The Capture and Evolution of Contextual Requirements: The Case of Adaptive Systems

by

Alessia Knauss
Dipl.-Math., Leibniz Universität Hannover, 2009

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

The Capture and Evolution of Contextual Requirements:
The Case of Adaptive Systems

by

Alessia Knauss
Dipl.-Math., Leibniz Universität Hannover, 2009

Supervisory Committee

Dr. Daniela Damian, Supervisor
(Department of Computer Science)

Dr. Hausi A. Müller, Departmental Member
(Department of Computer Science)

Dr. Xavier Franch, Outside Member
(Service and Information System Engineering Department, Universitat Politècnica de Catalunya)

Supervisory Committee

ABSTRACT

Today's software systems are becoming increasingly *integrated* into the lives of their end-users and their *ever-changing environments and needs.* These demands lead to a growing *complexity* of systems. The development of adaptive systems is a promising way to manage this complexity. Adaptive systems are able to adapt their behavior at operation time while considering the changing operational environment to maximize the satisfaction of end-user needs. However, adaptive systems have their own challenges to overcome. Especially, requirements engineering for adaptive systems is challenging given the fact that requirements are active runtime entities and can change at runtime. Requirements engineering activities have not only to take place at design but also at runtime. Requirements engineering for adaptive systems is an emerging research area that has so far received little attention, compared to other research areas (e.g., architecture) for adaptive systems.

Adaptive systems need to have a full understanding of the context in order to handle the complexity and satisfy end-user needs. Therefore, a new trend in requirements engineering for adaptive systems emerged to document requirements with the context in which the requirements are valid. Such *contextual requirements* necessitate adaptive systems to consider and define context in order to fully understand the requirements at operation time. Further, adaptive systems must be able to cope with *uncertainty* inherent in a changing runtime environment. Otherwise, adaptive systems will not be able to satisfy end-user needs. Therefore, after the system has been deployed, support for the evolution of contextual requirements is needed, too. The trend of considering context as part of a contextual requirement poses new challenges in the field of requirements engineering.

This dissertation investigates the *capture and evolution of contextual requirements* for adaptive systems, which leads to three contributions: First, this dissertation presents a framework that differentiates between context and requirements as two separate entities in contextual requirements that can be captured and can be evolved independently. It is especially necessary to capture and evolve the essential context to support the ability of a system to adapt to fulfilling the needs of its end-users, whose requirements and context are constantly changing.

The framework is then applied in two case studies. The first case study investigates the usefulness of existing requirements elicitation techniques for the elicitation of contextual requirements. This dissertation's second contribution is the empirical evidence that existing requirement elicitation techniques can be used for the capture of contextual requirements at design time. We propose a combination of interviews, focus groups and prototyping that we found useful in eliciting contextual requirements in our case study. The second study develops and evaluates techniques to support the evolution of context when contextual requirements are validated at runtime. For this purpose we propose an approach which uses machine learning and feedback loops to support the evolution of contextual requirements and which represents the third contribution of this dissertation.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I have been very fortunate to work with an outstanding dissertation committee. I would like to express my deepest gratitude and appreciation to my supervisor Daniela Damian. Thank you for believing in me and giving me the chance to study this exciting topic, despite the doubts and risks, for your permanent support, your patience and understanding. Without this, I would not have been able to accomplish this dissertation.

I am deeply indebted to my dissertation committee members Hausi Müller, Xavier Franch, and Bashar Nuseibeh. Hausi, thank you for introducing the topic of self-adaptive systems closer to me as well as for insightful discussions. I felt very welcome to the RIGI group, like an external group member. It was very encouraging. Xavier, thank you for seeing the potential in this research topic, for challenging me, your mentoring and support. It kept me going. Bashar, thank you for your precious time and stimulating questions. The content of this dissertation is based on two investigations. I would like to show my deepest gratitude to Kane Kilbey from the University of Victoria to involve us in the replacement of the applicant tracking system at UVic, as well as to all participants in the study. Additionally, I cannot express how thankful I am to the OAR Northwest team members Adam Kreek, Jordan Hanssen, Greg Spooner, Markus Pukonen, and Pat Fleming. They not only gave us the chance to study the evolution of contextual requirements based on their rowing trip, but took all possible effort to collect the data that we needed while being totally exhausted during their rowing trip. You guys are just amazing!

The life during my dissertation study was made very special by certain people being around. First and most important, the members of the SEGAL lab. Thank you Jordan, Braden, Angela, Adrian, Jorge, Germán, Arber, Aminah, Francis, Joyti, Prashant, Ville and Daniel. Thank you Eirini for taking time and effort in supporting me at times when needed. Thank you Norha, Lorena, Nina, Andy, and Pratik for our discussions on self-adaptive systems. Special thanks to the Computer Science secretaries, especially Wendy and Nancy for their support.

I would also like to thank Kurt Schneider for getting me involved into the topic of requirements engineering for adaptive systems and ecosystems. Being part of his group gave me the opportunity to gain some invaluable experiences.

This dissertation would not have been possible without the support and encouragement of my family and friends. I would like to show my deepest appreciation to my mom, who raised and taught me that you have to work hard and be patient to reach the goals you set for yourself. I am greatly indebted to my husband for going beyond boundaries to support me to make this dissertation possible and for being a great mentor to learn from. Supporting his academic path gave me a preview of what I have to expect, so I was prepared for the tough times. Thanks to my little sweetheart, Marie, for showing me that you sometimes have to pause to explore the beauty of the world.

DEDICATION

To my family.

Chapter 1

Introduction

Today's software systems are increasingly *complex* with regards to their integration in larger system landscapes and the trend towards their ubiquity [93, 26]. This complexity makes maintenance and evolution of such systems a challenge for software engineers. Adaptive systems promise to decrease the cost of handling the complexity of software systems at runtime [110].

Adaptive systems are defined as "computer-based systems that are capable of recognizing changes in the domain they share an interface with, and at the same time being able to change their behavior to adapt to the changing conditions without necessary direct user interaction" [123]. The goal of an adaptive system is to provide a better system for its users, which operates well in different situations through the use of context-awareness and adaptation [123, 18]. To fulfill this goal, an adaptive system must be given a thorough understanding of the context during its design phase in order to satisfy end-user needs in different situations [120, 69, 12].

Uncertainty makes it impossible to completely understand end-user needs and runtime environment during the design phase [135, 106]. Therefore, requirements engineering activities have not only to take place during design, but also at runtime [103]. During use of the system (we refer to it as *runtime*), the environment and end-user expectations change and require the system to evolve its knowledge of requirements and context in order to fulfill end-user needs [92].

Therefore, focusing solely on the requirements in requirements engineering activities becomes insufficient given the importance of specific context that can change at runtime [102, 70]. For certain systems adaptation can even benefit from keeping the human in the loop [72]. The importance of context in requirements engineering is not a new phenomenon. Understanding the context is a well known and important activity for capturing requirements [98]. For example, contextual inquiry is an integral part of customer focused research [20]. But for adaptive systems, in which the context is an equally important part and is documented together with requirements, existing requirements engineering approaches only focusing on understanding the context are insufficient.

Context has to become an active constituent of an adaptive system [56, 58, 111, 125, 123] and has to be monitored and captured not only at design, but also at runtime [14, 21, 96, 116, 117, 123, 134]. In this situation, there exists a need to further our understanding of the relationship between context and requirements in requirements engineering for adaptive systems.

1.1 Problem Statement

In our preliminary works we focused on investigating requirements approaches that equally emphasize requirements and their context, such as the capture of requirements in their context [24, 25, 122] and end-user involvement in requirements elicitation activities [23, 77, 76].[1] We discovered that in some cases even the decision about when a requirement is needed depends on context.

As an example, we conducted a study [24] in which a system guiding drivers to a vacant parking space in a multi-level parking garage was tested. The system at this stage tried to fill up the parking lot from the top. We found that users chose the parking space fitting a particular context regardless of the recommendation of the system.

In our case study the adaptive system was a guidance system with two LED arrows that directed drivers to a floor with free parking spaces. This system adapted by showing users the direction to the higher level floor with empty parking spaces. While users were happy with this system at the beginning of its operation as their requirements were fulfilled, their needs seemed to change: Users eventually did not follow the advice of the system and chose the lower (closer) level, even if free parking spaces were available on a higher (further) level. Observations of user behavior in the basement garage shed more light on this phenomenon: Users preferred free parking spaces not adjacent to pillars. The newly captured requirement represents a

[1]Please note that Brill is my maiden name.

contextual requirement and can be formulated in the following way: *If there exist free parking spaces not adjacent to pillars on ground floor, direct the car to ground floor.* The requirement of directing the cars to the ground level was needed in the context of free parking spaces not adjacent to pillars. The context "free parking spaces not adjacent to pillars" was not considered by the adaptive system, leaving end-user needs unfulfilled. If such parking spaces are not available on ground level and there are parking spaces available on the higher level, then the system should give directions to the higher level. This means that depending on the context, the system's behavior needs to change.

In a literature review we found similar observations of contextual requirements – requirements that depend on context [70, 7, 21]. Nevertheless, approaches to identify and discover requirements together with relevant context are currently underrepresented in research and practice, as also stated by Cheng et al. [33, 34]. While approaches to modeling system requirements and their context have been proposed to deal with complexity [9], capturing and evolving requirements and relevant context for adaptive systems continue to pose significant challenges for requirements engineering research [6]. To address topics like uncertainty affecting requirements and relevant context [127, 106] threatening the satisfaction of contextual requirements at runtime [56, 34], research needs to consider requirements engineering activities during design, but also at runtime.

1.2 Research Goal and Questions

To further the research on contextual requirements, particularly for adaptive systems, this dissertation investigates the following research goal:

Investigate how to capture contextual requirements and manage their evolution to address uncertainty during the design and operation of adaptive systems.

We address the research goal based on three research questions using exploratory research methods. Figure 1.1 gives an overview of our research methodology. The first research question investigates how to study the capture and evolution of contextual

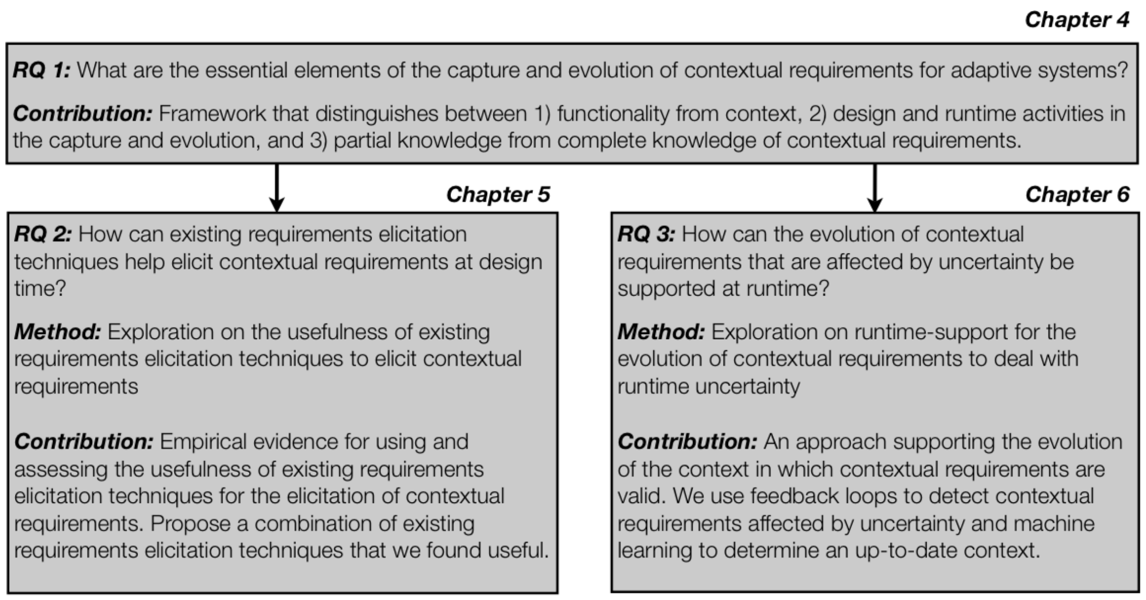

Chapter 4
RQ 1: What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?
Contribution: Framework that distinguishes between 1) functionality from context, 2) design and runtime activities in the capture and evolution, and 3) partial knowledge from complete knowledge of contextual requirements.
Chapter 5
RQ 2: How can existing requirements elicitation techniques help elicit contextual requirements at design time?
Method: Exploration on the usefulness of existing requirements elicitation techniques to elicit contextual requirements
Contribution: Empirical evidence for using and assessing the usefulness of existing requirements elicitation techniques for the elicitation of contextual requirements. Propose a combination of existing requirements elicitation techniques that we found useful.
Chapter 6
RQ 3: How can the evolution of contextual requirements that are affected by uncertainty be supported at runtime?
Method: Exploration on runtime-support for the evolution of contextual requirements to deal with runtime uncertainty
Contribution: An approach supporting the evolution of the context in which contextual requirements are valid. We use feedback loops to detect contextual requirements affected by uncertainty and machine learning to determine an up-to-date context.

Figure 1.1: Overview of our research questions and contributions.

requirements. The second research question addresses the capture (i.e., elicitation) of contextual requirements during the design phase through the use of existing requirements elicitation techniques. The third research question pursues evolution of contextual requirements at runtime to deal with uncertainty. In the following, we describe each of the research questions briefly. Further information on the research design can be found in Chapter 3.

Contextual requirements are a specific kind of requirements and it is unclear whether existing requirements engineering techniques can be used for capturing contextual requirements. As such kind of requirements include a detailed description of the context, both context and requirement (also referred to as expected system behavior from the view of the system) have to be captured. Capture of contextual requirements is used in this dissertation as a broad term encompassing the process of learning and understanding (contextual) requirements.

The captured context as well as system behavior can change at runtime. Therefore, we have to consider evolution of contextual requirements to ensure that the software system is constantly maximizing the satisfaction of end-user requirements at runtime and does not fail when it encounters uncertainty.

Our first research question (RQ 1) investigates how to study the capture and evolution of such requirements that depend on context:

Research Question 1: *What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?*

We present a framework to structure our understanding of as well as the activities of the capture and evolution of contextual requirements for adaptive systems at design and runtime. The framework consists of three main concepts: 1) distinguishing functionality from context in contextual requirements, 2) distinguishing between design and runtime activities in the capture and evolution of contextual requirements, and 3) distinguishing partial and complete knowledge of contextual requirements. The framework is intended to help requirements analysts, designers, and operators of adaptive systems explore techniques that support the capture and evolution of contextual requirements.

The framework divides the activity of capturing contextual requirements into two distinct activities – requirements elicitation and discovery. Motivated by literature that differentiates requirements engineering activities at design and runtime, the framework differentiates between capturing contextual requirements at design time (referred to as elicitation) as well as capturing contextual requirements at runtime (referred to as discovery). The evolution of contextual requirements is defined as identification of changes to existing contextual requirements at runtime.

The separation of system behavior and context in contextual requirements allows to reason about each part separately. The framework discusses the transitions from partial knowledge about contextual requirements to their correct and complete specification and in relation to the three activities of requirements elicitation, discovery, and evolution. This process can be applied at design and runtime. Additionally, partial knowledge about contextual requirements can be reused from the design phase and complemented during runtime.

Contribution 1: The framework facilitates reasoning about the activities necessary for the elicitation, discovery, and evolution of contextual requirements. It facilitates decisions about which techniques to use for each of these activities. The three essential elements of the capture and evolution of contextual requirements – the differentiation between the three activities elicitation, discovery, and evolution, the differentiation between context and system behavior in contextual requirements, and the differentiation between partial and complete knowledge of contextual requirements – offer value for practitioners as well as researchers in the field. Practitioners can use the framework to set up the requirements capture process for adaptive systems. For researchers the framework provides guidance to focus research activities and define research questions.

In RQ 2 and RQ 3 this dissertation investigates more deeply two cases in our framework, more specifically the case of capturing contextual requirements at design time and the case of evolving contextual requirements at runtime This dissertation does not investigate techniques on the discovery of contextual requirements at runtime.

Literature presents approaches to the elicitation of requirements as well as context separately. However, approaches that combine the elicitation of requirements and then their context of validity are to the best of our knowledge underrepresented and require further research attention. Instead of starting to investigate new techniques for the elicitation of contextual requirements, we investigate in research question 2 whether and how existing requirements elicitation techniques are useful for the elicitation of contextual requirements.

Research Question 2: *How can existing requirements elicitation techniques help elicit contextual requirements at design time?*

In a case study conducted with the Human Resources Department at the University of Victoria we explored the usefulness of existing requirements elicitation techniques in eliciting contextual requirements when revising its job applicant tracking system. Reports of actual requirements engineering practice in real projects are rare in literature [86]. In-depth studies of requirements engineering practice are difficult to carry out given the human, organizational, and political aspects that surround software projects. Yet they are very important in providing insights about the application of requirements engineering techniques in eliciting requirements in practice.

In our case study with the University of Victoria, we acted as the requirements analyst in the project. The applicant tracking system was to be replaced to better serve the needs of thousands of stakeholders. We applied different requirements elicitation techniques (e.g., interviews, prototyping, scenarios, goal-based approaches, and focus groups) to explore their usefulness in eliciting contextual requirements.

We were able to document a number of contextual requirements when applying existing requirements elicitation techniques in a particular order: First we identified requirements through interviews and focus groups and attempted to understand the rationale behind them through interviews. We then used prototyping to get a detailed understanding of these requirements in context. Next, we identified conflicts between different end-users when discussing requirements in detail together in focus groups. These discussions helped identify the need for contextual requirements. Finally we elicited, through interviews with the respective end-users, the different contexts related to requirements so that we could document the contextual requirements.

Contribution 2: This dissertation brings empirical evidence on using and assessing the usefulness of existing requirements elicitation techniques for the elicitation of contextual requirements in a real software acquisition project. We propose a combination of existing requirements elicitation techniques that we found useful for the elicitation of contextual requirements.

Keeping user requirements continuously satisfied at runtime requires evolution of contextual requirements. Current approaches either focus on requirements evolution or context evolution. Evolution of contextual requirements needs further investigations, especially the development of automatic support for the evolution of contextual requirements to be used in adaptive systems.

Therefore, the third research question (RQ 3) investigates support for the evolution of contextual requirements:

Research Question 3: *How can the evolution of contextual requirements that are affected by uncertainty be supported at runtime?*
Runtime uncertainty might affect contextual requirements at runtime [106]. In order to keep contextual requirements satisfied, a system must keep the knowledge about contextual requirements up-to-date. Hence an adaptive system has to support the evolution of its contextual requirements to fulfill end-user needs.

We present an approach that updates the knowledge about contextual requirements with up-to-date information about the context in which contextual requirements are valid at runtime. The approach detects contextual requirements that are affected by uncertainty and integrates data mining algorithms that are used on contextual data (i.e., sensor data) to update the context in which the system behavior is valid.

We evaluated the approach based on a case study in an unpredictable environment, the ocean, with a high impact from the environment on the requirements. We mined contextual data from 46 sensors for five contextual requirements to make the context measurable in which contextual requirements were valid. Further, we analyzed the cases in which our approach would trigger the evolution of contextual requirements that are affected by uncertainty. We could show that our approach would achieve great results for 4 out of 5 contextual requirements.

Contribution 3: This dissertation develops and evaluates an approach to support the evolution of the context in which contextual requirements are valid. The approach uses feedback loops to detect contextual requirement affected by runtime uncertainty, with a need for evolution. It applies data mining algorithms on sensor data to determine an up-to-date context in which contextual requirements are valid.

1.3 Dissertation Structure

This dissertation is organized as follows:

Chapter 2 – Background and Related Work. In this chapter we depict background information necessary to understand the concepts presented in this dissertation as well as related work.

Chapter 3 – Research Design. We detail the research design of this dissertation and outline the research methods for each of the research questions.

Chapter 4 – A Framework for Contextual Requirements. We present our first contribution of this dissertation, a framework on the capture and evolution of contextual requirements.

Chapter 5 – Eliciting Contextual Requirements at Design Time. This chapter presents a case study on the use of existing requirements elicitation techniques for the elicitation of contextual requirements at design time.

Chapter 6 – Support Evolution of Contextual Requirements at Runtime. Our last contribution is an approach developed to support the evolution of contextual requirements to deal with uncertainty affecting the execution of contextual requirements at runtime.

Chapter 7 - Contributions and Future Work. In this chapter we revisit the research questions along with the contribution of the dissertation, and discuss future work.

Chapter 2

Background and Related Work

This chapter presents background for the concepts of this dissertation as well as related works. The literature review was conducted in an iterative way, using the snow ball method. We studied existing related work at the beginning of the dissertation study. Over the last three years we continuously complemented the literature review using databases including IEEE Xplore, Springer Link, and Google Scholar. After identifying relevant papers, we applied the snowball approach making sure we have considered all relevant publications. Additionally, we monitored publications from related conferences, workshops, and journals, including the International Requirements Engineering Conference, Requirement Engineering Journal, International Conference on Software Engineering, Transactions on Software Engineering Journal, Software Engineering for Adaptive and Self-Managing Systems, International Working Conference on Requirements Engineering: Foundation for Software Quality, as well as workshops taking place at the mentioned conferences.

2.1 Capturing Requirements and Context in Requirements Engineering

Capturing Requirements

A *requirement* expresses a system behavior needed or wished for by stakeholders [97]. The process of learning and understanding stakeholder's needs and wishes is often referred to as *requirements elicitation* [140]. There exist multiple definitions for requirements elicitation, as well as overlapping concepts that are closely related, including requirements capture [140], gathering [36], discovery [119], acquisition [88], as well as inventing [87], and creating requirements [87].

In this dissertation we refer to the general process of learning and understanding requirements as requirements capture of contextual requirements. We refine this process and define requirements elicitation as a design time activity and requirements discovery as a runtime activity. At runtime, many requirements are already implemented and the requirements analyst is faced with the problem of discovering requirements that are currently missing. In contrast, at design time, the requirements analyst has to start from scratch and therefore we refer to this more established activity as requirements elicitation.

According to Nuseibeh and Easterbrook, the established requirements elicitation techniques can be classified as follows [95]:

1. *Traditional techniques* include questionnaires, interviews, and analysis of existing documentation.

2. *Group elicitation techniques* allow for the exploitation of team dynamics for requirements elicitation and include focus groups, JAD, and brainstorming.

3. *Prototyping* allows dealing with high levels of uncertainty and can provoke customer feedback and includes using prototypes to invoke discussions in group elicitation techniques.

4. *Model-driven techniques* focus on using models to drive the elicitation process and include goal-based methods.

5. *Cognitive techniques* allow for knowledge acquisition and include protocol analysis, laddering, card sorting, and repository grids.

6. *Contextual techniques* are used in situations where the local context is vital for understanding social and organizational behavior, and include ethnographic techniques such as participant observation.

Before starting with the elicitation of requirements, the requirements analyst has to determine suitable requirements elicitation techniques. Dieste and Juristo assess effective requirements elicitation techniques [47]. In a systematic literature review of empirical studies on elicitation techniques, they found that although many experiments to study different aspects of requirements elicitation techniques exist, studies run in real environments are missing. Based on the review, the most studied elicitation technique is interviewing, which also seems to be one of the most effective requirements elicitation techniques as presented by Dieste and Juristo in an earlier publication of this study [1].

Requirements at design time might be known or unknown [127]. In the case they are known, it is possible to get a detailed understanding of requirements. In cases where requirements are unknown, researchers differentiate between the known unknown and the unknown unknown. In the first case, we know that certain information is unknown and can implement techniques at runtime to get a detailed understanding. In the second case, we do not even know which information is missing. Capturing unknown unknowns is challenging.

Recent trends in supporting the capture of requirements at runtime involve techniques that automatically extract information from online data based on which new requirements can be derived. Guzman et al. [64] analyze app reviews to identify new requirements. The app reviews are filtered, aggregated, and analyzed using sentiment analysis. Rahimi et al. [104] present a data mining approach to automatically extract quality concerns from requirements, feature requests, and online forums. Such approaches could be a valuable first step, when capturing contextual requirements.

Capturing Context

Understanding the context is an important activity for capturing requirements [98]. For example, contextual inquiry is an integral part of customer focused research [20]. In fact, with advances in ubiquitous computing, context plays an increasingly important role in understanding stakeholder's needs [114]. Different approaches propose capturing context to better understand user needs.

Context plays an increased role in ubiquitous and embedded systems. Maiden and Seyff develop techniques that can be used to detect context in which new system behavior is needed [89, 119]. They validate scenarios in the original context of use. Seyff et al. present a tool to submit user needs at runtime [118]. Schneider et al. give users the opportunity to articulate their feedback in context at runtime to better understand the runtime context [116]. Daun et al. propose to document assumptions about the operational context for long-living collaborative embedded systems to allow the systems to cope with specific changes in their operational context [43].

With respect to the goal of this dissertation, these contextual approaches can be considered as an important first step, as they recognize the importance of context. Mostly, requirement engineering approaches focus on the capture of requirements, while understanding the context, or on the capture of context to understand real life situations for which end-users require new features. While modeling approaches exist to model and analyze requirements in context, approaches to systematically capture end-user requirements together with the context they are valid in are underrepresented. Before we investigate the development of new techniques for the purpose of capturing contextual requirements, **we investigate the usefulness of existing requirements elicitation techniques for the capture (i.e., elicitation) of contextual requirements in research question 2.**

2.2 Requirements Engineering for Adaptive Systems

Self-adaptive systems (also referred to as dynamically adaptive systems [135]) are systems with the ability to adjust their behavior in response to their perception of the environment and the system itself [34]. This operating environment includes end-user input, external hardware devices, and sensors [110]. To determine how to adjust behavior in a specific state, a self-adaptive system uses models to decide which action needs to be performed in order to reach an end-user goal [99, 9].

Designing self-adaptive systems demands the prediction of potential changes at runtime, and implementation of abilities to facilitate reflection of the system depends on a solid understanding of the runtime environment at design time [113]. Qureshi et al. confirm the importance of distinguishing between requirements engineering activities at *design time* and *runtime*, when engineering self-adaptive systems [102]. In traditional requirements engineering, the analyst is typically responsible for specifying requirements at design time. Nowadays, there is a move to collect runtime system data which can be used by the requirements engineer to better understand the environment [68]. However, self-adaptive systems are aware of their requirements and can execute basic requirements engineering activities themselves to cope with changing conditions that appear at runtime [100].

The PhD thesis of Nauman Qureshi [100] presents an important work for this dissertation. Qureshi presents a conceptual framework to perform requirements engineering for self-adaptive systems, in which he describes the importance of context for requirements in self-adaptive systems. Furthermore, Qureshi et al. present adaptive requirements [99] – "requirements that encompass the notion of variability associated to either a functionality or a system quality constraint". In contrast to Qureshi et al.'s focus on variability, we focus in this dissertation on one specific system behavior, which is needed in a specific context.

Cheng et al. argue that the central task of requirements engineering for adaptive systems is capturing the relevant context-attributes for the adaptability of the system at design time [34]. Qureshi et al. describe the adaptation problem at runtime [100, 102]. Their approach for identifying new context-attributes is to ask the end-users for missing context information. Including the users in the adaptation process has an advantage in that they are already in the right context and are able to provide relevant data. However, experience shows that stakeholders find it difficult to articulate the influences of context and their implications [95]. Sitou and Spanfelner propose a model-based approach to requirements engineering for (context-)adaptive systems [123]. This approach integrates a model of the usage context and distinguishes three dimensions: Changing participants, activities, and operational environment. Their methodology consists of two parts: Stability check and identification check. In contrast to this dissertation, they only consider a fixed set of context-attributes of the environment that they monitor for adaptation.

Based on these context-attributes, analysts have to assess which requirements might change during runtime. In the scope of requirements elicitation for adaptive systems, the focus is not primarily on understanding complex tasks but to understand the influence of context. However, a deep understanding of the required functionality and its connection to the context is required to address the uncertainty inherent in adaptive systems, and this dissertation proposes a framework that establishes this connection and allows continuous learning and adaptation of the knowledge about context influences at runtime. Hassan et al. [65] present a method that can be used to explore the design decisions for self-adaptive systems. They provide designers of self-adaptive systems with a basis for multi-dimensional what-if analysis to revise and improve the understanding of the environment.

Current research on adaptive systems concentrates mainly on modeling, architecture, and monitoring, see for example [127, 8, 137, 111]. However, research previews emphasize the importance of further investigations in requirements engineering for adaptive systems (e.g., [34, 110, 137]). Hong et al. identify important characteristics of context-adaptive systems based on a literature review, but also report a lack of requirements engineering in this field [66]. Especially linking requirements with context and defining which context can change at runtime is an important activity at design time that needs further investigation [34, 37]. A recent systematic literature review published in 2015 on requirements engineering for self-adaptive systems outlines the 101 publications on this topic [124]. It shows that the highest number of research in the area of requirements engineering for self-adaptive systems was published in 2012. Most publications focus on the activities requirements specification (34 out of 101), requirements modeling (15 out of 101), requirements monitoring (12 out of 101), and requirements verification (12 out of 101). Only three publications exist on requirements elicitation, three publications on system evolution, and only one on requirements evolution. This emphasizes the lack of research in the area of capture and evolution of requirements for self-adaptive systems.

2.3 Uncertainty in Adaptive Systems

Designing self-adaptive systems presents researchers with different challenges. One challenge is handling uncertainty that affects adaptive systems. Uncertainty in (dynamically) adaptive systems is defined as:

Definition 2.3.1. Uncertainty
Uncertainty is "a system state of incomplete or inconsistent knowledge such that it is not possible for a dynamically adaptive system to know which of two or more alternative environmental or system configurations hold at a specific point. This uncertainty can occur due to missing or ambiguous requirements, false assumptions, unpredictable entities or phenomena in the execution environment, and unresolvable conditions caused by incomplete and inconsistent information obtained by potentially imprecise, inaccurate, and unreliable sensors in its monitoring infrastructure." – Ramirez at al. [106]

One of the main reasons for uncertainty is that adaptive systems are often operated in uncertain environments to which they must adapt. Examples of such systems interacting in an uncertain environment include: 1) intelligent vehicle systems [106] that have to deal with differing and unforeseen traffic and weather conditions or obstacles that they have to detect and avoid, and 2) software systems in smart cities that are interacting with thousands of individuals in highly dynamic environments [42, 24].

In order to deal with uncertainty affecting requirements, Whittle et al. propose the RELAX language [138]. The language allows relaxed requirements that can be adjusted at runtime by giving the system flexibility to handle its requirements. By relaxing requirements, the system can determine the best solution to fulfill the requirement at runtime. Further approaches to solving the problem of missing runtime information at design time are to include users in decision-making [13] or in the software life cycle [84].

Even when using the RELAX language and including the users in decision-making, it is still challenging to detect certain context conditions in an uncertain operational environment in which certain system behavior is required. The definition of when an adaptive system is supposed to execute a certain action requires a full understanding of relevant context, which might not be possible at design time. In the last year several approaches on handling uncertainty have been presented: Horkoff et al. [67, 109] focus on methods to support early requirements decision-making. They model uncertainty in goal models to support the iterative reduction of uncertainty [67] and provide support for uncertainty capture, elaboration, and change [109]. Tran and Massacci [129] provide a means to capture the uncertainty of evolution in feature models. This dissertation presents a framework that differentiates between system behavior and the context in which it is valid, as well as between design and runtime activities to deal with uncertainty concerning contextual requirements at different levels of time in the development process as well as handling uncertainty of system behavior and context independently.

2.4 Requirements Evolution in Adaptive Systems

Antón [16] describes goal evolution when discussing the refinement of a requirements model from high-level objectives to lower level, technical requirements including the operationalization of goals with enough details. The goal of this process is to refine subgoals into an operational definition [17], which is an important task for adaptive systems. However, it is not always possible to refine subgoals at design time due to incomplete context information that are caused due to uncertainty [106]. One solution to deal with this situation is to support requirements evolution at runtime when more context information becomes available [34]. Specifically, even if context conditions can be captured at design time, assumptions might become invalid at runtime [10] or uncertainty might affect the system's ability to satisfy requirements otherwise.

Changing requirements and uncertainty concerning the operational environment require adaptive systems to evolve at runtime [92].

Fickas et al. and Oriol et al. propose to use requirements monitoring, the results of which can be of benefit for designers and maintainers to provide the required information to redesign the system [55, 96]. Additionally, to be able to execute requirements engineering activities at runtime, an adaptive system has to store a model containing the necessary information that can change at runtime (e.g., context and requirements information) [56]. This model is used to evolve the knowledge regarding changing end-user needs and context information.

Souza et al. [121] introduce *evolution requirements* that define the runtime evolution of existing requirements. Evolution requirements define how requirements can change at runtime and under which conditions. When identifying such conditions at runtime, the system can trigger the (predefined) evolution on its own. Nevertheless, evolution requirements are not easily applicable in uncertain environments that cannot be completely predicted at design time. In fact, uncertainty is the main reason that we cannot fully specify our knowledge of the requirements and environment of a system at design time [106]. Due to uncertainty, self-adaptive systems need to evolve and even consider unforeseeable changes at runtime [92]. Indeed, the ability of self-adaptive systems to adjust their behavior based on what they sense in the environment and the system itself [34] is linked to software evolution [110].

Inverardi et al. [69] present a framework on the evolution of contextual requirements. They propose integrating the user feedback as one source for evolution, focusing on the evolution of the system functionality and not on the evolution of the context representation. **In research question 3 we investigate support for the evolution of the context (operationalization) in which contextual requirements are valid.** Instead of relying (only) on user input, we present an approach that is based on machine learning and feedback loops to detect contextual requirements affected by uncertainty and supporting the knowledge of the context in which contextual requirements are valid.

2.5 Context in Requirements Engineering and Adaptive Systems

As software intensive systems pervade more aspects of life, the notion of context becomes increasingly important in requirements engineering. User needs have to be supported actively at runtime and context plays an increased role in supporting user needs efficiently [22].

Villegas et al. conducted a literature review on context-awareness, with a focus on the characterization of context information [132, 133]. Based on this literature review they propose a context definition that we reuse for the purpose of this dissertation:

Definition 2.5.1. Context
"Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject which can affect the behavior of the system and/or its interaction with the user. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and maintained to support its dynamic evolution." – Villegas et al. [133].

Furthermore, Villegas et al. [134] present separation of concerns between requirements, context, and adaptation. Each of the three levels needs to be controlled in self-adaptive systems, for which they use feedback loops. The work by Villegas et al. is important work that influenced this dissertation, especially the concepts of the framework on the capture and evolution of contextual requirements (cf. Chapter 4).

One important activity during requirement elicitation for adaptive systems is the identification of context-attributes [123]. Related work presents different context taxonomies. Context taxonomies that mainly present common context-attributes are described for example in [66, 61, 139, 31]. Still, most taxonomies are designed for a specific system. Different types of systems need to consider different context-attributes and therefore the existing taxonomies of context-attributes cannot always be transferred to other domains or systems. Once identified, the importance of context-attributes can change at runtime. Therefore, systematic approaches for capturing context and supporting its evolution can help in this situation.

Some of the requirements for context-aware systems (i.e., systems that implement the concept of context-awareness) are only valid at specific locations [35, 2], or in other specific contexts [54]. The concept of context is not limited to the spatial position. Schmidt et al. [115] propose a hierarchical model of context-attributes. Several researchers propose to use an ontology to document context information, for example Villegas et al. [132] and Qureshi et al. [100]. This work is an excellent starting point when searching for relevant context-attributes in a given situation. We focus on the capture and evolution of context information related to one particular system behavior that is captured in a contextual requirement.

Chen at al. argue that the implementation of context-awareness requires the capture of context on different levels: Low level context (e.g., location and time) and higher-level context (e.g., user's current activity) that can be expressed through low-level context [32]. Through the combination of several low level contexts, complex context can be recognized [32]. Mongiello et al. [91] present a runtime verification method for context-aware applications. They use cognitive psychology concepts for the adaptation of adaptive systems to changed context on the source code level.

Villegas [132] introduces situation-aware software systems, in which context information is collected and the system is able to recognize different situations from lower-level context at runtime. For this purpose, Villegas presents a context ontology and reasoning engine and demonstrates the concepts based on a smarter internet shopping case study. This demonstration is a good application example of system adaptation to context, in which the end-user is involved in the activities. Afanasov et al. [4] provide design concepts and language support for the design and implementation of cyberphysical systems. Their concepts include situation-awareness. All these approaches view the developed techniques from the system's (technical) adaptation point of view, while we focus on support for requirements engineering activities.

2.6 Contextual Requirements

First ideas on adapting requirements based on changes in the environment were presented around the year 2000 (e.g., by Eracar et al. [52]). Over the last decade there has been a growing interest in developing approaches to study contextual requirements engineering and contextual design. The first framework on contextual requirements engineering was presented by Sutcliffe et al. [126] considering the effect of context on personal goals. Müller and Villegas [133] propose a reference model that uses a separation of concerns between the system objectives, monitoring of the context representation, and adaptation to context using feedback loops. In a next step, Castañeda, Villegas and Müller [30] enrich the objectives by user's personal goals for web-tasking systems. Inverardi and Mori [70] present a framework that is centered around features. Their framework considers three elements – a requirement in form of a feature, a context entity, and the service implementing the feature. Afanasov et al. [4] present contextual design for cyberphysical systems. They focus on the definition of different context situations, and their monitoring. Acher et al. [3] model context together with software variants in feature models. Ali et al. [7] introduce contextual requirement models to specify requirements and their context at runtime. They describe contextual requirements as the interplay of two elements – requirements and context – and focus on the modeling of the variability of both context and requirements, and the detection of errors in contextual requirements models [9]. Ali et al. refer to context as activation context, required context, and context which leads to different quality levels. In this dissertation we concentrate on the activation context, which describes when exactly requirements are valid.

In building on this related work where the execution of requirements depends on the context, we define contextual requirements:

Definition 2.6.1. Contextual requirement
A contextual requirement consists of a 2-tuple of the expected system behavior and the specific context within which this expected behavior is valid.

Similarly to the approaches by Inverardi and Mori [70] and Ali et al. [7], this dissertation focuses on requirements that depend on context. The approach by Ali et al. concentrates on modeling and reasoning about the adaptation of systems to variable context. Contextual goal models are a powerful instrument to model socio-technical systems [41] and are recently being presented as an instrument for the analysis of security requirements [83]. However, the presented approach on contextual goal models does not deal with requirements uncertainty at runtime. Inverardi and Mori [70] present a research preview on the evolution of context variations to deal with uncertainty by using model checking to execute either predefined evolution or by letting users specify the evolution needs.

The approaches described above motivate the need for contextual requirements.

Requirements capture and evolution are largely unexplored research areas that motivate us to investigate **research question 1: What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?** This dissertation presents a framework that can be used to develop techniques to support the capture and evolution of contextual requirements.

2.7 Chapter Summary

This chapter reviewed the challenges in capturing contextual requirements. Traditional requirements elicitation techniques only focus on capturing requirements, while understanding the context as far as is necessary. Recent advances in requirements engineering research have presented approaches to document relevant context at runtime, but do not offer assistance when actively integrating context capture into (contextual) requirements capture. In addition to requirements elicitation at design time, requirements engineering for adaptive systems has to consider runtime requirements capture and evolution due to uncertainty. Literature on contextual requirements mostly concentrates on modeling, architecture, and monitoring of requirements and relevant context. The research presented in this dissertation focuses on the topic of capture and evolution of contextual requirements at design and runtime to deal with runtime uncertainty.

Chapter 3

Research Design

The research conducted in this dissertation is of an exploratory nature due to the unexplored area of the capture and evolution of contextual requirements for adaptive systems.

Figure 3.1 gives a detailed overview of the research design used in this dissertation. The structure was divided into two parts. In *phase 1* we explored the research area

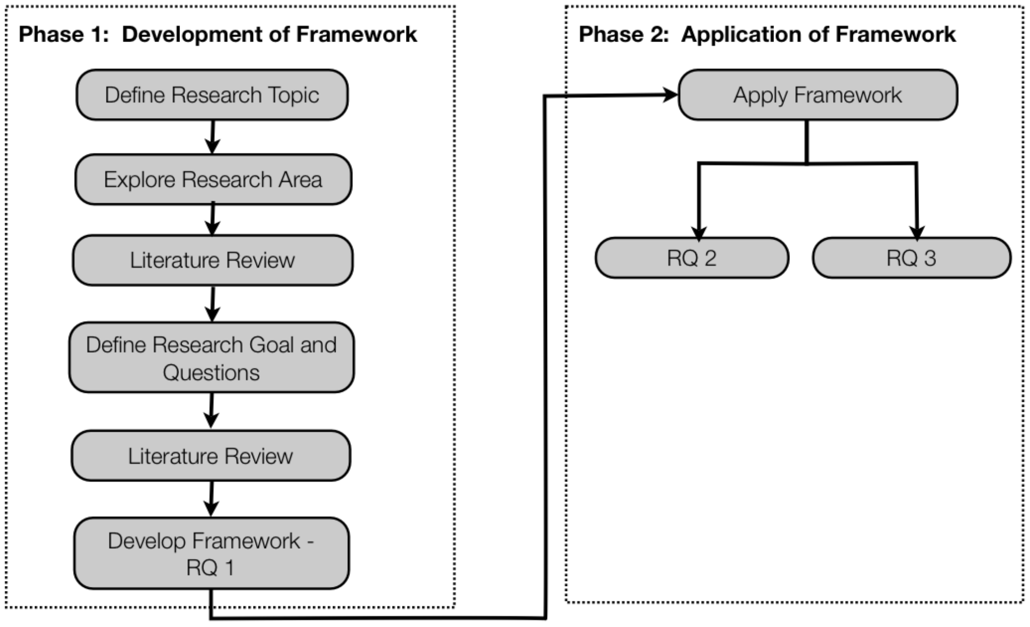


Figure 3.1: Research pathway

and developed a framework for the capture and evolution of contextual requirements (presented in Chapter 4). In *phase 2* we applied the framework and investigated techniques for elicitation (presented in Chapter 5) and evolution of contextual requirements (presented in Chapter 6).

3.1 Phase 1: Exploration of Research Area and Development of Framework

The goal of phase 1 was the development of a framework for capture and evolution of contextual requirements to deal with uncertainty. In the first step we defined the **research topic** of the capture and evolution of requirements considering context for adaptive systems, with a focus on capturing and communicating context together with requirements for system adaptation. To investigate this research topic we started to **explore the research area**. We complemented this exploration continuously through reviews of existing literature. An extensive **literature review** was conducted at the beginning of phase 1. During that time the area of requirements engineering for adaptive systems was quite unexplored and only just started gaining importance due to the trend in increasing ecosystems and decentralized systems [93]. The literature review included the areas of system adaptation, requirements engineering techniques with focus on requirements elicitation, and end-user involvement in requirements elicitation activities at runtime for system adaptation. Additionally, the literature review included contextual techniques in requirements engineering, in-situ techniques, and capturing context.

Due to the fact that this research area so far had little research activity, the methodology taken in this dissertation is of an exploratory nature. Three **research questions** were derived with respect to the **research goal** "investigate how to capture contextual requirements and manage their evolution to address uncertainty during design and operation of adaptive systems." The rest of this chapter details the three research questions and the research methodology that we used to investigate them.

Usually context helps clarifying requirements, e.g. by understanding context of use and human factors we are able to derive usability requirements more concretely [85]. For requirements that are contextual, analysts will have to capture and document the system behavior and the context it is valid in. To continue the satisfaction of contextual requirements, runtime support is needed to evolve contextual requirements where appropriate. Research question 1 investigates capturing and evolution of contextual requirements in general:

RQ 1: *What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?*

To capture all relevant information about contextual requirements and demonstrate how analysts can apply existing techniques, a **framework is developed**.

Literature differentiates between requirements engineering activities at design and runtime [103]. Motivated by their work, this dissertation differentiates between the activities of capturing requirements at design and runtime. The framework defines the design time activity as elicitation (the traditional requirements elicitation), and the runtime requirements capturing activity as discovery. Changes to existing requirements at runtime is defined as the evolution of contextual requirements.

This dissertation takes the view of contextual requirements as consisting of two parts – system behavior and context in which it is valid. Underlying this view is the notion that each of the requirement and its relevant context can be discovered and evolved separately at runtime. By evolving the context the adaptive system keeps satisfying its requirements continuously in the right context, and end-user needs are fulfilled.

3.2 Phase 2: Application of the Framework

To evaluate aspects of the framework and further an understanding of the application of existing techniques this dissertation **applies the framework** in two further studies. The first study investigates techniques for the elicitation of contextual requirements at design time (RQ 2) and the second study their evolution at runtime (RQ 3).

RQ 2: *How can existing requirements elicitation techniques help elicit contextual requirements at design time?*

To support adaptive systems, contextual requirements must be captured at design time. Requirements that depend on context have to be identified, as well as their valid context defined as precisely as possible to remove requirements uncertainty. Yet, while the usefulness of contextual requirements has been discussed, there is little research on how to analyze, or more specifically to use, existing requirements engineering practices for their elicitation at *design time*.

Case study research is chosen as the research method of choice to investigate this exploratory research question [108]. We had the opportunity to work with the Human Resources Department at the University of Victoria in revising its job applicant tracking system. To investigate the goal of *exploring the use of existing requirements elicitation techniques in identifying contextual requirements at design time* we acted as the requirements analyst for the project. The applicant tracking system was to be replaced to better serve the needs of thousands of stakeholders. We applied different requirements elicitation techniques (e.g., interviews, prototyping, scenarios, goal-based approaches, and focus groups) to this real world socio-technical system.

We were able to document a number of contextual requirements when applying particular requirements elicitation techniques in a particular order: First we identified requirements through interviews and focus groups, attempted to understand the rationale behind them through interviews, then used prototyping to get a detailed understanding of these requirements in context. Next we identified conflicts between different end-users when discussing requirements in focus groups in detail. These discussions helped identify the need for contextual requirements. Finally, we identified through interviews with the respective end-users the different context related to requirements so that we could document the contextual requirements.

In our case the need for system runtime adaptation (by identifying contextual requirements) became apparent based on end-user requirements and their complex operating environments. The context part offered a starting point to analyze triggers for adaptation, while the requirement part defined the adaptation goal. Designing such a system as an adaptive system would allow it to fulfill otherwise conflicting requirements of the different stakeholders by adapting to their context-specific needs.

Due to uncertainty we cannot fully specify our knowledge of requirements and the environment of a system during design time [106]. Therefore, contextual requirements that are elicited and implemented at design time can be affected by uncertainty at runtime. Uncertainty leads to the fact that contextual requirements will not be satisfied without runtime support for their evolution [92], enabling the system to react appropriately [54]. Therefore, the next research question investigates the evolution of contextual requirements that are affected by runtime uncertainty (i.e., unpredictable environment and sensor loss can be supported):

RQ 3: *How can the evolution of contextual requirements that are affected by uncertainty be supported at runtime?*

Supporting the evolution of existing contextual requirements requires runtime support that analyzes usage data as well as contextual data, and can be run automatically and use available data at runtime. In this research question we *explore the use of existing techniques such as data mining algorithms and feedback loops to support the evolution of contextual requirements in the face of uncertainty (i.e., unpredictable environment and sensor failure).* We concentrate on the evolution of the context part of contextual requirements, as there already exists work on the evolution of requirements, and the contribution of context evolution research would be more valuable for the research field.

Data mining provides techniques to detect patterns in large data sets to deal with their complexity. The application of data mining on contextual data is a promising strategy to use when detecting certain conditions that change over time [5]. When used by an adaptive system to support the evolution of contextual requirements, data mining shows promise in helping the system maintain updated knowledge about contextual requirements in the face of runtime uncertainty and herewith supports the evolution of contextual requirements. Feedback loops promise automatic support at runtime and can help in implementing automatic support for evolution.

Considering the integration of data mining and feedback loops, we present an approach that updates the knowledge about contextual requirements with up-to-date information about the context in which contextual requirements are valid at runtime. A feedback loop is used to detect indications that the satisfaction of contextual requirements is affected by runtime uncertainty, and data mining algorithms are integrated to determine the context in which contextual requirements are valid. With the help of data mining used on sensor data, the approach is able to update the software system's knowledge about contextual requirements where affected by uncertainty.

For the purpose of evaluation we developed an activity scheduling system we called ToTEM to be used in a wild and unpredictable environment, the ocean. ToTEM is an activity scheduling system that supports extremely demanding and potentially life-threatening situations in this wild environment. ToTEM provided alarms for the daily activities of a crew of four rowers crossing the Atlantic Ocean during a 72-day trip [94]. In the case of ToTEM we collected the rowers' system adaptation needs in context, elicited five contextual requirements, analyzed patterns in the contextual data from 46 sensors, and mined this data to make the context in which contextual requirements were valid and measurable. Further, we analyzed the cases in which the evolution of contextual requirements would be triggered due to uncertainty.

Chapter 4

A Framework for the Capture and Evolution of Contextual Requirements

When developing adaptive systems, requirements engineering becomes more difficult compared to the development of traditional systems. In addition to converting end-user needs into requirements, the requirements analysts and designers of the system have to determine when and how the system should adapt at runtime. At runtime, end-user needs and operational environments can change. Adaptive systems have to support the systematic evolution of requirements to maximize user satisfaction during the lifetime of a system.

We motivated the need for treating some requirements as contextual in Chapter 1. Capturing such contextual requirements and supporting their evolution is challenging as end-user needs have to be analyzed in context. Additionally, the context has to be documented together with the corresponding requirement in system models. Adaptive systems have to implement techniques to track the need for both evolution and execution of the evolution of contextual requirements to deal with uncertainty.

This chapter investigates the following research question:

RQ 1: *What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?*

We present a framework with essential elements of capture and evolution of contextual requirements for adaptive systems at design and runtime.

At its core are contextual requirements which are decomposed into two parts: the expected system behavior and the context in which this behavior is valid. Taking the view that system behavior and context can be regarded as two entities of contextual requirements allows us to examine each part separately. Because the framework is meant to be applicable to different domains and different kind of systems, we keep the concepts of the framework on a higher level of abstraction. The implications of this decision include the fact that parts of the concepts, discussed in the framework, are only formalized in Chapter 6. Furthermore, we do not distinguish between context description and measurable context when presenting the concepts of the capture and evolution of contextual requirements in the framework.

The framework differentiates between capturing contextual requirements at design and runtime. It also considers the evolution of existing contextual requirements. Motivated by the literature on differentiating between design and runtime activities in requirements engineering [103], the activity of capturing requirements is defined as *requirements elicitation* (at design time), and *requirements discovery* (of new contextual requirements at runtime). *Requirements evolution* is the activity of evolving existing contextual requirements at runtime. Furthermore, we treat contextual requirements as composed of the system behavior and the context in which the system behavior is valid. This separation of concern is motivated by the work presented by Villegas et al. [134]. They separate the system objectives, the context information, and the adaptation. Our framework considers transitions from partial knowledge about contextual requirements to their correct and complete specification for the three activities of requirements elicitation, discovery, and evolution (thus differentiating between design and runtime activities).

The framework is intended to help requirements analysts, designers, and operators of adaptive systems in exploring techniques that support the capture and evolution of contextual requirements through the following underlying concepts:

- *Distinguishing between design time and runtime* activities in the capture of contextual requirements allows adding new contextual requirements at runtime to deal for example with incomplete design time knowledge or changing end-user needs or operational environment at runtime.

- *Distinguishing functionality from context* in contextual requirements enables activities that lead to partial knowledge of any part of the contextual requirement to be completed later through analysis.

- *Distinguishing between partial and complete knowledge* of contextual requirements by leveraging the concept of known/unknown requirements presented by Sutcliffe et al. [127]. It allows capturing contextual requirements step by step by using partial knowledge about contextual requirements.

4.1 Contextual Requirements: Definition and Example

This section defines the concept of contextual requirements and illustrates it by two contextual requirements from the parking lot example.

4.1.1 Definition

Drawing on existing work as elaborated on in Chapter 2, we refine our definition of *contextual requirements*.

Definition 4.1.1. Contextual requirement
A contextual requirement cr is a requirement (an expected system behavior) that is only valid in a specific context, represented as a 2-tuple $cr = (b, c)$ consisting of an *expected behavior* b and the *context* c in which it is valid.

In order to have the flexibility of separately adding or changing the content of contextual requirements, the framework presented in this chapter supports the view of contextual requirements as composed of *the system behavior* and *context* as two entities that can be captured and can evolve separately. This allows us to analyze, at runtime, the evolution of context or behavior and identify possible *evolution* of previously identified contextual requirements. Further, it allows the elicitation and discovery of new contextual requirements step by step by using partial knowledge about contextual requirements.

Contextual requirements can be used by adaptive systems in order to provide the appropriate system behavior based on the current context. For this purpose an adaptive system monitors the execution context at runtime in order to decide which behavior needs to be provided in the currently sensed context. Therefore, in order to be quantifiable by an adaptive system, a certain context of validity c has to be measurable by the system. In contrast to the perceived context, the measurable context condition cannot always be provided by end-users during requirements elicitation. Therefore, this dissertation differentiates between a context description (i.e., the perceived context in which a user requires a certain system behavior) and a measurable context.

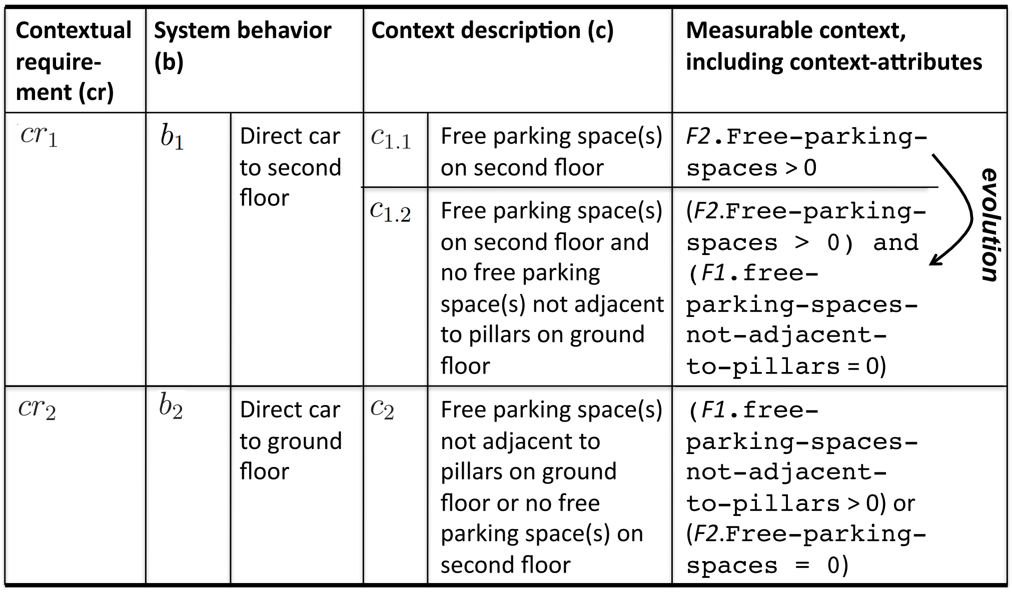
Definition 4.1.2. Measurable context
A measurable context is a certain condition that can be monitored and quantified by the system with the help of context-attributes.

4.1.2 Example of Contextual Requirements

We illustrate the introduced concepts based on the parking lot example presented in the introduction.

Figure 4.1 outlines two contextual requirements in detail. Each contextual requirement, cr_1 and cr_2, consists of the two parts – the system behavior and context. The context is separated into a context description, and the corresponding measurable context that includes context-attributes. In the following, we discuss the example and show one contextual requirement (cr_1) elicited at design time, one contextual requirement (cr_2) that was discovered at runtime, and the evolution of one contextual requirement (cr_1 with evolution of the context).

Elicitation of contextual requirement 1 (cr_1): If there are (is) (a) free parking space(s) on the second floor ($c_{1.1}$), the system should direct the car to the second floor (b_1).
Direct cars to the lower level floor (also referred to as ground floor - $F1$) was so far the standard system behavior (b_1). The context description $c_{1.1}$ adds a restriction to the system behavior. This represents a contextual requirement (cr_1): If there is at least one free parking space on the higher level floor (also referred to as second floor, $F2$, in the following) then the system gives directions to the second floor. In our



Contextual require-ment (cr)	System behavior (b)		Context description (c)		Measurable context, including context-attributes
cr_1	b_1	Direct car to second floor	$c_{1.1}$	Free parking space(s) on second floor	`F2.Free-parking-spaces` > 0
			$c_{1.2}$	Free parking space(s) on second floor and no free parking space(s) not adjacent to pillars on ground floor	(`F2.Free-parking-spaces > 0`) `and` (`F1.free-parking-spaces-not-adjacent-to-pillars` = 0)
cr_2	b_2	Direct car to ground floor	c_2	Free parking space(s) not adjacent to pillars on ground floor or no free parking space(s) on second floor	(`F1.free-parking-spaces-not-adjacent-to-pillars` > 0) or (`F2.Free-parking-spaces = 0`)

Figure 4.1: Examples of contextual requirements

example the first contextual requirement cr_1 consisting of b_1 and $c_{1.1}$ was *elicited at design time.*

Discovery of contextual requirement 2 (cr_2): If there are (is) (a) free parking space(s) not adjacent to pillars on the ground floor or no free parking space(s) on the second floor, the system should direct the car to the ground floor.
Our second example illustrates a contextual requirement *discovered at runtime.* By observing the user behavior together with the context, we were able to identify indications for the need of a second contextual requirement (cr_2). At design time the only emphasis was on providing a free parking space, discarding its location. When observing users we identified that users preferred to choose free parking spaces not adjacent to pillars. A new contextual requirement had to be introduced considering the context-attribute: `free-parking-spaces-not-adjacent-to-pillars`. If there are no free parking spaces not adjacent to pillars on ground floor, users go to the second floor to check whether there are parking spaces not adjacent to pillars. Only if there are no free parking spaces on the second floor, the system should direct cars to the ground floor. Hence, the second context condition for cr_2 is: no free parking spaces on the second floor.

Evolution of contextual requirement 1 (cr_1 - with $c_{1.2}$): If there are (is) (a) free parking space(s) on the second floor and no free parking space(s) not adjacent to pillars on the ground floor ($c_{1.2}$), the system should direct the car to the the second floor (b_1).

We identified the need for *evolution* of the first contextual requirement cr_1 *at runtime.* By identifying the need for the context-attribute `free-parking-spaces-not-adjacent-to-pillars`, the context part ($c_{1.1}$) of cr_1 had to evolve. $c_{1.2}$ adds another condition of "no free parking space(s) not adjacent to pillars on ground floor" to the context $c_{1.1}$ of free parking space(s) on the second floor. Users prefer choosing a parking lot that is not adjacent to pillars and prefer the ground floor. If no such parking spaces are available they choose other parking spaces. In the example it was necessary to evolve the contextual requirement concerning the context to satisfy end-user needs, whereas the expected system behavior of directing cars to the second floor stayed the same.

4.2 From Partial to Complete Knowledge of Contextual Requirements

The example of cr_2 shows that in some cases the context is known at design time, but the required system behavior is discovered at runtime (cr_2). The same can happen if the system behavior is known at design time, but its relevant context is not known at design time and discovered at runtime. To have the flexibility in capturing only parts of contextual requirements, we differentiate between partial and complete knowledge of contextual requirements. Partial knowledge means that only the context or the system behavior are known, but the corresponding system behavior or context respectively unknown.

To model the partial knowledge and the transitions from partial to complete knowledge, the framework uses the model of known/unknown requirements presented by Sutcliffe et al. [127]. This representation of known/unknown requirements is not only applied to the expected behavior, but also to the context in which this is valid. *Known* or *unknown* requirement refers to whether the expected behavior in the contextual requirement is known or unknown. Similarly, *unknown context* describes the situation where the influence of context on the validity of the system behavior is not

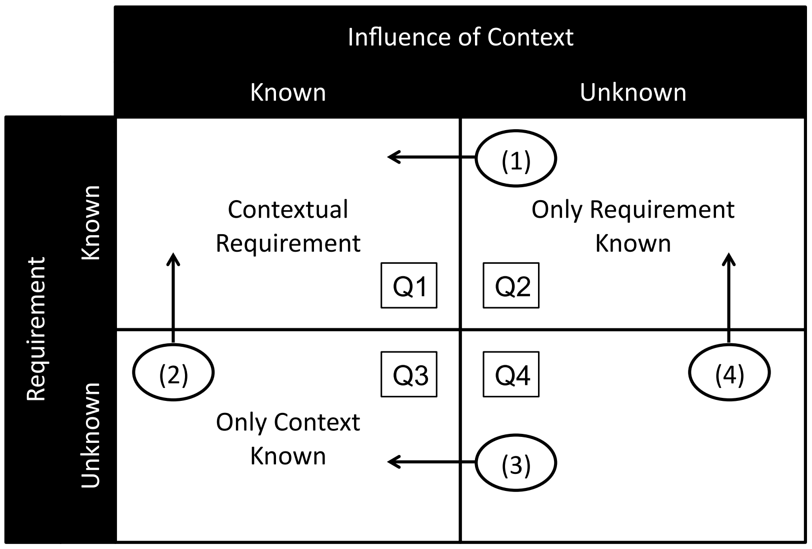


Figure 4.2: Four quadrants representing the combinations of known/unknown influence of context, and known/unknown system behavior (requirement)

known. Figure 4.2 illustrates the concept of known or unknown requirements and known or unknown context through four quadrants (Q1 - Q4). In the following we give more details on each quadrant:

Q1 - Known requirement and known context: Contextual requirement - The expected behavior as well as the influence of context on the functionality is known and defined. The behavior is only valid in the context that has been identified.

Q2 - Known requirement and unknown context: Only the expected behavior is known. This quadrant represents traditional requirements, for which the influence of context on the execution of the system behavior is not known.

Q3 - Unknown requirement and known context: Only the context in which a system behavior might be needed is known. The requirement has to be determined so that the system can satisfy end-user needs in this context.

Q4 - Unknown requirement and unknown context: We neither know the required system behavior nor the context in which this behavior is needed.

The four quadrants represent our knowledge of contextual requirements. In Q4 we do not have any knowledge. In Q2 the requirement is known, whereas in Q3 only the context is known. Both quadrants represent partial knowledge of contextual requirements. Q1 is representative of the complete knowledge of contextual requirements. This concept of partial to complete knowledge of contextual requirements allows us to discuss the given information, as well as the information that is missing.

4.3 The Capture and Evolution of Contextual Requirements at Design vs. Runtime

Capturing contextual requirements (represented in quadrant Q1 in Figure 4.2) requires that context and system behavior are known. To identify contextual requirements (reach quadrant Q1) there are certain steps that have to be followed in order to identify the system behavior and the context in which this behavior is valid. We outline two possibilities on the capture of contextual requirements, starting from anything known and reaching the complete knowledge of contextual requirements (i.e., system behavior and context known):

Path 1: Capture the context (i.e., transition (3) in Figure 4.2, from quadrant Q4 to Q3) and then the requirement which should be valid in this context (i.e., transition (2), from Q3 to Q1).

Path 2: Capture the requirement (transition (4), from Q4 to Q2) and then the context in which this requirement is valid (transition (1), from Q2 to Q1).

In the following we outline the four possible transitions presented in Figure 4.2 that an analyst could use to capture contextual requirements.

Transition (1) - From known requirement (influence of context unknown) to a contextual requirement (known requirement and known context): The task of this transition is to find the context in which a requirement is valid. Not all requirements will necessarily resolve in contextual requirements. For contextual requirements the context of validity has to be identified.

Transition (2) - From known context (and unknown requirement) to a contextual requirement (known context and known requirement): In this case the context in which a requirement is needed is known. In this transition the exact requirement has to be identified.

Transition (3) - From anything known to known context (and unknown requirement): This is the precondition of transition (2) in which the context for the validity of a system behavior is identified.

Transition (4) - From anything known to known requirement (and unknown context): This transition is representative of traditional requirements elicitation activities in which a requirement is captured. Additionally, it represents the precondition for transition (1).

All four transitions can take place at design or runtime. In our framework we distinguish between elicitation of contextual requirements (at design time) and discovery of contextual requirements (at runtime). We consider a contextual requirement as discovered even if only parts of the contextual requirement are captured at runtime and the rest at design time.

The evolution of contextual requirements takes place at runtime. There are two options for the evolution of contextual requirements at runtime: evolution of the knowledge of the system behavior or the context. Therefore, the evolution of contextual requirements has to support transition (2) (i.e., evolution concerning system behavior) or transition (1) (i.e., evolution concerning the context of validity). In the following, elicitation, discovery, and evolution of contextual requirements are defined and the concepts, introduced in this chapter, are discussed for each of the three activities.

4.4 Elicitation of Contextual Requirements

Definition 4.4.1. Elicitation of contextual requirements
The capture of contextual requirements at design time.

During the elicitation of contextual requirements all four transitions introduced in the previous section have to take place at design time (as depicted in Figure 4.3). Again, transition (3) and (2), as well as transition (4) and (1) are connected, as they represent the two paths of eliciting contextual requirements.

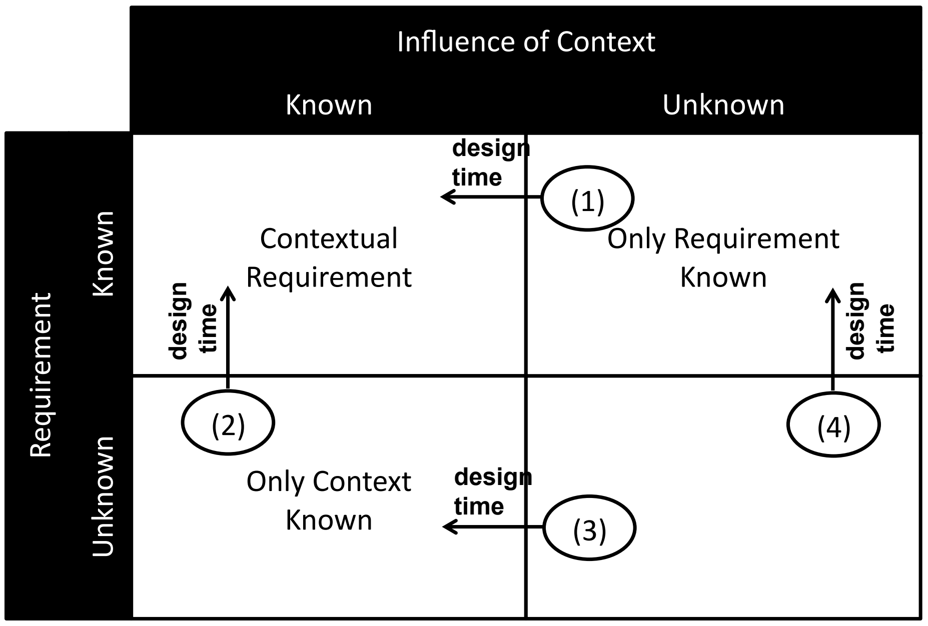


Figure 4.3: Design time: elicitation of contextual requirements

Transition (4) - This is the well known requirements elicitation: As this activity has been extensively studied for the last couple of decades it deserves no further attention in this dissertation.

Transition (1) - Elicit context in which requirements from (4) are valid: The context for which the requirement (system behavior) from transition (4) is valid has to be elicited. This implies a complete understanding of the circumstances in which the system behavior is needed at runtime. This dissertation explores (in research question 2) the use of existing requirements elicitation techniques, e.g. interviews, prototyping, and focus groups for the elicitation of contextual requirements from an existing requirements document, which basically represents this transition.

Transition (3) - Elicit context (for yet unknown system behavior): Starting with a context description in which a specific, yet unknown, system behavior should be valid is challenging at design time. The system is not implemented yet and therefore the operational context and user interaction with the system cannot be observed. Approaches for contextual design are a good starting point for the elicitation of the context in which some, yet undefined, system behavior is needed [20]. Additionally, there exist tools that can be used in the real environment at design time to collect context situation that would have been otherwise undiscovered [119]. Both sources are just a starting point as they present approaches to understanding the context, but not explicitly present a way on how to elicit and document the context in which contextual requirements are valid.

Transition (2) - Elicit system behavior for context from (3): For the context elicited in transition (3) we now have to determine the system behavior. Mostly, approaches combine both tasks of elicitation of context as well as system behavior, for example the approaches described in transition (3) (i.e.,[20, 119]).

4.5 Discovery of Contextual Requirements

Definition 4.5.1. Discovery of contextual requirements
The capture of contextual requirements or complementing partial knowledge of contextual requirements at runtime (when the system is operating).

Figure 4.4 illustrates the concept of discovery of contextual requirements with the view on the transitions that have to take place at design and/or runtime. The figure represents four transitions, from which two have to take place at runtime to call it discovery of contextual requirements. We purposely integrate the partial knowledge from design time in the activity of contextual requirements discovery. For example, a requirement is captured at design time and context of validity at runtime. Consequently, only transition (1) and transition (2) have to take place at runtime. Transition (3) and transition (4) take place at design time or runtime.

We give more details on each of the four transitions, with a focus on transition (1) and transition (2) as the activities that have to take place at runtime. Figure 4.5 summarizes steps that have to take place in transition (1) and (2).

Transition (1): In the process of discovering contextual requirements we start with a particular requirement (system behavior) that is either elicited at design time or runtime in transition (4), but for which the influence of context is not known at design time. If, at runtime, the system identifies that a system behavior

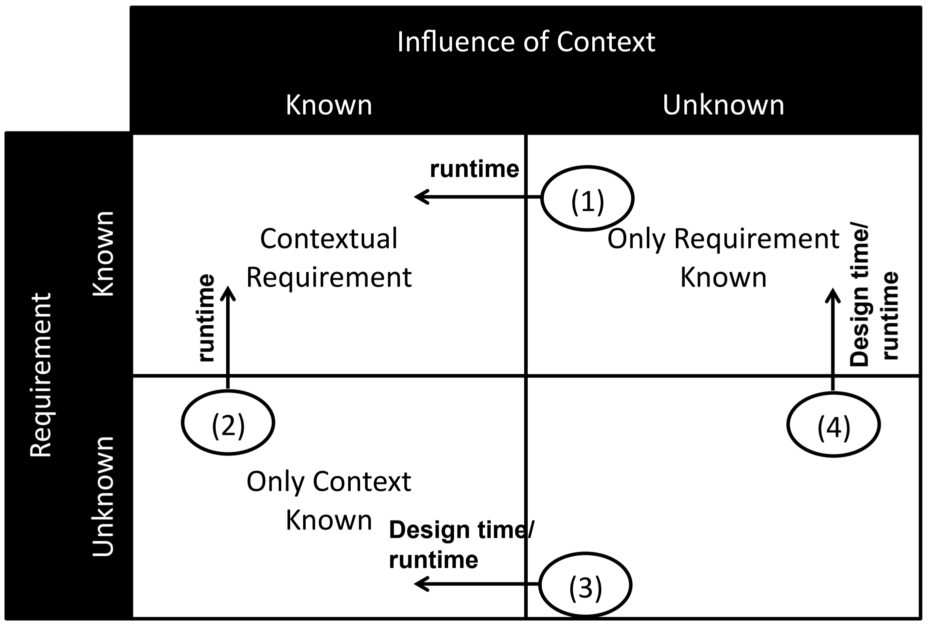


Figure 4.4: Discovery of contextual requirements

might be required *depending on the context*, then this dissertation proposes to use the following steps for the transition to a contextual requirement (transition (1) in Figure 4.2 and Figure 4.5):

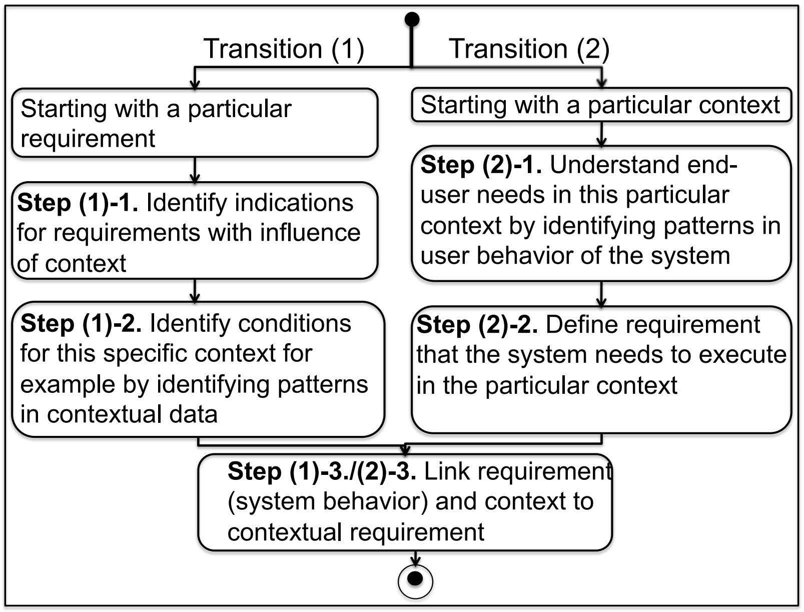

Transition (1)
Transition (2)
Starting with a particular requirement
Starting with a particular context
Step (1)-1. Identify indications for requirements with influence of context
Step (2)-1. Understand end-user needs in this particular context by identifying patterns in user behavior of the system
Step (1)-2. Identify conditions for this specific context for example by identifying patterns in contextual data
Step (2)-2. Define requirement that the system needs to execute in the particular context
Step (1)-3./(2)-3. Link requirement (system behavior) and context to contextual requirement

Figure 4.5: Details on transition (1) and transition (2) in the process of discovery of contextual requirements. Transition (1) starts from a known requirement and transition (2) from a known context.

Step (1)-1. Identify indicators for requirements (system behavior) that might only be valid in a specific context (e.g., by observing end-user needs and analyze in which context a particular system behavior is needed.)

Step (1)-2. Identify conditions for this specific context (if possible identify the relevant context-attributes and their values).

Step (1)-3. Link the requirements to the identified context in which they are valid to create contextual requirements.

Consider the parking lot example above: At design time the system behavior b_1, representing the requirement of directing cars to second floor, was elicited. At runtime, while using the system, we observed that users chose the ground floor if there were parking spaces not adjacent to pillars on that floor. This is a particular context in which the user needs the system to direct him to the ground floor, even if there are free parking spaces on the second floor as described by cr_1 (the only contextual requirement for the guidance system so far). This end-user behavior was an indicator that the requirement of directing cars to the ground floor was needed in a specific context (step (1)-1.), namely in the context c_2 of "free parking lots not adjacent to pillars" (step (1)-2.). The other context in which users chose the ground floor was when no parking lots were available on the second floor. The new contextual requirement of *if there are free parking lots not adjacent to pillars on the ground floor or no free parking lots on the second floor, direct car to the ground floor* is discovered by merging the system behavior with the context in which this behavior is needed (step (1)-3.).

Following this approach of distinguishing the context from the requirement, the system is able to "learn" the context in which a *particular requirement* needs to be executed over time even if the *influence of context is not known at design time*. The system learns the important context in which b_1 is needed, for example by observing the users in their interaction with the system and identifying situations in which the user is behaving differently than expected [24].

The integration of techniques to support the transition from requirements to contextual requirements at runtime is a necessary step in the discovery of contextual requirements. Existing requirements elicitation techniques should be complemented by methods that automatically identify requirements and analyze their context at runtime, especially for systems where the context is not even observable at design time.

Transition (2): Starting with a particular known context in which a requirement is needed (i.e., the system identifies in transition (3) a particular context in which the end-user needs require a certain system behavior). In transition (2) the system behavior for this particular context has to be identified.

This dissertation proposes the following steps for transition (2) to transfer our knowledge from partial to complete knowledge of a contextual requirement (Figure 4.5):

Step (2)-1. Understand end-user needs in the particular context, for example by identifying patterns in user behavior.

Step (2)-2. Define the requirement that the system needs to execute in the particular context.

Step (2)-3. Link the identified requirement to the particular context in which the requirement is valid to define a contextual requirement.

In the parking lot example above we could identify that every time a free parking space not adjacent to a pillar was available on the ground floor, end-users would drive their car there instead of to the second floor (as proposed by the system, because at the same time there were also free parking lots on the second floor).

The system might observe end-users in their behavior [24], collect usage data, and identify patterns in how the system is used in the particular context (step (2)-1.), thus identifying a new system behavior (b_2, step (2)-2.) and then link it with the context (step (2)-3.) to define the contextual requirement.

Transition (3): If the context of validity is not already captured at design time, at runtime the system might observe indications for the appearance of certain patterns in the context. After identifying a particular context that seems to have an influence on system behavior (Figure 4.2, transition (3)) in the next step, the system behavior has to be discovered (followed up with the case of unknown requirement/known context, transition (2) in Figure 4.5).

Transition (4): At runtime, the system might also discover that in certain situations (in a certain context) the end-users are using the system differently or do not use the system at all. Their behavior might give indications for a particular new requirement that has not been considered so far (Figure 4.2, transition (4)). Techniques that automate the discovery of requirements support this transition. For example, techniques that use machine learning techniques to (automatically) discover new requirements include mining online data [104] or sentiment analysis of app reviews [64]. By identifying the exact requirement, the system can then follow up with the case of known requirement/unknown context (Transition (1) in Figure 4.5).

4.6 Evolution of Contextual Requirements

During its lifetime, an adaptive system will face uncertainty in the operational environment or requirements. In such cases, the system needs to evolve the knowledge around existing contextual requirements at runtime to keep satisfying end-user needs. Figure 4.6 depicts the evolution of contextual requirements. As the evolution is based on existing contextual requirements, we start with the evolution in quadrant Q1 (i.e., the context and requirement are known, but have to evolve).

Definition 4.6.1. Evolution of contextual requirements
Evolution of the knowledge of the context or the system behavior of existing contextual requirements at runtime.

Transition (1) - Evolution of the context: If the context of validity is outdated or affected by runtime uncertainty, we have to determine an up-to-date context in which the contextual requirement is valid. Transition (1) is modeled such that it starts with an existing contextual requirement (Q1), migrates to Q2 (i.e., requirement known and context unknown), and finally moves back to a contextual requirement by determining the up-to-date context of validity.

Later in this dissertation, we investigate support for the evolution of the context in which contextual requirements are valid. We present an approach in which we use data mining and feedback loops for the purpose of evolution of contextual requirements.

Transition (2) - Evolution of the system behavior: Transition (2) follows a similar pattern as transition (1), in this case concentrating on the system behavior instead of the context. If the user needs evolve, we move from a contextual requirement (i.e., context and requirement known) to only context known. Determining the up-to-date system functionality allows to move back to the complete knowledge of the contextual requirement (i.e., context and requirement known).

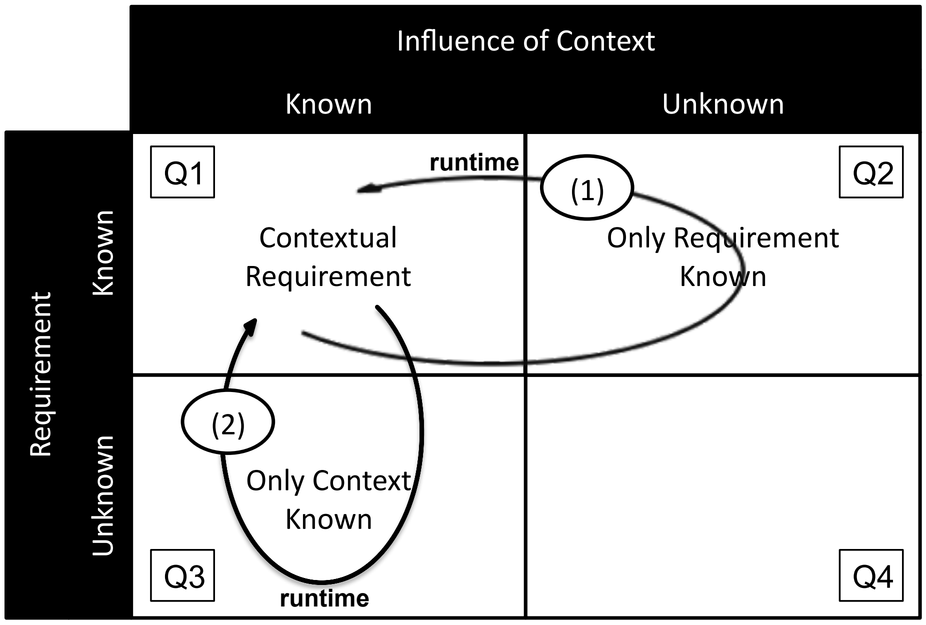


Figure 4.6: Evolution of contextual requirements

One example of the evolution of contextual requirements is shown in Section 4.1.2 for contextual requirement 1 (cr_1). For the system behavior of directing cars to the second floor, the knowledge of the context evolved from $c_{1.1}$ to $c_{1.2}$ in which cr_1 is valid (transition (1)). At the beginning, the system directed cars to the second floor when there were free parking spaces. After having identified contextual requirement cr_2, the original contextual requirement cr_1 had to evolve as users preferred the ground floor in case there were free parking spaces not adjacent to pillars. Only if this was not the case, they preferred the second floor if there were free parking spaces (here again the favorite choice was parking spaces not adjacent to pillars). Therefore, the context conditions in which cr_1 is needed evolved from $c_{1.1}$ of "free parking spaces on the second floor", to $c_{1.2}$ of "free parking spaces on the second floor and no free parking spaces not adjacent to pillars on the ground floor".

4.7 Chapter Summary

In this chapter we presented a framework for the capture and evolution of contextual requirements. Figure 7.1 summarizes the concepts of the framework. The framework presented three elements that play a key role in the capture and evolution of contextual requirements. It differentiated system functionality from context in contextual requirements (outlined in the black/white boxes in Figure 7.1). By this, it enabled us to consider the two parts of contextual requirements separately. Further, the framework differentiated between design time and runtime activities and therefore between elicitation (at design time) and discovery (at runtime) in respect to capturing contextual requirements. Finally, the framework distinguished between partial and complete knowledge of contextual requirements and allowed us for example to start with existing knowledge about the context and enrich it with the system behavior to have complete knowledge about a contextual requirement. This view enables a more flexible approach to the capture and evolution of contextual requirements, especially when more information about end-user needs and an (unpredictable) environment is available at runtime.

In the remaining sections of this dissertation we present our investigations into techniques to support the capture and evolution of contextual requirements. As the latter task takes place at runtime, we chose to investigate the capture of contextual requirements at design time (i.e., investigate the elicitation of contextual requirements) to cover different aspects of the framework. In our investigations we focus on applying the concept of partial to complete knowledge of contextual requirements. In research question 2 (Chapter 5) we investigate the use of existing requirements elicitation techniques for the elicitation of contextual requirements. We investigate path 2, starting with the elicitation of requirements represented through transition (4). For requirements that are contextual, we investigate techniques that support the elicitation of the context in which contextual requirements are valid (i.e., transition (1)). In research question 3 (Chapter 6) we investigate techniques to support the evolution of contextual requirements. As the new aspect of this dissertation are re-

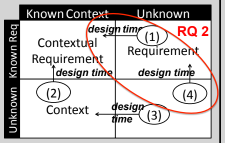
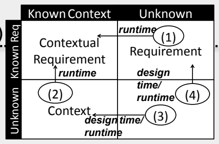
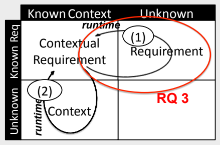
	Capture	Evolution
Design Time	*Elicitation:* **Partial to complete knowledge** Path 1: (3) + (2) = contextual req. Path 2: (4) + (1) = contextual req.  Known Context Unknown Known Req Unknown Contextual Requirement Requirement Context design time (1) (2) (3) (4) RQ 2 *Discovery:* **Partial knowledge** Path 1: (3) or path 2: (4)  Known Context Unknown Known Req Unknown Contextual Requirement Requirement Context runtime design time/ runtime (1) (2) (3) (4)	- not needed
Runtime	**Partial to complete knowledge (reuse knowledge from design time)** Path 1: (3 from design time) + (2) = contextual req. Path 2: (4 from design time) + (1) = contextual req. **Partial to complete knowledge (only runtime)** Path 1: (3) + (2) = contextual req. Path 2: (4) + (1) = contextual req.	*Evolution:*  Known Context Unknown Known Req Unknown Contextual Requirement Requirement Context runtime (1) (2) RQ 3 **Partial to complete knowledge** Path 1: contextual req. + (2) = evolved context in contextual req. Path 2: contextual req. + (1) = evolved requirement in contextual req.

Figure 4.7: Summary of the framework on the capture and evolution of contextual requirements.

quirements that are only valid in a specific context, we focus on the evolution of the context (operationalization) represented through transition (1), viewed in relation to the system behavior.

Chapter 5

Elicitation of Contextual Requirements at Design Time

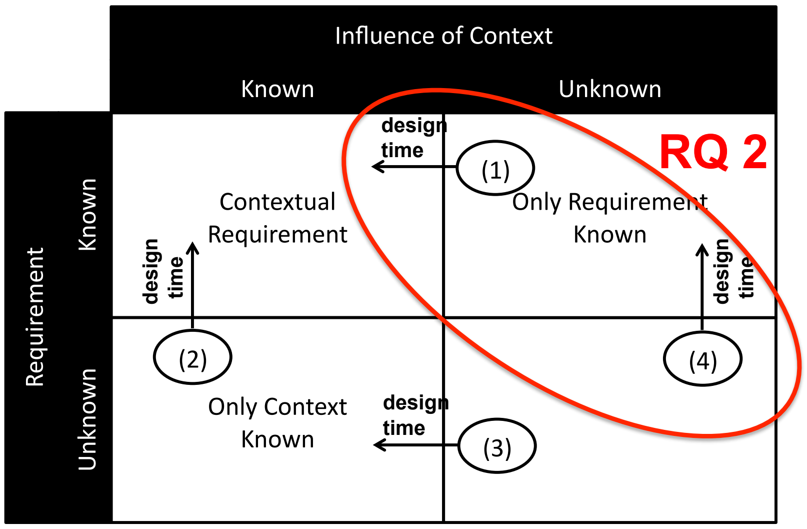


Figure 5.1: Investigating the support of the transition from known requirement and unknown context to contextual requirements, where context and functional requirement are known in the elicitation of contextual requirements.

In the previous chapter we defined the elicitation of contextual requirements as an activity that takes place at design time. To elicit contextual requirements we have to capture the system behavior as well as the context in which this is valid. The elicitation of requirements (transition (4) in Figure 5.1) has been intensely studied in the past. Established requirements elicitation approaches focus on eliciting the requirement, while understanding the context. In the process of eliciting contextual requirements, the challenge is mainly to detect the need for contextual requirements and elicit the context in which such contextual requirements are valid (transition (1)).

Before we develop new techniques to investigate the gap of eliciting requirements and their context of validity, we investigate whether existing requirements elicitation techniques can support the elicitation of contextual requirements (i.e., the elicitation system behavior, followed by the elicitation of context).

This chapter investigates the following research question[1]:

RQ 2: *How can existing requirements elicitation techniques help elicit contextual requirements at design time?*

In a case study we applied several existing requirements elicitation techniques and explored their use for the elicitation of contextual requirements. We had the opportunity to with the Human Resources Department at the University of Victoria in supporting them in their requirements engineering process for its job applicant tracking system. The system enables online job application submissions, managing the applications, and supporting recruiters for staff and faculty positions in the hiring process to find the right applicant. Furthermore, as soon as someone is recruited, the system supports the transition from an applicant to an employer for example by transferring information related to financing to the right place. This system had thousands of users and their requirements had to be considered for the implementation of the system.

We took the role of the requirements analyst in the software project at the University of Victoria. We applied several existing requirements elicitation techniques (e.g., interviews, prototyping, scenarios, goal-based approaches, and focus groups) to this real world socio-technical system. More precisely, we started with existing functional requirements and investigated how existing requirements elicitation techniques help eliciting the context in which a functional requirement is valid. For the elicitation of functional requirements the literature provides enough knowledge about the fact that and how existing requirements elicitation techniques support their elicitation. The important fact here is to elicit the context in which such requirements are valid.

We considered exploring ontology-based approaches (e.g., [51, 101, 28]) as well as

[1] A publication based on this chapter appears in Proceedings of the International Workshop on Empirical Requirements Engineering [78]

creativity techniques for the elicitation of contextual requirements. However, none proved applicable:

- Using context-ontologies proved to be difficult, as it required a lot of time and did not yield good information. Thus, the acceptance among users was low. Furthermore, we found the specific context provided by the ontologies restrictive. We concluded that it is not possible to elicit unknown unknowns [127].
- As we started from existing requirements, the acceptance of creativity techniques was low among our users.

We were able to document a number of contextual requirements by using existing requirements elicitation techniques. The use of particular requirements elicitation techniques in a particular order seemed to be important:

- First we identified requirements through *interviews* and *focus groups.* Interviews helped in understanding the rationale behind requirements.
- *Prototyping* was helpful to gain a detailed understanding of these requirements in context.
- *Focus groups* were helpful in identifying conflicts between different end-users when discussing requirements together in detail. These discussions helped identify the need for contextual requirements.
- Finally we identified, through *interviews* with the respective end-users, the different context related to requirements so that we could document the contextual requirements.

5.1 Example and Related Work

Before detailing the case study and its findings we give an example of contextual requirements elicitation from the case study and present background information that is important to understand our investigation in the case study. The example should give an understanding what a contextual requirements looks like for an applicant tracking system by differentiating between 1) the elicitation of a functional requirement and 2) the context in which this functionality is valid for the purpose of 3) the elicitation of a contextual requirement. The research question in this chapter investigates mainly transition (2), the transition from a known requirement and unknown context to a known requirement and known context as shown in Figure 5.1.

5.1.1 Example for Contextual Requirements Elicitation

An example of elicitation of contextual requirements from our case of the applicant tracking system:

1) *Requirement elicited:* Consider the requirement *"the system should be able to screen the applicants in case the recruiters have too many applications for one position"* and its rationale that screening helps with identifying qualified applicants.

2) *Relevant context identified:* A relevant context-attribute in this case is the number of applicants that applied to an open job posting and which can be small (e.g. 15) or high (e.g. 100). Another context is the department in which the job is posted, the qualification of applicants etc.

3) *Resulting contextual requirement:* If the number of applicants is high, the screening functionality should propose the 15 most qualified candidates. If the number of applicants is small, the functionality depends on the department. In some departments recruiters need to be able to get more information from the few candidates that applied. In this case chat functionality could enable recruiters to connect to the applicants and receive more information or to clarify information already submitted.

5.1.2 Related Work

In the preparation of our case study we identified the importance of different user viewpoints. Therefore, we assumed that different user viewpoints, as described in viewpoints-based approaches [140], have an impact on the requirements for the system under investigation. A *viewpoint* is defined as "a self-consistent description of an area of knowledge with an identifiable originator" [49, 50]. Because the system under investigation in our case study had thousands of end-users with different backgrounds and working in different domains (e.g., janitorial, medical, faculty) their viewpoints played an important role in the elicitation of the context in which contextual requirements were valid.

Zowghi and Coulin give an overview of requirements elicitation techniques in general and discuss which requirements elicitation techniques complement each other and which can be used as alternatives [140]. Based on this information we chose the requirements elicitation techniques that seemed promising to elicit and analyze information about viewpoints: interviews, groupwork, prototyping, and goal-based analysis.

In the elicitation of contextual requirements the identification of the exact context related to contextual requirements plays a major role. Van der Zanden argues for prototypes of the actual system as the requirements elicitation technique that helps users understand the system and their needs in order to support requirements engineering for a context-aware system [130]. His work is based on a literature review and is missing empirical investigation. In our case study we use mock-ups from the legacy system and similar systems to set users in context. Using mock-ups gives them an understanding what the system under investigation could look like.

Related work in the area of requirements engineering for socio-technical systems (e.g., Mate and Silva [90], Whittle et al. [138], Dalpiaz et al. [41], Lapouchnian et al. [81], and Ali et al. [11]) presents the importance of iterative development in socio-technical systems. For adaptive socio-technical systems, investigations mostly focus on requirements analysis. Dalpiaz et al. [41], Lapouchnian et al. [81], and Ali et al. [11] model and analyze contextual requirements that represent end-user needs in contextual goal models. An interplay of modeling and analysis as well as research that investigates the communication around contextual requirements is necessary. Our case study presents one of few case studies on requirements engineering in practice and extends their work by investigating the communication about requirements to enable modelling of contextual requirements. Models are used in our investigation as a foundation to support requirements communication.

5.2 Case Study on Eliciting Contextual Requirements at Design Time

This section gives an overview of our case study, presents the system under investigation and the requirements elicitation techniques we used in our exploration for the elicitation of contextual requirements.

5.2.1 Research Methodology

According to recommendations of Rubå and Cruzes [48] to study requirements elicitation processes based on field work and given contextualization, we conducted an exploratory case study [82, 40] with the office of Human Resources of the University of Victoria, Canada. While we had a practical role (analyst) in their project, our research goal was also to explore and continually reflect on the use of different requirements elicitation techniques (alone or in combination) in the elicitation of contextual requirements.

A case study of exploratory nature [108] allows us to observe natural behavior of stakeholders in socio-technical systems and the results of elicitation techniques. Such end-user behavior and the outcome of different requirements elicitation techniques cannot be observed from the outside, but needs to be studied in context. Therefore, a researcher has to stay a longer time in the field and become a part of local culture [82].

Table 5.1: Requirements elicitation activities in the project. The second iteration describes the case study we report on in this chapter.

Iteration	**Responsibility of**	**Purpose**	**Techniques used**
1	Professional req. analyst	Requirements elicitation	Interviews, focus groups
2	My responsibility	Contextual requirements elicitation & prioritization	- Interviews (3+24) in combination with: scenarios, prototyping, goal-based approaches. - Focus groups (7).

Table 5.1 shows the process of requirements engineering activities in the entire project. At the start of our investigation, the first iteration of requirements analysis had already been completed by a professional requirements analyst and integrated all end-user groups. At this point, the requirements engineering process was considered to be complete, with the prioritization of requirements missing. Our task (second iteration of requirements activities in this project) was to prioritize the requirements that were elicited and documented requirements in the first iteration. We

had the opportunity to meet with participants that got involved in the first iteration of requirements engineering activities.

For research purposes we were interested which of the elicited requirements depended on context and how this context interplays with the socio-technical system. Our task was to combine different existing requirements elicitation techniques for the elicitation of contextual requirements and related context-attributes that define the context. Systems integrating contextual requirements have to capture this context at runtime to decide about necessary adaptation. In the scope of this study we investigated how the dependence of requirements on context and the specific context itself can be determined based on discussing requirements with end-users.

Hence, our goal in this investigation was to explore existing requirements elicitation techniques with the goal to combine both, elicitation of requirements and related context that the requirements are valid in simultaneously at design time through the involvement of end-users. We used existing requirements elicitation techniques to investigate different user viewpoints: interviews, prototyping, goal-based analysis, and groupwork [140]. We added scenarios as a technique to be able to talk about requirements with users and that provided us with the possibility to understand requirements in context of other requirements, which is important in a complex setting. We explored the usefulness of the chosen requirements elicitation techniques for the identification of contextual requirements based on the need that we felt in the interview.

Before presenting our insights in the next section, we describe the system under investigation, users of the system, requirements elicitation techniques and how they were used in the case study.

5.2.2 System under Investigation: Applicant Tracking System

This study is based on a real project that aims at upgrading the staff and faculty applicant tracking system with thousands of users and introduces it into a bigger systems landscape. In the past the existing system was only used for hiring staff (e.g., library assistants, nurses, janitorial). Additionally to staff member a new group of end-users, faculty staff members, is introduced in the replacement process. Another goal for the replacement was to integrate the applicant tracking system in the complex environment of a socio-technical system – the University of Victoria system landscape as well as the thousands of users that use (parts of) this landscape – by integrating different systems that are used together. In the past the transition between the applicant tracking system and other systems was based on the manual work of end-users (e.g. printing documents and typing them in the other systems) who transferred the data into related systems. For example, the replacement system should facilitate to push or pull applicant data to and from a separate administrative system that is used to manage personal and accounting data for all employees.

For the requirements engineering effort, thousands of end-users of the system were classified into 6 large groups of end-users whose requirements are considered in the development of the system. The end-users and their goals are as follows:

- **Human resources** staff has different roles and goals that range from helping other end-users in solving their problems with the system to checking that regulations are adhered to collective agreement rules.
- **Finance** staff is interested mainly in confirming the availability of budget for new positions (also known as position control). Payroll/HRIS processes job and employee information for new hires.
- **Recruiters for staff** from different departments are looking to fill positions.
- **Recruiters for faculty** from different departments are looking to fill positions.
- **Operational unit** staff supports recruiters in the recruitment process.
- Finally, the biggest group consists of **applicants** that are interested in being hired and who use the system during the application process.

The case study we present offers highly diversified end-user groups. Some groups were substantially larger than others. The groups include people with a high level of responsibility that are quite busy and have little time to participate in the requirements engineering process, power-users that are using the system for hiring employees several times a week, novice users that have only used the system once, and users with other possible combinations of use in between these ranges.

Only a small sample size of each group could be considered in our study. Each sample covered 4-6 participants for each of the six core groups during the requirements elicitation activities. For some of these groups, the participants of the case study were only a very small sample size, whereas for the smallest group these 4-6 participants covered almost the entire end-user group.

5.2.3 Requirements Elicitation Techniques

In this section we describe the requirements elicitation techniques and how they were used in detail.

Interviews: Interviews with end-users constitute the elicitation technique most commonly used in practice [31]. This was also the preferred technique for our project. We started with a series of three interviews with end-users from different groups for the purpose of understanding the domain, identifying sources of requirements, analyzing the stakeholders, and selecting the requirements elicitation techniques to use [140]. The results of some of these activities were already well prepared and documented by the professional requirements analyst, minimizing our effort for example in studying legacy documents.

In the following, we conducted 24 additional interviews for our research purpose of identifying contextual requirements. We used voice recording for later analysis of meetings with end-users. We interviewed each end-user for approximately one hour. We enriched the interviews with task-analysis, scenario walkthroughs, and prototyping in different combinations. Goal-based approaches were used for requirements analysis in parallel. In some of the interviews we used all three techniques. In these interviews we started with discussions about end-users' goals to be supported by the system and the tasks they would be performing while using or aiming to use the new system. Following this, we asked the end-users to describe their scenarios. The last task was to talk about their (most important) requirements. This question sometimes helped end-users to think out of the box, and not only to think about their scenarios. Additionally, by talking about the most important requirements we got a feeling of what was perceived as important by the end-users.

Scenarios: The requirements documentation that we received from the professional requirements analyst did not include scenarios. The requirements were only mapped to high level user goals, e.g., "interviewing" or "applying for a job posting". During many of the interviews, we used scenarios for discovering requirements. Scenarios allow talking to end-users about their requirements step by step and to understand the goals that end-users aim to achieve through the completion of tasks.

We asked the end-users for their scenarios to be covered by the replacement system. If the end-user had difficulties to come up with scenarios, we divided this task in three subtasks. We asked the end-user to

(i) describe the scenarios for using the existing system,

(ii) how existing scenarios can be improved, and
(iii) which of the steps or even which scenarios can be added beyond the scope of the existing system.

We give a high level example of a scenario with the goal "fill a new position" that was discussed with one of the end-users. Note that this scenario can be divided into subgoals/tasks.

1. Looks at the application of each applicant.
2. Review applications.
3. Forward them to the hiring committee.
4. Create a short list (which is done by the hiring committee).
5. Email applicants that are shortlisted.
6. Telephone or email applicants (choice is on personal preference).
7. Schedule interview appointments and document information concerning the applicant in the system.
8. Put information into the system in the case when someone has been chosen.

After an applicant has been chosen to fill a position, a new entry with the employee data needs to be created in a separate administrative system. In this system all employees have an account, containing personal data, accounting data, etc. Both systems can be integrated in a way that data can be securely pushed and pulled between this system and the new applicant tracking system. There are some workflows (e.g., accounting approvals) that may need to bridge the two systems.

Prototyping: We explored the use of the following prototypes as visual representations to discuss requirements: 1) Screenshots from the existing system, 2) screenshots from another very successful system that fulfilled the same purpose, and 3) screenshots that were proposed by end-users.

Screenshots of the actual system and other systems with an extended set of features were used for clarifying requirements. In some interviews end-users talked about their experience with other systems used for the same purpose. They explained some of their requirements based on their experience. We asked them to show us the system and made screenshots of these systems. We used the screenshots in the following interviews.

Focus Groups: After discussing the requirements with a few end-users from the same group (the minimum was three interviews) we set up two hours focus groups. In total, seven focus groups with 3-7 participants were conducted to discuss the priorities of requirements. We employed a simple process: If end-users from the same end-user group did not give the same priority rating to a particular requirement this requirement was discussed in more detail.

We based our prioritization process on the work of Karlsson et al. and used the group sorting technique for prioritizing requirements with stakeholders [73]. At the end of our prioritization process, user requirements fell into three different groups of requirements: Requirements with an average of high, medium and low priority. The group with medium priority requirements contained many requirements with different individual ratings (high, medium and low), which we call here conflicting requirements. These requirements were analyzed in more detail, to identify the reason for the contradicting priorities.

Goal-based requirements analysis was used in parallel to document and analyze end-user goals as well as their tasks to fulfill these goals. We used task analysis in the beginning of most interviews in order to understand commonalities and compare the viewpoints of different end-users.

5.3 The Use of Requirements Elicitation Techniques to Elicit Contextual Requirements

Our main insight from this case study is the need for a combination of different requirements elicitation techniques for the elicitation of contextual requirements. We describe lessons learned from the application of these techniques as well as a combination that helped us elicit contextual requirements in our case study.

End-users from the same end-user group did not necessarily agree on requirements and priorities due to different user viewpoints. We identified the importance of viewpoints to identify context related to requirements early on in our investigation. One of the initial interviews was especially important for understanding the dynamics in the project. The person we interviewed was responsible for the help desk and knew the problems of the existing system as well as the needs of the different end-user groups. Through this interview we understood that project success would depend on the consideration of viewpoints in the prioritization of requirements. This prioritization was most challenging because of the complexity of the end-user groups. The scenarios for the six groups were partially heterogeneous and end-users from one group could not see the importance of the other groups, because of the limited understanding for the viewpoints of others. Therefore we considered different viewpoints as a source for requirements.

Requirements elicitation techniques should not assume that stakeholders have similar operating contexts. While analyzing the documented requirements that resulted from the first iteration we found 33 requirements with an average rating for high priority, 86 for medium priority, and 48 for low priority. From the 48 with low priority, there were even 22 requirements with low priority for the target group, from which the requirement originated. We investigated those 22 requirements out of 167 in more detail. We made an observation that requirements were important only for some end-users from the target group. They seem to have been documented when the requirements analyst encountered and understood them for the first time. In our case when we tried to understand those requirements in the interviews with the stakeholders the requirement was not investigated in detail, because the analyst, misled by her previous knowledge, failed to recognize the subtle differences that were caused by different end-user context. Prioritization would not help in this case as different end-user viewpoints are not considered for requirements. This is one of the main causes for the problems we encountered with prioritization. Asking users cannot solve this problem, because usually end-users do not have other end-users but only their own viewpoint. They think they are understood and cannot know that a similar requirement with potentially contradicting details already exists.

Designing a system as an adaptive system through the consideration of contextual requirements helped resolving conflicting requirements.

In our case study the indication that adaptation to context is necessary was given by the fact that no agreement could be achieved with respect to prioritization of some of the requirements (especially requirements with an average rating of medium that contained high, medium as well as low individual ratings) among end-users without considering the specific context for requirements.

One indicator for the need of systems adaptation to context is the fact that the execution of a requirement does not only depend on the input from the end-user, the system provides the functionality to, but also from other sources (e.g., for the contextual requirement of screening the applicants the system has to consider the context-attributes number of applicants, type of position, qualification of applicants). After identifying these context-attributes, the system should decide which functionality it offers to each end-user based on the exact value for the attribute at runtime. At design time this context-awareness and therefore consideration of context of the end-users for adaptation needs to be investigated. One opportunity for the identification is talking about different context with end-users and creating a shared understanding. Conflicts are great sources for contextual requirements and important context-attributes.

The original requirements elicitation based on established techniques was not complete and satisfying from the perspective of the project manager. The first indicator for this was the lack of prioritization of the requirements. Closer investigation on this issue revealed several problems:

- Different end-user groups did not agree on priorities. For this reason, the priorities of each group were weighted based on the importance of the group.
- Contradicting priorities of requirements even within groups: It was not possible to achieve an agreement on the priorities of requirements, not even within end-user groups, i.e. two end-users with the same role would often disagree on priorities.
- Contradicting requirements: End-users would agree on an abstract requirement, but would offer contradicting specific requirements when asked for details. Often, these detailed requirements would exclude each other.

Having finer grained end-user groups would not have been a solution, because the prioritization was planned on the given level of abstraction, i.e. based on end-user roles. Even with finer grained groups, an agreement about the prioritization and contextual requirements would have needed to be achieved. Finer grained groups only shift the problem up and require focus groups that consist of representatives from different end-user groups which would further increase the effort. Also, it was not possible to create finer grained groups in our case, because any two end-users would agree with one requirement and disagree with another requirement.

Conflicts among end-users helped resolve different viewpoints and identify important contextual requirements.

We combined different elicitation techniques to elicit end-user viewpoints. The need for a combination became apparent during a particular conflict in a focus group meeting. After one third of the scheduled time, one end-user left the room, apparently angry about the situation ("It was not a pleasure."). We conducted a follow-up interview in which we used prototyping and were able to understand the end-user viewpoint as well as identify context-attributes for the discussed contextual requirement. This interview revealed that the discussion during the focus group meeting was not aligned to the views of this end-user, even though all participants belonged to the same end-user group, had the same role, and no differences were visible. This specific end-user could not find herself in the discussion results. In order to achieve a consistent requirements description, it is important to be aware that conflicts can lead to great results and resolve them in a less frustrating way by facilitating them. Otherwise, the support of important end-users might be lost during elicitation and development, rendering the resulting system useless.

During our study, the following steps proved useful in our requirements elicitation approach:

1. Confront each end-user with a specific requirement in context.
2. If two end-users disagree, analyze their viewpoint and the context they are presuming.
3. Add as many context-attributes as needed to fulfill their needs through contextual requirements.

From the requirements perspective, this algorithm leads to important context-attributes for systems adaptation, i.e. the ones that determine when the system needs to adapt in order to fulfill the end-users' requirements.

Prototypes were useful in understanding the differences of end-users in context. When the first attempt to prioritize the requirements failed, our assumption was that the initial requirements elicitation was not sufficiently detailed. Therefore, we applied prototyping to get more detailed requirements from end-users [46]. For adaptive systems the activity of identifying and capturing the context for the development of the software system plays an important role. Therefore prototyping might have a really important role in identifying contextual requirements. Therefore our hypothesis was that prototyping would unveil enough context to identify important contextual requirements and thus overcome the limitations of only using one requirements elicitation technique. However, prototyping alone was not sufficient to overcome these limitations and to deliver consistent prioritization and requirements without contradictions. Our endeavor revealed that the initial requirements were complete and actually in a good shape. Prototyping lead to the same requirements, and simply added more details.

Prioritization was still not possible because of contradicting requirements, and no agreement could be achieved between different end-user groups, or even among the end-users of one group. The major problem was the wide range of end-users and their different needs (same end-user group, same requirement). It seemed that the documented requirements were only documented from the viewpoint of one end-user, probably the first who had communicated this requirement. While the other end-users from the same group seemed to have at first sight the same requirement, but when discussed in detail, they were, in fact, slightly different, demonstrating the need for different system functionality.

In our project, prototyping helped to identify these differences. By using the same prototypes when talking to end-users about the same requirement, an understanding of the differences in the needed functionality could be reached in most of the cases. In a few cases, the personal preferences were what made the difference.

No single elicitation technique was sufficient to capture contextual requirements on its own. To summarize we propose the following seven-step approach for a similar process that someone wants to use (see Table 5.2 for details). In this project none of the existing elicitation techniques was successful in eliciting contextual requirements on its own. Instead, our approach that consisted of a combination of techniques was successful because it allowed end-users agree on prioritization and resolve inconsistencies in viewpoints. Below we list the techniques and how they were used in the approach:

1. Identify all requirements from the (sample of) users (which in our case study was conducted by a professional requirements analyst).

2. Understand the rationale behind these requirements, to prepare for their discussion with other end-users.

3. Use appropriate prototypes to better understand the needs of end-users in context.

4. Let end-users prioritize the elicited requirements.

Table 5.2: Following these steps we could identify contextual requirements at design time.

	Technique	**Usefulness in identifying contextual requirements**	**Explanation**
Step 1	Interview, Focus Groups	Identify requirements	Having a complete list of requirements is necessary for a complete understanding.
Step 2	Interviews	Rationale behind priorities	Understanding why something is important helps to prepare negotiation.
Step 3	Prototyping	Detailed understanding	Establish reference for negotiation: Details for requirements are important to be able to compare these requirements to similar requirements from other end-users.
Step 4	Focus Groups	Prioritization of requirements	
Step 5	Focus Groups	Identify conflicting requirements	These conflicts between end-user requirements are sources of important context-attributes.
Step 6		Understand priorities and establish common ground on detailed requirements	How do those detailed requirements differ? Derive context-attributes and related contextual requirements.
Step 7	Interviews	Understand priorities and why they differ. Go back to step 2.	Probably information is missing for a complete understanding.

5. Identify conflicting requirements or prioritization of requirements.

6. Understand the rationales behind conflicting requirements. Are there cases where the end-users can agree on prioritization if documenting the requirement as contextual requirement?

7. If steps 1 to 6 do not lead to a successful understanding of the requirements then conduct further interviews with end-users who had contradicting priorities for the same requirement.

This approach was successful in our case study because it led to contextual requirements which allowed end-users to agree on prioritization. The discussion about different viewpoints helped end-users to understand the specific context of other end-users. This allowed the disagreeing end-users to accept that in a specific context, a given requirement is important. Discussing the complete picture during focus groups meetings, especially the end-user needs to differentiate a system behavior in different operational context of end-users from the same user group helped to establish a complete picture of the adaptation needs concerning this requirement.

5.4 Threats to Validity

5.4.1 Internal Validity

This exploratory case study is based on authentic data analyzed and interpreted by the author of this dissertation. The main concern here is that only one researcher was responsible for requirements elicitation. Therefore, there might be some limitations of our ability to objectively discuss the results. Firstly, it is hard to objectively judge if there were faults in the requirements elicitation techniques as applied by the first author. Secondly, the goal of this exploratory study was to investigate the use of requirements elicitation techniques for the elicitation of contextual requirements. With this goal in mind, the first author might have introduced a bias into the results. However, we are not reporting quantitative results, but share qualitative results in our example from this specific perspective.

A third threat to internal validity is caused by the fact that at the start of this case study a requirements specification for the project already existed. We were thus asked to help with the prioritization of these requirements and in return received the opportunity to explore the usefulness of elicitation techniques. However, the existence of requirements caused us to chose a specific path with respect to the framework of this dissertation, i.e. the path where requirements were captured first, followed by the capture of the context.

5.4.2 Construct Validity

In this exploratory case study, the selection of elicitation techniques and the classification of observations might be biased to some extent. For example, we discarded use of context ontologies and creativity techniques because of low acceptance and effectiveness with our users. Thus, the case presents some additional risks which we could not avoid in the activity of reflecting on the usefulness of elicitation techniques for elicitation of contextual requirements. However, we considered the opportunity to explore elicitation techniques in a real project to outweigh this threat to construct validity.

5.4.3 Conclusion Validity

As a single case study, our insights might have a low generalizability to other projects. We presented our approach as detailed as possible to support replication of our study. While it is possible or even likely that other researchers would identify a slightly different set of elicitation techniques or order to be useful for elicitation of contextual requirements, we are confident that our main results would be confirmed, i.e. the applicability of traditional elicitation techniques and the fact that conflict points towards relevant contextual requirements.

5.4.4 External Validity

As a qualitative exploratory case study, our ability to draw conclusions from our findings is limited to the scope of our study. In the applicant tracking case, the presented list of elicitation techniques was found useful to uncover contextual requirements, when executed exactly in the presented order.

Our insights are at least applicable to other applicant tracking systems for universities. Many issues we encountered seem to result from a wide range of end-users covered in this study. This indicates that a significant amount of our experiences are more generally applicable to the development of today's complex systems that often face the similar challenge of having to cover a wide range of end-users in the requirements engineering process.

While practitioners in comparable projects might profit from our experience, the existence of a good solution for our specific case is sufficient: we found indeed that existing elicitation techniques can be suitable for uncovering contextual requirements at design time.

5.5 Chapter Summary and Future Work on Elicitation of Contextual Requirements

In this chapter we shared our insights from an exploratory case study on contextual requirements elicitation. Specifically, we investigated the usefulness of existing requirements elicitation techniques in identifying contextual requirements at design time. Our main insight from this study was that no one elicitation technique is sufficient to elicit contextual requirements.

We were able to elicit contextual requirement by applying existing requirements elicitation techniques (interviews, focus groups, and prototyping) in a particular order. In summary, our insights from the case study included the following:

- Requirements elicitation techniques have to consider that end-users have different operation context. These different operation contexts can cause requirements conflicts among stakeholders, e.g. two stakeholders might agree on an abstract requirement but disagree on its specific refinements in a way that they cannot be fulfilled at the same time.
- Conflicts among stakeholders indicate the need for contextual requirements: Requirements of several stakeholders are not alternatives and are not consistent with each other - they rather apply under certain conditions that are determined by the context.
- Viewpoints are valuable to identify such context related to requirements and to analyze contextual requirements.
- Prototypes are particularly helpful to understand the context of conflicting requirements in detail.

Besides this investigation our plans for future work are the following:

Extend Investigation on Extisting Requirements Elicitation Techniques for Elicitation of Contextual Requirements

We presented one case study on the usefulness of existing requirements elicitation techniques for the elicitation of contextual requirements, focusing on the elicitation of the context in which contextual requirements were valid. Further research is needed to extend our insights into a more general approach that is applicable to different systems and domains for the elicitation of contextual requirements.

It would be important to *explore other existing requirements elicitation* techniques for the usefulness of eliciting contextual requirements. In our case study we were only able to explore an excerpt of existing requirements elicitation techniques. We believe that video-based requirements elicitation [25] could be quite useful in identifying differences in context when talking about a specific requirement. The reason for this believe is the fact that prototyping was very helpful in our case study in defining the context in which contextual requirements were valid. Prototyping enabled users to imagine what the system could look like. Videos could reinforce this fact through visualizations of different scenarios (which can consist of prototypes) [25] and the interaction between the user and the system.

It would be interesting to *use existing elicitation techniques* for *existing adaptive systems* to investigate how the presented work can be applied and extended to cases when an existing running adaptive system needs manual identification of context related to requirements. Conflicts are often based on different viewpoints and bringing them to surface is useful for adaptive systems in general (e.g., it has been shown that viewpoints are helpful to identify errors early in the development process in contextual goal models [9]).

Support Elicitation on Different Levels of Context

Beyond the use of existing requirements elicitation techniques, we envision the *level of context* to be an important topic in the elicitation (and discovery) of contextual requirements. In this dissertation, we differentiated between the higher level context (which we called context description) and the lower-level context (which we called measurable context). Elicitation of context-attributes (and related sensors) already at design time is challenging, but would be very valuable for adaptive systems. The more knowledge we gain at design time, the better we can make decisions on how to design adaptive systems.

Capturing context is partly a very *subjective task*. The perceived user context (as elicited in research question 2) is mostly on a higher level. The high level context description has to be made measurable through sensors. In most cases users will not be able to map such perceived context to sensors or even context-attributes. The task of mapping high level context to such specific, lower level context that can be measured by sensors is a difficult task and can be easily manipulated through subjectivity. This leads to another important research question that has so far not been addressed: How can we objectively derive the operationalization of context by considering the perceived context of different end-users?

Chapter 6

Supporting Evolution of Contextual Requirements at Runtime

Predicting, at design time, how a contextual requirement evolves is challenging as we do not know the changes in the context and end-user needs that will arise at runtime, especially in uncertain operating environments. Therefore, the ability to embed support for the evolution of contextual requirements into the requirements model already at design time is restricted. Instead, we posit that self-adaptive systems need to support the evolution of existing contextual requirements to learn from the data that they have available at runtime about user needs and operational environment.

In the framework (cf. Chapter 4) we defined the evolution of existing contextual

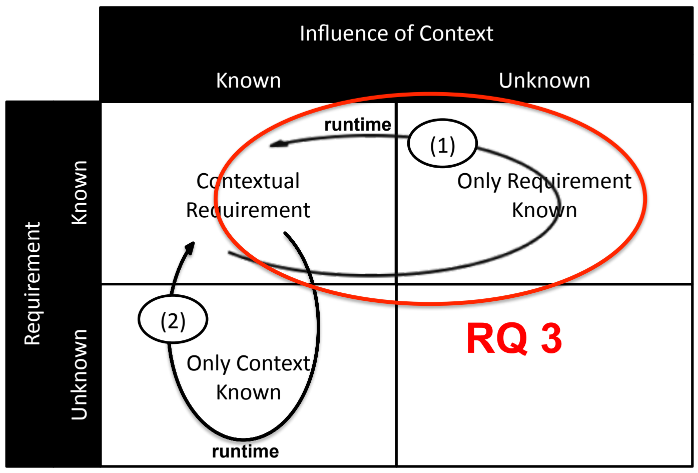


Figure 6.1: Research question 3 addresses the evolution of contextual requirements at runtime

requirements as the evolution of the knowledge of the context or system behavior. Figure 6.1 outlines the two possible transitions for the evolution of contextual requirements at runtime. Transition (1) visualizes the evolved knowledge of the context, respectively transition (2) the evolved knowledge of the system behavior.

Research into supporting the evolution of contextual requirements is limited and largely visionary in nature, focusing on the evolution of the system behavior and including the end-users in the evolution process [69]. Complementary, this chapter investigates the following research question:

RQ 3: *How can the evolution of contextual requirements that are affected by uncertainty be supported at runtime?*

In our investigation, we chose to concentrate on the evolution of the context part of contextual requirements (transition (1) in Figure 6.1). We focus on the context part, as uncertainty in self-adaptive systems at runtime arises mostly between a software system and its execution environment [106] and can affect the satisfaction of system requirements [107]. The two major root sources of runtime uncertainty tackled in this chapter are: 1) The dimension of unpredictability in the environment resulting from the fact that the requirements analyst cannot predict all possible environmental conditions when designing the system [107], and 2) runtime uncertainty due to the monitoring infrastructure (e.g., sensor imprecision, sensor noise, and sensor failures) [106].

Unpredictable environments have especially an impact on contextual require-

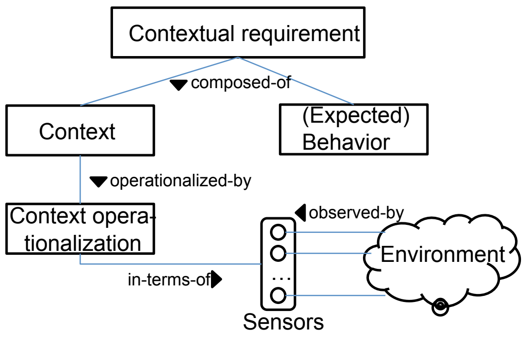


Figure 6.2: Contextual requirements decomposed of system behavior and context. The context is made measurable through a context operationalization in terms of sensors and their values.

ments. The context state in which a contextual requirement is valid, has to be made measurable (through sensors) for the system (see Figure 6.2). At runtime, uncertainty may result in invalid measurable context for contextual requirements. Additionally, runtime uncertainty may affect the system's ability to recognize a certain context in which contextual requirements are valid due to problems or changes in the monitoring infrastructure. Both conditions result in the system not being able to satisfy contextual requirements and to provide the system behavior.

The need to overcome this limitation calls for runtime support for the evolution of contextual requirements [92]. Based on the changes in the operational environment or the monitoring infrastructure the knowledge of the context in which contextual requirements are valid has to be updated. An up-to-date knowledge about contextual requirements enables the system to react appropriately at runtime [54].

In this section, we present an approach that updates the knowledge about contextual requirements with up-to-date information about the context in which contextual requirements are valid at runtime. Through an incremental runtime adaptation of the operationalized context in which contextual requirements are valid the approach supports the evolution of the context part of contextual requirements. We call the approach ACon (**A**daptation of **Con**textual requirements). ACon uses a feedback loop to detect indications that contextual requirements are affected by runtime uncertainty, and integrates data mining algorithms that are used on contextual data (i.e., sensorized environmental data) to determine the context in which contextual requirements are valid. ACon adapts the context in which contextual requirements are valid where appropriate and includes the interaction with end-users as part of a semi-automatic approach in which the human is in the loop.

For the purpose of validation we developed ToTEM - an activity scheduling system that was used in extremely demanding situations (with threat to life) in a wild and unpredictable environment with a complex monitoring infrastructure. ToTEM provided alarms for the daily activities of a crew of four rowers crossing the Atlantic Ocean during a 72-day trip [94]. In the case of ToTEM we collected contextual data from 46 sensors and mined this data to determine patterns guiding the appropriate adapted context for five selected contextual requirements.

Publication from this chapter has been submitted as "Alessia Knauss, Daniela Damian, Xavier Franch, Angela Rook, Hausi A. Müller, Alex Thomo. ACon: A Learning-Based Approach to Deal with Uncertainty in Contextual Requirements at Runtime. *Information and Software Technology*".

6.1 Background and Related Work

In this section we give an overview of related approaches and background information on feedback loops.

6.1.1 Related Work

Table 6.1 summarizes the characteristics of related approaches compared with ACon.

An adaptive system adapts to certain context conditions by monitoring the current operating environment (through sensors) to decide which actions it has to perform to fulfil system goals [34, 56]. Oriol et al. [96] argue that a *monitoring specification* (operationalization of context) is a prerequisite to designing requirements monitoring feedback loops to adapt to a certain goal, and derive this monitoring specification manually. Franch et al. [59] extend this work to integrate (again, manually) monitoring into an approach of goal-driven adaptation. Qureshi et al. [99] introduce adaptive requirements that contain variation points together with a monitoring specification, so that the system can make its own decision on how to best react to a particular goal with such variation points.

Even when using adaptive requirements, the challenge remains on how to detect certain context conditions in an uncertain operational environment when the measurable conditions might be unknown at design time to define the monitoring specification. Context operationalization is not trivial and requires a full understanding of relevant context. Even if context conditions can be operationalized, assumptions might become invalid at runtime [106], or sensor failure in the sensors monitoring these conditions make the re-operationalization necessary at runtime to provide the adaptive system with up-to-date contextual requirements. This situation requires techniques to track sensor data at runtime and adapt the operationalization of context in which contextual requirements are valid. We investigate how to update the monitoring specification of one (contextual) variation point automatically *at runtime* with the help of machine learning (i.e., data mining).

Related work shows that machine learning techniques are valuable for the development of self-adaptive mechanisms, where often great volumes of data are produced over time and can be used to derive decisions on self-adaptation. In this way, machine learning has been used successfully in different research areas on self-adaptive systems: Canavera et al. use data mining to determine the right time for system adaptation with the goal to avoid inconsistencies and system disruptions during and

Table 6.1: Summary of related work compared to ACon.

	Type of requirement	Purpose of usage in system	Consider uncertainty	Triggers changes	Lifecycle	Automatic execution	Techniques applied
Oriol et al. [96]	Monitoring requirements	Monitor adaptation	No	No	Feedback Loop	No	Event-Cond.-Action
Qureshi et al. [99]	Adaptive requirements	Capturing adaptation	Yes	No	N/A	No	Variability modeling
Canavera et al. [29]	N/A	Determine right time for adaptation	Yes	N/A	N/A	Yes	Data mining
Gullapalli et al. [63]	N/A	Self-management of adaptive controllers	Yes	N/A	N/A	N/A	Data mining
Esfahani et al. [53]	Features	Feature-oriented adaptation	Yes	Yes	Feedback loop	Yes	Machine learning
ACon	**Contextual requirements**	**Adaptation of contextual req.**	**Yes**	**Yes**	**Feedback loop**	**Yes**	**Machine learning (data mining)**

after adaptation [29]. They focus on situations (to which they refer to as "uncertainty factor") where the dependencies of system components are not captured in a model. They mine the execution history of a software system to infer a stochastic component dependency model, which represents interactions among the system's components and use this model to infer the right time for adaptation of a system component. Gullapalli et al. apply data mining to adaptive control in order to adapt feedback control based solutions to changes in the environment [63]. Their goal is to provide accurate decisions in tuning the control parameters in order to self-regulate distributed computing systems based on a time-series-analysis algorithm. Esfahani et al. use machine learning on a feature selection space to support system adaptation of features [53]. They focus on feature adaptation and suggest to treat the system as a blackbox. Thus, they do not base adaptation decisions on the managed system's internal structure. Instead, by defining a feature solution space, they are able to use machine learning to asses and reason about adaptation decisions. They map features to metrics that consider contextual factors to allow automatic learning of feature-oriented adaptation. While Esfahani et al. concentrate on making decisions on the adaptation of all possible features, we concentrate on only one particular system behavior to adapt the context operationalization.

Our work is motivated by the literature on the use of machine learning to support adaptive systems. Machine learning techniques seem to be valuable for the development of self-adaptive mechanisms where great volumes of data are produced over time and can be used to derive decisions on self-adaptation.

In comparison to these proposals (Table 6.1), our approach ACon uses machine learning to mine contextual data at runtime so that the system can continuously adapt its contextual requirements. Adaptive requirements capture points of adaptation [99]. ACon uses a MAPE-K feedback loop [38] to trigger adaptation of context operationalization concerning contextual requirements to deal with runtime uncertainty. Oriol et al. use feedback loops to monitor the context and provide system adaptation to current context conditions by using a monitoring specification [96]. They do not consider the adaptation of the monitoring specification. Three approaches – by Canavera et al. [29], Gullapalli et al. [63], and Esfahani et al. [53] – use machine learning (data mining) to support system adaptation. Only one of these approaches concentrates on the integration of requirements through the focus on features [53]. They concentrate on making decisions on the adaptation of all possible features, while we only concentrate on the adaptation of the context in which one system behavior is valid.

6.1.2 Background: MAPE-K Feedback Loops

Feedback loops are a key aspect of engineering self-adaptation [27]. Kephart and Chess introduced the notion of an autonomic element with its famous MAPE-K loop (cf. Figure 6.3), which culminated in IBM's architectural blueprint for autonomic computing and the Autonomic Computing Reference Architecture (ACRA) [38, 74]. These are key architectural elements of modern self-adaptive systems. Several autonomic elements can be composed to fulfill a common system goal.

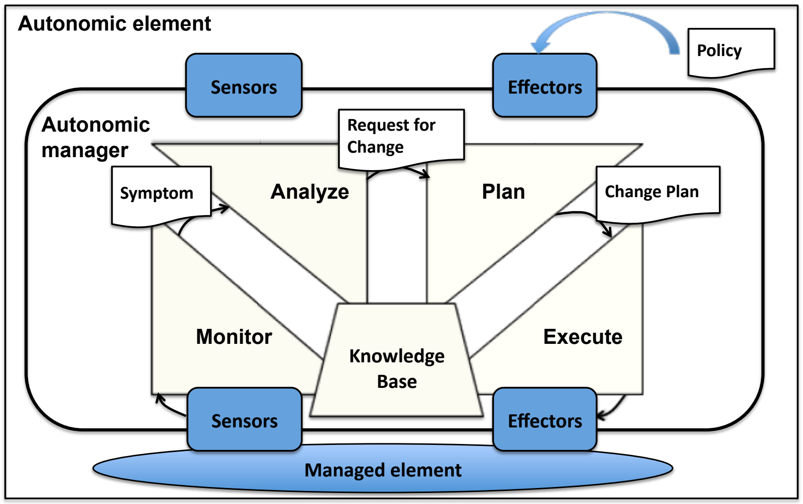


Figure 6.3: IBM's autonomic element consisting of a managed element and an autonomic manager with a MAPE-K feedback loop at its core [38, 74].

Each autonomic element consists of an *autonomic manager*, and one or more *managed elements.* The autonomic manager implements two *manageability interfaces – sensors and effectors.* Through sensors the autonomic manager gathers information from the environment or other autonomic elements. Through effectors the autonomic manager adjusts the managed element as needed. An autonomic element itself can be a managed element, therefore consisting of sensors and effectors at the top of the autonomic manager. Through the effectors at the top the autonomic manager can receive policies that drive the adaptation and evolution of the system.

The autonomic manager implements a feedback loop that is known as the *MAPE-K loop* because of the four components – monitor, analyze, plan, and execute – and the knowledge base. The *knowledge base* represents the major communication mechanism between the four components of the autonomic manager. The *monitor* senses the managed element and the context, filters the collected sensor data, and decides on relevant events that can indicate the need for an action of the autonomic manager. These *symptoms* are communicated to the *analyzer* for further analysis [39]. The analyzer correlates the received symptoms and in case it decides about the need to adapt the managed element, it sends a *request for change*. The *planner* defines the activity to be executed by considering the policy and creates a *change plan*. The *executor* adapts or evolves the system accordingly following the change plan [92].

6.2 Preconditions for the Application of ACon

This section formalizes the ACon framework by defining different functions over the domains introduced in Figure 6.2. Figure 6.4 outlines further details (e.g., variable names) that are necessary to understand the concepts presented in this section.

To illustrate these concepts we use a fictive but realistic example from a domain affected by runtime uncertainty: the intelligent vehicle domain. Consider the following contextual requirement *support lane keeping when end-user is sleeping*, represented with the 2-tuple cr = (end-user is sleeping, support lane keeping). In order for the system to be able to satisfy a contextual requirement, the context needs to be operationalized using available sensors (e.g., a camera installed in the car or some wearable devices).

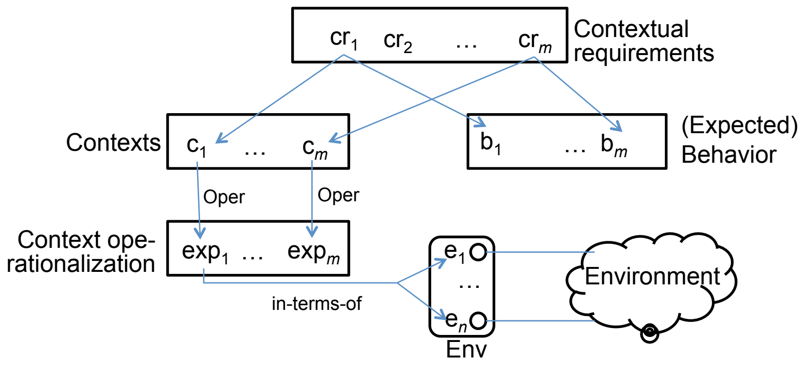


Figure 6.4: Variables and functions defined in ACon framework.

6.2.1 Definition of a System State

At any time t, the self-adaptive system will be in a given state concerning the requirements, the operationalization of their context and their satisfactibility. The state includes:

1. A set CR of m contextual requirements,

   $$CR = \{cr_i = (c_i, b_i) \mid 0 < i \leq m\},$$

   where c_i is the context and b_i the expected system behavior for a particular contextual requirement i.

2. $Contexts_{CR}$ and $Behav_{CR}$ are the sets of all contexts respectively expected behaviors of CR,

   $$Contexts_{CR} = \{c_i \mid 0 < i \leq m\},$$

   $$Behav_{CR} = \{b_i \mid 0 < i \leq m\}.$$

3. A set *Env* of typed variables that represents the environment measured through sensors,

   $$Env = \{e_i \mid 0 < i \leq n\}.$$

   The value of these variables is characterized by a measurement function

   $$Meas : Env \rightarrow Object,$$

   where *Object* is a supertype of all possible measure types.

4. The function *Oper* assigns to requirements' contexts its operationalization in terms of the environment variables

   $$Oper : Contexts_{CR} \rightarrow T(Env),$$

   where *T(Env)* is the term algebra formed from variables from *Env* combined with expression operators (i.e., relational, arithmetic, and logical). $vars(Oper(c))$

denotes the set of variables involved in the operationalization of c, $c \in Contexts_{CR}$.

We assume the following:

1. The contextual requirements in CR are elicited at design time and cannot change, which means that no unexpected system behavior may emerge and also that contexts cannot change, only their operationalization (represented by the *Oper* function) is allowed to change.

2. Variables in *Env* are fixed, too, meaning that sensors are deployed at design time.

3. $Meas$ is a partial function to reflect the fact that a particular sensor may become unavailable at a certain interval of time.

4. *Oper* is a partial function, meaning that at some points, the operationalization of a context is not yet known (typically because it is not known at design time and not yet determined at runtime).

5. Even if a context is operationalized, it may be not evaluable because some of the variables involved represent a sensor that is currently unavailable. To facilitate the detection of this situation, we introduce a predicate *evaluable* on contexts defined as: $\mathit{evaluable}(c) = \forall e : e \in vars(\mathit{Oper}(c)) : e \in \mathit{dom}(Meas)$, where *dom* is the actual domain of the specified function at a given point of time.

6. Even if the sensor is available, it may be sending wrong data. To detect such situations, we introduce a predicate

$$\mathit{outlier} : \mathit{Env} \rightarrow \mathit{Bool}$$

   over environment variables. *outlier* returns true if the current value of the variable is considered to be wrong.

7. At design time, an initial operationalization Oper_I of the defined contexts will be determined with all the operationalizations that are known beforehand.

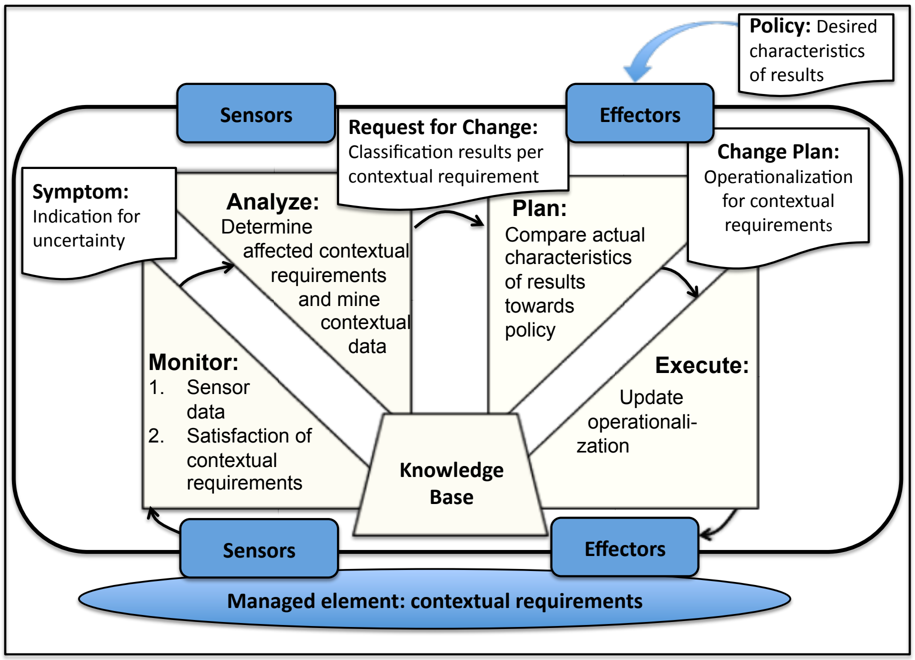

Policy: Desired characteristics of results
Sensors
Request for Change: Classification results per contextual requirement
Effectors
Change Plan: Operationalization for contextual requirements
Symptom: Indication for uncertainty
Analyze: Determine affected contextual requirements and mine contextual data
Plan: Compare actual characteristics of results towards policy
Monitor: 1. Sensor data 2. Satisfaction of contextual requirements
Execute: Update operationalization
Knowledge Base
Sensors
Effectors
Managed element: contextual requirements

Figure 6.5: The ACRFL in ACon responsible for the adaptation of contextual requirements.

6.2.2 Satisfaction of Contextual Requirements

The ultimate goal is to maximize the satisfaction of contextual requirements under uncertainty. With this objective, we introduce selected evaluation functions to the ACon framework:

1. $Eval_{Oper}$, a function to evaluate the current operationalization of a context,

$$Eval_{Oper} : Contexts_{CR} \rightarrow Bool,$$

   defined as

$$Eval_{Oper}(c_i) = evaluation(Oper(c_i)),$$

   where *evaluation* is a function that analyzes the operationalization of the context in some given logic that interprets the expressions formed with the term algebra *T(Env)* previously chosen.

2. $Eval_{Behav}$, a function to evaluate the satisfaction of a given system behavior,

$$Eval_{Behav} : Behav_{CR} \to Bool.$$

3. $Eval_{CR}$, a function to evaluate the satisfaction of a contextual requirement,

$$Eval_{CR} : CR \to Bool.$$

We define $Eval_{CR}((c_i, b_i))$ in the following manner:

- if $c_i \notin dom(Oper)$, then the context is not operationalized, therefore the context cannot be evaluated. In this case, we consider that the contextual requirements is not satisfied
- if $\neg evaluable(c_i)$, the contextual requirement is considered not satisfied since the current context operationalization cannot be evaluated
- if $\neg c_i$, the contextual requirement is considered satisfied since the context acts as guard when evaluating the satisfaction
- if c_i and b_i, this means that both the context and the behavior are satisfied, yielding thus to the satisfaction of the contextual requirement
- if c_i and $\neg b_i$, this yields clearly to insatisfaction of the contextual requirement

Given this, we define:

$$\begin{aligned} &Eval_{CR}((c_i, b_i)) = (c_i \in dom(Oper)) \\ &\wedge evaluable(c_i) \wedge (c_i \Rightarrow b_i) \end{aligned}$$

Note that, in order to make ACon as generic as possible, we leave the choice of formal framework both for expressing and evaluating contexts and requirements open. These formalisms do not impact the design of ACon as long as the three required satisfaction functions are provided.

6.2.3 ACon Objective Function

ACon aims at reacting to changes that make requirements unsatisfied. In the case of contextual requirements, the particular situation to avoid is having requirements

Table 6.2: Each contextual requirement is stored as a 2-tuple of context and expected system behavior in the knowledge base, together with the operationalization (rules) produced through the application of data mining.

Contextual requirements knowledge base

cr $\in$ CR	**Context ($\mathbf{c_i}$)**	**Exp. Behavior ($\mathbf{b_i}$)**	**Operationalization of context ($\mathbf{Oper(c_1)}$)**	**Sensors involved ($\mathbf{vars(Oper(c_i))}$)**
cr_1	End-user is sleeping	Support lane keeping	(BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75)	BlinkOfEye, PositionOfHead
cr_1	End-user is sleeping	Support lane keeping	(RightArmOffSteeringwheel = 1) and (LeftArmOffSteeringwheel = 1) and (PositionOfHead > 0.95)	RightArmOff-Steeringwheel, LeftArmOff Steeringwheel, PositionOfHead
cr_2	It is raining	Activate wind-shield wiper	...	...

whose context is not operationalized or is not satisfied while the behavior is. These are the two cases that violate the notion of satisfaction of contextual requirements. This objective can be formalized in the following way:

$$\boxed{\text{minimize } cr_i : 0 < i \leq m : \neg Eval_{CR}(cr_i)}$$

In order to accomplish this goal, ACon will continuously try to operationalize contexts not yet operationalized, and will react as soon as possible to unpredictable environment changes and monitoring problems.

6.3 ACon - An Approach to Support the Evolution of Contextual Requirements

Since ACon intends to support systems in self-adaptation, we adopt a feedback loop-based approach as an essential part to realize the adaptation. The feedback loop, which we call *Adaptation of Contextual Requirements Feedback Loop* (short: ACRFL), identifies conditions in which contextual requirements are affected by uncertainty and adapts the context operationalization for such contextual requirements.

Figure 6.5 gives an overview of the elements and activities taking place in the ACRFL. The managed element of the feedback loop consists of all contextual requirements of the system. The *monitor* keeps track of the managed elements. The monitoring component continuously executes two main tasks: 1) evaluating the satisfaction of contextual requirements, and 2) collecting available sensor data related (see Figure 6.5). The monitor communicates *symptoms* to the *analyzer*: every requirements violation and sensor relevant change (i.e., indications of potential uncertainty). A symptom contains relevant information for the indication on uncertainty so that the analyzer can analyze the situation and make a decision on whether to act on the indications or not. If the analyzer decides to act on it, the analyzer uses data mining on the collected sensor data to determine an appropriate operationalization. The *planner* decides whether the results of the data mining algorithm are good enough and generates a *change plan*. The change plan contains the rules produced by the data mining algorithms representing a new operationalization. Based on the change plan the *executer* updates the knowledge base that contains information about contextual requirements as well as the operationalization for the contextual requirements involved.

We describe the tasks of the four components as well as the knowledge base of the ACRFL step by step in the remainder of this section.

6.3.1 Knowledge Base

The knowledge base is an important part of ACon and stores relevant information about 1) contextual requirements, and 2) sensor data as preparation to apply data mining on.

Information Related to Contextual Requirements

For simplicity we store all information about requirements in the knowledge base. Every requirement is stored as a 2-tuple (context, expected behavior).

Table 6.3 shows the knowledge base containing the contextual requirement from the intelligent vehicle example. The expected behavior (b_1) is *support lane keeping*

Table 6.3: Sensor data and related information are stored in the knowledge base.

Sensor data in knowledge base							
Time	**IA for cr_1**	**IA for cr_2**	**IA for cr_3**	**...**	**e_1**	**e_2**	**...**
...	...	...	...	...	...	...	...
$t-1$	true	true	false	...	0.34	0.8	...
t	false	true	false	...	0.34	0.7	...
$t+1$	false	true	false	...	0.34	0.7	...
...	...	...	...	...	...	...	...

in the context (c_1) of *end-user is sleeping*. To be able to satisfy this contextual requirement the system has to identify when an end-user is sleeping using the sensors or cameras installed in a car. Such an operationalization (represented through sensors and their values) could be (BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75), measurable by the sensors BlinkOfEye and PositionOfHead.

The operationalization allows the system to exactly recognize the context state (end-user is sleeping) at runtime. The different operationalizations created along time for the requirement are stored in the operationalization column in the knowledge base (each rule represented in one row in Table 6.3). We introduce the function former-operationalizations: $Contexts_{CR} \Rightarrow Powerset(T(Env))$ which returns the set of such existing operationalizations.

Information Related to Sensor Data for Data Mining

Typically, the data used for data mining will come from sensors of different types. Before data mining can be applied on the sensor data, the incoming raw sensor data from the available sensors has to be preprocessed. Preprocessing of the contextual data depends on the characteristics of the sensors and data collected. The frequency of sensor readings that are taken into account for the operationalization have to be determined and can change at runtime (the time t, $t+1$, $t+2$ that is used to in ACon). For some sensors (e.g., sensors integrated in other devices and reused for the purpose of system adaptation) the readings might be less often than the defined time interval t_{int}. In such cases missing data for these less frequent readings has to be determined for example by keeping the value of the last reading till the next reading. Finally, the sensor data has to be normalized to lay within a range between 0 and 1 so that data mining algorithms that rely on Euclidian distance can be applied for operationalization.

After preprocessing of sensor data, another step is necessary to prepare the contextual data for data mining. An additional attribute, the *indicator attribute* (short: IA), is stored together with the sensor data in the knowledge base (see Table 6.3) and is needed to determine the operationalization of the context in which a contextual requirement is valid. Every time the expected system behaviour was satisfied it stores the current context together with a value true for the indicator attribute, false otherwise.

For the initial operationalization of the "driver is sleeping" context the user might help by manually activating the lane keeping feature when he has the feeling to fall asleep soon. Tracking these situations allows to exactly define conditions which indicate that the driver is falling asleep and in which the expected behavior of supporting lane keeping is needed. When the system starts executing the lane keeping feature on its own the system monitors the actions of the user. Slow actions might indicate that the user is sleeping. In addition end-user feedback can be used to determine whether the action was right or wrong.

ACon uses the evaluation function of the system behavior ($Eval_{Behav}(b)$ for $cr = (c, b)$) as the indicator attribute. $Eval_{Behav}(b) = true$ marks contextual data in which the expected system behavior is linked to the contextual requirement. $Eval_{Behav}(b) = false$ marks contextual data in which the expected behavior is not needed. Note that we expect the user correct the system if the executed behavior is not correct for example by terminating the system behavior. The indicator attribute together with the contextual data define the input for data mining algorithms to identify patterns that show correlation between contextual conditions with $Eval_{Behav}(b) = true$.

6.3.2 Monitor the State of the System

In the monitoring component, ACon keeps track of the satisfaction of contextual requirements in certain intervals of time. We define the time interval t_{int} between two measurements t and $t + 1$ of the monitor. ACon considers four different cases that can point to uncertainty affecting the satisfaction of contextual requirements (Table 6.4, case 1 - case 4).

Table 6.4: Monitoring of requirements affected by runtime uncertainty.

Cases	**Detection of uncertainty**	**Condition on context (c)**	**Condition on behavior (b)**
Case 1	No operationalized context.	$c \notin dom(Oper)$	
Case 2 a)	Sensor lost	$\exists e \in vars(Oper(c))$: $e \notin dom(Meas)$	
Case 2 b)	Sensor decallibrated	$\exists e \in vars(Oper(c))$: $outlier(e)$	
Case 2 c)	Sensor up	$\exists e \in vars(Oper(c))$: Time $t-1$: $\neg correctSensor(e)$ Time t : $correctSensor(e)$	
Case 3	Violation	$Eval_{Oper}(c)$ =true	$Eval_{Behav}(b)$ =false
Case 4	Potentially wrong context	$Eval_{Oper}(c)$ =false	$Eval_{Behav}(b)$ =true

Case 1: No operationalized context

This case will arise typically when the context has not been operationalized at design time and still there is not enough data for the data mining algorithm to propose a context. Eventually, it also may happen that a context becomes unoperationalized at runtime since no feasible operationalization can be found at a certain moment.

Affected requirements: $\{cr = (c, b) \in CR \mid c \notin dom(Oper)\}$

For each new car which does not include the sensor PositionOfHead, the analyst does not know how to operationalize the context *end-user is sleeping*. As a consequence, the operationalization is left to runtime, expecting than the ACon approach will be able to discover other ways to operationalize the context.

Case 2: An unpredictable monitoring framework state

When the context is operationalized, some of the sensors might become unavailable temporarily or permanently. Even if the signal is not lost, the data may be incorrect because the sensor requires recallibration. In any case, the system will not be able to satisfy those contextual requirements which contain the sensor that is malfunctioning in their operationalization. To identify this situation, all sensors that are part of the current operationalized context for the contextual requirements are monitored.

Three situations are possible:

a) The sensor stops sending data.

Affected requirements: $\{cr = (c, b) \in CR \mid c \in dom(Oper) \wedge (\exists e : e \in vars(Oper(c) : e \notin dom(Meas))\}$

b) The sensor is sending wrong data.

Affected requirements: $\{cr = (c, b) \in CR \mid c \in dom(Oper) \wedge (\exists e : e \in vars(Oper(c)) : e \in dom(Meas) \wedge outlier(e))\}$

c) The sensor regains a correct state. Either because it gets recallibrated or comes back into operation again. Requirements that have this sensor in their operationalization are affected, as well as all other requirements that contained the sensor in an operationalization in their history. This second type includes both requirements whose context is not currently operationalized as well as others that are operationalized because it may happen that the current operationalization does not behave as well as the one involving the recovered sensor. To detect conditions that trigger this case, we define $correctSensor(e) = e \in dom(Meas) \wedge \neg outlier(e)$.

Affected requirements: $\{cr = (c, b) \in CR \mid e \in vars(Oper(c)) \vee (\exists op \in$ former-operationalizations(c): $e \in vars(op))\}$

Note that it is not necessary to refer to unoperationalized requirements because these ones will be identified in Case 1 (see above).

Example for case a: Consider a user that is suddenly wearing sunglasses. So far the eye tracking sensor was the most accurate sensor to measure when a user is sleeping. After wearing sunglasses the sensor shows an error in capturing the sensor values. Hence, the monitor will communicate this loss of sensor. Later in the cycle, all other available sensors will be used for re-operationalization for "end-user is sleeping" context so that the system can continue monitoring the satisfaction of this contextual requirement.

Case 3: Contextual requirement not satisfied

This situation may occur in two completely different circumstances. First, it may happen that the context has been operationalized in an incorrect or inaccurate way (e.g., because the data mining algorithms have not learned enough yet or because the context has been reoperationalized recently due to some malfunctioning in a sensor of the previous operationalization) or because the action that leads to the satisfaction of the behaviour has not been executed yet. The reason will be found out in the next phase of the cycle (analysis), determining then the adequate actions for the next loop.

Affected requirements: $\{cr = (c,b) \in CR \mid Eval_{Oper}(c) = true \wedge Eval_{Behav}(b) = false\}$

Imagine the data mining algorithm was mostly used for daily trips where the user is awake. With the resulting operationalization, the system would only be able to detect "end-user is sleeping" situation with the end-user's fast blinking of eyes. Now during the night it might happen that the user needs to blink often due to the different light. Therefore, the operationalization will be satisfied while the system behavior is not, as it is not needed.

Case 4: Effects of environmental uncertainty

Cases occur in which the operationalization is not satisfied but the expected system behavior is. Especially at the beginning, when the data mining classifier is not properly trained because it does not have enough historical data, this case will appear as the system has to learn over time. In this case the user executes an action that makes the specified system behavior to be fulfilled, triggering the system to re-operationalize the context considering the current context state in the new operationalization.

Affected requirements: $\{cr = (c, b) \in CR \mid Eval_{Oper}(c) = false \wedge Eval_{Behav}(b) = true\}$

For example, a user who is quite tired realizes that he might be falling asleep in the next couple of minutes. Therefore, he switches on the lane keeping feature which makes context situation "end-user is sleeping" to become active. The current context should be considered for (re-)operationalization.

6.3.3 Analyze - Apply Data Mining to Operationalize Context for Contextual Requirements affected by Uncertainty

Given the output of the monitor, the analyzer determines which contextual requirements are affected by uncertainty. For these contextual requirements, it applies data mining to (re-)operationalize the context (determine $Oper(c_i)$). In cases of sensor loss, the analyzer will identify the missing sensor e_1 and ensure that it is not included in the operationalization (because $e_1 \notin dom(Meas)$).

In the analyzer, ACon relies on lightweight data mining algorithms (i.e. rule-based classifiers) which can be easily rerun many times as newer data instances become available. The outcome of applying such data mining algorithm is the identification of patterns, so called *rules* which represent the operationalization of context $Oper(c_i)$. Rules are produced on contextual data collected by the monitor at runtime and represent the operationalization of the desired context in which the contextual requirement is valid. Many different rules can exist for one contextual requirement, depending on how many patterns the data mining algorithm finds in the contextual data. This means that there will exist many operationalizations for the context of one contextual requirement.

For the sake of illustration, we show two hypothetical rules for our example in Table 6.3. The rule describes a situation in which the eyes of the end-user are almost closed (BlinkOfEye <= 0.9), and the head directed to the left side ((PositionOfHead <= 0.85) and (PositionOfHead >= 0.75)).

6.3.4 Plan - Decide whether Operationalization is Good Enough

The decision as to whether the rules that represent the operationalization of context are to be updated in the knowledge base depends on the system's tolerance of error in the rules produced by the data mining algorithm. In the planer the actual error in classifying correct context conditions (e.g. number of context conditions misclassified by the rules generated by the data mining algorithm and calculated by the 10-fold cross validation) is compared against the policy that is given. The policy can represent either an error threshold given by the system provider or given by the users. The users can define this threshold for the error at design time (and change it at runtime). If the error is smaller than the threshold, the system updates the contextual requirements with the new rules produced by the data mining algorithm at runtime.

6.3.5 Execute - Update Contextual Requirements with Operationalized Context

Last, the system updates the information about contextual requirements in the knowledge base with the operationalized context. The rules generated by the data mining algorithm become the candidate operationalization of context for the particular contextual requirement. Using these rules, the system can automatically recognize the context in which contextual requirements are valid at runtime and satisfy the expected behavior.

6.4 Evaluation of ACon

In this section we present the evaluation of our ACon approach. As shown in the previous section, ACon is designed to deal with two different types of runtime uncertainty, i.e. unpredictable environment and unpredictable monitoring infrastructure.

The main idea of ACon is to use data mining algorithms for context operationalization at runtime. Feedback loops are a means to automate this operationalization to tackle runtime uncertainty.

In our evaluation of ACon we focus on the data mining part of the approach to evaluate its applicability for the operationalization of context in cases where contextual requirements might be affect by an unpredictable environment and unpredictable monitoring infrastructure. We assume that users would be willing to help train the classifier to deal with uncertainty and therefore do not evaluate the cases that only relate to user-system interactions, but simulate when such events would happen. To illustrate these cases of uncertainty, we explored an extremely demanding situation (with threat to life) in a wild environment where lots of variables are involved.

We examined in execution an activity scheduling system, ToTEM, to support a crew of four athletes rowing in extreme conditions crossing the Atlantic Ocean, from Dakar (Africa) to Miami (USA). In the following we refer to this rowing trip as *DtM - Dakar to Miami*. To reach Miami within 60 to 100 days the rowers had to adhere to a strict schedule of required activities and daily routines (e.g., sleeping/rowing in four-hour shifts, and also adequate rest time to maintain their biometric rhythms [112], [75]), hence the reliance on ToTEM that ideally would consider the uncertain environmental conditions when adapting at runtime.

The rowers volunteered to provide feedback in situations in which they needed new system behavior *during the trip*, which presented us with a unique opportunity to record and examine the situations when new requirements were needed and in which context they should be valid in. Our interviews with the rowers *after the trip* as well as the analysis of their input at runtime allowed us to identify a number of contextual requirements.

In the remainder of this section, we describe the post-trip analysis we conducted of the runtime contextual data in relation to these contextual requirements. More specifically, we applied data mining on the contextual data to operationalize the context (i.e., identify measurable context conditions) in which the rowers' contextual requirements were valid. The post-trip analysis allowed us to apply data mining algorithms on parts of the contextual data and evaluate the results on the rest of the data (representing future contextual data from the viewpoint of this time). Implementing ACon and using it at runtime would not allow us to evaluate the applicability of data mining algorithms (which is crucial to show that implementing data mining in ACon is actually fulfilling the intended purpose).

We manually went through the following activities (that ACon would automate) to operationalize and evaluate the context in which the contextual requirements were valid:

1. *Preparation for the application of data mining algorithms:*
    (a) Elicitation of requirements: Based on the knowledge from the past trip as well as through interviews, and prototyping we elicited requirements and developed ToTEM for DtM.
    (b) Collecting end-user needs and sensor data: ToTEM was used by the rowers during DtM. We collected contextual data and users' runtime adaptation needs in context.
    (c) Identification of contextual requirements: We used the collected data to elicit and understand contextual requirements.
2. *Operationalization of context:* We applied data mining for the operationalization of the context in which the contextual requirements were valid.
3. *Evaluation of data mining (rules):* We validated the results of our data mining algorithms 1) through statistical analysis and 2) in interviews with the rowers.

6.4.1 Preparation for the Application of Data Mining Algorithms

(a) Elicitation of requirements

Drawing on the domain knowledge gained in the analysis of the contextual data when shaping the ACon approach, we elicited the rowers' goals and requirements for the ToTEM scheduler to be used in their DtM trip.

The major ToTEM functionality was to *alert the rowers about scheduled activities according to the current local (boat) time*. Other requirements included the system *allowing the rowers to manually assign activities*, *alerting rowers by using configured alarms*, and *allowing the rowers to configure the representations of the alerts*. Besides one contextual requirement, however, we were not able to understand the impact the context would have on ToTEM requirements. The rowers indicated their preference for automated rescheduling in certain context conditions but were unable to indicate which activities were to be rescheduled or under which conditions.

(b) Collecting end-user needs and sensor data

We implemented ToTEM which was used during DtM. The boat was configured with 46 onboard sensors that recorded biometric and environmental data. Biometric data was captured through ReadiBands[1] on the arm of each rower, and measured actigraphy (movement), effectiveness (fatigue level), and whether the rower was in bed. Environmental data included GPS position, ship roll, wind direction/speed, and altitude.

Unfortunately the trip did not reach its destination due to capsizing at about 1600 km from Miami. However the rowers recovered measurements from all 46 sensors, containing about 90,748 measurements per sensor, from the first 64 days of the trip. The rowers also recorded (in daily audio and written logs), as much as their busy schedule and harsh conditions allowed, their desired functionality for system self-adaptation in particular context conditions.

(c) Identification of contextual requirements

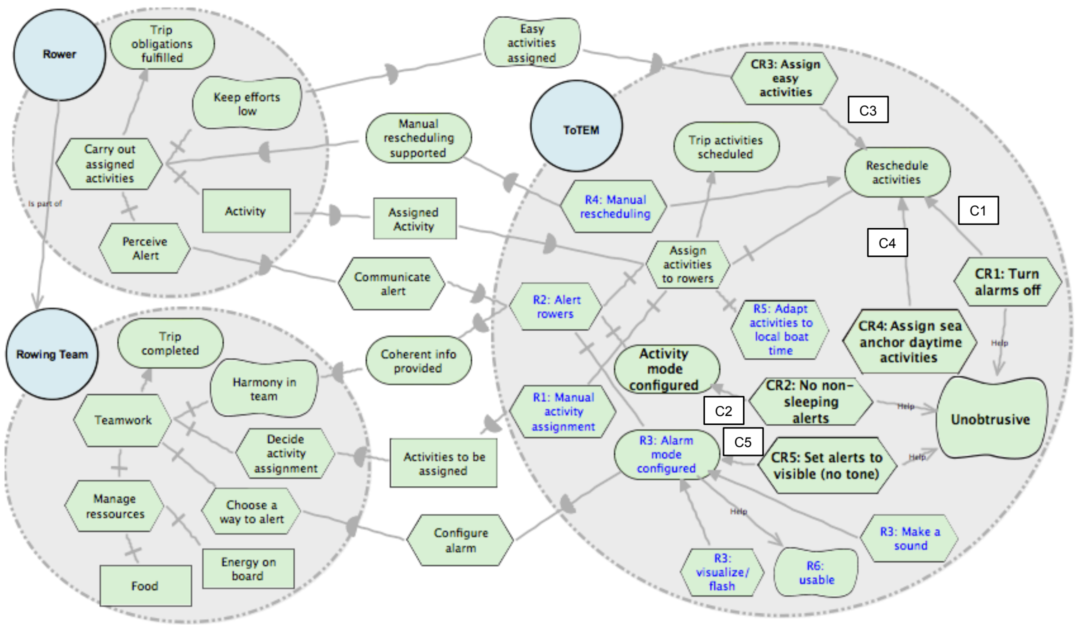
After the trip we analyzed these adaptation scenarios from the rowers' audio files as well as from interviews we conducted with the rowers. We identified that the rowers' goals for ToTEM adaptation to certain context conditions at sea became clearer during the trip, and we were able to identify five contextual requirements (shown in Figure 6.5).

We model the five contextual requirements in a contextual goal model (Figure 6.6). To integrate the contextual requirements into an adaptive system, we have to make contextual requirements runtime entities (as suggested by Bencomo et al. [19]). Contextual goal models have been recognized as a modeling technique to model requirements together with context conditions [7]. The goal model presents the system behavior as tasks (e.g., `turn alarms off` for cr_1) and the context of validity as boxes close to the arrows which connect the task with the goal. For the first contextual requirement the context (C1) is `on sea anchor at night`.

6.4.2 Operationalization of Context

With knowledge of these five contextual requirements, we turned our endeavors to the application of ACon. To operationalize context for each contextual requirement,

[1]http://fatiguescience.com/solutions/readiband



Context: C1: On sea anchor at night; C2: Two rowers are sleeping; C3: Low performance; C4: On sea anchor during the day; C5: One rower is sleeping

Figure 6.6: Contextual goal model representing the five contextual requirements for ToTEM.

we analyzed the contextual data from the entire trip, as well as the rowers' input, to identify correlations between contexts in which contextual requirements *appeared* to be valid and the actual sensor data collected from such contexts.

For cr_2, cr_3, and cr_5 (shown in Figure 6.5), where certain sensors clearly indicated relevant context conditions, the operationalization was trivial. c_2 (`two rowers are sleeping`) and c_5 (`one rower is sleeping`) directly correlated with the rowers band sensors capturing the times the rowers were sleeping; similarly for c_3 (`low performance`) we used the fatigue sensor-measurements to determine the rowers performance. The outstanding research challenge for measuring context for these contextual requirements was when there was loss of these sensors and different sensors had to be identified. Such cases appeared in the sensor data and ACon would have recognized such situations and would have (re-)operationalized the context, using sensors that are currently available. Because ACon uses data collected in the past, it can exactly give correlations to other sensors that are not malfunctioning with the help of data mining algorithms. Using the new operationalization the system now is able to recognize the context situations c_2, c_3, and c_5.

In contrast, for cr_1 and c_4 the situation was completely different because the boat was not equipped with a sensor to directly detect sea anchor conditions. As rowers

Table 6.5: Contextual requirements with most important sensors (sorted based on the importance calculated through data mining) and number of rules generated for day 53.

ID	**Valid context**	**Expect. behavior**	**Most important sensors (up to first 13 important)**	**# Rules**
cr_1	On sea anchor at night	Turn alarms off	Rower1InBed, Rower2InBed, Rower3-InBed, Hour, WindDirectionRelative, Rower2Effectiveness, Rower1Effective-ness, ShipPitch, Rower4InBed, Rower3-Effectiveness, GPSSpeedOverGround, WindSpeedRelative, GPSCourse OverGround	29
cr_2	Two rowers are sleeping	No non-sleeping alerts	Rower2SleepWake, Rower3SleepWake, Rower1SleepWake, Rower4SleepWake	7
cr_3	Low performance	Assign easy activities	Rower1Effectiveness, WindDirection-Relative, GPSSpeedOverGround, DistanceOverGround, Atmospheric-Pressure, Rower3InBed, Hour, Rower1InBed, WindDirectionBow, Rower4Effectiveness	8
cr_4	On sea anchor during the day	Assign sea anchor day-time activities	WindDirectionRelative, Rower4-Effectiveness, Hour, ShipRoll, Rower4-InBed, Rower1Effectiveness, Rower3-Effectiveness, GPSCourseOverGround, SpeedOverGround, Atmospheric-Temperature, WindDirectionBow, Rower1InBed, WindSpeedRelative	29
cr_5	One rower is sleeping	Set alerts to visible (no tone)	Rower1SleepWake, Rower2InBed, Rower4SleepWake, Rower3SleepWake, Rower2SleepWake, Rower4InBed, Rower1InBed	6

reported, the contexts for sea anchor (at day or night) were totally unpredictable and variable. Possible (though not fully understood) conditions for (sea) anchoring could be rower fatigue or sickness or bad weather conditions. In any of these cases, identifying which sensors would indicate sea anchor was not possible at design time.

Following the steps in ACon we first had *to identify contextual data in which the contextual requirement appeared to be valid.* We manually identified sea anchor conditions through an iterative process of inspecting log data, listening to audio files, and analyzing the sensor data visually. The Speed Over Ground sensor was not relevant as its values close to 0 could have indicated sea anchor but also rowing against heavy winds. The visual inspection of the graph showing the combination of

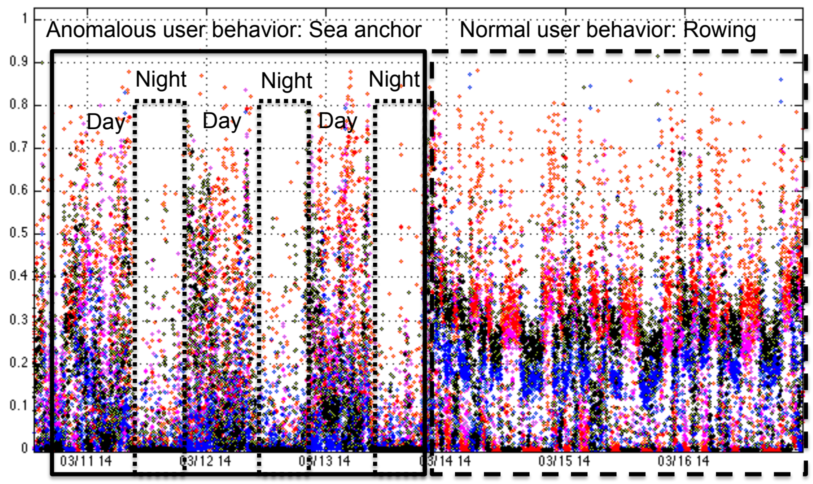


Figure 6.7: Anomalous user behavior indicates sea anchor conditions

actigraphy measurements for all four rowers was more useful. Figure 6.7 shows the rowers movements in different colors for each rower on the y-axis over a period of eight days (x-axis). Alternating clusters of similar movements show normal user behavior in the right dotted box — two pairs of rowers rowing in alternating shifts of 4 hours, 24 hours a day. The left continuous lined box shows anomalous user behavior for three sequential days. This anomalous behavior coincided with the times when the daily logs indicated that the rowers were actually on sea anchor. These are the conditions that represent the context for sea anchor at night and during the day. For the shown time at night it was discovered that at least three rowers are resting/sleeping during the times shown in "night" boxes. Given that there were only two sleeping spots in the cabin, these conditions could not be predicted by the analyst at design time.

Having identified the context "on sea anchor at night" we next had to identify the sensors and their values that correlated with this context. We used *data mining algorithms on the sensor readings to identify frequent patterns* that correlated with this context. First, we preprocessed the sensor data for the data mining by merging sensor data from all sensors into a single data set. As some sensors had measurements every minute, some every couple of minutes, and some every 15 minutes, we filled the entries in between by taking the last sensed measurement. Finally, we normalized all sensor readings.

Next, we considered data mining algorithms appropriate to the system's available processing resources (e.g., battery power and CPU). Because ToTEM is implemented on a mobile smartphone we choose JRip [128, 79], a rule-based classifier that uses

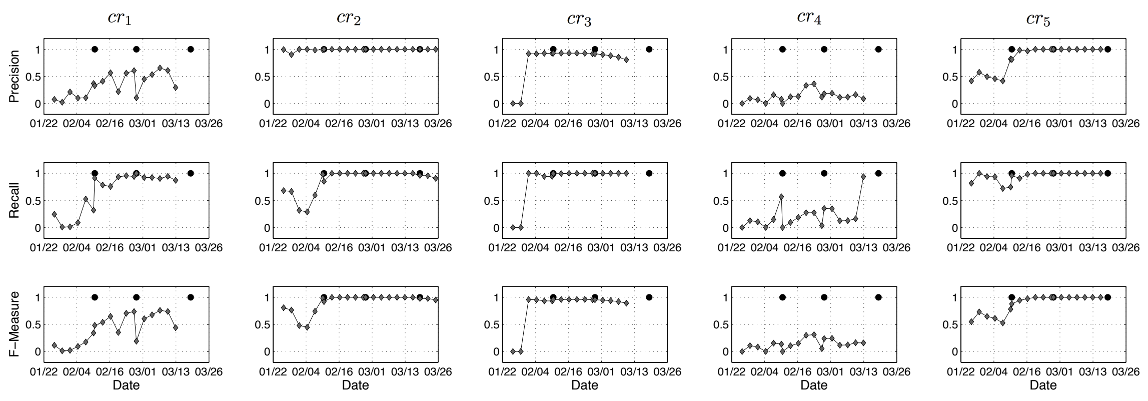


Figure 6.8: Time series analysis of contextual rules generated by the JRip algorithm for the five contextual requirements from Table 6.5. Each column shows precision, recall, and f-measure of correctly identified context related to each contextual requirement.

relatively few system resources. JRip produces a series of rules that can be converted into a series of "if...then statements" for system implementation. These represent the sensors and associated threshold values that characterize the context conditions for a particular contextual requirement.

Table 6.5 shows the first 13 sensors with the highest frequency in the rules generated by JRip for each of the five contextual requirements for day 53 in the trip. These rules represent the *operationalization of context* for the contextual requirements in our case study.

6.4.3 Evaluation of Data Mining (Rules):

We validated our context operationalization results on all five contextual requirements through 1) statistical analysis of the data mining algorithms and 2) interviews with the rowers.

1) Statistical analysis. We performed two statistical analyses based on accumulated data to validate the rules (operationalized context) generated by the JRip algorithm.

(A) For any day of the trip, we used the accumulated data to up to that day in a *10-fold cross validation* to check the data mining algorithms' performance on the data collected up to that point. During the 10-fold cross validation the collected data is randomly partitioned into 10 data sets of equal size. The data mining classifier is trained on 9 data sets (the training data set) and tested on the data set that is left (validation data set). This procedure is repeated 10 times, making sure that each of the 10 data sets is used only once as the validation data set. The means is determined out of the 10 results.

For the 10-fold cross validation we used Weka[2], which accounts for skewed data sets (relevant for most of our contextual requirements) by stratifying the folds accordingly (see [57], page 50). The 10-fold cross validation trains and classifies the context conditions on past collected data. The results were very high with average values between 98-99% for each of the precision, recall, and f-measure respectively for all five contextual requirements. We conclude from this analysis that the classifier performs well if applied on past data to classify events in the past.

(B) To assess the predictive power of the data mining classifier, we also applied the classifier on "future sensor data". We were interested in how rules, produced at one point in time, would classify context conditions at future points during the rowing trip. An accurate runtime classification of ACon is the major step for the satisfaction of contextual requirements. Therefore, we conducted a time series analysis of the classifier which trains on past contextual data and tests on future contextual data for the evaluation of the data mining approach in ACon. This is a stricter, but more realistic evaluation of the generated rules as it shows results as if ToTEM would have been used at runtime during the rowers trip. For our evaluation of the classifier we use the measurements *precision, recall and F-measure.* Precision shows how many of the cases that were identified as the desired context condition by the classifier are actually correct (compared to real life). Recall shows how many of all desired context instances existing in the sensor data set are also found by the classifier. F-measure is the harmonic means of precision and recall.

We demonstrate our results from the analysis for precision, recall and F-measure for each of the five contextual requirements (cr_1 - cr_5) in Figure 6.8, represented each in a separate sub-graph. The x-axis of each sub-graph shows the time of the trip for which we had sensor data (January 22nd to March 26th), the y-axis the normalization of the represented measures precision/recall/F-measure. Each grey diamond that occurs along the x-axis indicates our evaluation of the data mining classifier produced at that date (e.g., the first grey diamond represents our analysis for the day January 25th). The measurement is determined based on a classifier built on the sensor data up to that day (e.g., January 25th) and tested on the rest of the sensor data from that point on to March 26th.

[2]http://www.cs.waikato.ac.nz/ml/weka

The times when we did the validation of the operationalization (grey diamonds) represent times when ACon could have triggered the (re-)operationalization based on the four cases of the ACRFL when used at runtime. In our post-trip analysis we chose to operationalize the context on day three for the first time, followed by a (re-)operationalization every third day. Hence, day three covers *case 1* as there did not exist an operationalization before. Because the classification results of the 10-fold cross validation are quite good for these instances already, ACon would store this operationalization in the knowledge base. Every following grey diamond (besides the ones that are following the circles) represents cases where the users might have indicated cases where contextual requirements are not satisfied (*case 3*) or indicate effects of uncertainty (*case 4*). In the absence of real-time data on the user-system-interaction we applied heuristics in which such cases would appear every third day and therefore operationalize the context for these days.

In Figure 6.8, the times when sensor loss occurred are indicated by black filled circles. The first sensor loss (representing *case 2 a)*) was due to the need for energy conservation and affected the environmental measurements (wind speed, wind direction, ship roll, ship pitch, atmospheric temperature, and atmospheric pressure). The second black filled circle shows sensors gained for the previously lost environmental sensors (representing *case 2 c)*). The third circle shows the second sensor loss (again *case 2 a)*), the biometric sensor for one of the rowers.

For cr_2, cr_3, and cr_5 we observe that the classifier results improved in the first 18 days, going up to over 95% for each measurement. This indicates that ACon adapts to the runtime conditions and shows great results for the three contextual requirements after a learning phase of about 18 days. After the first sensor loss (first black filled circle) cr_2 drops in recall, but recovers very fast. Similar with cr_5, which also drops precision a bit, again it recovers fast. For the second sensor loss we do not have enough sensor data to evaluate the effect of this sensor loss for all contextual requirements, besides cr_2. For cr_2 recall drops about 10%, whereas precision stays the same. For contextual requirements that are prone to an unpredictable environment (cr_1 and cr_4), precision, recall, and f-measure fluctuate more than for the other three contextual requirements over time. Recall of cr_1 is better than for cr_4. This might be due to the fact that being on sea anchor allows the rowers to have an additional rest, resulting in patterns in the sensor data that measure the sleeping times, whereas the daily activities of the rowers when on sea anchor seem to be quite unpredictable (e.g., cleaning the boat, catching up with activities that they do not have time for on regular days).

In summary, the time series analysis shows that for three out of five contextual requirements ACon would have adapted at runtime after 20 days if used in our ocean rowing example, producing very high results for precision, recall and F-measure. For all three contextual requirements it was possible to achieve similar results even after (the first) sensor loss occured which seemed to affect the classification of the context conditions in first place. For cr_1 the data mining classifier achieved at certain time periods results of about 90% and even more after 27 days of the trip for recall, but only about 60% in precision. Depending on the circumstances this result is better than having no support for the adaptation. Only for cr_4 the data mining classifier delivered poor results over the entire trip.

2) *Two interviews with the rowers to validate the discovery and operationalization of the five contextual requirements* confirmed that the context we identified was correct and that the rules produced by data mining were appropriate (we discussed an excerpt of them). Furthermore, the rowers indicated that they would like to use the functionality of the contextual requirements (even if the classification of the context delivers low performance results) as early as possible during their next trip. They were willing to help train the classifier (by giving feedback about the correctness of the classification) rather than not using the systems adaptation.

6.5 Discussion

In this chapter we presented ACon, an approach that uses data mining techniques to operationalize context in contextual requirements at runtime. ACon uses a feedback loop to maintain an up-to-date knowledge about contextual requirements based on an up-to-date information about the context in which contextual requirements are valid at runtime. Upon detecting that contextual requirements are affected by runtime uncertainty, ACon integrates data mining algorithms that analyze contextual data to determine the context in which contextual requirements are valid, thus adapting the context information in the knowledge base. ACon includes the interaction with end-users as part of a semi-automatic approach in which the human is in the loop. In a preliminary evaluation, we evaluated the performance of the data mining algorithms, which lie at the core of ACon. In discussing our approach, we first review the threats to validity in our evaluation, then discuss the applicability of ACon to other domains, ACon's relationship to other research approaches in the literature, as well as possible extensions of ACon.

6.5.1 Threats to Validity

Internal Validity

With respect to internal validity, we see possible threats in the fact that we might have been biased in our evaluation. While we took great care to not influence the outcome during (manual) preparation of data, it cannot be guaranteed that no problem was introduced in this step. Also, we relied on interviews with the rowers for establishing the usefulness of our results. Although we carefully designed the interview guide, it is possible that the way we asked could have influenced the outcome. In addition, the rowers might have been biased towards helping the researchers and towards confirming the researchers goal. We believe that plans to actually use the system on the rowers' next trip mitigated this problem to a large extend.

Construct Validity

A threat to construct validity is that our evaluation of ACon was conducted through a post-trip analysis of the runtime contextual data in relation to the identified contextual requirements. We applied data mining on the contextual data to operationalize the context (i.e. identify measurable context conditions) in which the rowers' contextual requirements were valid. ACon was not used during TOTEM's runtime execution and our evaluation could be affected by our interpretation of the context where the contextual requirements appeared to be valid. At the same time however, the post-trip analysis provided us with an unique evaluation setup not possible if ACon would have been implemented and evaluated at runtime. It enabled us to apply data mining algorithms on part of the contextual data and evaluate the results on the rest of the data (which would have been future contextual data if ACon had been evaluated at runtime).

Conclusion Validity

In our evaluation we used five contextual requirements that we identified after the trip. We achieved results of over 90% for three out of five contextual requirements for the measurements precision, recall and F-measurement. The classifier for the other two requirements was at about 90% and even higher after 27 days of the trip for recall, and about 60% for precision. However, for one contextual requirement the results were very poor. This might be due to two reasons: 1) the data set for the contextual data that had to be classified as the valid context was the smallest data set of all five contextual requirements and might not have been sufficient to find a correlation, and 2) the data points that we had for the valid context consisted of times with little repeatability (i.e., the rowers being on sea anchor during the day and executing random activities).

External Validity

Our evaluation is highly specific to the case study and provides one example of how it is possible to automatically update context operationalization to improve the mapping of system behavior to valid context. In the specific case, even moderate accuracy promised value to the rowers and failure to execute a requirement would not imply immediate danger. For safety critical systems, an extensive assessment would be needed, which might prove difficult because of the high level of uncertainty in system, environment, and technical approach (i.e. machine learning). In any case we suggest to provide an option for users to override a decision of the system in the case that it fails to adapt correctly to the users' needs. Observing such user interference could prove to be a valuable information source for the adaptive system by itself and we encourage future research in that direction.

6.5.2 Application of ACon to other domains

ACon is well suited for domains with an unobservable environment, like the ocean rowing domain –which represents a dynamic, uncertain environment. We found that ACon can be applied for the operationalization of context and also trigger continuous adaptation of contextual requirements when runtime uncertainty is identified. Nevertheless, ACon only works semi-automatically as it needs the help of end-users in using the functionality manually before ACon starts to adapt to the context in which the user executes the functionality. Additionally, the end-user has to correct the system when requirements are not satisfied by the system or affected by environmental uncertainty.

ACon requires human-system interaction to achieve best results. Therefore, Acon is suited for not safety-critical systems, where human intervention is possible and timely responses not critical. The applicability of ACon for safety-critical systems would require further research on validation and verification of the data mining results. Additionally, other artificial intelligence techniques might be better suited in the area of safety-critical system (e.g., Artificial Neural Networks [80]).

The applicability of ACon to complex environments deserves some reflection. On the one hand, though the operational setting and direct users of ToTEM are relatively unique (i.e., four elite athletes on an open-ocean rowing trip), the overall challenge of developing a system for an uncertain operating environment is not. The analysis of uncertain operating environments for context becomes increasingly significant as mobile and cloud system developers create products that are used by *extremely broad audiences in unexpected settings*. Considering the example of a typical *smart city* scenario, with literally *thousands of sensors* continuously sending information that may be used for example by e-mobility services, ACon may provide significant value. For instance, one of the most important challenges that smart cities face is to *counteract sensor damage* or calibration loss. Since the cost to repair an individual unit is high due to the human involvement required, it is often the case that the sensor is disconnected until there are several damaged sensors in a relatively small area. Developers in these cases cannot anticipate the full scope of scenarios, and context changes that may occur at runtime. ACon can support developers in designing and implementing systems for such uncertain environments to support self-adaptive systems in changing contexts evolving under unknown conditions.

The smart city example naturally raises the concern of *scalability* of ACon. On the one hand, it becomes necessary to manage hundreds of contextual requirements that are represented in the system. Managing a large amount of contextual requirements is inherently complex, but ACon partially mitigates this problem thanks to the *feedback loop* and due to having one central place where adaptation is triggered and updates are executed.

On the other hand, we need to consider the *behavior of data mining techniques* for these large data sets. Jacobs discusses performance for big data, including examples of sensor data bases [71]. Jacobs argues that sequential access is very fast and is suitable for big data. The algorithm we used in our evaluation (JRip) performs only "sequential passes" over contextual data. Therefore, we may reasonably expect that ACon would scale even in this extreme smart city setting, although of course validation by experimentation is required.

6.6 Chapter Summary and Future Work on Evolution of Contextual Requirements

In this chapter we presented ACon, a novel approach for effective integration of operationalization and continuous evolution of the context in which contextual requirements are valid. ACon is targeted to systems that operate in uncertain environments and it uses data mining techniques to analyze sensor data to determine an up-to-date operationalization of context. ACon supports (semi-)automatically the long-term evolution of contextual requirements through the use of a feedback loop to detect contextual requirements affected by runtime uncertainty.

The development of ACon was based on the framework presented in Chapter 4 and focused on identifying contextual requirements that are affected by runtime uncertainty and supporting the evolution of the context (operationalization) of such contextual requirements at runtime. The application of data mining techniques in the evaluation scenario of the ocean rowing domain has demonstrated its great potential and has shown that supporting the evolution of contextual requirements leads to a better adaptation to end-users' needs over time.

Future Work on Evolution of Contextual Requirements

Further research on the evolution of contextual requirements has to include, based on the framework, support for two tasks: the evolution of the system behavior and the context in which the behavior is valid. For the purpose of evaluation, existing implementations could be used (e.g., the Tele Asisstance System [136]).

Transition (1) - Evolution of the Context of a Contextual Requirement

We have presented an approach (ACon) as a first step towards using machine learning techniques to support self-adaptive systems in the evolution of contextual requirements. Our evaluation of ACon showed great results for four out of five contextual requirements. The next step would be to further *explore the strengths and limitations of ACon*, as well as to explore the application of ACon to other domains. An extensive evaluation would allow us to experiment with different options and improve details of ACon. For example, one area to explore is the application of adaptive monitoring. We could monitor the context more often in cases where many requirements are violated. Furthermore, we can investigate different policies in the determination of the quality of data mining results. Currently, we are suggesting to use a threshold for the characteristics of data mining as policy.

In ACon we have viewed the context on the lower level, on the sensor level, while keeping the context description firm. The next step would be to *support the evolution of the higher level context*. There will be cases at runtime in which the context has to evolve on the higher level (completely new context of validity). We need techniques to identify the need for the evolution of the higher level context and to support its evolution.

Contextual requirements that are never executed are a source for contextual requirements that need evolution. Reasons for never executed contextual requirements might be a context (in which the contextual requirement is valid) that never appears in reality. It is important that a self-adaptive system identifies such cases and acts on them, potentially with humans in the loop to investigate this situation. This would extend our ACon approach, which so far triggers operationalization for cases when no operationalization is given, problems with the monitoring infrastructure occur, a contextual requirement is violated, or a contextual requirement is executed in a wrong context.

Another research direction in the area of evolution of contextual requirements is to combine ACon with RELAX-based approaches to uncertainty mitigation [105]. In particular, we could consider *applying the principle of goal relaxation* for the particular situations where the data mining techniques applied in ACon are not able to discover a clear correlation between context conditions and contextual requirement satisfaction. Hence, the evolution of the context description would be a relaxation of the context conditions.

It would be worth exploring how the evolution of the *social context* in contextual requirements can be supported at runtime. Understanding the human context is an important part of requirements elicitation activities [60]. Recently, Georg et al. [62] have presented the use of Activity Theory and goal/scenario modeling for requirements evolution, considering the social context. Their methodology uses monitoring capabilities to determine the effects of potential changes in the system (e.g., changes in the stakeholders' social constraints). These changes are used to determine system evolution. It would be worth exploring how such approaches, that consider the evolution of requirements due to social constraints could be used to support the (automatic) evolution of the (social) context in which contextual requirements are valid.

Transition (2) - Evolution of the System Behavior in a Particular Context

At runtime, the system behavior for some of the contextual requirements might be outdated, for example because the end-user needs have changed. Support for the evolution of the system behavior might consider the context of validity and observe the end-users in their needs in this particular context. For example, the interaction of the end-users with a cyber-physical system can be observed [24]. We can derive the expected user behavior through considerations of the expected system behavior. Tracking deviations from expected end-user behavior in the particular context might indicate the need for the evolution of the corresponding contextual requirement.

For example, imagine the expected system behavior is ”alarming the end-user about a task”. During the alarm we expect the end-user to pick up the phone, switch the alarm off and execute the task. If the user does not execute this task, but instead focuses on another task, this might require the system to consider the "new" task and evolve.

Furthermore, the exploration of automated support for the automatic identification of outdated requirements might be a valuable input to support the evolution of contextual requirements. Daun et al. [44, 45] recently presented an approach that supports the automatic identification of outdated requirements to support the requirements engineer in validating requirements against stakeholder intentions and correcting the system behavior. It is worth exploring whether and how such techniques can be used in adaptive systems to identify and evolve system behavior of contextual requirements. Recently, Vassev and Hinchev [131] introduced autonomy requirements – requirements that are able to learn and adapt autonomously. The presented concepts might be helpful in the case of contextual requirements as well.

End-users might be a valuable source for information concerning the evolution of the system behavior in contextual requirements. Almaliki et al. [15] recently presented an empirical study in which they explore the adaptive acquisition of user feedback at runtime. Such source of information might be valuable in developing support for the evolution of contextual requirements.

Chapter 7

Conclusion

In this chapter we summarize this dissertation, and outline its contributions and future work.

7.1 Dissertation Summary

Based on our preliminary works as well as a literature review, we identified the need for requirements that are only valid in a specific context, which we call contextual requirements. The capture and evolution of contextual requirements poses significant challenges for research in requirements engineering. These challenges include the capture of contextual requirements at design time as well as runtime when additional information about operational environment and end-user needs becomes available. In addition, we have to support the evolution of contextual requirements at runtime, for example in cases where runtime uncertainty affects the satisfaction of existing contextual requirements, to ensure the system is able to satisfy all end-user requirements and not fail due to uncertainty that it will encounter in its lifetime. Hence, the goal of this dissertation was to *investigate how to capture contextual requirements and support their evolution to address uncertainty during the design and operation of adaptive systems.*

In addressing this research goal, the dissertation tackled the following research questions:
Research question 1: *What are the essential elements of the capture and evolution of contextual requirements for adaptive systems?*

The research methodology included a literature survey in the area of requirements engineering and context management for self-adaptive systems. We studied this literature and derived the essential elements of the capture and evolution of contextual requirements for adaptive systems.

Research question 2: *How can existing requirements elicitation techniques help elicit contextual requirements at design time?*

In a case study we explored the use of existing requirements elicitation techniques in the elicitation of contextual requirements. Taking the role of the requirements analyst, we explored this question within a software project at the University of Victoria and applied different requirements elicitation techniques (e.g., interviews, prototyping, scenarios, goal-based approaches).

Research question 3: *How can the evolution of contextual requirements that are affected by uncertainty be supported at runtime?*

In this research question we explored the use of existing techniques such as data mining algorithms and feedback loops to support the evolution of contextual requirements in the face of uncertainty (i.e., unpredictable environment and sensor failure). We concentrated on the evolution of the context part of contextual requirements.

This dissertation therefore brings contributions of both theoretical and empirical nature, as follows:

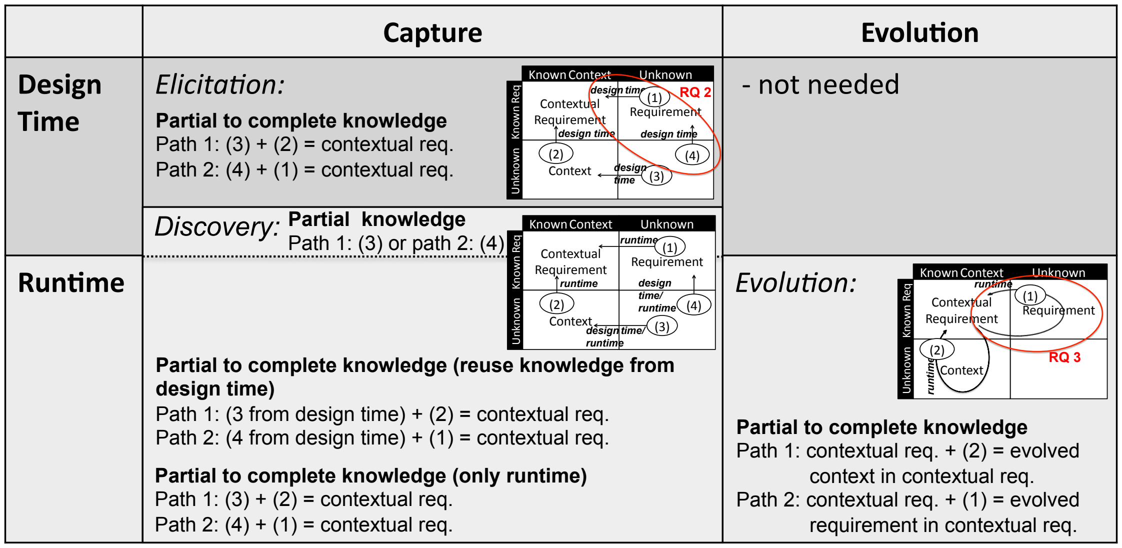
Contribution 1 (theoretical): This dissertation presented a framework for the capture and evolution of contextual requirements for adaptive systems (cf. Section 7.1.1). We applied parts of the framework in two investigations, which result in two further contributions.

Contribution 2 (empirical): Findings from an empirical investigation on the usefulness of existing requirements elicitation techniques for the capture of contextual requirements at design time result in the second contribution of this dissertation (cf. Section 7.1.2).

Contribution 3 (theoretical): As the third contribution, we present an approach for supporting the runtime evolution of the context in which contextual requirements are valid (cf. Section 7.1.3).

7.1.1 Contribution 1: A Framework for the Capture and Evolution of Contextual Requirements

We presented a framework for the capture and evolution of contextual requirements. Figure 7.1 summarizes the concepts of the framework, which is based on three elements:



	Capture	Evolution
Design Time	*Elicitation:* **Partial to complete knowledge** Path 1: (3) + (2) = contextual req. Path 2: (4) + (1) = contextual req. *Discovery:* **Partial knowledge** Path 1: (3) or path 2: (4)	- not needed
Runtime	**Partial to complete knowledge (reuse knowledge from design time)** Path 1: (3 from design time) + (2) = contextual req. Path 2: (4 from design time) + (1) = contextual req. **Partial to complete knowledge (only runtime)** Path 1: (3) + (2) = contextual req. Path 2: (4) + (1) = contextual req.	*Evolution:* **Partial to complete knowledge** Path 1: contextual req. + (2) = evolved context in contextual req. Path 2: contextual req. + (1) = evolved requirement in contextual req.

Figure 7.1: Summary of the framework on the capture and evolution of contextual requirements.

1) Context vs. system behavior in contextual requirements: Adaptive systems need to handle runtime uncertainty, which makes it impossible to have complete knowledge at design time about end-user needs and operational environment. Therefore, adaptive systems require a flexible approach in the capture and evolution of contextual requirements. To enable flexibility, our framework defined a contextual requirement consisting of two parts: the system behavior and the context in which the contextual requirement is valid. By considering the system behavior and context as two separate entities in contextual requirements we are able to capture and evolve only parts of contextual requirements, for example when more information on the context of validity is available at runtime.

2) Runtime vs. design time activities: Adaptive systems are developed at design time and can be extended as well as evolve at runtime. Hence, we have to consider design and runtime activities in capturing and supporting the evolution of contextual requirements. Motivated by the literature, the framework differentiates between the capture of contextual requirements at design and runtime. We referred to the capture at design time as elicitation of contextual requirements, and the capture at runtime as discovery of contextual requirements. The evolution of contextual requirements takes place at runtime and is related to existing contextual requirements.

3) Partial vs. complete knowledge of contextual requirements: Based on the previous concept of conceptually separating requirement and context, we had to consider partial knowledge of contextual requirements. Studying the capture and evolution of contextual requirements involved two parts: capture and evolution of the system behavior, and capture and evolution of the context in which the system behavior is valid. To enable the capture and evolution of a part of contextual requirements, the framework included a model that transitioned from partial knowledge to complete knowledge of contextual requirements for both, elicitation and discovery of contextual requirements. In supporting the evolution of contextual requirements we assumed that both, the context and system behavior, are known at runtime and the evolution is supported for (a part of) the contextual requirement.

The framework facilitates reasoning about the activities necessary for the elicitation, discovery, and evolution of contextual requirements. It facilitates decisions about which techniques to use for each of these activities. The three essential elements of the capture and evolution of contextual requirements offer value for practitioners as well as researchers in the field. Practitioners can use the framework to set up the requirements capture process for adaptive systems. For researchers the framework provides guidance to focus research activities and define research questions.

The framework tackles a gap in current requirements engineering research activities for adaptive systems. Most research in requirements engineering for adaptive systems focuses on requirements specifications, requirements modeling, and requirements monitoring (e.g., contextual goal modeling [7], requirements monitoring [96]) leaving a gap in the early phases of requirements engineering. A recent literature review has shown that there exist only three publications on the topic of requirements elicitation as well as three for system evolution of self-adaptive systems [124].

The next two contributions are dedicated to investigations of techniques for the capture (i.e., elicitation) and evolution of contextual requirements. Both contributions focused on the context part of contextual requirements (i.e., transition (1)). The elicitation and evolution of requirements are both quite well studied topics. For the elicitation and evolution of contextual requirements the challenge is to extend requirements with the context of validity. Therefore, we chose to focus on the elicitation and evolution especially of the context part in contextual requirements.

7.1.2 Contribution 2: Empirical Findings from Using Existing Requirements Elicitation Techniques for the Elicitation of Contextual Requirements

This dissertation brings empirical evidence from using and assessing the usefulness of existing requirements elicitation techniques for the elicitation of contextual requirements in a real software acquisition project. The techniques we used were interviews, focus groups, prototyping, scenarios, and goal-based analysis. Our findings suggest that there is no need for new elicitation techniques for the elicitation of contextual requirements. However, none of the existing requirements elicitation techniques used in our case study was sufficient on its own. A combination of these techniques was sufficient to elicit contextual requirements.

In the exploration in our case study we were able to document a number of contextual requirements when applying particular requirements elicitation techniques. We presented a step-based approach that helped us elicit contextual requirements. The approach combined the elicitation techniques — interviews, prototyping, and focus groups — in a particular order: After identifying requirements, we used interviews to understand priorities of requirements. In the next step, prototyping was used to gain a detailed understanding and establish a reference for negotiation. Finally, requirements were prioritized in focus group. Conflicts between end-users concerning requirements with different end-user priorities indicated requirements that were only valid in a specific context. The specific context was further explored in interviews to define contextual requirements.

Actual requirements engineering practice is rarely studied in real projects due to the difficulty to carry out such studies given the human, organizational, and political aspects that surround software projects [86]. Therefore, this case study is extra valuable as it contributes first hand insights on the application of requirements engineering techniques in eliciting contextual requirements.

7.1.3 Contribution 3: Approach for Supporting the Evolution of Contextual Requirements at Runtime

We presented an approach, ACon, that supports the evolution of contextual requirements affected by uncertainty. ACon updates the knowledge of contextual requirements by up-to-date information about the operationalization of context in which contextual requirements are valid at runtime. In ACon, we used a feedback loop to monitor contextual requirements and detect contextual requirements affected by uncertainty. We applied data mining algorithms on contextual data (i.e., sensor data) to update the context in which the system behavior was valid.

For evaluation, we used a wild and unpredictable environment (the ocean) with a high impact on the volatility of requirements. Based on five contextual requirements and contextual data from 46 sensors we applied data mining to make the context, in which contextual requirements were valid, measurable. Further, we analyzed the cases in which ACon would trigger the evolution of contextual requirements that are affected by uncertainty. We could show that our approach would achieve excellent results for four out of five contextual requirements after a period of 20 days into the rowing trip.

When applying ACon, contextual requirements are continuously kept updated in reaction to situations in which contextual requirements are affected by runtime uncertainty. ACon covers a current gap in the state of the art and nourishes related lines of research. For instance, it complements existing requirements monitoring approaches (e.g., the work by Oriol et al. [96]): The context operationalizations, derived through ACon, can be used as monitoring specifications for contextual requirements, thus allowing *timely adaptation* of the monitor in response to context changes. Thus, ACon adds to the field of evolution requirements [121] by enabling systems to evolve requirements flexible at runtime, instead of defining possible evolution at design time.

7.2 Future Work

In the following we sketch future work to devise support for the discovery of contextual requirements at runtime. Future work for the elicitation and evolution of contextual requirements has been presented in Chapter 5 and Chapter 6 respectively.

7.2.1 Discovery of Contextual Requirements at Runtime

We have investigated techniques for the elicitation and evolution of contextual requirements. The discovery of contextual requirements is so far a relatively unexplored research area and leaves room for future work. We have to develop techniques for each of the four transitions described in the framework. Figure 7.1 gives an overview of the framework.

Transition (4) - Discovery of Requirements

Traditional requirements elicitation techniques can be used for the purpose of capturing the requirement. However, techniques that automate the discovery of (new) requirements at runtime will be needed in self-adaptive systems. The discovery of a requirement is a first step in the discovery of contextual requirements.

Transition (1) - From Requirement to Contextual Requirement

Existing requirements might require to be contextual requirements at runtime. Support for the discovery of requirements that are contextual has to be investigated. Automatic support to identify such requirements, as well as automatic discovery of the context in which such requirements are valid is challenging, but would be helpful for self-adaptation of systems. Machine learning is worth exploring in this scenario, similar to our application of data mining in the ACon approach. Cluster analysis might help in discovering requirements that are always executed in one specific context.

Transition (3) - Discovery of Context

In this step we start with the discovery of a context in which a system behavior is needed. Again, at runtime techniques are needed for the discovery of context in which a system behavior is needed. An automation of this task would be helpful, but is challenging for self-adaptation of systems.

Transition (2) - From Context to Contextual Requirement

After discovering the context in which a particular system behavior is needed, we have to discover the particular system behavior for this context. An automation of this task would allow systems to self-adapt and discover new contextual requirements at runtime. For the automation of each of the transitions data mining and feedback loops might be useful, similar to our ACon approach. While developing ACon, we have partly applied data mining to operationalize the context in which contextual requirements were valid (transition from context to contextual requirements). We captured the higher level context (and requirements) manually through interviews with the users and runtime data analysis. Further investigation on this topic is needed to develop a structured approach for the different tasks of contextual requirements discovery.

The new area of self-adaptive systems is an exciting research domain that has already arrived. The limits of traditional requirements engineering are being stretched. However, luckily new methods such as data mining show promise to leverage the large amounts of runtime data and thus open up new directions for research into requirements engineering at runtime. In the last couple of years, workshops emerged on "Artificial Intelligence for Requirements Engineering (AIRE)", "Requirements at Runtime", "Just-in-Time Requirements (JIT RE)", and "Requirements Engineering for Self-Adaptive and Cyber Physical systems", which show the importance and attention of the community on the topic of requirements engineering at runtime. This thesis has provided a first, though hopefully significant step to support this exciting future.

Bibliography

[1] Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review. In *Proceedings of the International Requirements Engineering Conference (RE'06)*, pages 179–188. IEEE , 2006.

[2] Gregory D. Abowd and Elizabeth D. Mynatt. Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[3] Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, and Jean Paul Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *International Workshop Models@run.time at Models (MRT 2009)*, pages 89–98, 2009.

[4] Mikhail Afanasov, Luca Mottola, and Carlo Ghezzi. Towards Context-oriented Self-adaptation in Resource-constrained Cyberphysical Systems. In *Proceedings of the International Computer Software and Applications Conference Workshop (COMPSACW)*, pages 372–377, 2014.

[5] Charu C. Aggarwal, editor. *Managing and Mining Sensor Data.* Springer, San Mateo, USA, 2013.

[6] IA Al-Fataftah and AA Issa. A Systematic Review for the Latest Development in Requirement Engineering. *World Academy of Science, Engineering and Technology*, 64:800–807, 2012.

[7] Raian Ali. *Modeling and Reasoning about Contextual Requirements: Goal-based Framework.* PhD thesis, University of Trento, Italy, 2010.

[8] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458, July 2010.

[9] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology*, 55(1):35–57, January 2013.

[10] Raian Ali, Fabiano Dalpiaz, Paolo Giorgini, and Vítor E. Silva Souza. Requirements evolution: From assumptions to reality. In *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *LNBIP*, pages 372–382. Springer, 2011.

[11] Raian Ali, Alberto Griggio, Anders Franz, Fabiano Dalpiaz, and Paolo Giorgini. Optimizing Monitoring Requirements in Self-adaptive Systems. In *BPMDS 2012 and EMMSAD 2012*, pages 362–377, 2012.

[12] Raian Ali, Carlos Solis, Inah Omoronyia, Mazeiar Salehie, and Bashar Nuseibeh. Social Adaptation: When Software Gives Users a Voice. In *Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 28–30, 2012.

[13] Raian Ali, Carlos Solis, Mazeiar Salehie, Inah Omoronyia, Bashar Nuseibeh, and Walid Maalej. Social Sensing: When Users Become Monitors. In *Proceedings of European Software Engineering Conference (ESEC 2011)*, pages 476–479, 2011.

[14] Raian Ali, Yijun Yu, Ruzanna Chitchyan, Armstrong Nhlabatsi, and Paolo Giorgini. Towards a Unified Framework for Contextual Variability in Requirements. *International Workshop on Software Product Management*, pages 31–34, 2009.

[15] Malik Almaliki, Cornelius Ncube, and Raian Ali. The Design of Adaptive Acquisition of Users Feedback: an Empirical Study. In *Proceedings of International Conference on Research Challenges in Information Science*, pages 1–12, 2014.

[16] Annie I. Antón. Goal-Based Requirements Analysis. In *Proceedings of International Requirements Engineering Conference (RE 1996)*, pages 136–144. IEEE, 1996.

[17] Annie I. Antón and Colin Potts. Functional Paleontology: System Evolution as the User Sees It. In *Proceedings of International Conference on Software Engineering (ICSE 2001)*, pages 421–430, 2001.

[18] Twan Basten, Roelof Hamberg, Frans Reckers, and Jacques Verriet, editors. *Model-Based Design of Adaptive Embedded Systems*. Springer, 2013.

[19] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier. Requirements Reflection: Requirements as Runtime Entities. In *International Conference on Software Engineering*, pages 199–202, 2010.

[20] Hugh Beyer and Karen Holtzblatt. *Contextual Design: Defining Customer-centered Systems*. Interactive Technologies Series. Morgan Kaufmann, 1998.

[21] N. Uday Bhaskar and Dr. P. Govindarajulu. Context Exploration For Requirements Elicitation In Mobile Learning Application Development. *International Journal of Computer Science and Network Security*, 8(8):292–299, 2008.

[22] Patrick Brézillon. Using Context for Supporting Users Efficiently. In *Proceedings of Hawaii International Conference on System Sciences*, pages 1–9. IEEE, 2003.

[23] Olesia Brill, Constanze Deiters, Ursula Goltz, Sandra Lange, Benjamin Mensing, and Kurt Schneider. The RuleIT Methodology. Technical Report December, NTH Focused Research School for IT Ecosystems, 2010.

[24] Olesia Brill and Eric Knauss. Structured and Unobtrusive Observation of Anonymous Users and their Context for Requirements Elicitation. In *Proceedings of International Requirements Engineering Conference (RE 2011)*, pages 175–184. IEEE, 2011.

[25] Olesia Brill, Kurt Schneider, and Eric Knauss. Videos vs. Use Cases: Can Videos Capture More Requirements Under Time Pressure? In *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010)*, pages 30–44. Springer, 2010.

[26] Manfred Broy. The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems. *Computer*, 39(10):72–80, 2006.

[27] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi A. Müller, Mauro Pezzè, and Mary Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Self-Adaptive Systems*, pages 48–70, 2009.

[28] Oscar Cabrera, Xavier Franch, and Jordi Marco. A Context Ontology for Service Provisioning and Consumption. *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12, May 2014.

[29] Kyle R. Canavera, Naeem Esfahani, and Sam Malek. Mining the Execution History of a Software System to Infer the Best Time for Its Adaptation. In *Proceedings of International Symposium on the Foundations of Software Engineering (FSE 2012)*, pages 1–11. ACM, 2012.

[30] Lorena Castañeda, Norha M Villegas, and Hausi a Müller. Self-Adaptive Applications: On The Development Of Personalized Web-Tasking Systems. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*, pages 49–54, 2014.

[31] Viviana Castelli, Rodolfo Bertone, Pablo Thomas, and Alejandro Oliveros. A Requirements Engineering Process extended to Context Information Management. In *Proceedings of International Conference on Research Challenges in Information Science (RCIS 2011)*, pages 1–6. IEEE, 2011.

[32] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth Computer Science Technical Report TR2000-381, 2000.

[33] Betty H. C. Cheng and Joanne M. Atlee. Current and Future Research Directions in Requirements Engineering. In *Workshop on Design Requirements*, pages 11–43, 2009.

[34] Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Betty H. C. Cheng, editor, *Self-Adaptive Systems*, pages 1–26, Dagstuhl, Germany, 2009. Springer-Verlag Berlin Heidelberg.

[35] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences. In *Proceeding of annual SIGCHI Conference on Human Factors in Computing Systems*, pages 17–24. ACM Press, 2000.

[36] Bee Bee Chua, Dan Bernardo, and June Verner. Understanding the Use of Elicitation Approaches for Effective Requirements Gathering. In *Proceedings*

of International Conference on Software Engineering Advances (ICSEA 2010), pages 325–330. IEEE, 2010.

[37] Andreas Classen, Arnaud Hubaux, Franciscus Sanen, Eddy Truyen, Jorge Vallejos, Pascal Costanza, Wolfgang De Meuter, Patrick Heymans, and Wouter Joosen. Modelling Variability in Self-Adaptive Systems: Towards a Research Agenda. *Workshop on Modularization, Composition and Generative Techniques for Product-Line Engineering*, 1(2):19–26, 2008.

[38] IBM Corporation. An architectural blueprint for autonomic computing. *White paper*, Fourth Edition, 2006.

[39] IBM Corporation. Symptoms Reference specification, 2006.

[40] John W. Creswell. *Research Design - Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 2nd edition, 2008.

[41] Fabiano Dalpiaz. *Exploiting Contextual and Social Variability for Software Adaptation*. PhD thesis, University of Trento, Italy, 2011.

[42] Renata Paola Dameri and Camille Rosenthal-Sabroux, editors. *Smart City*. Progress in IS. Springer International Publishing, 2014.

[43] Marian Daun, Bastian Tenbergen, Jennifer Brings, and Thorsten Weyer. Documenting Assumptions about the Operational Context of Long - Living Collaborative Embedded Systems. In *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering*, pages 115–117, 2015.

[44] Marian Daun, Thorsten Weyer, and Klaus Pohl. Validating the Functional Design of Embedded Systems against Stakeholder Intentions. In *Proceedings of the International Model-Driven Engineering and Software Development (MODELSWARD 2014)*, pages 333–339, 2014.

[45] Marian Daun, Thorsten Weyer, and Klaus Pohl. Detecting and Correcting Outdated Requirements in Function-Centered Engineering of Embedded Systems. In *Proceedings of the International Working Conference on Requirements Engineering: Foundations for Software Quality*, pages 65–80, 2015.

[46] Alan M. Davis. : A New Development Approach. *IEEE Software*, 9(5):70–78, September 1992.

[47] Oscar Dieste and Natalia Juristo. Systematic review and aggregation of empirical studies on elicitation techniques. *Transactions on Software Engineering*, 37(2):283–304.

[48] Tore Dybåand Daniela S. Cruzes. Process Research in Requirements Elicitation. In *Proceedings of International Workshop on Empirical Requirements Engineering (EmpiRE 2013)*, pages 36–39, 2013.

[49] Steve Easterbrook. Handling Conflict Between Domain Descriptions With Computer-Supported Negotiation. *Knowledge Acquisition: An International Journal*, 3:255–289, 1991.

[50] Steve Easterbrook. Domain Modelling with Hierarchies of Alternative Viewpoints. In *Proceedings of International Symposium on Requirements Engineering*, pages 65–72. IEEE, 1993.

[51] Sahar Ebrahimi, Norha M Villegas, Hausi A. Müller, and Alex Thomo. SmarterDeals: A Context-aware Deal Recommendation System based on the SmarterContext Engine. In *Conf. of the Center for Advanced Studies on Collaborative Research*, pages 116–130. IBM Corp., 2012.

[52] Yönet a. Eracar and Mieczyslaw M. Kokar. Architecture for software that adapts to changes in requirements. *Journal of Systems and Software*, 50(3):209–219, 2000.

[53] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems. *Transactions on Software Engineering*, 39(11):1467–1493, 2013.

[54] Michael Fahrmair, Bernd Spanfelner, and Wassiou Sitou. Unwanted Behavior and its Impact on Adaptive Systems in Ubiquitous Computing. In *Workshop on Adaptivity and User Modeling in Interactive Systems*, Hildesheimer Informatik-Berichte, pages 36–41, 2006.

[55] Stephen Fickas and Martin S. Feather. Requirements Monitoring in Dynamic Environments. In *Proceedings of International Symposium on Requirements Engineering (RE 1995)*, pages 140–147, 1995.

[56] Anthony Finkelstein and Andrea Savigni. A Framework for Requirements Engineering for Context-Aware Services. In *Proceedings of International Workshop From Software Requirements to Architectures*, pages 2–7, 2001.

[57] George Forman and Martin Scholz. Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement. *Special Interest Group on Knowledge Discovery and Data Mining*, 12(1):49–57, 2010.

[58] Konstantinos G. Fouskas, Adamantia G. Pateli, Diomidis D. Spinellis, and Heli Virola. Applying Contextual Inquiry for Capturing End-Users Behaviour Requirements for Mobile Exhibition Services. In *M-Business*, volume 81, pages 1–23, 2002.

[59] Xavier Franch, Paul Grunbacher, Marc Oriol, Benedikt Burgstaller, Deepak Dhungana, Lidia López, Jordi Marco, and João Pimentel. Goal-Driven Adaptation of Service-Based Systems from Runtime Monitoring Data. *Proceedings of Annual Computer Software and Applications Conference Workshops*, pages 458–463, 2011.

[60] Rubén Fuentes-Fernández, Jorge J. Gómez-Sanz, and Juan Pavón. Understanding the human context in requirements elicitation. *Requirements Engineering*, 15(3):267–283, August 2009.

[61] Hans W. Gellersen, Albrecht Schmidt, and Michael Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. In *Mobile Networks and Applications 7*, pages 341–351. Kluwer Academic Publishers, 2002.

[62] Geri Georg, Gunter Mussbacher, Daniel Amyot, Dorina Petriu, Lucy Troup, Saul Lozano-fuentes, and Robert France. Synergy between Activity Theory and goal/scenario modeling for requirements elicitation, analysis, and evolution. *Information and Software Technology*, 59:109–135, 2015.

[63] Ravi Kumar Gullapalli, Chelliah Muthusamy, and A. Vinaya Babu. Data Mining in Adaptive Control of Distributed Computing System Performance. *Journal of Computer Trends and Technology*, 2(2):128–133, 2011.

[64] Emitza Guzman and Walid Maalej. Do Users Like this Feature? A Fine Grained Sentiment Analysis of App Reviews. In *Proceedings of the International Requirements Engineering Conference (RE 2014)*, pages 153–162. IEEE, 2014.

[65] Sara Hassan, Nelly Bencomo, and Rami Bahsoon. Minimizing Nasty Surprises with Better Informed Decision-Making in Self-Adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, pages 134–144, 2015.

[66] Jong-Yi Hong, Eui-Ho Suh, and Sung-Jin Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509–8522, May 2009.

[67] Jennifer Horkoff, Rick Salay, Marsha Chechik, and Alessio Di Sandro. Supporting Early Decision-Making in the Presence of Uncertainty. In *Proceedings of the International Requirements Engineering Conference (RE 2014)*, pages 33–42. IEEE, 2014.

[68] Burkhard Igel, Erik Kamsties, Fabian Kneer, Bernd Kolb, and Markus Voelter. Feedback-Aware Requirements Documents for Smart Devices. In *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2014)*, pages 119–134. Springer, 2014.

[69] Paola Inverardi and Marco Mori. Feature Oriented Evolutions for Context-Aware Adaptive Systems. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 93–97, Antwerp, Belgium, 2010. ACM.

[70] Paola Inverardi and Marco Mori. Requirements Models at Run-time to Support Consistent System Evolutions. In *Proceedings of International Workshop on Requirements@Run.Time*, pages 1–8. IEEE, 2011.

[71] Adam Jacobs. The Pathologies of Big Data. *Communications of the ACM*, 52(8):36–44, 2009.

[72] C'amara Javier, Gabriel A. Moreno, and David Garlan. Reasoning about Human Participation in Self-Adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, pages 146–156, 2015.

[73] Joachim Karlsson, Claes Wohlin, and Björn Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14-15):939–947, January 1998.

[74] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, January 2003.

[75] Karl E. Klein and Hans M. Wegmann. Significance of Circadian Rhythms in Aerospace Operations. Technical report, Neuilly-Sur-Seine: NATO-AGARD, AGARDograph No.247, 1980.

[76] Alessia Knauss. On the Usage of Context for Requirements Elicitation: End-User Involvement in IT-Ecosystems. In *Proceedings of International Requirements Engineering Conference (RE 2012), Doctoral Symposium*, pages 345 – 348. IEEE, 2012.

[77] Alessia Knauss and Daniela Damian. Requirements Elicitation Driven by End-Users. In *Proceedings of Requirements Engineering: Foundation for Software Quality (REFSQ 2012)*, 2012.

[78] Alessia Knauss, Daniela Damian, and Kurt Schneider. Eliciting Contextual Requirements at Design Time: A Case Study. In *Proceedings of International Workshop on Empirical Requirements Engineering (EmpiRE 2014)*, pages 56–63. IEEE, 2014.

[79] Sotiris B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, 31:249–268, 2007.

[80] Zeshan Kurd, Tim Kelly, and Jim Austin. Developing artificial neural networks for safety critical systems. *Neural Computing and Applications*, 16(1):11–19, 2007.

[81] Alexei Lapouchnian and John Mylopoulos. Modeling domain variability in requirements engineering with contexts. In *Proceedings of 28th International Conference on Conceptual Modeling*, pages 115 – 130, 2009.

[82] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10:311–341, 2005.

[83] Tong Li, Jennifer Horko, and John Mylopoulos. Integrating Security Patterns with Security Requirements Analysis Using Contextual Goal Models. In *The Practice of Enterprise Modeling*, pages 208–223. Springer Berlin Heidelberg, 2014.

[84] Walid Maalej and Dennis Pagano. On the Socialness of Software. In *Proceedings of the International Conference on Dependable, Autonomic and Secure Computing (DASC 2011)*, pages 864 – 871. IEEE, 2011.

[85] Martin Maguire. Context of Use within usability activities. *International Journal of Human-Computer Studies*, 55(4):453–483, October 2001.

[86] Neil Maiden. Exactly How Are Requirements Written? *IEEE Software*, 29(1):26–27, 2012.

[87] Neil Maiden, Alexis Gizikis, and Suzanne Robertson. Provoking Creativity : Imagine What Your Requirements Could Be Like. *IEEE Software*, 21(5):68 – 75, 2004.

[88] Neil Maiden and Gordon Rugg. ACRE: selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3):183–192, 1996.

[89] Neil Maiden, Norbert Seyff, Paul Grünbacher, Omo Otojare, and Karl Mitteregger. Making Mobile Requirements Engineering Tools Usable and Useful. In *Proceedings of International Requirements Engineering Conference (RE 2006)*, pages 29–38. IEEE, 2006.

[90] Jose Luis Mate and Andres Silva. *Requirements Engineering for Sociotechnical Systems*. 2005.

[91] Marina Mongiello, Patrizio Pelliccione, and Massimo Sciancalepore. AC-contract: Run-time verification of context-aware applications. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, pages 24–34, 2015.

[92] Hausi A. Müller and Norha M. Villegas. Runtime Evolution of Highly Dynamic Software. In *Evolving Software Systems*, pages 229–264. Springer, 2014.

[93] Linda Northrop, Richard P. Gabriel, Mark Klein, and Douglas Schmidt. *Ultra-Large-Scale Systems: The Software Challenge*. Software Engineering Institute, 2006.

[94] OAR Northwest. http://oarnorthwest.com/, last visit March 8th, 2015.

[95] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the International Conference on Software Engineering (ICSE 2000)*, pages 35–46. ACM Press, 2000.

[96] Marc Oriol, Nauman A. Qureshi, Xavier Franch, Anna Perini, and Jordi Marco. Requirements Monitoring for Adaptive Service-Based Applications. *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2012)*, pages 280–287, 2012.

[97] Shari Lawrence Pfleeger and Joanne M. Atlee. Capturing the Requirements. In *Software Engineering: Theory and Practice, Fourth Edition.* 2010.

[98] Colin Potts and Idris Hsi. Abstraction and context in requirements engineering: toward a synthesis. *Annals of Software Engineering*, 3:23–61, 1997.

[99] Nauman Qureshi and Anna Perini. Engineering Adaptive Requirements. In *Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009)*, pages 126–131. IEEE, 2009.

[100] Nauman A. Qureshi. *Requirements Engineering for Self-Adaptive Software: Bridging the Gap between Design-Time and Run-Time.* PhD thesis, University of Trento, Italy.

[101] Nauman A. Qureshi, Ivan J. Jureta, and Anna Perini. Requirements Engineering for Self-Adaptive Systems: Core Ontology and Problem Statement. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'11)*, pages 33–47, 2011.

[102] Nauman A. Qureshi and Anna Perini. Requirements Engineering for Adaptive Service Based Applications. In *Proceedings of International Requirements Engineering Conference (RE 2010)*, pages 108–111. IEEE, 2010.

[103] Nauman A. Qureshi, Anna Perini, Fondazione Bruno, Kessler Irst, Neil A. Ernst, and John Mylopoulos. Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems. In *Proceedings of International Workshop on Requirements@RunTime*, pages 9–16, 2010.

[104] Mona Rahimi, Mehdi Mirakhorli, and Jane Cleland-Huang. Automated Extraction and Visualization of Quality Concerns from Requirements Specifications.

In *Proceedings of the International Requirements Engineering Conference (RE 2014)*, pages 253–262. IEEE, 2014.

[105] Andres J. Ramirez, Erik M. Fredericks, Adam C. Jensen, and Betty H.C. Cheng. Automatically RELAXing a Goal Model to Cope with Uncertainty. In *Proceedings of Symposium on Search-Based Software Engineering*, pages 198–212. Springer, 2012.

[106] Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng. A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*, pages 99–108. IEEE, 2012.

[107] Andres J. Ramirez, Adam C. Jensen, Betty H. C. Cheng, and David B. Knoester. Automatically Exploring How Uncertainty Impacts Goal Satisfaction. In *Proceedings of International Conference on Automated Software Engineering (ASE 2011)*, pages 568 – 571. IEEE, 2011.

[108] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, December 2008.

[109] Rick Salay, Marsha Chechik, Jennifer Horkoff, and Alessio Di Sandro. Managing requirements uncertainty with partial models. *Requirements Engineering*, 18(2):107–128, 2013.

[110] Mazeiar Salehie and Ladan Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *Transactions on Autonomous and Adaptive Systems*, V(N):1–40.

[111] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying Monitoring and Switching Problems in Context. In *Proceedings of International Requirements Engineering Conference (RE 2007)*, pages 211–220. IEEE, 2007.

[112] Charles Samuels. Sleep, Recovery, and Performance: The New Frontier in High-Performance Athletics. *Physical medicine and rehabilitation clinics of North America*, 20(1):149–159, 2009.

[113] Peter Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-Aware Systems: A research agenda for RE for Self-adaptive Systems. In *Proceedings of International Requirements Engineering Conference (RE 2010)*, pages 95–103. IEEE, September 2010.

[114] Albrecht Schmidt. *Ubiquitous Computing – Computing in Context*. PhD thesis, Lancaster University, 2002.

[115] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to Context than Location. *Computers & Graphics*, 23(6):893–901, 1999.

[116] Kurt Schneider, Sebastian Meyer, Maximilian Peters, Felix Schliephacke, Jonas Mörschbach, and Lukas Aguirre. Feedback in Context: Supporting the Evolution of IT-Ecosystems. In *Proceedings of International Conference on Product Focused Software Process Improvement (PROFES 2010)*, pages 191–205. Springer, 2010.

[117] Norbert Seyff and Florian Graf. Mobile Discovery of Requirements for Context-Aware Systems. In *Conference on Requirements Engineering: Foundations for Software Quality (REFSQ 2008)*, pages 183–197, 2008.

[118] Norbert Seyff, Florian Graf, and Neil Maiden. Using Mobile RE Tools to Give End-Users their Own Voice. In *Proceedings of International Conference on Requirements Engineering*, pages 37–46, 2010.

[119] Norbert Seyff, Neil Maiden, Kristine Karlsen, James Lockerbie, Paul Grünbacher, Florian Graf, and Cornelius Ncube. Exploring how to use scenarios to discover requirements. *Requirements Engineering*, 14(2):91–111, 2009.

[120] Boris Shishkov and Marten Van Sinderen. From User Context States to Context-Aware Applications. In *Proceedings of International Conference on Enterprise Information Systems (ICEIS 2008)*, pages 225–239, 2008.

[121] Vítor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos. (Requirement) Evolution Requirements for Adaptive Systems. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*, pages 155–164. IEEE, 2012.

[122] Leif Singer, Olesia Brill, Sebastian Meyer, and Kurt Schneider. Utilizing Rule Deviations in IT Ecosystems for Implicit Requirements Elicitation. In *Proceedings of International Workshop on Managing Requirements Knowledge (MaRK 2009)*, pages 22 – 26. IEEE, 2009.

[123] Wassiou Sitou and Bernd Spanfelner. Towards Requirements Engineering for Context Adaptive Systems. In *Proceedings of Annual International Computer Software and Applications Conference*, volume 2, pages 593–600. IEEE, 2007.

[124] Slamet Sucipto and Romi S. Wahono. A Systematic Literature Review of Requirements Engineering for Self-Adaptive Systems. *Journal of Software Engineering*, 1(1):17–27, 2015.

[125] Alistair Sutcliffe, Stephen Fickas, and McKay Moore Sohlberg. Personal and Contextual Requirements Engineering. In *Proceedings of International Conference on Requirements Engineering (RE 2005)*, pages 19–28. IEEE, Aug. 2005.

[126] Alistair Sutcliffe, Stephen Fickas, and McKay Moore Sohlberg. PC-RE: a method for personal and contextual requirements engineering with some experience. *Requirements Engineering*, 11(3):157–173, 2006.

[127] Alistair Sutcliffe and Pete Sawyer. Requirements Elicitation: Towards the Unknown Unknowns. In *Proceedings of International Requirements Engineering Conference (RE 2013)*, pages 92–104, 2013.

[128] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Publishing, 2005.

[129] Le M. S. Tran and Fabio Massacci. An Approach for Decision Support on the Uncertainty in Feature Model Evolution. In *Proceedings of the International Conference on Requirements Engineering (RE 2014)*, pages 93–102. IEEE, 2014.

[130] Guido van der Zanden. Requirements Engineering for Context-Aware applications. In *Twente Student Conference on IT*, 2008.

[131] Emil Vassev and Mike Hinchey. Autonomy Requirements Engineering: A Case Study on the BepiColombo Mission. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 31–41, 2013.

[132] Norha M. Villegas. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems.* Phd thesis, University of Victoria, Victoria BC, Canada, 2013.

[133] Norha M. Villegas and Hausi A. Müller. Managing Dynamic Context to Optimize Smart Interactions and Services. In M. Chignell, editor, *The Smart Internet*, pages 289–318. Springer Berlin Heidelberg, 2010.

[134] Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, Laurence Duchien, and Ruby Casallas. DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. *Software Engineering for Self-Adaptive Systems II - Lecure Notes in Computer Science*, 7475:265–293, 2013.

[135] Kristopher Welsh and Peter Sawyer. Understanding the Scope of Uncertainty in Dynamically Adaptive Systems. In *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010)*, pages 2–16. Springer, 2010.

[136] Danny Weyns and Radu Calinescu. Tele Assistance : A Self-Adaptive Service-Based System Examplar. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, pages 88–92, 2015.

[137] Danny Weyns, M. Usman Iftikhar, Sam Malek, and Jesper Andersson. Claims and Supporting Evidence for Self-Adaptive Systems: A Literature Study. In *Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*, pages 89–98, 2012.

[138] John Whittle, Peter Sawyer, and Nelly Bencomo. RELAX: a Language to Address Uncertainty in Self-Adaptive Systems Requirements. *Requirement Engineering*, 15 (2):177–196, 2010.

[139] Andreas Zimmermann, Andreas Lorenz, Reinhard Oppermann, and Sankt Augustin. An Operational Definition of Context. In *CONTEXT*, pages 558–571. Springer-Verlag Berlin Heidelberg, 2007.

[140] Didar Zowghi and Chad Coulin. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In *Engineering and Managing Software Requirements*, pages 19–46. 2005.