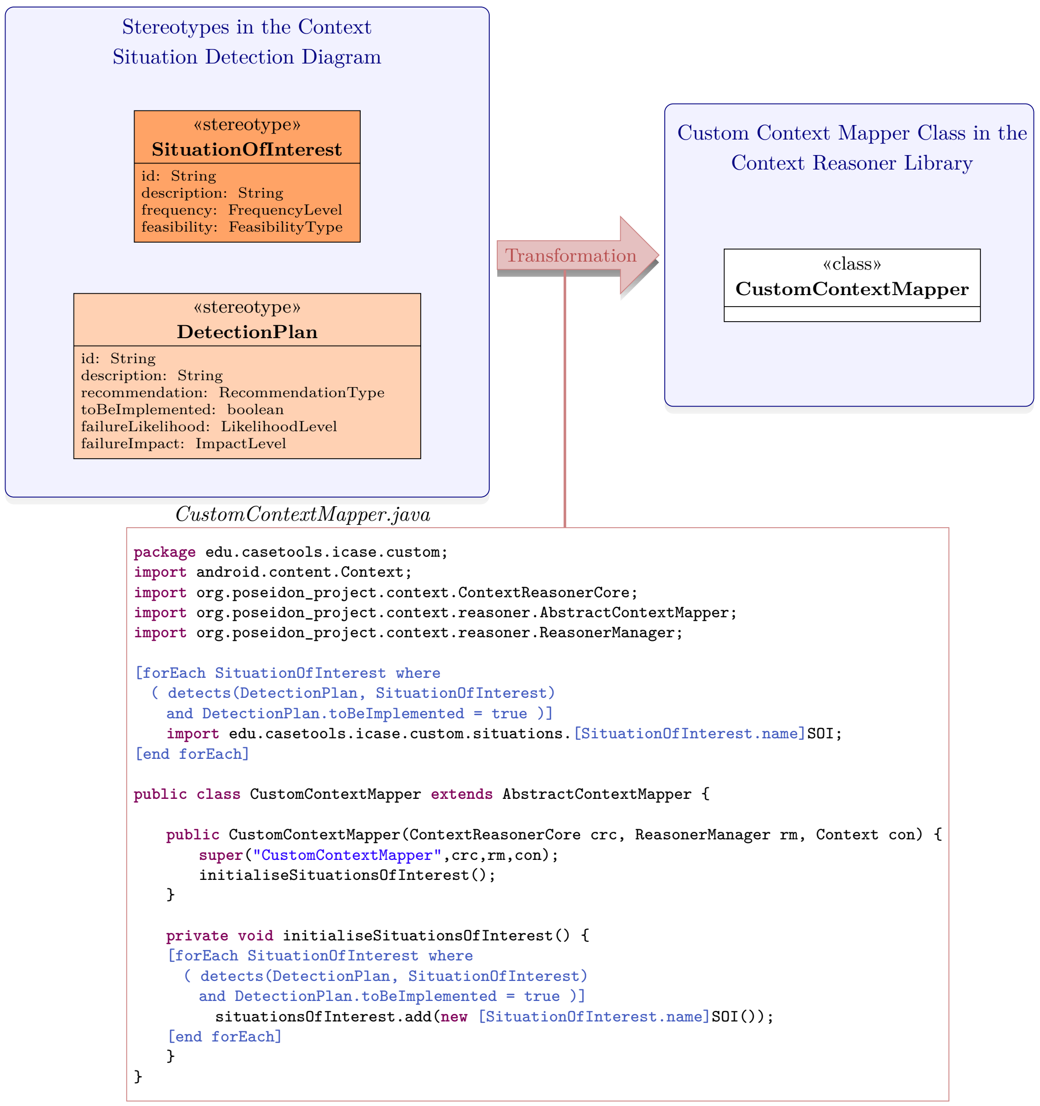
Figure 8.8 shows the code transformation for observers. The specific pseudo-code rules for getting the transformation of each Observer type can be found in Appendix A.4.

#### 8.7.1.3 CustomContextMapper Class

This class contains all the SituationsOfInterest required by the application. It is a centralised way of handling the reasoning and modelling rules to be applied by the reasoner. Figure 8.9 shows the pseudo-code rules for getting the transformation. Note that only one CustomContextMapper is generated for each project.

#### 8.7.1.4 Prefs Class

This class contains all the preference values which are customisable by the users and which can be changed after the system is implemented. It is a centralised way of handling the reasoning and modelling rules to be applied by the reasoner. Figure 8.7 shows the pseudo-code rules for getting the transformation. Note that only one Prefs file is generated for each project.


Stereotypes in the Context Situation Detection Diagram
«stereotype»
SituationOfInterest
id: String
description: String
frequency: FrequencyLevel
feasibility: FeasibilityType
«stereotype»
DetectionPlan
id: String
description: String
recommendation: RecommendationType
toBeImplemented: boolean
failureLikelihood: LikelihoodLevel
failureImpact: ImpactLevel
Transformation
Custom Context Mapper Class in the Context Reasoner Library
«class»
CustomContextMapper
*CustomContextMapper.java*

```
package edu.casetools.icase.custom;
import android.content.Context;
import org.poseidon_project.context.ContextReasonerCore;
import org.poseidon_project.context.reasoner.AbstractContextMapper;
import org.poseidon_project.context.reasoner.ReasonerManager;

[forEach SituationOfInterest where
  ( detects(DetectionPlan, SituationOfInterest)
    and DetectionPlan.toBeImplemented = true )]
    import edu.casetools.icase.custom.situations.[SituationOfInterest.name]SOI;
[end forEach]

public class CustomContextMapper extends AbstractContextMapper {

    public CustomContextMapper(ContextReasonerCore crc, ReasonerManager rm, Context con) {
        super("CustomContextMapper",crc,rm,con);
        initialiseSituationsOfInterest();
    }

    private void initialiseSituationsOfInterest() {
    [forEach SituationOfInterest where
      ( detects(DetectionPlan, SituationOfInterest)
        and DetectionPlan.toBeImplemented = true )]
          situationsOfInterest.add(new [SituationOfInterest.name]SOI());
    [end forEach]
    }
}
```

**Figure 8.7: Illustration of the pseudo-code rules for transforming ContextPreference stereotypes in the Situation Detection Diagram, to the Java class for the preferences in the Android Context Reasoner.**

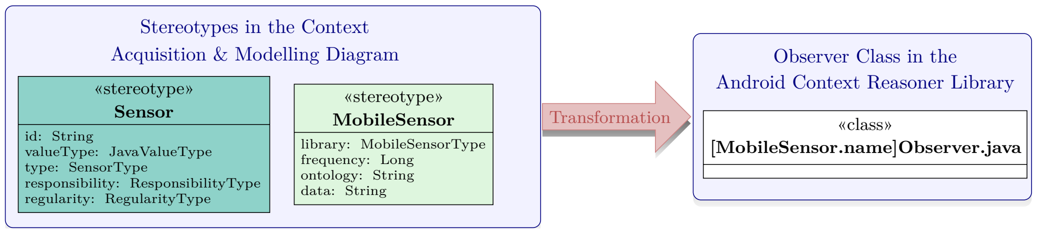
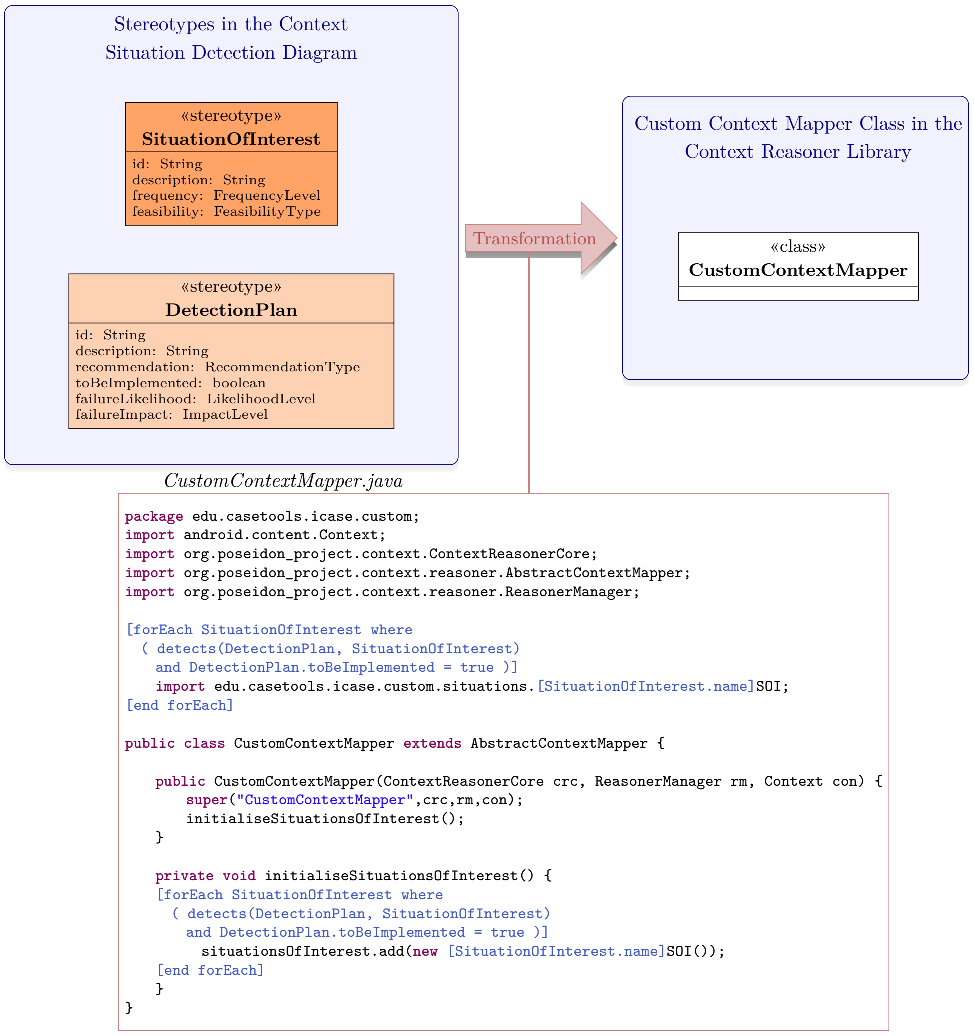

Stereotypes in the Context
Acquisition & Modelling Diagram
«stereotype»
Sensor
id: String
valueType: JavaValueType
type: SensorType
responsibility: ResponsibilityType
regularity: RegularityType
«stereotype»
MobileSensor
library: MobileSensorType
frequency: Long
ontology: String
data: String
Transformation
Observer Class in the
Android Context Reasoner Library
«class»
[MobileSensor.name]Observer.java

**Figure 8.8: Illustration of the Stereotype transformation in the Context Acquisition & Modelling Diagram, to the Java class for the context mapper in the Android Context Reasoner. Where text between square brackets represents an attribute from the stereotype in the form of *Stereotype.name*.**


Stereotypes in the Context Situation Detection Diagram
«stereotype»
SituationOfInterest
id: String
description: String
frequency: FrequencyLevel
feasibility: FeasibilityType
«stereotype»
DetectionPlan
id: String
description: String
recommendation: RecommendationType
toBeImplemented: boolean
failureLikelihood: LikelihoodLevel
failureImpact: ImpactLevel
Transformation
Custom Context Mapper Class in the Context Reasoner Library
«class»
CustomContextMapper
CustomContextMapper.java

```
package edu.casetools.icase.custom;
import android.content.Context;
import org.poseidon_project.context.ContextReasonerCore;
import org.poseidon_project.context.reasoner.AbstractContextMapper;
import org.poseidon_project.context.reasoner.ReasonerManager;

[forEach SituationOfInterest where
  ( detects(DetectionPlan, SituationOfInterest)
    and DetectionPlan.toBeImplemented = true )]
    import edu.casetools.icase.custom.situations.[SituationOfInterest.name]SOI;
[end forEach]

public class CustomContextMapper extends AbstractContextMapper {

    public CustomContextMapper(ContextReasonerCore crc, ReasonerManager rm, Context con) {
        super("CustomContextMapper",crc,rm,con);
        initialiseSituationsOfInterest();
    }

    private void initialiseSituationsOfInterest() {
    [forEach SituationOfInterest where
      ( detects(DetectionPlan, SituationOfInterest)
        and DetectionPlan.toBeImplemented = true )]
          situationsOfInterest.add(new [SituationOfInterest.name]SOI());
    [end forEach]
    }
}
```

**Figure 8.9: Illustration of the pseudo-code rules for transforming stereotypes in the Situation Detection Diagram, to the Java class for the context mapper in the Android Context Reasoner.**

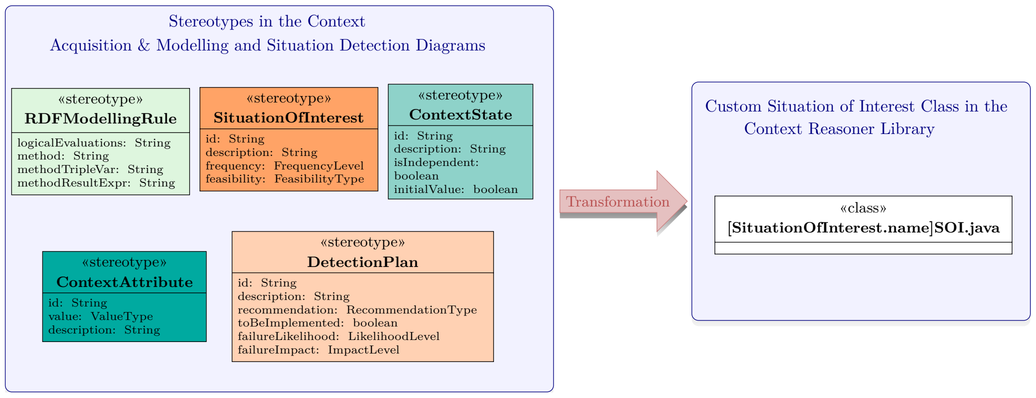


**Figure 8.10: Illustration of the pseudo-code rules for transforming stereotypes in the Situation Detection Diagram, to the Java class for the context mapper in the Android Context Reasoner. For space reasons, the tranformation rules can be found in Appendix A.4.C.**

### 8.7.1.5 Situation of Interest Class

This abstract class enables registering and de-registering of the context modelling and reasoning rules of a situation of interest at the same time. In this way it allows the handling of the different situations of interest separately. Figure 8.10 illustrates the pseudo-code transformations rules for the automatic code transformation of this class.

### 8.7.1.6 Ontology Manager Class

This is a simple class containing (in different Java Strings) the base of the main ontology, and the IRI for the stream which will be used in the C-SPARQL engine. Firgure 8.6 illustrates the pseudo-code rules for the automatic code transformation of this class.

### 8.7.1.7 Example

This example is a continuation of the example introduced in Section 6.4.1.1 and it assumes that all the corresponding tools have been installed as indicated in Appendix A. As it is illustrated in Figure 8.4, the generated code is related to the context library, context reasoner library and the developed application. Although the code could be managed directly in the developed application, in order to make the libraries more reusable for future projects, it is suggested divide the files as indicated in Figure 8.4. In order to generate the code corresponding to the acquisition of context, related to the models appearing in Figure 6.6, the developer (interacting with the tool GUI) needs to select and click with the right mouse button the root folder of the Modelio model (left folder hierarchy panel). Then, the user needs to click on *Design for C-ASE -> Context Acquisition and Modelling -> Generate mobile acquisition*. This will generate the NavigationSystemReceiver.java file. Note that for this example, no observers and no actuators are generated. In this case, the driver required for obtaining the temperature can directly be the LocationWeatherObserver.java class of the context library. In order to generate the code corresponding to the modelling and reasoning of each situation of interest, the developer needs to right click on the root folder of the Modelio model (left folder hierarchy panel). Click on *Design for C-ASE -> Context Acquisition and Modelling -> Generate mobile modelling and reasoning rules*. This will generate the CustomContextMapper.java file, containing all the references to the situations of interest, and a Java class for each situation of interest in the project, containing all the modelling and reasoning rules. A code snippet of the CSPARQL rules produced for part of the situation of interest represented in the example in Figure 6.6, is shown in Figure 8.11. As it can be observed in this figure, the implementation allows the preference corresponding to *$$ColdTemperaturePreference* to be changed at execution time. In order to relocate the files, the following project hierarchy is suggested. The CustomContextMapper.java file is located into the *edu.casetools.icase.custom* package. Each of the situations of interest is located into the *edu.casetools.icase.custom.situations* package.

### 8.7.2 Process for stationary platforms

This subsection explains in more detail how the code is automatically generated, compiled, executed and deployed, for stationary platforms as it is illustrated in Figure 8.12. The figure maps the different Design models to the different software artefacts of the reasoner, as well as the steps that the developers need to follow for this purpose. The first step consists of generating the code from the different Design models, which can be done as further explained in Section 8.7.2.8. Note that there are two types of code that will be generated in this case, Java code and text files. The Java files contain code that needs to be integrated with the reasoner library. Particularly with the graphical user interface version of the M Reasoner [206]. The text files will be loaded once the

```
private static final String coldTemperature_query =
  "REGISTER QUERY coldTemperature_query AS
  PREFIX
  ex:<http://ie.cs.mdx.ac.uk/POSEIDON/envir#>

  CONSTRUCT {
  ex:temperaturesensor
  http://ie.cs.mdx.ac.uk/POSEIDON/context/is
   \"coldTemperature\"
  }

  FROM STREAM
  <http://poseidon-project.org/context-stream>
  [RANGE 10s STEP 4s]

  WHERE {
  ?m ex:hasTemperatureValue ?tempValueIRI .
  ?tempValueIRI ex:temperatureValue ?tempValue .
  FILTER ( ?tempValue < $$ColdTemperaturePreference )
  }";

public String getcoldTemperature_query(String value) {
  String coldTemperature_queryString =
    new String(coldTemperature_query);
  return coldTemperature_queryString.replace(
  "$$ColdTemperaturePreference",
  String.valueOf(value));
}
```

**Figure 8.11: Code snippet of the C-SPARQL code generated by Modelio with regard to the model of Figure 6.6. Note that this code generates Java clases. The engine for producing the content inside the string has been reused from that presented in [3], which is publicly available in [4].**

reasoner has been compiled and it is executing. The second step consists of relocating the Java code into the corresponding reasoner library [206]. Appendix A describes how the library can be downloaded and installed. Once the code is relocated inside the library, the next step is to compile the project and run it, which can be done following the instructions in Appendix A. Then, the specifications and configurations are loaded into the M Reasoner, as explained in Appendix A, A, and A. Finally, following the instructions of Appendix A, the M Reasoner is executed. This will automatically deploy all the required elements and run the system. Note that developers can also simulate the system. The remainder of this section will explain the different model to text transformations required to produce each of the files that can be observed in Figure 8.12. Note that for the rest of the chapter, model to text transformations will be illustrated in figures, where text between square brackets represents an attribute from the stereotype in the form of *Stereotype.name*.

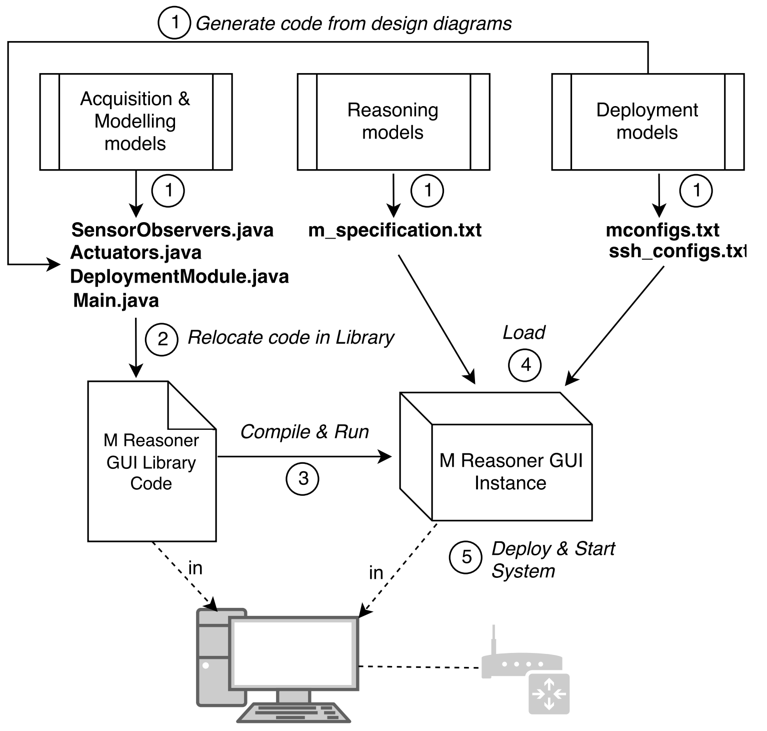
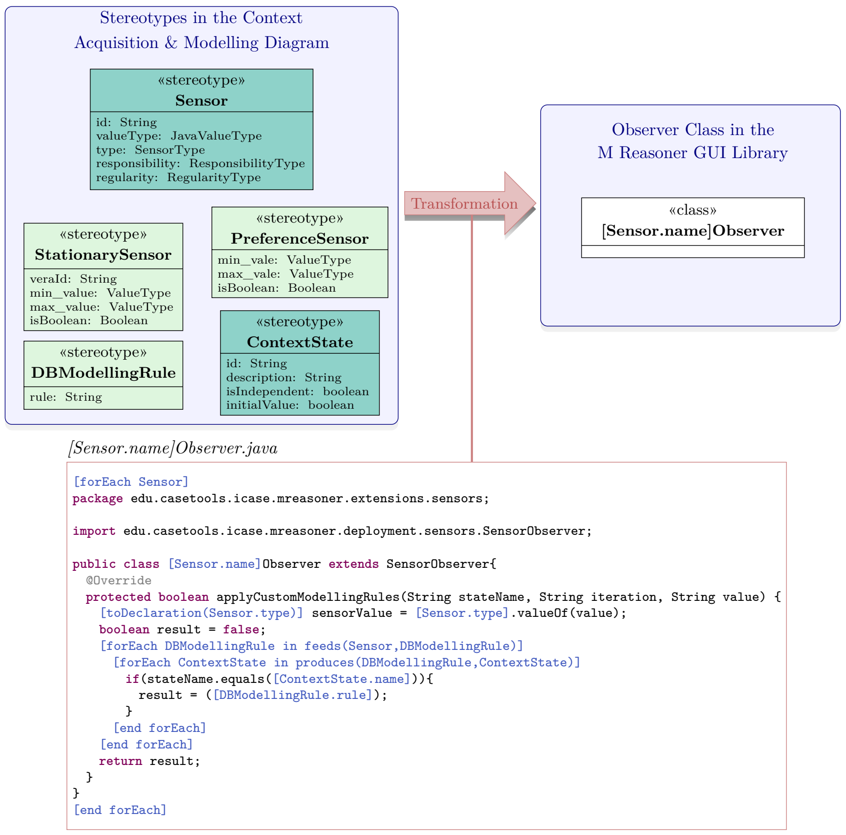


**Figure 8.12: Map of the software artefacts produced by the Acquisition & Modelling, Reasoning and Deployment diagrams, and their relation to the M Reasoner code and system.**


Stereotypes in the Context Acquisition & Modelling Diagram
«stereotype» Sensor
id: String
valueType: JavaValueType
type: SensorType
responsibility: ResponsibilityType
regularity: RegularityType
«stereotype» StationarySensor
veraId: String
min_value: ValueType
max_value: ValueType
isBoolean: Boolean
«stereotype» PreferenceSensor
min_vale: ValueType
max_vale: ValueType
isBoolean: Boolean
«stereotype» DBModellingRule
rule: String
«stereotype» ContextState
id: String
description: String
isIndependent: boolean
initialValue: boolean
Transformation
Observer Class in the M Reasoner GUI Library
«class» [Sensor.name]Observer
[Sensor.name]Observer.java

```
[forEach Sensor]
package edu.casetools.icase.mreasoner.extensions.sensors;

import edu.casetools.icase.mreasoner.deployment.sensors.SensorObserver;

public class [Sensor.name]Observer extends SensorObserver{
  @Override
  protected boolean applyCustomModellingRules(String stateName, String iteration, String value) {
    [toDeclaration(Sensor.type)] sensorValue = [Sensor.type].valueOf(value);
    boolean result = false;
    [forEach DBModellingRule in feeds(Sensor,DBModellingRule)]
      [forEach ContextState in produces(DBModellingRule,ContextState)]
        if(stateName.equals([ContextState.name])){
          result = ([DBModellingRule.rule]);
        }
      [end forEach]
    [end forEach]
    return result;
  }
}
[end forEach]
```

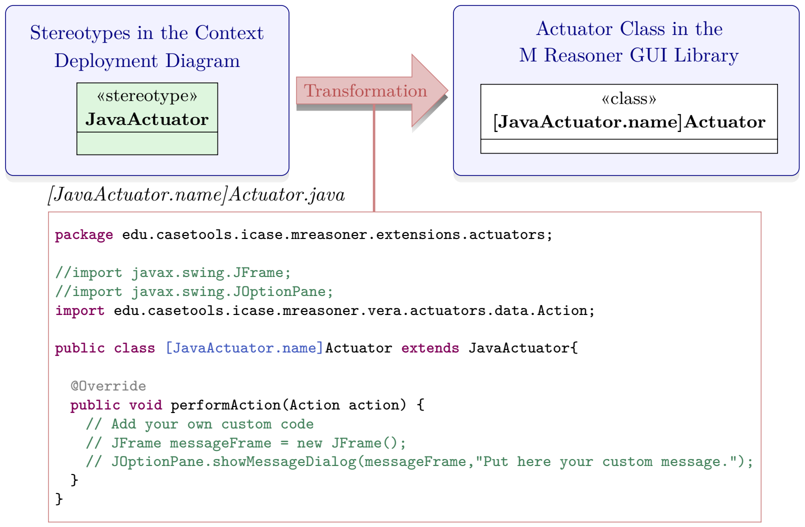
**Figure 8.13: Illustration of the pseudo-code rules for transforming stereotypes in the Context Acquisition & Modelling Diagrams, to the SensorObserver type of class in the M Reasoner GUI Library. Where $toDeclaration()$ is a function that returns the corresponding Java declaration type for each JavaValueType (*e.g.,* for an Integer value of the JavaValueType it returns *int*).**

#### 8.7.2.1 Sensor Observer Class

For each Sensor stereotype in the design models, a SensorObserver Class is generated, which contains the set of modelling rules which are applied to the same sensor, and that produce a set of context states. Figure 8.13 shows the pseudo-code of the rules for generating this code, using the stereotypes from the Context Acquisition & Modelling Diagram.

#### 8.7.2.2 Actuator Class

In the Deployment Diagrams, there are two stereotypes corresponding to actuators in the stationary platform: VeraActuator and JavaActuator. In the case of the VeraActu-


Stereotypes in the Context Deployment Diagram
«stereotype»
JavaActuator
Transformation
Actuator Class in the M Reasoner GUI Library
«class»
[JavaActuator.name]Actuator
[JavaActuator.name]Actuator.java

```
package edu.casetools.icase.mreasoner.extensions.actuators;

//import javax.swing.JFrame;
//import javax.swing.JOptionPane;
import edu.casetools.icase.mreasoner.vera.actuators.data.Action;

public class [JavaActuator.name]Actuator extends JavaActuator{

  @Override
  public void performAction(Action action) {
    // Add your own custom code
    // JFrame messageFrame = new JFrame();
    // JOptionPane.showMessageDialog(messageFrame,"Put here your custom message.");
  }
}
```

**Figure 8.14: Illustration of the pseudo-code rules for transforming stereotypes in the Context Deployment Diagrams, to the Actuator type of class in the M Reasoner GUI Library.**

ator stereotype, the reasoner library listens to changes in the state that triggers a corresponding actuator. If a change in that state is detected, the reasoner library sends an http request to the Vera router, containing the action that the device should perform. Then, the vera router makes that change to the corresponding ZWave actuator. In the case of VeraActuator stereotypes, there is no need to generate an additional class, as the existing class VeraActuator of the reasoner can be used for this purpose. On the other hand, for the JavaActuator stereotypes, the reasoner is equally listening for changes in the associated states, in the same way that it listens to associated states for VeraActuators. Once a change in the state that triggers the actuator is detected, the code in the $performAction$ function is executed. This function is executed in Java code that is controlled from the reasoner library. As it can be observed in Figure 8.14, the JavaActuator stereotypes generate the code skeleton for implementing the functionality of the actuator. In this case, the developers need to implement the functionality manually.

### 8.7.2.3 Deployment Module Class

The DeploymentModule class contains information about all the actuators and sensor observers to be deployed in the system. Note that the SensorObserver class also contains the Sensor class, which is intended to contain the information required by the database about the relevant sensor information and its associations to the M Reasoner states, as it contains a list with all the Actuators of the system, and all its SensorObservers. Figure 8.15 shows how the DeploymentModule class of the reasoner is generated.

#### 8.7.2.4 Main File Class

The main file is the Java file that is used for executing the M Reasoner GUI. Figure 8.18 shows how this file is generated. As it can be observed, it simply initialises the controller, model and view of the Model-View-Control (MVC) architecture of the library. The system requires an AbstractDeploymentModule that contains information to help the deployment of the system in the real environment modality. The automatically generated class from Figure 8.15 extends from the AbstractDeploymentModule.

#### 8.7.2.5 Specification

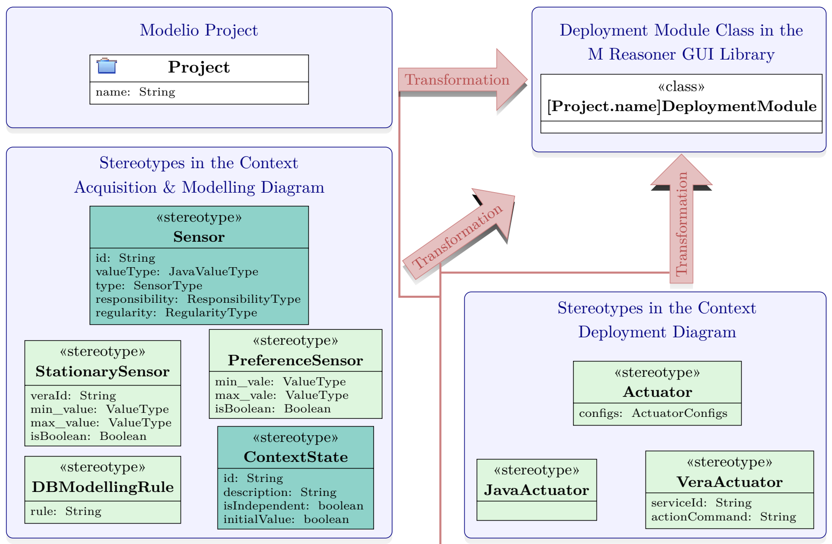
The M specification is a simple text file containing the states and rules which indicate the reasoning to be performed by the M Reasoning system. This file can easily be loaded to the reasoner, and it contains information about all the states and the rules that are going to be applied to them [162]. Figure 8.16 shows the mapping between Reasoning Diagram elements and the specification.

#### 8.7.2.6 Configurations

The M configurations is a simple text file containing information about how the algorithm should be executed and the data required for connecting to the database. This file can easily be loaded into the reasoner. Figure 8.17 shows the mapping between the Deployment Diagram elements and the configurations.

#### 8.7.2.7 Secure Shell Configurations

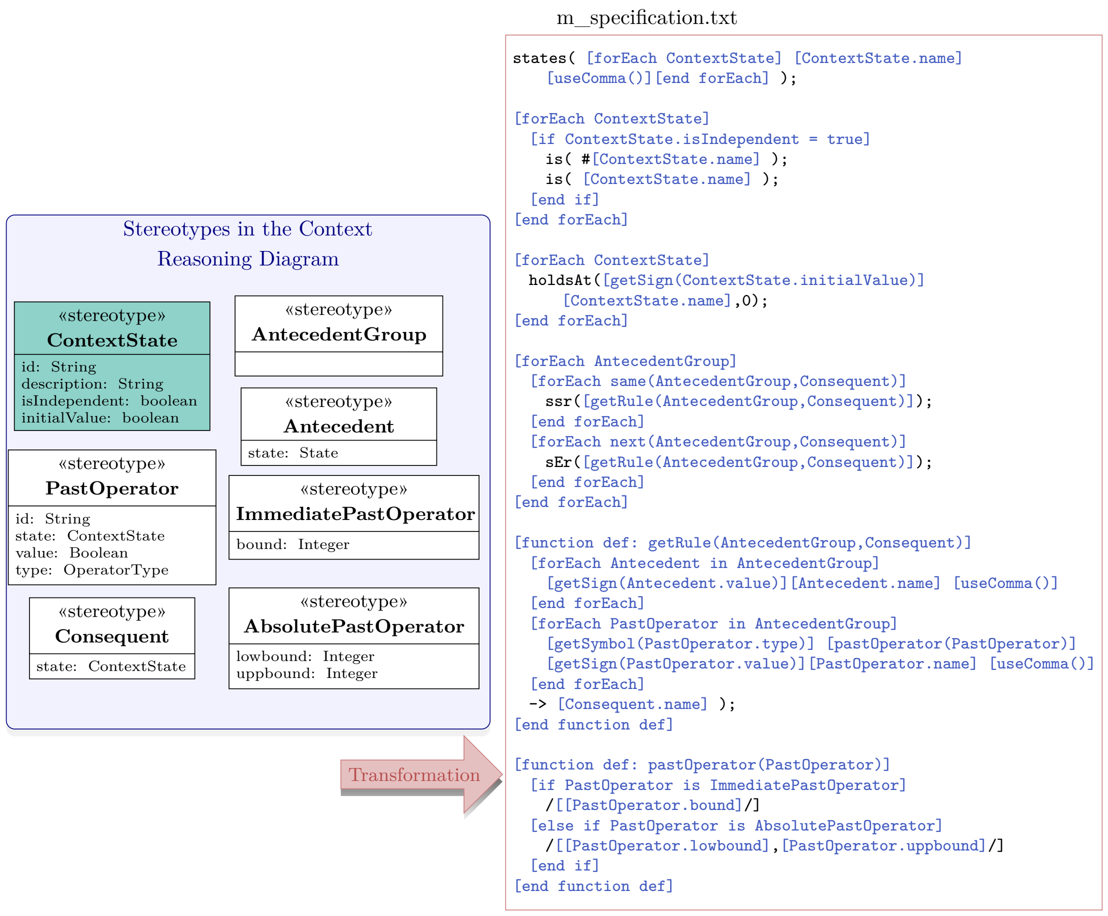
There are other additional configurations required to connect to a specific Vera router through Secure Shell (SSH), in order to read the information about changes in the



*[Project.name]DeploymentModule.java*

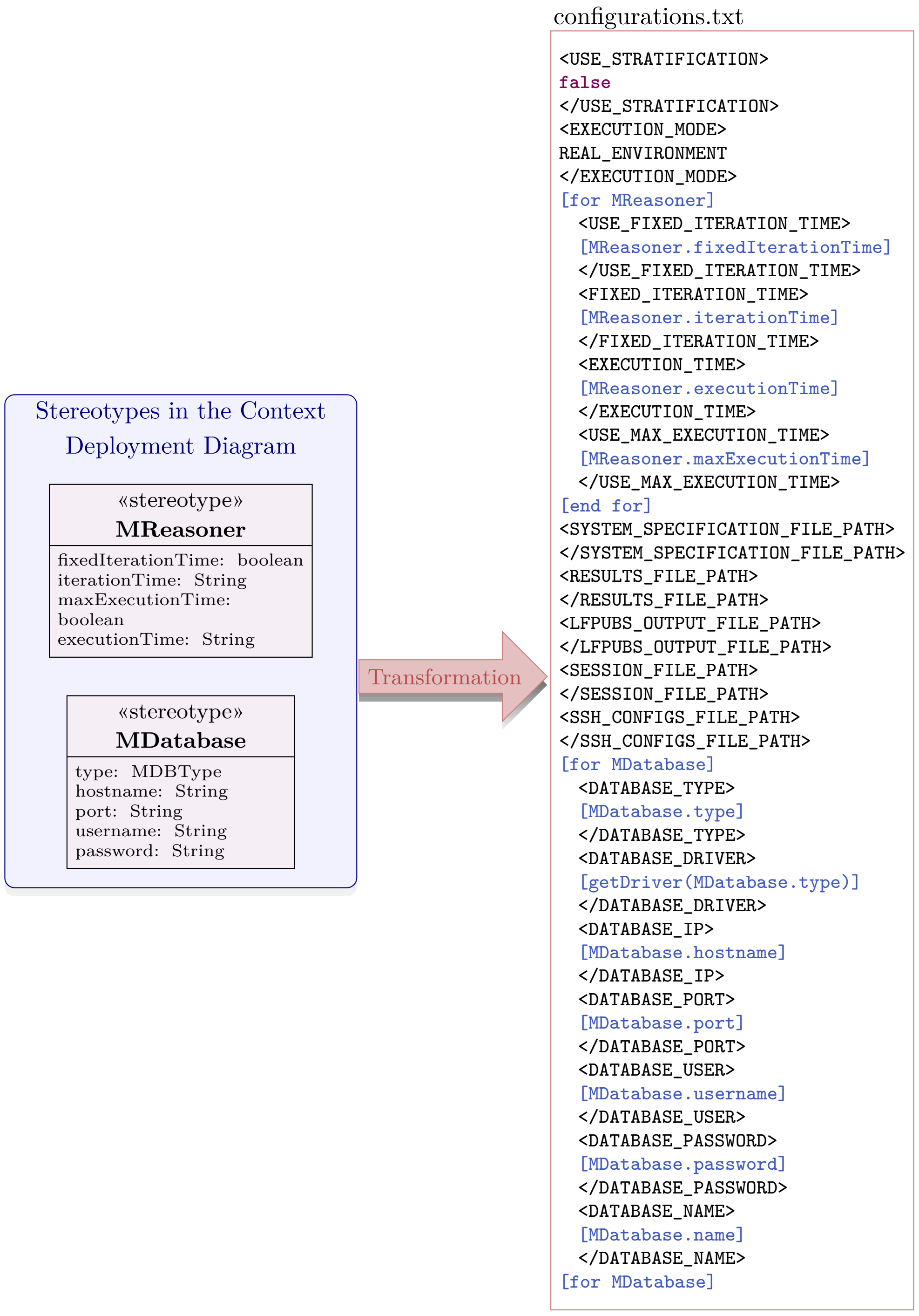
```
package edu.casetools.icase.mreasoner.extensions.modules;

import java.util.Vector;
import edu.casetools.icase.mreasoner.database.core.MDBImplementations;
import edu.casetools.icase.mreasoner.deployment.realenvironment.AbstractDeploymentModule;
import edu.casetools.icase.mreasoner.deployment.sensors.Sensor;
[forEach Sensor]
  import edu.casetools.icase.mreasoner.extensions.sensors.[Sensor.name]Observer;
[end forEach]
import edu.casetools.icase.mreasoner.vera.actuators.device.VeraActuator;
[forEach VeraActuator or JavaActuator]
import edu.casetools.icase.mreasoner.extensions.actuators.[Actuator.name]
[end forEach]
public class [Project.name]DeploymentModule extends AbstractDeploymentModule {

  public [Project.name]DeploymentModule(){
    super();
  }

  @Override
  protected void initialiseSensorObservers() {
    [forEach Sensor]
      initialise[Sensor.name]SensorObserver();
    [end forEach]
  }

  [forEach Sensor]
    private void initialise[Sensor.name]SensorObserver() {
      Vector<String> states      = initialise[Sensor.name]SensorStates();
      [Sensor.name]Observer temperatureObserver = new [Sensor.name]Observer();
      Sensor [Sensor.name]Sensor     = new Sensor("[Sensor.id]","[Sensor.name]",
      "[Sensor.model]", "[Sensor.location]", "[Sensor.valueType]".toUpperCase(),
      "[Sensor.min_value]", "[Sensor.max_value]",
      "[Sensor.isBoolean()]".toLowerCase(), states);
      [Sensor.name]Observer.setSensor(temperatureSensor);
      this.sensorObservers.add([Sensor.name]Observer);
    }

    private Vector<String> initialise[Sensor.name]SensorStates() {
      Vector<String> states = new Vector<String>();
      [forEach DBModellingRule in feeds(Sensor,DBModellingRule)]
        [forEach ContextState in produces(DBModellingRule,ContextState)]
          states.add("[ContextState.name]");
        [end forEach]
      [end forEach]
      return states;
    }
  [end forEach]

  @Override
  protected void initialiseActuators(){
    [forEach VeraActuator]
      VeraActuator [VeraActuator.name]Actuator = new VeraActuator(
      "[VeraActuator.serviceId]","[VeraActuator.action]");
      this.actuators.add(lampActuator);
    [end forEach]
    [forEach JavaActuator]
      ExampleMessageDisplayActuator messageActuator = new ExampleMessageDisplayActuator();
      this.actuators.add(messageActuator);
    [end forEach]
  }
}
```

**Figure 8.15: Illustration of the pseudo-code rules for transforming stereotypes in the Context Acquisition & Modelling Diagrams, and Deployment Diagrams, to the Deployment Module class in the M Reasoner GUI Library.**



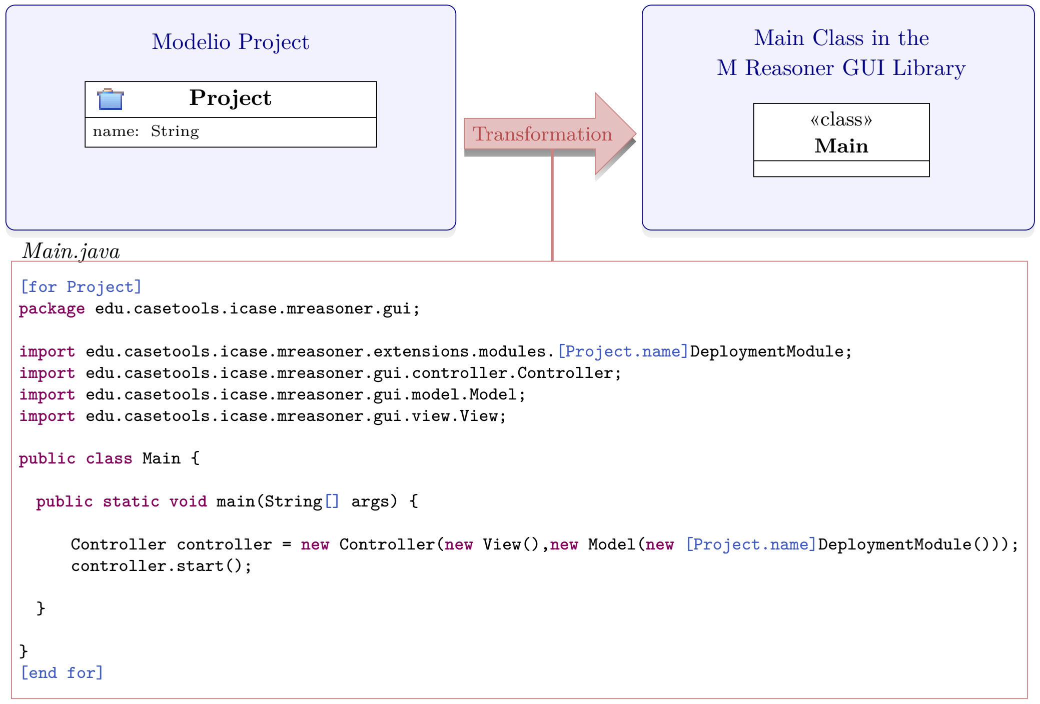
```
states( [forEach ContextState] [ContextState.name]
    [useComma()][end forEach] );

[forEach ContextState]
  [if ContextState.isIndependent = true]
    is( #[ContextState.name] );
    is( [ContextState.name] );
  [end if]
[end forEach]

[forEach ContextState]
  holdsAt([getSign(ContextState.initialValue)]
      [ContextState.name],0);
[end forEach]

[forEach AntecedentGroup]
  [forEach same(AntecedentGroup,Consequent)]
    ssr([getRule(AntecedentGroup,Consequent)]);
  [end forEach]
  [forEach next(AntecedentGroup,Consequent)]
    sEr([getRule(AntecedentGroup,Consequent)]);
  [end forEach]
[end forEach]

[function def: getRule(AntecedentGroup,Consequent)]
  [forEach Antecedent in AntecedentGroup]
    [getSign(Antecedent.value)][Antecedent.name] [useComma()]
  [end forEach]
  [forEach PastOperator in AntecedentGroup]
    [getSymbol(PastOperator.type)] [pastOperator(PastOperator)]
    [getSign(PastOperator.value)][PastOperator.name] [useComma()]
  [end forEach]
  -> [Consequent.name] );
[end function def]

[function def: pastOperator(PastOperator)]
  [if PastOperator is ImmediatePastOperator]
    /[[PastOperator.bound]/]
  [else if PastOperator is AbsolutePastOperator]
    /[[PastOperator.lowbound],[PastOperator.uppbound]/]
  [end if]
[end function def]
```

**Figure 8.16: Illustration of the pseudo-code rules for transforming stereotypes in the Reasoning Diagrams, to the specification text file for the M Reasoner GUI Library. Where $useComma()$ is a function that inserts a comma if there is another element remaining, $useAnd()$ is a function that inserts the $\wedge$ symbol if there is another element remaining, $getSymbol\ (OperatorType)$ returns "[-]" for STRONG OperatorType, and "<->" for WEAK OperatorType, $getSign(boolean)$ returns "#" for *false* value of the input variable, and nothing for *true* value. Note that brackets with a slash ("/[" "/]") represent text to be transformed.**



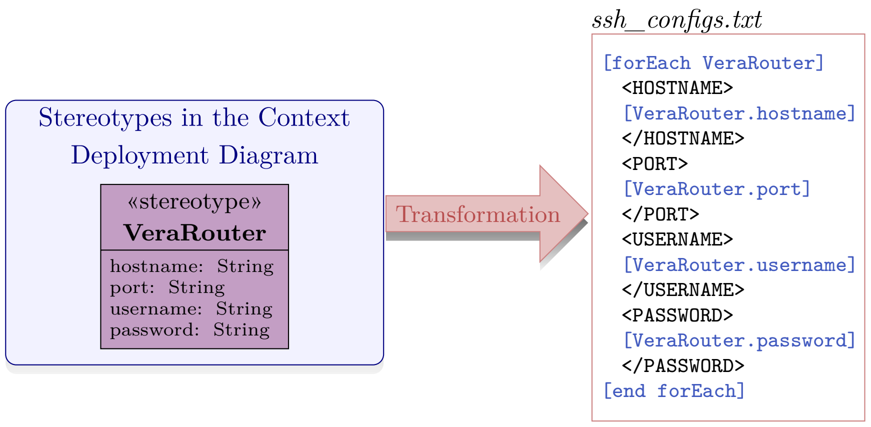
```
<USE_STRATIFICATION>
false
</USE_STRATIFICATION>
<EXECUTION_MODE>
REAL_ENVIRONMENT
</EXECUTION_MODE>
[for MReasoner]
  <USE_FIXED_ITERATION_TIME>
  [MReasoner.fixedIterationTime]
  </USE_FIXED_ITERATION_TIME>
  <FIXED_ITERATION_TIME>
  [MReasoner.iterationTime]
  </FIXED_ITERATION_TIME>
  <EXECUTION_TIME>
  [MReasoner.executionTime]
  </EXECUTION_TIME>
  <USE_MAX_EXECUTION_TIME>
  [MReasoner.maxExecutionTime]
  </USE_MAX_EXECUTION_TIME>
[end for]
<SYSTEM_SPECIFICATION_FILE_PATH>
</SYSTEM_SPECIFICATION_FILE_PATH>
<RESULTS_FILE_PATH>
</RESULTS_FILE_PATH>
<LFPUBS_OUTPUT_FILE_PATH>
</LFPUBS_OUTPUT_FILE_PATH>
<SESSION_FILE_PATH>
</SESSION_FILE_PATH>
<SSH_CONFIGS_FILE_PATH>
</SSH_CONFIGS_FILE_PATH>
[for MDatabase]
  <DATABASE_TYPE>
  [MDatabase.type]
  </DATABASE_TYPE>
  <DATABASE_DRIVER>
  [getDriver(MDatabase.type)]
  </DATABASE_DRIVER>
  <DATABASE_IP>
  [MDatabase.hostname]
  </DATABASE_IP>
  <DATABASE_PORT>
  [MDatabase.port]
  </DATABASE_PORT>
  <DATABASE_USER>
  [MDatabase.username]
  </DATABASE_USER>
  <DATABASE_PASSWORD>
  [MDatabase.password]
  </DATABASE_PASSWORD>
  <DATABASE_NAME>
  [MDatabase.name]
  </DATABASE_NAME>
[for MDatabase]
```

**Figure 8.17: Illustration of the pseudo-code rules for transforming stereotypes in the Context Deployment Diagram, to the configuration txt file of the M Reasoner. Where text between square brackets represents an attribute from the stereotype in the form of *Stereotype.name*; and where $getDriver()$ is a function that returns the driver used by each database type (*e.g.,* for an Integer value of the MDBType it returns *int*).**



```
[for Project]
package edu.casetools.icase.mreasoner.gui;

import edu.casetools.icase.mreasoner.extensions.modules.[Project.name]DeploymentModule;
import edu.casetools.icase.mreasoner.gui.controller.Controller;
import edu.casetools.icase.mreasoner.gui.model.Model;
import edu.casetools.icase.mreasoner.gui.view.View;

public class Main {

  public static void main(String[] args) {

      Controller controller = new Controller(new View(),new Model(new [Project.name]DeploymentModule()));
      controller.start();

  }

}
[end for]
```

**Figure 8.18: Illustration of the pseudo-code rules for transforming the modelio project into the Main class in the M Reasoner GUI Library.**



```
[forEach VeraRouter]
  <HOSTNAME>
  [VeraRouter.hostname]
  </HOSTNAME>
  <PORT>
  [VeraRouter.port]
  </PORT>
  <USERNAME>
  [VeraRouter.username]
  </USERNAME>
  <PASSWORD>
  [VeraRouter.password]
  </PASSWORD>
[end forEach]
```

**Figure 8.19: Illustration of the pseudo-code rules for transforming stereotypes in the Context Acquisition & Modelling Diagrams, to the configuration text file for the M Reasoner GUI Library.**

sensors, to then store them in the database. For each VeraRouter stereotype, a corresponding SSH configurations file is generated, as it is shown in Figure 8.19.

#### 8.7.2.8 Example

This example is a continuation of the example introduced in Section 6.4.1.2 and it assumes that all the corresponding tools have been installed as indicated in Appendix A. As it is illustrated in Figure 8.12, some of the generated code (.java) is related to the mreasoner-gui [206] library, and some other code (.txt) is related to the execution of the mreasoner-gui. In order to generate the code corresponding to the acquisition and modelling of context, related to the models appearing in Figure 6.7, the developer needs to select and click with the right button the root folder of the Modelio model (left folder hierarchy panel). Then the user should right click on *Design for C-ASE -> Context Acquisition and Modelling -> Generate stationary platform acquisition and modelling code (PostgreSQL)*. There is also a choice to generate the same code, but oriented to MySQL if this is the preferred database. This action will generate the following files for this example: *PresenceSensorObserver.java, CookerSwitchSensorObserver.java, CookerUnattendedExampleDeploymentModule.java, CookerActuator.java* and *Main.java*.

These files have to be merged with the existing project which uses the mreasoner-gui library. The following project hierarchy is suggested. The Main.java generated file should be located in the main package of the project, in the *edu.casetools.icase.mreasoner.gui* package. The *CookerUnattendedExampleDeploymentModule.java* file goes into the *edu.casetools.icase. mreasoner.extensions.modules* package of the mreasoner-gui project. The remaining two observers (*PresenceSensorObserver.java* and *CookerSwitchSensorObserver.java*) go into the *edu.casetools.icase.mreasoner.extensions.sensors* package. Finally, the actuator CookerActuator.java goes into the *edu.casetools.icase. mreasoner.extensions.actuators* package. Once the code is merged with the current project, there is a need to compile and execute the code using the Main.java file. This will launch the mreasoner-gui interface. In order to generate the corresponding code for this platform, the developer should click on *Design for C-ASE -> Context Reasoning -> Generate stationary reasoning rules* and on *Design for C-ASE -> Context Deployment*

-> *Generate stationary platform configurations*. This will produce the specifications and the configurations for the M Reasoner for this example, as shown in Figure 8.20. The user just needs to click on the main menu of the mreasoner-gui interface and go to Session -> Load Session. Then, select the generated file in order to load the rules and configurations to the mreasoner-gui execution instance. Finally, the developers have to configure the Z-Wave sensors into the Vera router. For this, simply follow the instructions provided by the company, which depend on the particular router used. The stationary system is now ready to be deployed.

## 8.8 Conclusions

This chapter proposes a framework for implementing, deploying and maintaining context-aware systems, that facilitates code generation from previously introduced requirements and design models. The overall purpose is to speed up the development process for supporting the dynamic changes caused by the dynamic nature of context. The main contributions of this chapter are:

- A guide for the implementation, deployment and maintenance of context-aware systems, supporting the creation of systems in mobile and stationary platforms.
- A set of tools to support the automatic code generation for requirements and Design models.
- The adaptation of reasoning tools for stationary platforms according to the objectives introduced in Section 1.4.
- The adaptation of reasoning tools for mobile platforms according to the objectives introduced in Section 1.4.

```
/***********************************************************
*               *
* This file has been automatically generated as part of the *
* DCASE module for the project NavigationSystem.
*               *
***********************************************************/

 states(atKitchen, cookerOn, cookerUnattendedState, hazardState);

 is(atKitchen);
 is(#atKitchen);
 is(cookerOn);
 is(#cookerOn);

 holdsAt(#cookerUnattendedState,0);
 holdsAt(#hazardState,0);

 ssr( (cookerOn ^ [-][120]#atKitchen ) ->cookerUnattended);
 ssr( ([-][60]cookerUnattended ) ->hazard);
 sEr( (#hazard ) ->#cookerOn);
 sEr( (#cookerOn ) ->#hazard);
```

**Figure 8.20: Code snippet of the M specification code generated by Modelio with regard to the model of Figure 6.7.**

# Part V

# Evaluation and critical reflection

Chapter 9

# Conclusions and future work

## 9.1 Introduction

The research work presented in this dissertation has aimed to cover the existing gaps in the specification and design of context-aware systems. Initial analysis of the different methodologies for creating these types of systems has revealed that existing approaches do not cover sufficiently the most common development stages of the life-cycle of a software application. This motivated research focused on the key stages of a software process (*i.e.* requirements, analysis and design, implementation, verification, deployment and maintenance ) each of the different stages, leading to the conclusions that the state-of-the-art approaches were scattered, and typically disconnected. This thesis has covered the creation of a backbone framework which covers the most common stages of the development process, and which is open-source, for other researchers and developers to contribute. This, is specifically targeted at the design and implementation of context-aware systems, and aims to support developers through the most common development life-cycle stages. It supports the creation of systems in any area that uses context-awareness, such as Ubiquitous Computing, Ambient Intelligence, Ambient Assisted Living or Intelligent Environments. Figure 3.2 represents the whole framework. More explanations and the full example used along this thesis can be found in [? ]. This research has resulted in the following contributions:

- C1 A novel conceptualisation approach for developing context-aware systems which: A) Takes into account the real capabilities of such systems; B) Is tailored to the needs and preferences of the end-user stakeholders for which context awareness is a primary consideration.
- C2 An enhanced requirements engineering model that specifically addresses the elicitation of context-aware features of software systems that have a primary focus of enabling a platform for an intelligent environment. This technique introduces a novel approach to the discernment of 'service triggers' with respect to situational detection. The work has demonstrated an improved understanding for developers with respect to two dimensions: A) The discovery of services which are tailored to the end-user stakeholder's needs and preferences in a particular situation; and B) The real capabilities, taking into account particular project constraints, of the system providing adequate services in those situations.

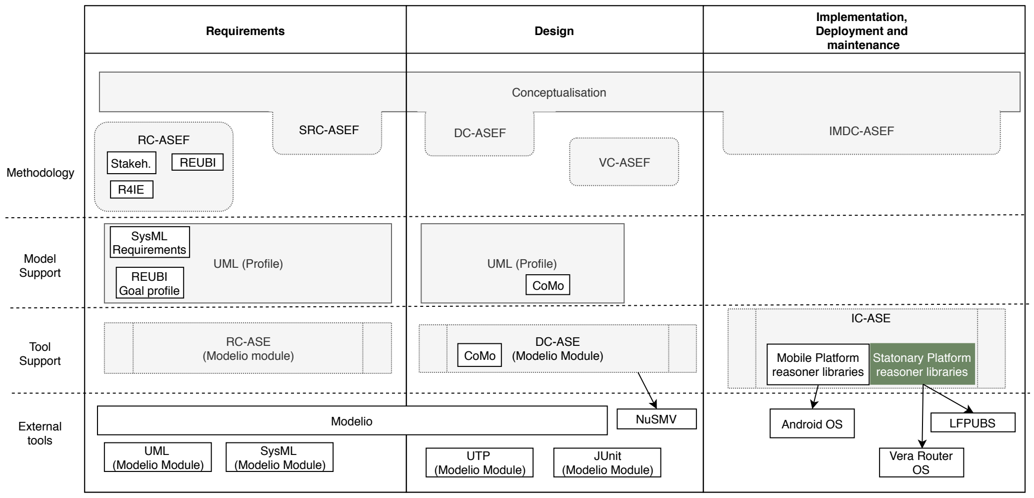


**Figure 9.1: Representation of the different parts conforming the C-ASEF. Light grey boxes represent parts which have been developed as part of the contribution of this thesis. White boxes represent parts that have been adopted. The darkest grey box represents a mixture of both.**

C3 A bespoke UML/SysML diagram-based profile model that supports the enhanced requirements elicitation methodology [C2], and thus further enabling:

- The representation of the concepts indicated by contribution C2.
- The representation of the M reasoning language [162].
- The representation of context-reasoning verification specifications and its relation to particular requirements.

C4 The extension of an open-source tool to support model-based elicitation of requirements using the enhanced requirements engineering model [C2]. The extension takes a pure requirements analysis approach with respect to an integrated support tool that facilitates the handling of complex relations between the models. The extension includes/enables:

- The full implementation of SysML Requirements diagrams, including all its elements and relations, the requirements table and traceability matrix.
- Automatic generation of documentation
- The implementation of tables to manage the properties of context
- The visualization of the traceability relations between the different ele-

ments in the system. This enables trace management by, for example, letting the developer observe the specific elements that will be affected after adding/modifying/removing a situation of interest from the system.

C5 The extension of an open-source tool that supports the design of rules [C3], compatible with different reasoning middleware libraries that support the implementation and deployment of context awareness in mobile or stationary platforms, by:

- C5.A Automatically generating code for an external context-aware sensor middleware for mobile devices.
- C5.B Automatically generating rule models for an external smart-home environment based middleware.

C6 A conceptual approach that enables the mapping of models specified in the M language [162] to models specified in the NuSMV [202] verification language. This enables the verification of M rules in the NuSMV model checker.

C7 The implementation of C6 as an integrated open-source module which enables the automatic generation of verification models from M rule models. The implementation integrates with other, already existing, open source tools:

- C7.A CoMo [3]: A context modelling tool that leverages the level of complexity in the definition of rules, to enable context-aware stream reasoning in mobile applications.
- C7.B M-Reasoner [162]: A graphical interface tool for reasoning in smart-home environments that facilitates the deployment of systems which use Z-Wave (radio) based sensors.

The original tool corresponding to [C7.A] has been refactored to facilitate the automation of code generation. Additionally, the library corresponding to the reasoner for mobile platforms has also been refactored in order to better accommodate the code generation. The tool presented in [C7.B] has been made open-source [206], mavenised, and some major refactorisations have been applied to the code, including its division into different reusable libraries.

## 9.2 Reflecting on the objectives and future lines

Section 1.1 introduces the Reader to the main challenges in the development of context-aware systems. It is not the intention of this dissertation to solve them all, but rather to use them for guiding the achievement of the main objective of this thesis: *Laying the foundations of a user-centred and open-source tool-supported framework for aiding the developing of context-aware systems throughout the most common stages of their development life-cycle*.

### 9.2.1 Conceptualisation

This dissertation has studied the different existing definitions of the concept of context, concluding that there is no consensus on what this term means. Guided by the first challenge [Chg1] of the analysis presented in Section 1.1, a deeper investigation into the motives behind this lack of consensus has shed light on the main cause: the philosophical tradition behind the traditional understanding of context is opposite to that underlying computer science. Although the resolution of this particular challenge is out of the scope of this dissertation, an approach to the conceptualisation of context in the area of context-aware computing has been presented [C1] [C2], acknowledging the synergies between these two opposite philosophical traditions. On the one hand, the identification of situations of interest is understood as a subjective process of understanding that the developer performs. On the other hand, guidance is provided for developers to translate their understanding into a set of programming models which can help to reproduce, to some extent, the identification of situations and the triggering of its corresponding services.

The conceptualisation of context presented in this thesis, does not intend to provide solutions to all of the conceptualisation challenges for the development of context-aware systems. Rather, this thesis intends to encourage future research towards creating more usable context-aware systems by mitigating the negative aspects of open challenges, and focusing on the positive aspects of what it can be achieved with state-of-the-art techniques. The following reflection on the current conceptualisation is presented, compared against the first objective of this thesis, introduced in Section 1.4.

- Even under the scope of this novel conceptualisation, the development of context-aware systems remains a difficult task, and there is a high chance that developed systems will take actions which might not be the most appropriate in certain situations. In order to mitigate this drawback, a conceptualisation of the interactions between humans and context-aware systems has been introduced. Enabling developers to choose the most adequate interaction type according to the situation, it helps to minimise the effects of the context-aware system not being able to fully understand a certain situation.

- It is not the aim of this conceptualisation to enable developers to discover all possible situations of interest which may occur, as this might be a very difficult or even an impossible task. Rather, the aim is to force developers to reflect on these situations of interest as part of the development process, and during early stages of the development life-cycle. Also, in order to provide better services, there is a greater emphasis in the identification and understanding of stakeholders and their particular profiles. This information helps developers to understand better the meaning of their actions, and to provide services which are more adequate to their preferences and needs.
- It may occur that developers come up with great services to provide in certain situations, but it might also happen that it is not possible for a computer to identify such situations. In such circumstances, it is important to reject the implementation of the detection mechanisms of the situation and its associated services at early stages of the development process. For this reason, an evaluation technique has been created to consider which features should not be implemented. Such an approach can help to balance the expectations that some stakeholders might have on the real abilities of the context-aware system under development.
- There are some situations that developers identify that might seem similar a priori, but can greatly differ from the actual instantiation of the situation. In order to mitigate this, the proposed framework enables the tracing of the different situations of interest to the different associated design elements, and to evaluate any change that could be done to the situation of interest. In this way, the dynamic change of the context-aware system is enabled, by facilitating the maintenance of situations of interest which are already implemented. This can facil-

itate a relatively quick implementation of some nuances to distinguish between different situations of interest.

The current conceptualisation of context has room for further exploration. The following research directions can be explored more in depth:

- Researching traditional techniques for discovering adequate situations of interest and services for C-AS. Particular approaches such as phenomenology [219] [220], activity theory [221], ethnomethodology [79], or Merleau-Ponty's phenomenology [222] have been proposed. Further investigation might be beneficial in order to perfect current approaches to situation of interest discovery.
- Researching state-of-the-art techniques for discovering adequate situations of interest and services for C-AS. Advanced state-of-the-art computer technologies such as those provided by data science, machine learning or cognitive computing, look promising for facilitating developers with greater understanding of the meaning of the actions of the users in particular situations. An analysis of the data that is already available in the sensors of the system can also be used for giving feedback to developers, and help them unearth new SOIs or enhance/-modify the provision of existing context-aware features. There is scope for a theoretical foundation that allows developers to reach such understanding of user action through information technologies, rather than trying to make the machines do this process for them. For this particular goal, the LFPUBS tool for user behaviour discovery can be further studied, as well as its relation to the maintenance and evolution of requirements and situations of interest.
- The framework might also benefit from the inclusion of mechanisms to make the systems intelligible to the end-users, to make them aware about what the system is thinking, and the reasons why it is taking such decisions. This transparency exercise in systems can reduce end-user rejection [87].
- The algorithm introduced in Section 4.6.2 has been informally applied. This evaluation procedure can also be formalised. Also, further research can be done in what other formal techniques could be applied between the requirements and design models in order to evaluate other aspects.
- The conceptualisation presented in this thesis has provided a set of tools to deal

with the preferences of the users. Some issues still need to be explored, and are currently being investigated [223], which are particularly related to the handling of ambiguous preferences and the addressing of conflicting knowledge. For instance, in a smart-house scenario there might be a preference conflict if a user likes music with loud volume and another user prefers silence at the same time. The M reasoning theory can be further extended in order to handle the conflicts between preferences.

### 9.2.2 Software process framework and open-source development

The research work conducted during this thesis has investigated the challenges with respect to the creation of context-aware systems, observing that the difficulties in the development of these systems make their development significantly different from that of traditional computing. Research on development techniques for context-aware systems is typically scattered and disconnected from existing software development methods. The thesis concludes that there is a need for a more holistic approach in the development of context-aware systems. This has resulted into the assembly of a framework which embeds the conceptualisation across the most common phases of a development process. By doing so, the aim is to achieve an integrated consistency across those phases towards the achievement of a methodology that holistically addresses the concerns related to the development of more usable context-aware systems. The following reflection on the framework presented as part of this dissertation, is compared below against the second objective of this thesis, introduced in Section 1.4. From the analysis of challenges presented in Section 1.1, the fourth challenge [Chg4] is related to the development cost of context-aware applications. In order to reduce this cost, a typical approach is to create a framework in order to facilitate the development of these types of systems. As it is concluded in the state-of-the-art analysis from Chapter 3, the use of a model-driven approach could reduce development costs, as the code for implementing the system can be automatically (or semi-automatically) generated. Reducing the development speed can not only be beneficial for reducing the cost, but also to manage the context information through the life-cycle of the context-aware application [Chg3], and to help in managing change [Chg6]. Although the method has been satisfactorily used during the POSEIDON project case-study, the next logical stage is to apply the approach in a wide range of different projects related to context-awareness for gaining more experience about its usability, and completing some common and particular aspects to context-awareness that might arise in the application process. The tools of this software process framework are not intended to be fully mature, but rather to start a coherent framework which can become a mature tool for the development of context-aware systems. This framework has been applied to different scenarios of the project POSEIDON. The framework influenced specifically the navigation and weight management applications [224]. Several master students used the application for implementing different parts of them as part of their master projects. The weight manager application is available in [225]. Although these projects were useful for giving feedback and insights on the usage of the different tools, there is still a need to create further experiments that help to show until what degree the tools can help to resolve the initial challenges.

#### 9.2.2.1 Requirements Elicitation

The initial analysis on the state-of-the-art techniques for requirements elicitation with a specialisation in context-aware computing concludes that there are techniques which individually give response to certain aspects[1] considered as relevant according to the main challenges[2] of context-aware systems development. Consequently, an enhanced requirements framework for context-aware systems is created by assembling different parts of different existing methodologies.

As part of the perspectives obtained during the creation of the context conceptualisation, a more end-user stakeholder centred perspective is demanded for the requirements elicitation framework. In consequence, developers are required to spend significant effort during the initial stages in identifying and analysing stakeholders. The analysis facilitates greater knowledge about the different aspects of the stakeholders, which is then used to create profiles and activities that these stakeholders perform, and ultimately used for gaining information to better understand the meaning behind the actions of stakeholders. This information is useful not only for creating (contextual and non-contextual) services that are tailored to the preferences and needs of the users,

[1]These aspects are introduced in Section 3.2.

[2]These challenges are introduced in Section 1.1.

but also to identify situations in which personalised context-aware features can be displayed.

In order to facilitate the identification of needs, two different abstraction layers are introduced. The first layer gathers the high level objectives, which are then refined into lower level objectives, particularly helping to identify non-functional requirements as soft-goals. Then, the particular functional requirements are refined from the lowest level objectives, and the non-functional requirements from the lowest level soft-goals. Then, the initial stakeholder analysis is refined for end-user stakeholders, from which specific user profiles are derived. These profiles are used to elicit new goals and requirements to give response to their specific demands. Also, another salient feature of the framework is that it introduces an ethical analysis and profiling of users and requirements.

The goal is to provide a framework with integrated consistency across those phases towards the achievement of a methodology that holistically addresses the concerns related to the development of more usable context-aware systems. The presented approach integrates several existing methodologies and tools, but it also has a hybrid approach that uses goals to identify the requirements of the system on one hand, and uses a scenario based approach for the identification, observation and understanding of the user activities. Nevertheless, this assembled framework is more oriented to the requirements elicitation of the non-contextual aspects of context-aware systems. In order to include the perspectives gained during the conceptualisation of context, through the inclusion of requirements elicitation of the context-aware aspects, a specialisation of this framework is introduced. This framework particularly focuses on including the three main aspects to get the context right. Based on the scenario-based analysis of the end-user stakeholder activities, different situations of interest are identified. Then two main aspects of these situations are analysed: How the system is going to detect these situations, and which context-aware features is going to trigger in consequence, as well as how users are going to interact with these [C2]. The analysis is concluded with an evaluation of the implementation feasibility of the different situation of interest detection plans and the different triggered context-aware features.

One of the major challenges of this approach is the complexity of the information that needs to be handled. In order to better represent all this information, a model-based approach is taken [C3], introducing an enhancement of some existing UML/SysML models/profiles to produce three main diagrams. The first diagram represents information related to the stakeholders. The second diagram is used to represent the different goals and soft-goals of the system, as well as their relation to the different stakeholders. Finally, the last diagram represents the requirements of the system, as well as their relation to the goals and stakeholders or stakeholder profiles. The major advantage of using model-based techniques at this development stage is that it allows the visualisation of the different relationships between all the different elements represented in the system. This constitutes an advantage over other text-based techniques for documenting the requirements. In addition, a formal evaluation of the system can be applied to check whether or not the proposed requirements meet the specified goals. As part of the context-oriented extension of the framework, two additional novel diagrams are introduced. The first diagram aims to analyse the proposed context-aware features, to be triggered on specific situations of interest. The second diagram is related to the creation of a plan for making the system detect a certain situation of interest. In this case, the situation of interest element is specially useful to handle the detection mechanisms and the corresponding context-aware features to be triggered. Additionally, an evaluation technique is presented, to determine whether or not the related context-aware services and situation detection mechanisms should be implemented or not. This is especially useful at maintenance stages, where the analysis of the addition/edition/removal of situation of interests can be analysed, in order to give response to the constant demand for change that is expected from the dynamic nature of context-aware systems. In order to reduce the complexity of diagrams, the tool developed for the framework includes a link view, which enables the customisation of the graphical visualisation of the different relations between the different elements.

As part of the goal to produce an open-source tool supported framework, a module based on an existing open-source modelling tool has been created, which implements the different UML/SysML diagrams [C4]. This module implements not only the above mentioned diagrams, but also the missing SysML features that the free version has, including *traceability matrices* and *requirements tables*, as well as other relevant functionality such as partial documentation generation. Also, it includes the application of the different algorithms for operationalising goals into requirements, and assessing the feasibility of implementing situation of interest detection plans and associated context-aware features.The code of the Modelio module can be accessed from [194], and the latest binaries can be directly downloaded from [226]. Further instructions for installation and usage can be found in Appendix A.

#### 9.2.2.2 Design

The creation of the framework for the design stage of the development life-cycle of context-aware systems presented in this thesis [C5] has been influenced by the challenges presented in Section 1.1. Taking into account the diversity of context-aware systems [Chg2], the aim of this framework is to maximise the scope of the systems that can be developed with this framework, while remaining within the scope of this dissertation. For this, the scope of the context-aware systems which can be developed with this framework has been constrained to rule-based context-aware systems, as being the most common approach in these kinds of systems [1]. Also, the scope of the systems developed with the framework is aimed to develop both stationary and mobile platforms. These platforms are constrained to Z-Wave sensors for the mobile platform and mobile sensors for the Android operating system.

The design framework is divided into four main stages. The first stage is related to general system design, focusing mostly on the implementation of the non-contextual aspects of the context-aware system. The framework guides developers to translate the requirements obtained during the previous development stages, into the design of the system. The aim of this stage is to provide coherence between the design and requirements frameworks. The second stage is based on the conceptualisation perspectives presented in Section 2.4.2. Specifically, it is related to the implementation of the different context-aware features to be triggered by the system, following the models created for context-aware features during the requirements elicitation stage. It also takes into account the conceptualisation of interaction modalities presented in Section 2.4.1. As part of the support provided for the information display context-aware feature, a novel diagram is introduced, the Information Display Diagram, which can be used to represent how the information is going to be displayed in the context-aware application. The rest of context-aware features can be implemented using existing UML diagrams. For this, the classification of interaction modalities and types of context-aware features introduced in Chapter 2 is used. Then, the third stage of the framework is more focused on the design of context information [Chg2], based on the conceptualisation of the life-cycle of context-information presented in [1]. The first two and the last stages of the context information life-cycle are supported with a novel diagram, the Acquisition and Modelling Diagram. This diagram enables developers to specify the ways in which context-attributes are going to be measured by sensors, and how the information produced by these sensors is going to be modelled within the system. The diagram supports the creation of elements for both stationary and mobile platforms. The reasoning stage of the information life-cycle is supported by another novel diagram, the Context Reasoning Diagram. This diagrams allows the production of higher level information from low level information, which is previously modelled within the system using a set of logical rules using temporal references. Designed rules are applicable for both stationary and mobile platforms. An additional diagram for specifying the deployment details in both stationary and mobile platforms is introduced, the Deployment Diagram. Finally, the last stage of the framework aims to give response to the fifth challenge [Chg5] from Section 1.1, providing support for the design evaluation. This process covers four main dimensions to evaluate the design. The first dimension consists of the elaboration of test-cases that will be used to evaluate the produced system against the requirements specification. The second and third dimensions are related to identifying possible errors caused by the developers when creating the model, and focus on the syntax of certain attributes of the elements that contain queries or elements that will be used for generating code. Additionally, the design of elements is traced in a particular way from requirements to design. For example, the context-aware features are traced to sensors, which are traced to modelling rules that ultimately produce context states. Anomalies in the way this tracing is done, such as a missing relation, can be used to discover errors in the design, which can potentially be passed into the generated code. The evaluation procedures also include to include the evaluation of the traceability in order to check if some elements have been left without design. Finally, the evaluation also considers evaluating the behaviour of context reasoning rules against certain properties, in order to prevent the system from exhibiting any undesired behaviour.

Based on the fifth challenge [Chg5] of the analysis presented in Section 1.1, an extension to increase the reliability of the design evaluation of this framework is presented, which is a proof of concept for enabling the verification of rule-based context-aware systems. The approach consists of translating models specified in M, or in the Reasoning Diagram from the design framework, into models of a model checker, enabling the indirect verification of a system modelled in M against some formalized requirements. A scenario exemplifies how to use the framework for creating a context-aware reasoning component that includes learnt user behaviour patterns, how to establish different specification properties, and how the framework can help to achieve a more reliable automatic inclusion of inferred behavioural patterns. Early performance experiments show good performance for model checking regular-sized context-aware systems, with running times proportional to the size of the model M. The negative effect of size on performance could be reduced by separating rule chains that are independent from each other.

As part of the goal to produce an open-source tool supported framework, all the diagrams presented as part of the framework have been developed as an open-source tool for the design of context aware systems (DC-ASE) [C5], which has been implemented as a Modelio module. In addition, this module includes the implementation of the four mentioned dimensions for evaluating the design, providing a more reliable way of creating the design. This implementation includes the creation of an open-source library [C6] [205] for enabling the translations from M into NuSMV models, in order to enable the model-checking of reasoning rules. The library has been applied to both the DC-ASE Modelio module and the M Reasoner IDE.

The design evaluation procedure has satisfactorily been covered with respect to the scope of this dissertation, but future work could also address the automated generation of test-cases through the whole life-cycle of the context-aware system. As an extension of the main design evaluation, this thesis provides a theory and tools for enabling the verification of context reasoning rules, by translating the design of rules into models that can be checked with a formal model checker. In the future, mechanisms for verifying the modelling of rules could also be studied. The requirements currently guide the test-case generation using the UML Testing Profile (UTP) [127] tool for Modelio. The overall idea is to be able to design the test once, and apply the same tests in each of the different stages to check whether of not the application satisfies the requirements.

Tests would check the results for the design. Additionally, another application will check how the application runs once implemented, reusing the same tests from UTP [127] , and loading them to an external application that checks the system under test. Finally, guidelines for developers to check the system after it is installed could be automated (or semi-automated). Such a tool will significantly help in reducing the time employed for testing, and would facilitate the creation of more reliable systems. Additionally, UTP [127] profiles could be used for automating the testing process. But the process itself of creation could also be automated or semi-automated using model to model transformations from the different existing diagrams.

It can also occur that models can get very complex. In order to handle this complexity, future work could address the definition of UML views over the proposed models in order to increase their readability, not only for the design framework but also for the requirements framework. Finally, the design framework could enable the design of ontologies which can be coupled with the C-SPARQL reasoning engine is an interesting idea. During the research conducted for this dissertation, an approach to automatically translate context-attributes into ontologies was explored. It is theoretically feasible to generate ontologies from context-attributes in the Requirements Diagrams, which can be further edited and even translated into code using an external tool. Namely, Protégé, an open-source development environment that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. The experimental prototype of this idea can be found at [227].

#### 9.2.2.3 Implementation, deployment and maintenance

One of the advantages of using a model-driven approach is that it can help to reduce the implementation costs [Chg4], by enabling the automated (or semi-automated) generation of implementation code. As part of the coherent framework to cover the different stages of the development process, an additional framework for the implementation, deployment and maintenance of context-aware systems is presented. The framework provides guidance on how developers should generate the code. Then, it guides them through the implementation and deployment stages, enabling corresponding testing. Finally, the framework also includes maintenance tasks which cover the general system dimension, and a situation of interest centred maintenance of the system, which is likely to occur due to the dynamic nature of context. Future work could also delve into the best maintenance techniques, perhaps including more sophisticated evaluation algorithms for evaluating the removal or modification of situations of interest.

Another contribution of this framework is the theory for creating code based on the models from the design framework presented in this dissertation. The framework introduces a total of 21 sets of rules for generating different code files. It is not the intention of this thesis to provide the automatic generation of code for all the elements introduced in the diagrams. The sets of rules presented satisfactorily cover the objective under the scope of this thesis. Further work could be focused on automatising the code generation for some aspects that still have to be manually implemented. Also, code could be generated for both mobile and stationary platforms using the models from the Information Display Diagrams, but this would require some modification on the current libraries for mobile and stationary platforms. As part of the goal to produce an open-source tool supported framework, the DC-ASE open-source Modelio module, presented as part of the design framework, has been extended to support the code generation of these models. The code generation is fully automated for most of the pieces of code generated, although it still requires some developer intervention for some other pieces of code.

Also as part of the goal to produce an open-source tool supported framework, existing tools for mobile and stationary platforms have been enhanced in order to adopt this generated code. The stationary platform uses an integrated development environment (IDE) to facilitate the deployment and execution of reasoning rules that interact with Z-Wave radio protocol based sensors. This platform has been enhanced as part of the contribution of this thesis. The rule specifications for this module are automatically generated from the design diagrams. These can be further modified in the future and can be translated into NuSMV models again for checking them against certain properties. The translation of verification properties is still not supported in the IDE, and developers have to specify them manually. Further work could focus on the integration of specifications as part of the M language specifications. The IDE could also be further extended in the future in order to deploy and execute automatically generated rules by the learning module. These rules would first be automatically checked against some developer defined properties in order to prevent the system from exhibiting some undesired behaviours.

The mobile platform adopts two existing libraries, developed by Dean Kramer as part of the POSEIDON project, in order to facilitate the development of reasoning components into mobile platforms. The reasoning library has been refactored in order to better accommodate the generation of code. The learning platform associated with the Android reasoner can be used for analysing relevant behavioural traits, use historical context data and generate visualisations. This engine can be further improved to include other data science techniques such as pattern detection, classification, clustering and/or regression analysis. Future work can also be afforded to the centralisation of both mobile and stationary platforms. The databases of the two reasoners (MReasoner and Android Context Reasoner) could be merged into a single context database over a server. This could facilitate big data analytics of the learning platforms.

# Part VI

# Appendixes

# Appendix A

The aim of this Appendix is to introduce a guide for developers so that they can freely download, install and use the different tools of the framework. Note that all the tools presented as part of the framework of this thesis are publicly available. This appendix contains information is intended for two different types of audience, each of which is assumed to have the following prerequisite knowledge:

- Developers of context-aware systems: The developers of context-aware systems that aim to use C-ASEF, do not require as much knowledge as the developers of the framework do.
  - RC-ASE and DC-ASE Modelling tools: The developers require to be familiar with UML/SySML modelling languages. Also, it is recommended to have some familiarity with Modelio. Developers need to be familiar with RDF and NuSMV model checker specifications.
  - Stationary Platform: Although it is not strictly required, since the specification code can be generated with the modelling tools, it is recommended for developers to be familiar with the M specification language in order to make small changes directly to the specifications. In addition, developers need to know about model-checking and verification, preferably with the NuSMV model-checker.
  - Mobile platform: The developers require knowing Android development, or at least to have some familiarity with the Java language. Particularly with the sensors they are going to use.
- Developers/Researchers of the C-ASEF tools: This audience should be familiarised with the same topics as the developers of context-aware systems that use the C-ASEF. In addition, they require knowing Java programming language, as well as being familiar with the Git version control system. Preferably, also being familiar with the Github free web-based hosting service for Git. Also, de-

velopers/researchers should have some familiarity with the Maven and or Gradle software project management tools. For developing these tools, at least the same knowledge required for using the tools is required, but also:

– Stationary Platform: It is recommended that developers are familiarised with the M reasoning theory, the Vera routers (linux commands), databases, the LFPUBS reasoner, and NuSMV.
– Mobile platform: It is recommended that developers know about C-SPARQL, SPARQL and/or RDF, databases.

## A.1 Modelio

Modelio is an open source modelling tool which is freely available to be downloaded from its official website [148] [228]. The current version of Modelio for which the framework modules introduced in this thesis are compatible is v3.7. The program is available for Linux, Windows and Mac. Follow the quick-start guide [229] provided in the official Modelio website to install the program in your preferred operating system. The guide also includes how to download .jmdac modules from the official Modelio store [5], as well as how to use them in the different projects.

### A RC-ASE

If there is an intention of developing context-aware systems using the RC-ASE module, it can be freely downloaded from its official repository [194]. The compiled binaries of the project can be found in the folder "rcase/rcase/target/", where the .jmdac Modelio module file can be directly obtained. The installation of this module is the same as any other regular .jmdac Modelio module. It is recommended to have installed the SySML Architect Module [230].

If there is an intention of programming an extension or reusing the code of the RC-ASE module, it is recommended to have Eclipse RCP neon with Maven (M2e) and Git (EGit) plugins correctly installed beforehand. The repository can then be cloned to your hard drive. Click on Import -> Existing Maven Projects -> (Select as root directory the folder where you have cloned your repo). EGit should automatically detect that you are using a repository. In order to compile an updated .jmdac Modelio module file, click with the right button of the mouse on the main folder in Eclipse -> Run as -> 7 Maven install. This produces a new version of the .jmdac module.

### B DC-ASE

The usage of the DC-ASE module is the same of the RC-ASE module. It is also recommended to have installed the RC-ASE module and the SysML Architect Module [230].

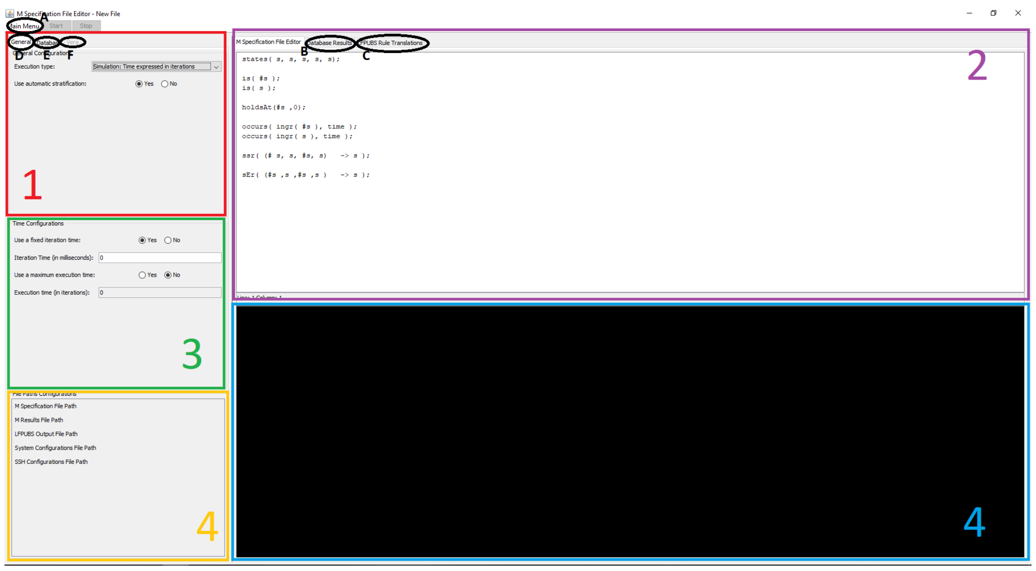
## A.2 Stationary platform

The M integrated development environment can be downloaded from the official repository [206]. The users can directly download the latest executable jar version from the folder "mreasoner-gui/mreasoner-gui/bin/".

### A Preparation and basic usage

The graphical user interface can be executed by simply double clicking the jar file. The prerequisite is to have Java v1.8 and PostgreSQL v9.6.3 installed either in a Linux-based, Mac or Windows operating system. Launching the application will display the content shown in Figure A.1.

1. The developer can choose between three different development modes: Simulation, simulation in real time, and deployment. Simulations enable to observe the behaviour of the system without having to deploy all the sensors and actuators in the real world. The rules need to be in stratification order [162]. This can be done manually by the user or it can be done automatically by the application.
2. The developer can specify the rules written in M language, these will be executed against a grammar parser.
3. The time taken by the computer can be approximately fixed to a certain time window. For example, each iteration can occur in a second. The system will try to adjust as much as possible to this window. On the other hand, the system can just execute without trying to adapt to a specific time-window. Also, the system



General Configuration
Execution type: Simulation: Time expressed in iterations
Use automatic stratification: [✓ Yes] No

Time Configurations
Use a fixed iteration time: [✓ Yes] No
Iteration Time (in milliseconds): 0
Use a maximum execution time: Yes [✓ No]
Execution time (in iterations): 0

File Paths Configurations
M Specification File Path
M Results File Path
LFPUBS Output File Path
System Configurations File Path
SSH Configurations File Path

```
states( s, s, s, s, s);

is( #s );
is( s );

holdsAt(#s ,0);

occurs( ingr( #s ), time );
occurs( ingr( s ), time );

ssr( (# s, s, #s, s)   -> s );

sEr( (#s ,s ,#s ,s )   -> s );
```

**Figure A.1: Screenshot of the main window of the M IDE (mreasoner-gui) application, which appears immediately after executing it.**

can run until the user stops it, or until it reaches a certain number of iterations. This can help developers to do quick simulations and observe specific results.

4. All the output results of the execution of the application will appear in this window. The log of the window can also be stored in the preferred location of the user.
5. There are certain results which are saved on different paths. This window helps the user to know where each of the files resides.

The program operates under the concept of a session. The session is a set of configurations required for executing and/or deploying a system, and which are saved and managed simultaneously. These include:

- M Specification: The rule specification written in M language.
- M log results: This file saves the results appearing in the panel identified as 4 in Figure A.1.
- LFPUBS output: This is the output of LFPUBS containing the user behavioural patterns in the LFPUBS specification language.
- M Configurations: Configurations related to panels identified as 1 and 3 in Fig-

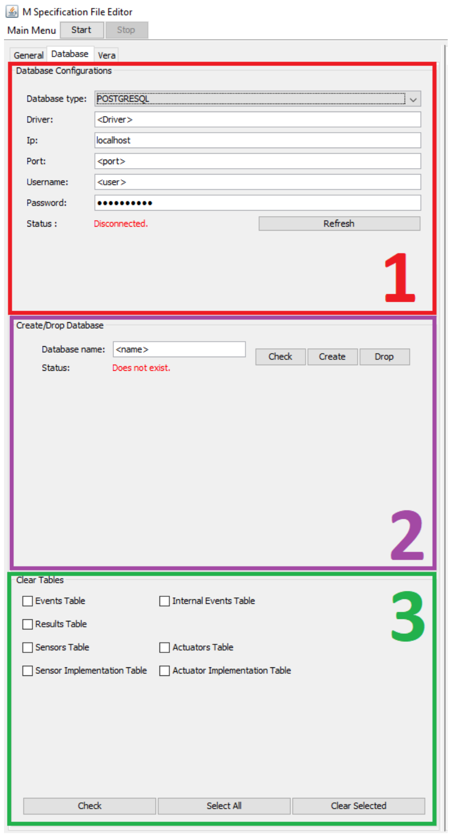
ure A.1, as well as the database configuration from .

- SSH Configurations: Configurations related to the SSH connection to the Vera Router for deployment purposes.

Additionally, the database can be configured using the panel shown in Figure A.2. For the panel A, in the same figure, the default driver is *org.postgresql.Driver*, but the application enables the use of other drivers, which are required to be installed independently or using Maven. This step requires access to a database through the username and password. The application will automatically handle the creation of tables in the database. The database can be named from Panel 2, in Figure A.2. Tables can also be cleared at will, using the panel labeled as 3 in the same figure. The SSH configurations are edited in the panel shown in Figure A.3, where the hostname of the router has to be specified, as well as the main port for the ssh connection. This requires an SSH account in the vera router, that is Linux based. Just create a new user in the router to start using this service, or use an existing one. This requires the username and the password.

The user can decide whether to create a session, load a previous one, save the current session or save the current session with another name just by clicking on the main menu (Letter A in Figure A.1) and then on session. Similarly, the developer can also do the same actions for an M specification or the logs, by clicking on the main menu and then on M Reasoner (M Specification or M Result Log). Also, the same can be done with the SSH configuration by clicking on the main menu and then on Vera. There are several full session examples that can be loaded from the mreasoner-gui/examples folder of the repository.

There are two other functions that can be used with the M IDE. The first function consists of translating the LFPUBS output into M specifications. For this, first load the LFPBUS output file by clicking on the main menu, LFPUBS, Load LFPUBS output file. Then, the loaded file can be observed on the LFPUBS Rule Translations tab (Letter C from Figure A.1). For obtaining the automatic translation of rules, click on the main menu, LFPUBS, Translate LFPUBS rules. The translation will be made available in the lower part of the main panel of the LFPUBS Rule Translation tab. The specifications generated from LFPUBS rules can be directly copied and merged with the current specifications. Finally, the other function is that of translating current



Database Configurations

Database type: POSTGRESQL
Driver: <Driver>
Ip: localhost
Port: <port>
Username: <user>
Password: ••••••••••
Status : Disconnected.

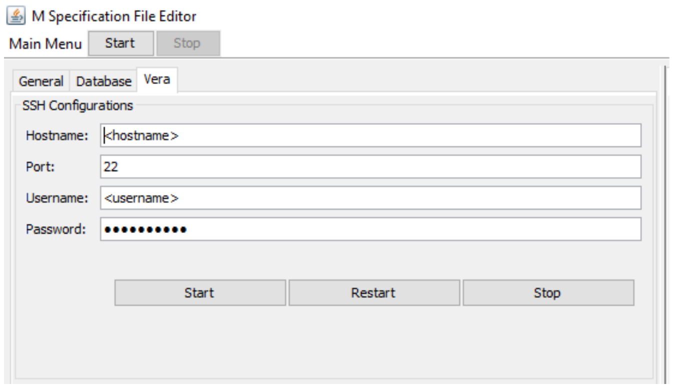
Create/Drop Database

Database name: <name>
Status: Does not exist.

Clear Tables

- [ ] Events Table
- [ ] Internal Events Table
- [ ] Results Table
- [ ] Sensors Table
- [ ] Actuators Table
- [ ] Sensor Implementation Table
- [ ] Actuator Implementation Table

Figure A.2: Screenshot of the database and SSH configurations panel of the M IDE (mreasoner-gui) application.



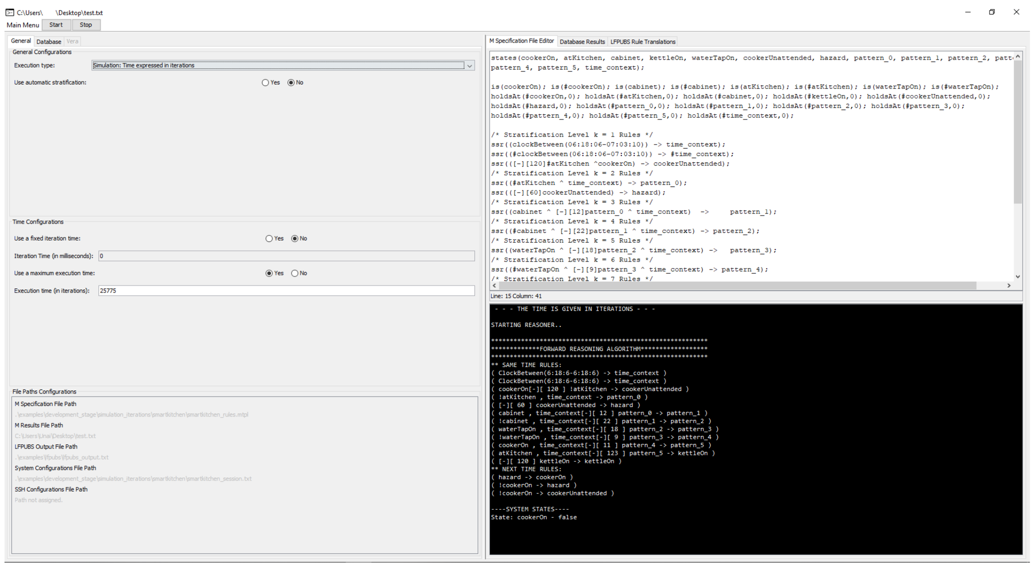
SSH Configurations

Hostname: <hostname>
Port: 22
Username: <username>
Password: ••••••••••

Figure A.3: Screenshot of the SSH configurations panel of the M IDE (mreasoner-gui) application.

specifications into NuSMV ones, for verification purposes. For this, click on the main menu, verification, export to NuSMV. Select the desired place for the NuSMV model and click save.

Once the specifications and configurations are loaded, the system can be simulated or deployed by pressing the start button, next to the main menu, as shown in Figures A.4 and A.5.



General Configurations

Execution type: Simulation: Time expressed in iterations

Use automatic stratification: Yes [✓ No]

Time Configurations

Use a fixed iteration time: Yes [✓ No]

Iteration Time (in milliseconds): 0

Use a maximum execution time: [✓ Yes] No

Execution time (in iterations): 25775

File Paths Configurations

M Specification File Path
.\examples\development_stage\simulation_iterations\smartkitchen\smartkitchen_rules.mtpl

M Results File Path
C:\Users\Unai\Desktop\test.txt

LFPUBS Output File Path
.\examples\lfpubs\lfpubs_output.txt

System Configurations File Path
.\examples\development_stage\simulation_iterations\smartkitchen\smartkitchen_session.txt

SSH Configurations File Path
Path not assigned.

```
states(cookerOn, atKitchen, cabinet, kettleOn, waterTapOn, cookerUnattended, hazard, pattern_0, pattern_1, pattern_2, patt
pattern_4, pattern_5, time_context);

is(cookerOn); is(#cookerOn); is(cabinet); is(#cabinet); is(atKitchen); is(#atKitchen); is(waterTapOn); is(#waterTapOn);
holdsAt(#cookerOn,0); holdsAt(#atKitchen,0); holdsAt(#cabinet,0); holdsAt(#kettleOn,0); holdsAt(#cookerUnattended,0);
holdsAt(#hazard,0); holdsAt(#pattern_0,0); holdsAt(#pattern_1,0); holdsAt(#pattern_2,0); holdsAt(#pattern_3,0);
holdsAt(#pattern_4,0); holdsAt(#pattern_5,0); holdsAt(#time_context,0);

/* Stratification Level k = 1 Rules */
ssr((clockBetween(06:18:06-07:03:10)) -> time_context);
ssr((#clockBetween(06:18:06-07:03:10)) -> #time_context);
ssr(([-][120]#atKitchen ^cookerOn) -> cookerUnattended);
/* Stratification Level k = 2 Rules */
ssr((#atKitchen ^ time_context) -> pattern_0);
ssr(([-][60]cookerUnattended) -> hazard);
/* Stratification Level k = 3 Rules */
ssr((cabinet ^ [-][12]pattern_0 ^ time_context)  ->     pattern_1);
/* Stratification Level k = 4 Rules */
ssr((#cabinet ^ [-][22]pattern_1 ^ time_context) -> pattern_2);
/* Stratification Level k = 5 Rules */
ssr((waterTapOn ^ [-][18]pattern_2 ^ time_context) ->   pattern_3);
/* Stratification Level k = 6 Rules */
ssr((#waterTapOn ^ [-][9]pattern_3 ^ time_context) -> pattern_4);
/* Stratification Level k = 7 Rules */
```

Line: 15 Column: 41

```
 - - - THE TIME IS GIVEN IN ITERATIONS - - -

STARTING REASONER..

**********************************************************
************FORWARD REASONING ALGORITHM*****************
**********************************************************
** SAME TIME RULES:
( ClockBetween(6:18:6-6:18:6) -> time_context )
( ClockBetween(6:18:6-6:18:6) -> time_context )
( cookerOn[-][ 120 ] !atKitchen -> cookerUnattended )
( !atKitchen , time_context -> pattern_0 )
( [-][ 60 ] cookerUnattended -> hazard )
( cabinet , time_context[-][ 12 ] pattern_0 -> pattern_1 )
( !cabinet , time_context[-][ 22 ] pattern_1 -> pattern_2 )
( waterTapOn , time_context[-][ 18 ] pattern_2 -> pattern_3 )
( !waterTapOn , time_context[-][ 9 ] pattern_3 -> pattern_4 )
( cookerOn , time_context[-][ 11 ] pattern_4 -> pattern_5 )
( atKitchen , time_context[-][ 123 ] pattern_5 -> kettleOn )
( [-][ 120 ] kettleOn -> kettleOn )
** NEXT TIME RULES:
( hazard -> cookerOn )
( !cookerOn -> hazard )
( !cookerOn -> cookerUnattended )

----SYSTEM STATES----
State: cookerOn - false
```

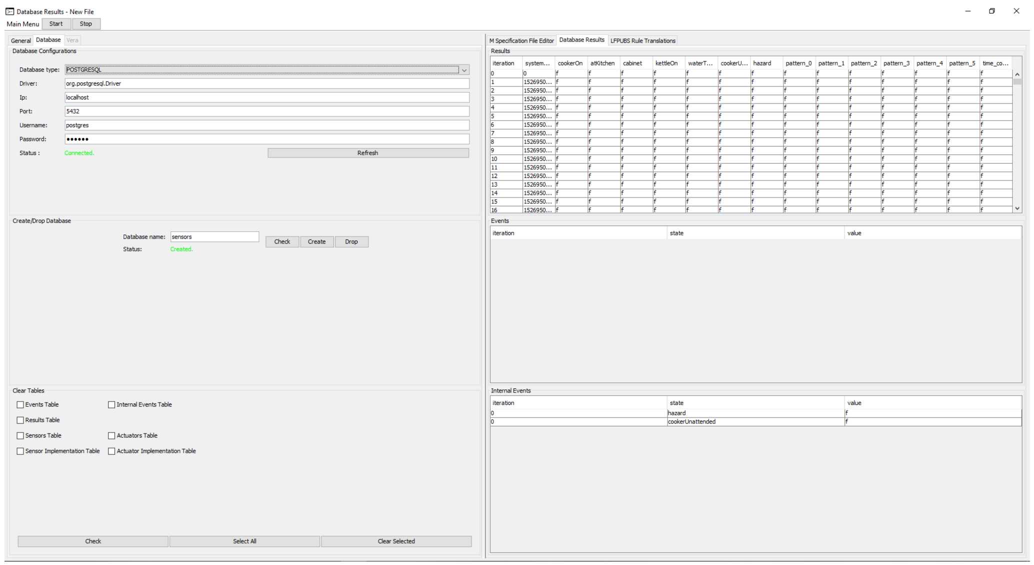
**Figure A.4: Screenshot of the main window of the M IDE (mreasoner-gui) application, just immediately after executing a specification by pressing the start button.**

## B Development

If there is an intention to extend the reasoner in any way, any of its parts can be downloaded from [212] [231] [206] [208] [205] and extended using any Java IDE.

## A.3 Mobile platform

While the stationary platform provides an integrated development environment for its execution, the mobile platform is just a set of libraries, which can be used as part of



Database Configurations

Database type: POSTGRESQL
Driver: org.postgresql.Driver
Ip: localhost
Port: 5432
Username: postgres
Password: ••••••
Status : Connected.

Create/Drop Database

Database name: sensors
Status: Created.

Clear Tables

☐ Events Table ☐ Internal Events Table
☐ Results Table
☐ Sensors Table ☐ Actuators Table
☐ Sensor Implementation Table ☐ Actuator Implementation Table

Results

| iteration | system... | cookerOn | atKitchen | cabinet | kettleOn | waterT... | cookerU... | hazard | pattern_0 | pattern_1 | pattern_2 | pattern_3 | pattern_4 | pattern_5 | time_co... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 1 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 2 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 3 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 4 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 5 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 6 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 7 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 8 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 9 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 10 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 11 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 12 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 13 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 14 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 15 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |
| 16 | 1526950... | f | f | f | f | f | f | f | f | f | f | f | f | f | f |

Events

| iteration | state | value |
|---|---|---|

Internal Events

| iteration | state | value |
|---|---|---|
| 0 | hazard | f |
| 0 | cookerUnattended | f |

**Figure A.5: Screenshot of the database window panel of the M IDE (mreasoner-gui) application, after executing a specification by pressing the start button.**

the development of a regular Android application. Any Android development platform can be used. For the development of these libraries, with regard to this thesis, Android Studio v3.0.1 has been used. There are two main libraries in this framework: Android Context Library and Android Context Reasoner. The first library,

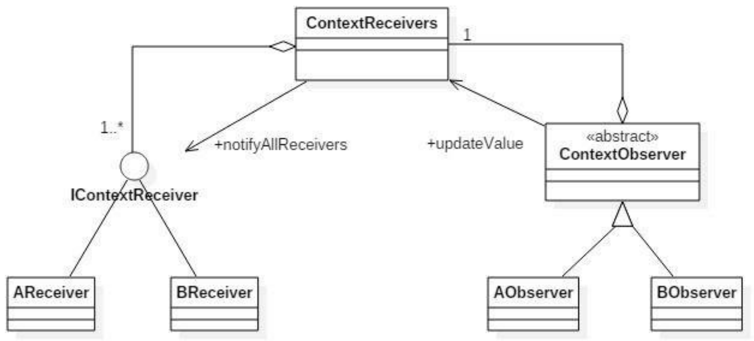


**Figure A.6: Main structure of the modified observer pattern in the Android Context Library.**

developed by Dean Kramer, it is related to the acquisition of information from the sensors. The main package contains the elements for providing a modified observer structure, as shown in Figure A.6. Additionally, it also contains the main types of observers: Pull and Push Observers, and its different implementations (SensorObserver, BroadcastObserver, LocationObserver and BluetoothLEObserver). There is another division of packages inside the main package: Environment, Hardware, Personal and Utility. Each package contains related observers. If any observer which is not in the library is required, it is recommended to continue the development in the library itself, as this would increase the reusability of the library. The second library is related to the modelling and reasoning of context. It was also developed by Dean Kramer, and later refactored for facilitating the automation of code generation, as part of the contribution of this thesis. The main package of this library is located in *org.poseidon_-project.context*. This library also contains a mobile version of CSPARQL, a SAT solver, the Tellu client API for connecting to the navigational services, and contains the custom code for including different situations of interest.

## A.4 Code generation rules

For spatial reasons, some code generation rules introduced in Chapter 8 have been omitted from the main text. This section introduces the different code generation rules that have been omitted from the main document.

### A Pull observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;

import android.content.Context;
import java.lang.Override;
import uk.ac.mdx.cs.ie.acontextlib.PullObserver;

public class [MobileSensor.name]Observer extends PullObserver {
  public static final String NAME = "[MobileSensor.name]Observer";

  public [MobileSensor.name]Observer(Context c) {
    super(c,10000,"[Observer.name]");
  }

  @Override
  public void checkContext() {
    // Object object) {
    //
    /*  // Include your main code to observe context here
      // This is an example of checking the external
    //storage of the phone
      long v = Environment.getExternalStorageDirectory()
    .getUsableSpace();
      if (mCurrentSpace != v) {
        sendToContextReceivers("sensor.external_storage_remaining",
      v / SIZE_MB);
        mCurrentSpace = v;
      }
    */
  }
}
[end for]
```

Listing A.1: Pseudo-code rules for code transformation of Pull type of observers.

## B Push observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;

import android.content.Context;
import java.lang.Override;
import uk.ac.mdx.cs.ie.acontextlib.PushObserver;

public class BadtemperatureContext extends PushObserver {
  public static final String NAME = "[MobileSensor.name]Observer";

  public BadtemperatureContext(Context c) {
    super(c);}

  public void checkContext() {
    checkContext(data);
  }

  @Override
  public void resume() {
    return start();}

  @Override
  public void stop() {
    return stop();}
}
[end for]
```

Listing A.2: Pseudo-code rules for code transformation of Push type of observers.

## B.1 Bluetooth low energy observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;
import android.content.Context;
import java.lang.Override;
import uk.ac.mdx.cs.ie.acontextlib.BluetoothLEDevice;

public class [MobileSensor.name]Observer extends BluetoothLEDevice {
  public static final String NAME = "[MobileSensor.name]Observer";

  public [MobileSensor.name]Observer(Context c) {
  /* //This is an example of a heart rate measuring
     //bluetooth wristband:
     super(c, UUID.fromString(HEART_RATE_SERVICE),
     UUID.fromString(HEART_RATE_MEASUREMENT));
  */
  }
  @Override
  public void checkContext() {

    /* BluetoothGattCharacteristic characteristic =
  (BluetoothGattCharacteristic) data;

    int flag = characteristic.getProperties();
    int format = -1;

    if ((flag & 0x01) != 0) {
      format = BluetoothGattCharacteristic.FORMAT_UINT16;
    } else {
      format = BluetoothGattCharacteristic.FORMAT_UINT8;
    }

    final int heartRate = characteristic.getIntValue(format, 1);
    sendToContextReceivers("sensor.heartrate", heartRate); */
  }
}
[end for]
```

Listing A.3: Pseudo-code rules for code transformation of Bluetooth Low Energy type of observers.

## B.2 Android sensor framework observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;
import android.content.Context;
import java.lang.Override;
import uk.ac.mdx.cs.ie.acontextlib.SensorContext;

public class [MobileSensor.name]Observer extends SensorContext {
  public static final String NAME = "[MobileSensor.name]Observer";
  public [MobileSensor.name]Observer(Context c) {
    /*   //This is an example of a super call for the light sensor:
       super(c, Sensor.TYPE_LIGHT,
     SensorManager.SENSOR_DELAY_NORMAL, Exercisetime)); */
  }
  @Override
  public void checkContext() {
    /* float[] values) {
        //Include your code here
        //This is an example for the light sensor in Android
        long value = Math.round(values['['/]0]);
            long difference = Math.abs(mCurrentValue - value);
            long threshold;
            if (value > mCurrentValue) {
                threshold = mContextDifferenceHigher;
            } else {
                threshold = mContextDifferenceLower;
            }
            if (difference >= threshold) {
                mCurrentValue = value;
                sendToContextReceivers("sensor.light_lumens",
        mCurrentValue);
                mContextDifferenceHigher = value * 2;
                mContextDifferenceLower = value / 2;
            }     */
  }
}
[end for]
```

Listing A.4: Pseudo-code rules for code transformation of Android Sensor Framework type of observers.

## B.3 Broadcast observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;

import android.content.Context;
import java.lang.Override;
import uk.ac.mdx.cs.ie.acontextlib.BroadcastContext;

public class [MobileSensor.name]Observer extends BroadcastContext {

  public static final String NAME = "[MobileSensor.name]Observer";
    /* //Include your variables here
     //This is an example for checking the battery level
    private int mBatteryLevel; */

  public [MobileSensor.name]Observer(Context c) {
    /*  //This is an example of a battery level measuring observer
       super(c, "Intent.Action_BATTERY_CHANGED",NAME); */
  }

  @Override
  public void checkContext() {
    /* //Bundle bundle) {
      // Include your variables here
      //This is an example for checking the battery level
      int rawlevel = bundle.getInt(BatteryManager.EXTRA_LEVEL, -1);
      int scale = bundle.getInt(BatteryManager.EXTRA_SCALE, -1);
      if (rawlevel >= 0 && scale > 0) {
        mBatteryLevel = (rawlevel * 100) / scale;

        //Send the receiver the context update
        sendToContextReceivers("sensor.battery_level",
      mBatteryLevel);
      }  */
  }
}
[end for]
```

Listing A.5: Pseudo-code rules for code transformation of Broadcast type of observers.

## B.4 Location observer

```
[for MobileSensor]
package package edu.casetools.icase.custom.observers;

import android.content.Context;
import java.lang.Override;
import java.lang.String;
import uk.ac.mdx.cs.ie.acontextlib.LocationContext;

public class [MobileSensor.name]Observer extends LocationContext {
  public static final String NAME = "[MobileSensor.name]Observer";

  public [MobileSensor.name]Observer(Context c, String provider) {
    super(c,provider,"[MobileSensor.name]");
  }

  @Override
  public void checkContext() {
    //  Object object) {
    //  sendToContextReceivers("device.current_loc", object);
  }
}
[end for]
```

Listing A.6: Pseudo-code rules for code transformation of Location type of observers.

# C Custom situation of interest class

```
[forEach SituationOfInterest where detects(DetectionPlan,
SituationOfInterest) and
(DetectionPlan.toBeImplemented = TRUE)]
package edu.casetools.icase.custom.situations;
import android.content.SharedPreferences;
import java.util.Map;
import eu.larkc.csparql.core.engine.CsparqlQueryResultProxy;
import org.poseidon_project.context.reasoner.SituationOfInterest;
import org.poseidon_project.context.ContextReasonerCore;
import org.poseidon_project.context.logging.DataLogger;
import org.poseidon_project.context.reasoner.AbstractContextMapper;
import org.poseidon_project.context.reasoner.ReasonerManager;
public class [SituationOfInterest.name]SOI extends
SituationOfInterest {
    public [SituationOfInterest.name]SOI(){
        super("[SituationOfInterest.name]");
    }
  [for DetectionPlan where aggregateComposition(DetectionPlan,
  ContextAttribute) and isPrimary(ContextAttribute)]
    [forEach ContextAttribute where
    trace(ContextAttribute,ContextState)]
      [for ContextState where
      produce(RDFModellingRule, ContextState)]
        private static final String [ContextState.name]Query =
        "[generateModellingRule(RDFModellingRule)]";
        [generateModellingRuleGetter
        (ContextState,RDFModellingRule)]
      [end for]
    [end forEach]
    [generateNextLevelReasoningRules(ContextAttribute,
    DetectionPlan)]
  [end for]
    @Override
    public boolean registerSituationOfInterest(
  ReasonerManager mReasonerManager,
  AbstractContextMapper contextMapper, SharedPreferences
  mRuleSettings, DataLogger mLogger, String logTag,
Map parameters) {
        boolean okExit = true;

   [registerModellingRule()]
        CsparqlQueryResultProxy c1 =
   mReasonerManager.registerCSPARQLQuery(getBatteryLOWQuery());
        CsparqlQueryResultProxy c2 =
   mReasonerManager.registerCSPARQLQuery(getBatteryOkQuery());

        okExit = contextMapper.registerModellingRule(
   getBatteryLOWQuery(), c1, okExit);
        okExit = contextMapper.registerModellingRule(
   getBatteryOkQuery(), c2, okExit);
        mLogger.logVerbose(DataLogger.REASONER, logTag,
   "Registered [SituationOfInterest.name] Situation of Interest");

   [addObserverRequirements()]
        return contextMapper.addObserverRequirement("engine",
   "BatteryContext",okExit);
   [for DetectionPlan where aggregateComposition(DetectionPlan,
   ContextAttribute) and isPrimary(ContextAttribute)]
   [forEach ContextAttribute where
   trace(ContextAttribute,ContextState)]
     [for ContextState where
     produce(RDFModellingRule, ContextState)]
        private static final String [ContextState.name]Query =
        "[generateModellingRule(RDFModellingRule)]";
        [generateModellingRuleGetter(ContextState,
        RDFModellingRule)]
     [end for]
   [end forEach]
   [generateNextLevelReasoningRules(ContextAttribute,
   DetectionPlan)]
   [end for]
   }
   @Override
   public boolean unRegisterSituationOfInterest(ContextReasonerCore
mReasonerCore, ReasonerManager mReasonerManager,
AbstractContextMapper contextMapper,
DataLogger mLogger, String logTag)  {
        boolean okExit = contextMapper.removeObserverRequirement(
    "engine", "BatteryContext");

        okExit = contextMapper.unregisterModellingRule(
    getBatteryLOWQuery(),okExit);
        okExit = contextMapper.unregisterModellingRule(
    getBatteryOkQuery(),okExit);
        mReasonerCore.removeContextValue("BATTERY");
        mLogger.logVerbose(DataLogger.REASONER, logTag,
    "Unregistered battery");

        return okExit;
    }
}
[end forEach]
```

Listing A.7: Pseudo-code rules for transforming stereotypes in the Situation Detection Diagram to the java class for custom situations of interest in the Android Context Reasoner.

```
[function def: generateModellingRule(RDFModellingRule)]
  [for ContextState where produce(RDFModellingRule,ContextState)]
  REGISTER QUERY [ContextState.name]_query AS
  [end for]
  PREFIX ex: <[RDFModellingRule.ontology]>
  CONSTRUCT { ex: [for MobileSensor where
  feedsInWindow(MobileSensor,RDFModellingRule)]
  [MobileSensor.name][end for]
  [RDFModellingRule.predicate]
  [for ContextState where produces(RDFModellingRule,ContextState)]
  [ContextState.name][end for]
  FROM STREAM [feedsInWindow.stream] \[RANGE [feedsInWindow.for]
  STEP [feedsInWindow.every]\]
  WHERE {
    [for MobileSensor where feedsInWindow(MobileSensor,
    RDFModellingRule)]
      [MobileSensor.data] . {
      [for (i=1;i<4;i++)]
        SELECT ([RDFModellingRule.method] AS subqres_[i])
        WHERE{  [MobileSensor.data] .
```

```
            FILTER(
      [getTripleFromString(RDFModellingRule.methodTripleVar)] )
        }

      }
      [end for]
    [end for]
    [getFiltersFromString(RDFModellingRule.logicalEvaluations)]
  }
[end function def]

[function def: generateModellingRuleGetter(ContextState,
RDFModellingRule)]
  public String get[ContextState.name]Query(String value){
    String query = new String([ContextState.name]Query);
    [forEach PreferenceSensor where feeds(PreferenceSensor,
    RDFModellingRule)]
      query.replace("$$[PreferenceSensor.name]", value);
    [end forEach]
    return query;
  }
[end function def]

[function def: generateNextLevelReasoningRules(ContextAttribute,
DetectionPlan)]
  [forEach ContextAttribute2 where
  derive(ContextAttribute,ContextAttribute2) and
  aggregateComposition(DetectionPlan,ContextAttribute2)]
    [for ContextState where trace(ContextState,ContextAttribute2)]
    [forEach AntecedentGroup where
    produces(AntecedentGroup,ContextState)]
      [forEach Consequent where same(AntecedentGroup,Consequent)]
        private static final String new[ContextAttribute2.name] =
        "[getAntecedents(AntecedentGroup)] implies
        [Consequent.name]";
        public String getNew[ContextAttribute2.name](){
          return new[ContextAttribute2.name];
        }
      [end forEach]
    [end forEach]

    [end for]
    [generateNextLevelReasoningRules(ContextAttribute2,
    DetectionPlan)]
  [end forEach]
[end function def]

[function def: getAntecedents(AntecedentGroup)]
  [for AntecedentGroup where
  produce(AntecedentGroup,ContextAttribute)]
    [forEach Antecedent in AntecedentGroup]
      [getSign(Antecedent.value)][Antecedent.name] [useAnd()]
    [end forEach]
  [forEach PastOperator in AntecedentGroup]
    [getSign(PastOperator.value)][PastOperator.name]
    \[[getSymbol(PastOperator.type)]
    [pastOperator(PastOperator)]\]
    [useAnd()]
  [end forEach]
  [end for]
[end function def]

[function def: pastOperator(PastOperator)]
  [if PastOperator is ImmediatePastOperator]
    [PastOperator.bound]
  [else if PastOperator is AbsolutePastOperator]
    [PastOperator.lowbound]-[PastOperator.uppbound]
  [end if]
[end function def]
```

Listing A.8: Pseudo-code functions appearing in Listing A.7.

## A.5 NuSMV specification counter-example

```
-- specification AG !(atKitchen = FALSE & waterTapOn = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
```

```
-> State: 1.1 <-
  time = 0
  cookerOn = FALSE
  atKitchen = FALSE
  cabinet = FALSE
  kettleOn = FALSE
  waterTapOn = FALSE
  cookerUnattended = FALSE
  hazard = FALSE
  pattern_0 = FALSE
  pattern_1 = FALSE
  pattern_2 = FALSE
  pattern_3 = FALSE
  pattern_4 = FALSE
  pattern_5 = FALSE
  time_context = FALSE
  time_context_aux = FALSE
  cookerUnattended_aux = FALSE
  pattern_0_aux = FALSE
  hazard_aux = FALSE
  pattern_1_aux = FALSE
  pattern_2_aux = FALSE
  pattern_3_aux = FALSE
  pattern_4_aux = FALSE
  pattern_5_aux = FALSE
  kettleOn_aux = FALSE
  atKitchen_sip_120.counter = 0
  atKitchen_sip_120.live = FALSE
  cookerUnattended_sip_60.counter = 0
  cookerUnattended_sip_60.live = FALSE
  pattern_0_sip_12.counter = 0
  pattern_0_sip_12.live = FALSE
  pattern_1_sip_22.counter = 0
  pattern_1_sip_22.live = FALSE
  pattern_2_sip_18.counter = 0
  pattern_2_sip_18.live = FALSE
  pattern_3_sip_9.counter = 0
  pattern_3_sip_9.live = FALSE
  pattern_4_sip_11.counter = 0
  pattern_4_sip_11.live = FALSE
  pattern_5_sip_123.counter = 0
  pattern_5_sip_123.live = FALSE
  kettleOn_sip_120.counter = 0
  kettleOn_sip_120.live = FALSE
-> State: 1.2 <-
  time = 1
```

```
    atKitchen = TRUE
    waterTapOn = TRUE
    atKitchen_sip_120.counter = 1
  -> State: 1.3 <-
    time = 2
    atKitchen = FALSE
    atKitchen_sip_120.counter = 0
```

Listing A.9: A counter example for the specification of Line 3 in Listing A.9.

# Appendix B

It is important to note that the work done during this PhD dissertation led to a number of published articles that, whilst not derived directly from this thesis, are based on work that is related to that presented by this dissertation:

- *(2018) RC-ASEF: An open-source tool-supported requirements elicitation framework for context-aware systems development.*
  Unai Alegre-Ibarra, Juan Carlos Augusto, and Carl Evans.
  24th Conference on Knowledge Acquisition and Management (IEEE).
- *(2018) Perspectives on engineering more usable context-aware systems.*
  Unai Alegre-Ibarra, Juan Carlos Augusto, and Carl Evans.
  Journal of Ambient Intelligence and Humanized Computing (Springer).
- *(2018) A survey on the evolution of the notion of context-awareness.*
  Juan Carlos Augusto, Asier Aztiria-Goenaga, Dean Kramer, and Unai Alegre-Ibarra.
  Applied Artificial Intelligence (Taylor & Francis).
- *(2017) Is 'Context-Aware Reasoning= Case-Based Reasoning'?.*
  Nawaz Khan, Unai Alegre-Ibarra, Dean Kramer, and Juan Carlos Augusto.
  International and Interdisciplinary Conference on Modeling and Using Context (Springer).
- *(2017) The user-centred intelligent environments development process as a guide to co-create smart technology for people with special needs.*
  Juan Carlos Augusto, Dean Kramer, Unai Alegre-Ibarra, Alexandra Covaci, and Adityarajsingh Santokhee.
  Universal Access in the Information Society (Springer).
- *(2016) Co-creation of Smart Technology with (and for) People with Special Needs.*
  Juan Carlos Augusto, Dean Kramer, Unai Alegre-Ibarra, Alexandra Covaci and

Adityarajsingh Santokhee.
Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (ACM).

- *(2016) Engineering context-aware systems and applications: A survey.*
Unai Alegre-Ibarra, Juan Carlos Augusto, and Tony Clark.
Journal of Systems and Software 117, 55-83 (Elsevier).

# Bibliography

[1] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos, "Context aware computing for the internet of things: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 414–454, 2014.

[2] OMG, "OMG Systems Modeling Language (OMG SysML), Version 1.3," Tech. Rep., Object Management Group, 2012.

[3] Dean Kramer and Juan Carlos Augusto, "Supporting context-aware engineering based on stream reasoning," in *International and Interdisciplinary Conference on Modeling and Using Context*. Springer, 2017, pp. 440–453.

[4] Dean Kramer, "CoMo Context Modeller library," `https://github.com/deankramer/ContextModeller`, [Online; Last accessed 25-June-2018].

[5] "Modelio Store," `http://store.modelio.org/resource/modules.html`, [Online; Last accessed 25-June-2018].

[6] Mark Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[7] Bill Schilit, Norman Adams, and Roy Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 1994, pp. 85–90.

[8] Anind K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4–7, 2001.

[9] Thomas Kleinberger, Martin Becker, Eric Ras, Andreas Holzinger, and Paul Müller, "Ambient intelligence in assisted living: enable elderly people to handle future interfaces," in *International Conference on Universal Access in Human-Computer Interaction*. Springer, 2007, pp. 103–112.

[10] J Augusto, Maurice Mulvenna, Huiru Zheng, Haiying Wang, Suzanne Martin, P McCullagh, and Jonathan Wallace, "Night optimised care technology for users needing assisted lifestyles," *Behaviour & Information Technology*, vol. 33, no. 12, pp. 1261–1277, 2014.

[11] Hong Sun, Vincenzo De Florio, Ning Gui, and Chris Blondia, "Promises and challenges of ambient assisted living systems," in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. Ieee, 2009, pp. 1201–1207.

[12] Jose Bravo, Diego López-De-Ipiña, Carmen Fuentes, Ramón Hervás, Rocío Peña, Marcos Vergara, and Gregorio Casero, "Enabling NFC technology for supporting chronic diseases: A proposal for alzheimer caregivers," *Ambient Intelligence*, pp. 109–125, 2008.

[13] Ilia Adami, Margherita Antona, and Constantine Stephanidis, "Ambient Assisted Living for People with Motor Impairments," *Disability Informatics and Web Accessibility for Motor Limitations*, pp. 76–104, 2013.

[14] Juan Carlos Augusto, Terje Grimstad, Reiner Wichert, Eva Schulze, Andreas Braun, Gro Marit Rødevand, and Vanda Ridley, "Personalized smart environments to increase inclusion of people with down's syndrome," in *International Joint Conference on Ambient Intelligence*. Springer, 2013, pp. 223–228.

[15] Vernor Vinge, "The coming technological singularity: How to survive in the post-human era," in *Proceedings of a Symposium Vision-21: Interdisciplinary Science & Engineering in the Era of CyberSpace, held at NASA Lewis Research Center (NASA Conference Publication CP-10129).—1993*, 1993.

[16] Amnon H Eden, Eric Steinhart, David Pearce, and James H Moor, "Singularity hypotheses: an overview," in *Singularity Hypotheses*, pp. 1–12. Springer, 2012.

[17] Thomas Erickson, "Some problems with the notion of context-aware computing," *Communications of the ACM*, vol. 45, no. 2, pp. 102–104, 2002.

[18] Anind K. Dey and Gregory D. Abowd, "Towards a better understanding of context and context-awareness," in *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. 1999, pp. 304–307, Springer-Verlag.

[19] Karen Henricksen and Jadwiga Indulska, "Modelling and using imperfect context information," in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 2004, pp. 33–37.

[20] Jason Pascoe, Nick Ryan, and David Morse, "Issues in developing context-aware computing," in *Handheld and ubiquitous computing*. Springer, 1999, pp. 208–221.

[21] Seyed Hossein Siadat and Minseok Song, "Understanding Requirement Engineering for Context-Aware Service-Based Applications," *Journal of Software Engineering and Applications*, vol. 5, no. 8, pp. 536–544, 2012.

[22] Karen Henricksen and Jadwiga Indulska, "Developing context-aware pervasive computing applications: Models and approach," *Pervasive and mobile computing*, vol. 2, no. 1, pp. 37–64, 2006.

[23] Bin Guo, Daqing Zhang, and Michita Imai, "Toward a cooperative programming framework for context-aware applications," *Personal and Ubiquitous Computing*, vol. 15, no. 3, pp. 221–233, 2011.

[24] Svein Hallsteinsen, Kurt Geihs, Nearchos Paspallis, Frank Eliassen, Geir Horn, Jorge Lorenzo, Alessandro Mamelli, and George Angelos Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2840–2859, 2012.

[25] Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland, and Charles Consel, "DiaSuite: A tool suite to develop Sense/Compute/Control applications," *Science of Computer Programming*, vol. 79, pp. 39–51, 2014.

[26] Juan Carlos Augusto, "User-Centric Software Development Process," in *Intelligent Environments (IE), 2014 International Conference on*. IEEE, 2014, pp. 252–255.

[27] Davy Preuveneers and Paulo Novais, "A survey of software engineering best practices for the development of smart applications in Ambient Intelligence," *Journal of Ambient Intelligence and Smart Environments*, vol. 4, no. 3, pp. 149–162, 2012.

[28] Iara Augustin, Adenauer C Yamin, Luciano Cavalheiro da Silva, Rodrigo Araújo Real, Gustavo Frainer, and Cláudio FR Geyer, "ISA-Madapt: abstractions and tools for designing general-purpose pervasive applications," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1231–1256, 2006.

[29] Aekyung Moon, Hyoungsun Kim, Hyun Kim, and Soowoo Lee, "Context-aware active services in ubiquitous computing environments," *ETRI journal*, vol. 29, no. 2, pp. 169–178, 2007.

[30] Achilleas Achilleos, Kun Yang, and Nektarios Georgalas, "Context modelling and a context-aware framework for pervasive service creation: A model-driven approach," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 281–296, 2010.

[31] Lei Tang, Zhiwen Yu, Hanbo Wang, Xingshe Zhou, and Zongtao Duan, "Methodology and Tools for Pervasive Application Development," *International Journal of Distributed Sensor Networks*, vol. 2014, 2014.

[32] Tomás Ruiz-López, Carlos Rodríguez-Domínguez, María José Rodríguez, Sergio F Ochoa, and José Luis Garrido, "Context-Aware Self-adaptations: From Requirements Specification to Code Generation," in *Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction*, pp. 46–53. Springer, 2013.

[33] Tom Gross, "Towards a new human-centred computing methodology for cooperative ambient intelligence," *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, no. 1, pp. 31–42, 2010.

[34] Juan Carlos Augusto and Miguel J Hornos, "Software simulation and verification to increase the reliability of Intelligent Environments," *Advances in Engineering Software*, vol. 58, pp. 18–34, 2013.

[35] Saul Greenberg, "Context as a dynamic construct," *Human-Computer Interaction*, vol. 16, no. 2, pp. 257–268, 2001.

[36] "DiaSuite," http://diagen.gforge.inria.fr/diasuite/repository/release/diasuite/, [Online; Last accessed 25-June-2018].

[37] Middlesex University Research Group on the development of Intelligent Environments, Fraunhofer-Gesellschaft Institute, and Tellu, "POSEIDON," http://www.poseidon-project.org/, [Online; Last accessed 25-June-2018].

[38] Dean Kramer, Alexandra Covaci, and Juan Carlos Augusto, "Developing Navigational Services for People with Down's Syndrome," in *Intelligent Environments (IE), 2015 International Conference on*. IEEE, 2015, pp. 128–131.

[39] POSEIDON Project, "POSEIDON Deliverable 2.1 - Report on requirements," Tech. Rep., PersOnalized Smart Environments to increase Inclusion of people with DOwn's syNdrome, 2015.

[40] Eamonn Slevin, I Lavery, David Sines, and J Knox, "Independent travel and people with learning disabilities: the views of a sample of service providers on whether this nee is being met," *Journal of Learning Disabilities for Nursing, Health, and Social Care*, vol. 2, no. 4, pp. 195–202, 1998.

[41] Dean Kramer and Tellu, "Poseidon Application," https://play.google.com/store/apps/details?id=no.tellu.poseidon, [Online; Last accessed 25-June-2018].

[42] Tellu, "POSEIDON," https://play.google.com/store/apps/details?id=no.tellu.poseidon.routecreator, [Online; Last accessed 25-June-2018].

[43] Alexandra Covaci, Dean Kramer, Juan Carlos Augusto, Silvia Rus, and Andreas Braun, "Assessing Real World Imagery in Virtual Environments for People with Cognitive Disabilities," in *Intelligent Environments (IE), 2015 International Conference on*. IEEE, 2015, pp. 41–48.

[44] Unai Alegre-Ibarra, Juan Carlos Augusto, and Carl Evans, "Perspectives on engineering more usable context-aware systems," *Journal of Ambient Intelligence and Humanized Computing*, 2018.

[45] Unai Alegre-Ibarra, Juan Carlos Augusto, and Tony Clark, "Engineering context-aware systems and applications: A survey," *Journal of Systems and Software*, vol. 117, pp. 55–83, 2016.

[46] Mary Bazire and Patrick Brézillon, "Understanding context before using it," in *Modeling and using context*, pp. 29–40. Springer, 2005.

[47] Bill N Schilit and Marvin M Theimer, "Disseminating active map information to mobile hosts," *Network, IEEE*, vol. 8, no. 5, pp. 22–32, 1994.

[48] Peter J Brown, "The stick-e document: a framework for creating context-aware applications," *Electronic Publishing-Chichester-*, vol. 8, pp. 259–272, 1995.

[49] Anind K Dey, "Context-aware computing: The CyberDesk project," in *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, 1998, pp. 51–54.

[50] Richard Hull, Philip Neaves, and James Bedford-Roberts, "Towards situated computing," in *Wearable Computers, 1997. Digest of Papers., First International Symposium on*. IEEE, 1997, pp. 146–153.

[51] Nick Ryan, Jason Pascoe, and David Morse, "Enhanced reality fieldwork: the context aware archaeological assistant," *Bar International Series*, vol. 750, pp. 269–274, 1999.

[52] International Organisation for Standardisation, "ISO 13407: Human-centred design processes for interactive systems," Tech. Rep., International Standards Organization, 1999.

[53] Albrecht Schmidt, *Ubiquitous computing-computing in context*, Ph.D. thesis, Lancaster University, 2003.

[54] Andreas Zimmermann, Andreas Lorenz, and Reinhard Oppermann, "An operational definition of context," in *Modeling and using context*, pp. 558–571. Springer, 2007.

[55] Prodromos Makris, Dimitrios N Skoutas, and Charalabos Skianis, "A survey on context-aware mobile and wireless networking: On networking and computing environments' integration," *IEEE communications surveys & tutorials*, vol. 15, no. 1, pp. 362–386, 2013.

[56] Stephen S Yau, Huan Liu, Dazhi Huang, and Yisheng Yao, "Situation-aware personalized information retrieval for mobile internet," in *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*. IEEE, 2003, pp. 639–644.

[57] Virpi Roto et al., *Web browsing on mobile phones: Characteristics of user experience*, Ph.D. thesis, Helsinki University of Technology, 2006.

[58] Juan Ye, Simon Dobson, and Susan McKeever, "Situation identification techniques in pervasive computing: A review," *Pervasive and mobile computing*, vol. 8, no. 1, pp. 36–66, 2012.

[59] Arthur H Van Bunningen, Ling Feng, and Peter MG Apers, "Context for ubiquitous data management," in *Ubiquitous Data Management, 2005. UDM 2005. International Workshop on*. IEEE, 2005, pp. 17–24.

[60] Karen Henricksen, *A framework for context-aware pervasive computing applications*, Ph.D. thesis, Computer Science, School of Information Technology and Electrical Engineering, University of Queensland, 2003.

[61] John McCarthy and Patrick J Hayes, "Some philosophical problems from the standpoint of artificial intelligence," *Readings in artificial intelligence*, pp. 431–450, 1969.

[62] Ray Reiter, "The situation calculus ontology," *Electronic News Journal on Reasoning about Actions and Changes*, 1997.

[63] Christos Anagnostopoulos and Stathes Hadjiefthymiades, "Advanced inference in situation-aware computing," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 5, pp. 1108–1115, 2009.

[64] Tomás Ruiz-López, *Un enfoque dirigido por modelos para el desarrollo de servicios para sistemas ubicuos basado en propiedades de calidad*, Ph.D. thesis, Universidad de Granada, 2014.

[65] "Google Now," `https://www.google.co.uk/landing/now/`, 2014, [Online; Last accessed 25-June-2018].

[66] Jason Pascoe, "Adding generic contextual capabilities to wearable computers," in *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*. IEEE, 1998, pp. 92–99.

[67] Louise Barkhuus and Anind Dey, "Is context-aware computing taking control away from the user? Three levels of interactivity examined," in *UbiComp 2003: Ubiquitous Computing*. Springer, 2003, pp. 149–156.

[68] Alan M Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[69] John R Lucas, "Minds, machines and Gödel," *Philosophy*, vol. 36, no. 137, pp. 112–127, 1961.

[70] Drew McDermott, "Artificial intelligence meets natural stupidity," *ACM SIGART Bulletin*, , no. 57, pp. 4–9, 1976.

[71] John R Searle, "Minds, brains, and programs," *Behavioral and brain sciences*, vol. 3, no. 03, pp. 417–424, 1980.

[72] Rodney A Brooks, "Intelligence without representation," *Artificial intelligence*, vol. 47, no. 1-3, pp. 139–159, 1991.

[73] Hubert L Dreyfus, *What computers still can't do: a critique of artificial reason*, MIT press, 1992.

[74] J.A. Fodor, *The Mind Doesn't Work that Way: The Scope and Limits of Computational Psychology*, Bradford book. MIT Press, 2001.

[75] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig, "Achieving human parity in conversational speech recognition," *arXiv preprint arXiv:1610.05256*, 2016.

[76] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *arXiv preprint arXiv:1609.08144*, 2016.

[77] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang, "Where does AlphaGo go: from Church-Turing thesis to AlphaGo thesis and beyond," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 2, pp. 113–120, 2016.

[78] Dharmendra S Modha, Rajagopal Ananthanarayanan, Steven K Esser, Anthony Ndirango, Anthony J Sherbondy, and Raghavendra Singh, "Cognitive computing," *Communications of the ACM*, vol. 54, no. 8, pp. 62–71, 2011.

[79] Lucy A Suchman, *Plans and situated actions: the problem of human-machine communication*, Xerox Corporation, Palo Alto Research Center, 1985.

[80] Leila Takayama, "The motivations of ubiquitous computing: revisiting the ideas behind and beyond the prototypes," *Personal and Ubiquitous Computing*, vol. 21, no. 3, pp. 557–569, 2017.

[81] Paul Dourish, "What we talk about when we talk about context," *Personal and ubiquitous computing*, vol. 8, no. 1, pp. 19–30, 2004.

[82] Matthew Ball, Vic Callaghan, and Michael Gardner, "An adjustable-autonomy agent for intelligent environments," in *Intelligent Environments (IE), 2010 Sixth International Conference on*. IEEE, 2010, pp. 1–6.

[83] Gerhard Fischer, "Context-aware systems: the'right'information, at the'right'time, in the'right'place, in the'right'way, to the'right'person," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM, 2012, pp. 287–294.

[84] Betty HC Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*, pp. 1–26. Springer, 2009.

[85] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, pp. 1–32. Springer, 2013.

[86] Brian Y Lim, "Improving trust in context-aware applications with intelligibility," in *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct*. ACM, 2010, pp. 477–480.

[87] Brian Y Lim and Anind K Dey, "Toolkit to support intelligibility in context-aware applications," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, 2010, pp. 13–22.

[88] Asier Aztiria, Juan Carlos Augusto, Rosa Basagoiti, Alberto Izaguirre, and Diane J Cook, "Learning frequent behaviors of the users in intelligent environments," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, vol. 43, no. 6, pp. 1265–1278, 2013.

[89] Gerhard Fischer, "End-user development and meta-design: Foundations for cultures of participation," in *End-user development*, pp. 3–14. Springer, 2009.

[90] Jadwiga Indulska and Peter Sutton, "Location management in pervasive systems," in *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*. Australian Computer Society, Inc., 2003, pp. 143–151.

[91] Klaus Pohl, *Requirements engineering: fundamentals, principles, and techniques*, Springer Publishing Company, Incorporated, 2010.

[92] Bashar Nuseibeh and Steve Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 35–46.

[93] Suzanne Robertson and James Robertson, *Mastering the requirements process: getting requirements right*, Addison-Wesley, 2012.

[94] Annie I Anton, "Goal-based requirements analysis," in *Requirements Engineering, 1996., Proceedings of the Second International Conference on*. IEEE, 1996, pp. 136–144.

[95] Anthony Finkelstein and Andrea Savigni, "A Framework for Requirements Engineering for Context-Aware Services," in *In Proc. of 1 st International Workshop From Software Requirements to Architectures (STRAW)*, 2001, pp. 200–1.

[96] Katsunori Oyama, Hojun Jaygarl, Jinchun Xia, Carl K Chang, Atsushi Takeuchi, and Hiroshi Fujimoto, "Requirements analysis using feedback from context awareness systems," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. IEEE, 2008, pp. 625–630.

[97] Katsunori Oyama, Atsushi Takeuchi, and Hiroshi Fujimoto, "CAPIS model based software design method for sharing experts' thought processes," in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*. IEEE, 2006, vol. 1, pp. 307–316.

[98] Russell L Ackoff, "From data to wisdom," *Journal of applied systems analysis*, vol. 16, no. 1, pp. 3–9, 1989.

[99] Luciano Baresi, Liliana Pasquale, and Paola Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*. IEEE, 2010, pp. 125–134.

[100] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas, "Goal-directed requirements acquisition," *Science of computer programming*, vol. 20, no. 1, pp. 3–50, 1993.

[101] Robert Darimont, Emmanuelle Delor, Philippe Massonet, and Axel van Lamsweerde, "GRAIL/KAOS: an environment for goal-driven requirements engineering," in *Proceedings of the 19th international conference on Software engineering*. ACM, 1997, pp. 612–613.

[102] Axel Van Lamsweerde and Emmanuel Letier, "From object orientation to goal orientation: A paradigm shift for requirements engineering," in *Radical Innovations of Software and Systems Engineering in the Future*, pp. 325–340. Springer, 2004.

[103] Alistair Sutcliffe, "Scenario-based requirements engineering," in *Requirements engineering conference, 2003. Proceedings. 11th IEEE international*. IEEE, 2003, pp. 320–329.

[104] Norbert Seyff, Florian Graf, Paul Grünbacher, and Neil Maiden, "Mobile discovery of requirements for context-aware systems," in *Requirements Engineering: Foundation for Software Quality*, pp. 183–197. Springer, 2008.

[105] Norbert Seyff, Florian Graf, Neil Maiden, and Paul Grünbacher, "Scenarios in the wild: Experiences with a contextual requirements discovery method," in *Requirements Engineering: Foundation for Software Quality*, pp. 147–161. Springer, 2009.

[106] Alistair Sutcliffe, Stephen Fickas, and McKay Moore Sohlberg, "PC-RE: a method for personal and contextual requirements engineering with some experience," *Requirements Engineering*, vol. 11, no. 3, pp. 157–173, 2006.

[107] Wassiou Sitou and Bernd Spanfelner, "Towards requirements engineering for context adaptive systems," in *Computer Software and Applications Conference,*

*2007. COMPSAC 2007. 31st Annual International*. IEEE, 2007, vol. 2, pp. 593–600.

[108] Carl Evans, Lindsey Brodie, and Juan Carlos Augusto, "Requirements Engineering for Intelligent Environments," in *Intelligent Environments (IE), 2014 International Conference on*. IEEE, 2014, pp. 154–161.

[109] Tomás Ruiz-López, Manuel Noguera, MaríA José Rodríguez, José Luis Garrido, and Lawrence Chung, "REUBI: A requirements engineering method for ubiquitous systems," *Science of Computer Programming*, vol. 78, no. 10, pp. 1895–1911, 2013.

[110] Michael Jesse Chonoles and James A Schardt, *UML 2 for Dummies*, John Wiley & Sons, 2011.

[111] Alberto Rodrigues da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015.

[112] "Object Management Group's official website," https://www.omg.org/, [Online; Last accessed 25-June-2018].

[113] R Geoff Dromey, "From requirements to design: Formalizing the key steps," in *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*. IEEE, 2003, pp. 2–11.

[114] Tadao Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[115] Armand Hatchuel and Benoit Weil, "A new approach of innovative Design: an introduction to CK theory.," in *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*, 2003.

[116] Terry Halpin and Tony Morgan, *Information modeling and relational databases*, Morgan Kaufmann, 2010.

[117] "Unified Modelling Language," http://www.omg.org/, [Online; Last accessed 25-June-2018].

[118] OMG, "OMG Universal Modeling Language (UML), Version 2.5," Tech. Rep., Object Management Group, 2015.

[119] Sinan Si Alhir, *Guide to Applying the UML*, Springer Science & Business Media, 2006.

[120] Marcos Vinicius Linhares, Alexandre Jose da Silva, and Rômulo Silva De Oliveira, "Empirical evaluation of SysML through the modeling of an industrial automation unit," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 2006, pp. 145–152.

[121] Rubén Fuentes-Fernández, Jorge J Gómez-Sanz, and Juan Pavón, "Understanding the human context in requirements elicitation," *Requirements engineering*, vol. 15, no. 3, pp. 267–283, 2010.

[122] Manzoor Ahmad, Jean-Michel Bruel, Régine Laleau, and Christophe Gnaho, "Using RELAX, SysML and KAOS for ambient systems requirements modeling," *Procedia Computer Science*, vol. 10, pp. 474–481, 2012.

[123] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Jean-Michel Bruel, "RELAX: a language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, no. 2, pp. 177–196, 2010.

[124] Eric SK Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*. IEEE, 1997, pp. 226–235.

[125] John Mylopoulos, Lawrence Chung, and Brian Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[126] Hugo Estrada, Alicia Martínez Rebollar, Oscar Pastor, and John Mylopoulos, "An empirical evaluation of the i* framework in a model-based software generation environment," in *International Conference on Advanced Information Systems Engineering*. Springer, 2006, pp. 513–527.

[127] U2TP Consortium, "UML 2.0 Testing Profile, Version 2.0," Tech. Rep., Object Management Group, 2017.

[128] Quan Z Sheng and Boualem Benatallah, "ContextUML: a UML-based modeling language for model-driven development of context-aware web services," in *Mobile Business, 2005. ICMB 2005. International Conference on*. IEEE, 2005, pp. 206–212.

[129] James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley Professional, 2005.

[130] Estefanía Serral, Pedro Valderas, and Vicente Pelechano, "A model driven development method for developing context-aware pervasive systems," in *Ubiquitous Intelligence and Computing*, pp. 662–676. Springer, 2008.

[131] Estefanía Serral, Pedro Valderas, and Vicente Pelechano, "Towards the model driven development of context-aware pervasive systems," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 254–280, 2010.

[132] J. Miller and J. Mukerji, "MDA Guide Version 1.0.1," Tech. Rep., Object Management Group (OMG), 2003.

[133] Ricardo Tesoriero, José A Gallud, María Dolores Lozano, and Victor M Ruiz Penichet, "CAUCE: Model-driven Development of Context-aware Applications for Ubiquitous Computing Environments.," *Journal of Universal Computer Science*, vol. 16, no. 15, pp. 2111–2138, 2010.

[134] Andrea Sindico and Vincenzo Grassi, "Model driven development of context aware software systems," in *International workshop on context-oriented programming*. ACM, 2009, p. 7.

[135] José Ramón Hoyos, Jesús García-Molina, and Juan Antonio Botía, "MLContext: a context-modeling language for context-aware systems," *Electronic Communications of the EASST*, vol. 28 (2010), 2010.

[136] Anind K Dey, Timothy Sohn, Sara Streng, and Justin Kodama, "iCAP: Interactive prototyping of context-aware applications," in *Pervasive Computing*, pp. 254–271. Springer, 2006.

[137] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 803–812.

[138] Justin Huang and Maya Cakmak, "Supporting mental model accuracy in trigger-action programming," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 215–225.

[139] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf, *End-User Development: An Emerging Paradigm*, vol. 9 of *Human-Computer Interaction Series*, Springer Netherlands, Dordrecht, 2006.

[140] Alessandro A Nacci, Bharathan Balaji, Paola Spoletini, Rajesh Gupta, Donatella Sciuto, and Yuvraj Agarwal, "BuildingRules: a trigger-action based system to manage complex commercial buildings," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 2015, pp. 381–384.

[141] Gabriella Lucci and Fabio Paternò, "Understanding end-user development of context-dependent applications in smartphones," in *Human-Centered Software Engineering*, pp. 182–198. Springer, 2014.

[142] Jongmyung Choi, "Context-driven requirements analysis," in *Computational Science and Its Applications–ICCSA 2007*, pp. 739–748. Springer, 2007.

[143] Jongmyung Choi and Youngho Lee, "Use-Case Driven Requirements Analysis for Context-Aware Systems," in *Computer Applications for Bio-technology, Multimedia, and Ubiquitous City*, pp. 202–209. Springer, 2012.

[144] Brecht Desmet, Jorge Vallejos, Pascal Costanza, Wolfgang De Meuter, and Theo D'Hondt, "Context-oriented domain analysis," in *Modeling and Using Context*, pp. 178–191. Springer, 2007.

[145] Carlos Rodrıguez-Domınguez, Tomás Ruiz-López, Kawtar Benghazi, and José Luis Garrido, "Designing a Middleware-Based Framework to Support Multiparadigm Communications in Ubiquitous Systems," *Ambient Intelligence-Software and Applications*, p. 163, 2012.

[146] Tomás Ruiz-López, Carlos Rodríguez-Domínguez, Manuel Noguera, and María José Rodríguez, "A Model-Driven Approach to Requirements Engineering in Ubiquitous Systems," in *Ambient Intelligence-Software and Applications*, pp. 85–92. Springer, 2012.

[147] Javier Munoz, Vicente Pelechano, and Joan Fons, "Model driven development of pervasive systems," in *International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*. Citeseer, 2004, vol. 1, pp. 3–14.

[148] Modeliosoft, "Modelio," https://www.modelio.org/, [Online; Last accessed 25-June-2018].

[149] "Papyrus," https://eclipse.org/papyrus/, [Online; Last accessed 25-June-2018].

[150] "Eclipse," https://eclipse.org/, [Online; Last accessed 25-June-2018].

[151] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," Tech. Rep., Object Management Group, 2011.

[152] Object Management Group, "Service oriented architecture Modeling Language (SoaML) Specification," Tech. Rep., Object Management Group, 2012.

[153] Van (Open Group) Haren, *TOGAF Version 9.1*, Van Haren Publishing, 2011.

[154] Alessandra Bagnato, Andrey Sadovykh, Etienne Brosse, and Tanja EJ Vos, "The OMG UML Testing Profile in Use–An Industrial Case Study for the Future Internet Testing," in *Software maintenance and reengineering (csmr), 2013 17th european conference on*. IEEE, 2013, pp. 457–460.

[155] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," Tech. Rep., W3C, 2017.

[156] Thomas Strang and Claudia Linnhoff-Popien, "A Context Modeling Survey," in *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*. 2004, pp. 31–41, Springer-Verlag.

[157] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca, "A Data-oriented Survey of Context Models," *SIGMOD Rec.*, vol. 36, no. 4, pp. 19–26, Dec. 2007.

[158] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[159] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[160] Daniele Riboni and Claudio Bettini, "Context-aware activity recognition through a combination of ontological and statistical reasoning," in *International Conference on Ubiquitous Intelligence and Computing*. Springer, 2009, pp. 39–53.

[161] Hristijan Gjoreski, *Context-based Reasoning in Ambient Intelligence*, Ph.D. thesis, PhD Thesis, IPS Jožef Stefan, Ljubljana, Slovenia, 2015.

[162] Unai Alegre-Ibarra, Juan Carlos Augusto, and Asier Aztiria, "Temporal Reasoning for Intuitive Specification of Context-Awareness," in *Intelligent Environments (IE), 2014 International Conference on*. IEEE, 2014, pp. 234–241.

[163] Antony Galton and Juan Carlos Augusto, "Stratified causal theories for reasoning about deterministic devices and protocols," in *Temporal Representation and Reasoning, 2002. TIME 2002. Proceedings. Ninth International Symposium on*. IEEE, 2002, pp. 52–54.

[164] Unai Alegre-Ibarra, "Context-awareness in intelligent environments," M.S. thesis, Mondragon University faculty of engineering, 2014.

[165] M. Estibalitz Aranbarri-Zinkunegi, "Improving the Pattern Learning System integrating the reasoning system," M.S. thesis, Mondragon University faculty of engineering, 2017.

[166] Dean Kramer, "AContextLib library," `https://github.com/deankramer/aContextLib`, [Online; Last accessed 25-June-2018].

[167] Dean Kramer and Unai Alegre-Ibarra, ," `https://github.com/ualegre/aContextReasoner`, 2015, [Online; Last accessed 25-June-2018].

[168] "C-SPARQL," `http://streamreasoning.org/resources/c-sparql`, [Online; Last accessed 25-June-2018].

[169] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, EMANUELE DELLA VALLE, and Michael Grossniklaus, "C-SPARQL: a continuous query language for RDF data streams," *International Journal of Semantic Computing*, vol. 4, no. 01, pp. 3–25, 2010.

[170] Edsger W Dijkstra, "The humble programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, 1972.

[171] Juan Carlos Augusto, "Increasing reliability in the development of intelligent environments," in *Intelligent Environments 2009: Proceedings of the 5th International Conference on Intelligent Environments, Barcelona 2009*. IOS Press, 2009, vol. 2, p. 134.

[172] Juan Carlos Augusto, Huiru Zheng, Maurice Mulvenna, Haiying Wang, William Carswell, and P Jeffers, "Design and modelling of the nocturnal AAL care system," in *Ambient Intelligence-Software and Applications*, pp. 109–116. Springer, 2011.

[173] Gerard J Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[174] Davy Preuveneers and Yolande Berbers, "Consistency in Context-Aware Behavior: a Model Checking Approach.," in *Intelligent Environments (Workshops)*, 2012, pp. 401–412.

[175] Liliana D'Errico and Michele Loreti, "Context aware specification and verification of distributed systems," in *Trustworthy Global Computing*, pp. 142–159. Springer, 2012.

[176] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese, "KLAIM: A kernel language for agents interaction and mobility," *Software Engineering, IEEE Transactions on*, vol. 24, no. 5, pp. 315–330, 1998.

[177] Yepang Liu, Chang Xu, and SC Cheung, "Afchecker: Effective model checking for context-aware adaptive applications," *Journal of Systems and Software*, vol. 86, no. 3, pp. 854–867, 2013.

[178] Michele Sama, Sebastian Elbaum, Franco Raimondi, David S Rosenblum, and Zhimin Wang, "Context-aware adaptive applications: Fault patterns and their automated identification," *Software Engineering, IEEE Transactions on*, vol. 36, no. 5, pp. 644–661, 2010.

[179] JoonSeok Park, Mikyeong Moon, Seongjin Hwang, and Keunhyuk Yeom, "CASS: A context-aware simulation system for smart home," in *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*. IEEE, 2007, pp. 461–467.

[180] Zhimin Wang, Sebastian Elbaum, and David S Rosenblum, "Automated generation of context-aware tests," in *Software Engineering, 2007. ICSE 2007. 29th International Conference on*. IEEE, 2007, pp. 406–415.

[181] Julien Bruneau, Wilfried Jouve, and Charles Consel, "DiaSim: A parameterized simulator for pervasive computing applications," in *Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International*. IEEE, 2009, pp. 1–10.

[182] Lian Yu, Wei Tek Tsai, Yanbing Jiang, and Jerry Gao, "Generating Test Cases for Context-Aware Applications Using Bigraphs," in *Software Security and Reliability, 2014 Eighth International Conference on*. IEEE, 2014, pp. 137–146.

[183] Roberto Cavada, Alessandro Cimatti, Gavin Jochim, Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev, "NuSMV 2.6 User Manual," .

[184] John M Bryson, Michael Quinn Patton, and Ruth A Bowman, "Working with evaluation stakeholders: A rationale, step-wise approach and toolkit," *Evaluation and program planning*, vol. 34, no. 1, pp. 1–12, 2011.

[185] John M Bryson, "What to do when stakeholders matter: stakeholder identification and analysis techniques," *Public management review*, vol. 6, no. 1, pp. 21–53, 2004.

[186] Simon Jones, Sukhvinder Hara, and Juan Augusto, "e-FRIEND: an Ethical Framework for Intelligent Environment Development," in *Ethics and Information Technology*. Springer, 2015, vol. 17, pp. 11–25.

[187] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos, *Non-functional requirements in software engineering*, vol. 5, Springer Science & Business Media, 2012.

[188] R Edward Freeman, *Strategic management: A stakeholder approach*, Boston: Pitman/Ballinger, 1984.

[189] ISO25000 Software Product Quality, "ISO25010," http://iso25000.com/index.php/en/iso-25000-standards/iso-25010, [Online; Last accessed 25-June-2018].

[190] David Maulsby, Saul Greenberg, and Richard Mander, "Prototyping an intelligent agent through Wizard of Oz," in *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. ACM, 1993, pp. 277–284.

[191] Hyun-Seok Min, "Traceability Guideline for Software Requirements and UML Design," *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, no. 01, pp. 87–113, 2016.

[192] Michel dos Santos Soares and Jos Vrancken, "Requirements specification and modeling through SysML," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*. IEEE, 2007, pp. 1735–1740.

[193] Matthew Hause, Andrew Stuart, David Richards, and Jon Holt, "Testing safety critical systems with SysML/UML," in *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*. IEEE, 2010, pp. 325–330.

[194] Unai Alegre-Ibarra, "Requirements for Context-Aware Systems Engineering (RCASE) Tool," https://github.com/ualegre/rcase, 2016, [Online; Last accessed 25-June-2018].

[195] Alegre-Ibarra Unai and Julian Hallett, "POSEIDON questionnaires on the situations of interest," https://www.dropbox.com/sh/velul2rmos8d3en/AABnIVZZWNfdpDq4jrTqcKYaa?dl=0, [Online; Last accessed 25-June-2018].

[196] Weiping Wang, Qiang Chang, Qun Li, Zesen Shi, and Wei Chen, "Indoor-Outdoor Detection Using a Smart Phone Sensor," *Sensors*, vol. 16, no. 10, pp. 1563, 2016.

[197] Citymapper Limited, "Citymapper Transport Application, Official Website," https://citymapper.com/, 2011, [Online; Last accessed 25-June-2018].

[198] Uber Technologies Inc., "Uber API," https://developer.uber.com/, 2009, [Online; Last accessed 25-June-2018].

[199] J Augusto, Dean Kramer, Unai Alegre-Ibarra, Alexandra Covaci, and Adityarajsingh Santokhee, "The user-centred intelligent environments development process as a guide to co-create smart technology for people with special needs," *Universal Access in the Information Society*, pp. 1–16, 2017.

[200] Unai Alegre-Ibarra, "Official repository of the Design for Context-Aware Systems Engineering (DCASE) tool," https://github.com/ualegre/dcase, [Online; Last accessed 25-June-2018].

[201] Lukasz Golab and M Tamer Özsu, "Processing sliding window multi-joins in continuous queries over data streams," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, pp. 500–511.

[202] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 359–364.

[203] Michel dos Santos Soares and Jos LM Vrancken, "Model-Driven User Requirements Specification using SysML.," *JSW*, vol. 3, no. 6, pp. 57–68, 2008.

[204] Modeliosoft, "UML Testing Profile for Modelio," http://store.modelio.org/resource/modules/utp.html, [Online; Last accessed 25-June-2018].

[205] Unai Alegre-Ibarra, ," https://github.com/ualegre/m2nusmv, 2016, [Online; Last accessed 25-June-2018].

[206] Unai Alegre-Ibarra, ," https://github.com/ualegre/mreasoner-gui, 2014, [Online; Last accessed 25-June-2018].

[207] Asier Aztiria-Goenaga and M. Estibalitz Aranbarri-Zinkunegi, "Learning Frequent Patterns of User Behaviour (LFPUBS)," https://github.com/estibalitz/lfpubs, [Online; Last accessed 25-June-2018].

[208] Unai Alegre-Ibarra and Estibaliz Aranbarri-Zinkunegi, ," https://github.com/ualegre/lfpubs2m, 2016, [Online; Last accessed 25-June-2018].

[209] Eclipse, "Eclipse IDE for Java Developers," https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neon3, [Online; Last accessed 25-June-2018].

[210] JUnit Team, "JUnit Official Website," https://junit.org/junit5/, [Online; Last accessed 25-June-2018].

[211] Modeliosoft, "JUnit Model Tester for Modelio," http://store.modelio.org/resource/modules/junit.html, [Online; Last accessed 25-June-2018].

[212] Unai Alegre-Ibarra, ," https://github.com/ualegre/mreasoner-core, 2014, [Online; Last accessed 25-June-2018].

[213] Vera Control Ltd., "Vera official webpage," http://getvera.com/, [Online; Last accessed 25-June-2018].

[214] Unai Alegre-Ibarra, ," https://github.com/ualegre/vera-manager, 2016, [Online; Last accessed 25-June-2018].

[215] Fabio Somenzi, "CUDD: CU Decision Diagram Package, Release 3.0.0," in *http://vlsi.colorado.edu*. Department of Electrical, Computer, and Energy Engineering, University of Colorado at Boulder, 2015.

[216] Center for Information Technology at FBK-IRST Embedded Systems Unit, Model Checking group at Carnegie Mellon University, Mechanized Reasoning Group at University of Genova, and The Mechanized Reasoning Group at University of Trento, "NuSMV Official Webpage," http://nusmv.fbk.eu/, [Online; Last accessed 25-June-2018].

[217] Dean Kramer, ," https://github.com/deankramer/aContextLib, 2016, [Online; Last accessed 25-June-2018].

[218] "Documentation on Android Sensors," https://developer.android.com/guide/topics/sensors/sensors_overview.html, [Online; Last accessed 25-June-2018].

[219] Terry Winograd and Fernando Flores, *Understanding computers and cognition: A new foundation for design*, Norword, NJ: Ablex Publishing Corporation, 1986.

[220] Paul Dourish, "Seeking a foundation for context-aware computing," *Human–Computer Interaction*, vol. 16, no. 2-4, pp. 229–241, 2001.

[221] Bonnie A Nardi, *Context and consciousness: activity theory and human-computer interaction*, Mit Press, 1996.

[222] Dag Svanaes, "Context-aware technology: a phenomenological perspective," *Human–Computer Interaction*, vol. 16, no. 2-4, pp. 379–400, 2001.

[223] CL Oguego, Juan Carlos Augusto, A Muñoz, and Mark Springett, "Using argumentation to manage users' preferences," *Future Generation Computer Systems*, vol. 81, pp. 235–243, 2018.

[224] POSEIDON project, "Poseidon applications download guide," http://www.poseidon-project.org/secondary-users/app/, 2017.

[225] Mikel Mateo, ," https://github.com/mmateom/Weight-Manager, 2017, [Online; Last accessed 25-June-2018].

[226] Unai Alegre-Ibarra, "Official repository of the Requirements for Context-Aware Systems Engineering (RCASE) tool," https://github.com/ualegre/rcase/raw/master/rcase/target/RCase_0.4.7.jmdac, 2016, [Online; Last accessed 25-June-2018].

[227] Unai Alegre-Ibarra, "Official repository of the Ontologies for Context-Aware Systems Engineering (OCASE) tool," https://github.com/ualegre/ocase, 2015, [Online; Last accessed 25-June-2018].

[228] Modeliosoft, "Modelio Module Installation Guide," https://www.modelio.org/downloads/download-modelio.html, [Online; Last accessed 25-June-2018].

[229] "Modelio: Quickstart guide," https://www.modelio.org/quick-start-pages-35.html, [Online; Last accessed 25-June-2018].

[230] "Modelio: Official website," http://store.modelio.org/resource/modules/sysml-architect-open-source.html, [Online; Last accessed 25-June-2018].

[231] Unai Alegre-Ibarra, ," https://github.com/ualegre/mreasoner, 2016, [Online; Last accessed 25-June-2018].