A Framework for Robust Control of Uncertainty in Self-Adaptive Software Connectors

Pooyan Jamshidi, M.Sc., B.Sc.

A dissertation submitted in partial fulfilment

of the requirement for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University

Faculty of Engineering and Computing

School of Computing

Supervisor: Dr. Claus Pahl

September 2014

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: ———————————

Pooyan Jamshidi

Student ID: 10112308

Date: 18/09/2014

Contents

List of Abbreviations

Continuous Stochastic Logic: CSL, 31
Continuous-Time Markov Chain: CTMC, 59

Discrete-Time Markov Chain: DTMC, 59

Event-Condition-Action: ECA, 129

Footprint of Uncertainty: FOU, 34
Fuzzy Logic Systems: FLS, 22
Fuzzy Set: FS, 33

Hidden Markov Models: HMM, 59

Interval Type-2: IT2, 33

Linear Temporal Logic: LTL, 129
Lower Membership Function: LMF, 34

Membership Function: MF, 33

Non-Functional Requirements: NFR, 6

Probabilistic Computation Tree Logic: PCTL, 31

Quality of Service: QoS, 50

Robust Control of Uncertainty: RCU, 11

Self-Adaptive Software: SAS, 126
Service Level Agreement: SLA, 7
State-Space Model: SSM, 59

Type-1: T1, 33
Type-2: T2, 33

Upper Membership Function: UMF, 34

List of Figures

List of Tables

A Framework for Robust Control of Uncertainty in Self-Adaptive Software Connectors

Pooyan Jamshidi

Abstract

Context and motivations. The desired behavior of a system in ubiquitous environments considers not only its correct functionality, but also the satisfaction of its non-functional properties, i.e., its quality of service. Given the heterogeneity and dynamism characterizing the ubiquitous environments and the need for continuous satisfaction of non-functional properties, self-adaptive solutions appear to be an appropriate approach to achieve interoperability. In this work, self-adaptation is adopted to enable software connectors to adapt the interaction protocols run by the connected components to let them communicate in a timely manner and with the required level of quality. However, this self-adaptation should be dependable, reliable and resilient to be adopted in dynamic, unpredictable environments with different sources of uncertainty. The majority of current approaches for the construction of self-adaptive software ignore the uncertainty underlying non-functional requirement verification and adaptation reasoning. Consequently, these approaches jeopardize system reliability and hinder the adoption of self-adaptive software in areas where dependability is of utmost importance.

Objective. The main objective of this research is to properly handle the uncertainties in the non-functional requirement verification and the adaptation reasoning part of the self-adaptive feedback control loop of software connectors. This will enable a robust and runtime efficient adaptation in software connectors and make them reliable for usage in uncertain environments.

Method. In the context of this thesis, a framework has been developed with the following functionalities: 1) Robust control of uncertainty in runtime requirement verification. The main activity in runtime verification is fine-tuning of the models that are adopted for runtime reasoning. The proposed stochastic approach is able to update the unknown parameters of the models at runtime even in the presence of incomplete and noisy observations. 2) Robust control of uncertainty in adaptation reasoning. A general methodology based on type-2 fuzzy logic has been introduced for the control of adaptation decision-making that adjusts the configuration of component connectors to the appropriate mode. The methodology enables a systematic development of fuzzy logic controllers that can derive the right mode for connectors even in the presence of measurement inaccuracy and adaptation policy conflicts.

Results. The proposed model evolution mechanism is empirically evaluated, showing a significant *precision* of parameter estimation with an acceptable *overhead* at runtime. In addition, the fuzzy based controller, generated by the methodology, has been shown to be *robust* against uncertainties in the input data, efficient in terms of *runtime overhead* even in *large-scale* knowledge bases and *stable* in terms of control theory properties. We also demonstrate the applicability of the developed framework in a *real-world* domain.

Thesis statement. We enable reliable and dependable self-adaptations of component connectors in unreliable environments with imperfect monitoring facilities and conflicting user opinions about adaptation policies by developing a framework which comprises: (a) mechanisms for robust model evolution, (b) a method for adaptation reasoning, and (c) tool support that allows an end-to-end application of the developed techniques in real-world domains.

Keywords: *Uncertainty, Self-Adaptive Software, Software Connector, Models at Runtime, Fuzzy Logic System, Type-2 Fuzzy Logic Control, Mode-based Reconfiguration*.

Acknowledgement

The completion of this thesis has been made possible through the support and encouragement of several people and I wish to acknowledge their contribution. In particular, I would like to thank my supervisor, Claus Pahl, for his constant support and inspiration, and for all the time and effort he has devoted towards my research. Claus has been my principal resource and guide throughout my PhD. He is a remarkable advisor for providing the right amount of guidance at the right time. He has entrusted me with independence and responsibility when I had learned the ropes. I could not have hoped for a better supervisor. I am also grateful to my thesis external and internal examiners, Pete Sawyer and Kevin Casey for their constructive comments. David Sinclair, my transfer examiner, also provided a number of useful comments that have helped to shape the direction of my thesis research right in the middle of this journey.

Over the years, I have been fortunate to receive useful input on my research from great people. I would like to particularly acknowledge the contributions of Aakash Ahmad, Yasunari Inamura and Colette Real. Aakash has been a continual source of helpful comments, guidance, and encouragement throughout my research. Yasunari helped me a lot when I was struggling in the implementation of model calibration approach in this thesis. Colette assisted me by proofreading the final draft of this thesis. I am especially indebted to Saeed Mansour, Fereidoon Shams and Mohsen Sharifi for introducing me into the field of system and software engineering research and providing me the opportunity to participate in practical research already as a master student.

I have been fortunate to have great friends during my PhD: Irina Roznovat, Kosala Bandara, Wang Ming Xue, Muhammad Javed, Yalemisew Abgaz, Paul Clarke, Fattah Alizadeh, Alireza Dehghani, Mohsen Moheimany, Marija Bezbradica, Na Li, John Pendlebury, Huanhuan Xiong, Frank Fowley, Graham Hunt, Lei Xu and Sergej Svorobej. Thanks for all your help and for making my PhD so much fun! A big thank you to Patricia Lacey, Clare McInerney, Susan Mitchell, Patsy Finn and Dara O'Connor for their invaluable administrative support at DCU and Lero.

I am very obliged to all the organizations that have provided support for this work. I would like to particularly express my gratitude to Lero- the Irish Software Engineering Research Centre. The steady support provided by the Lero over the three years of my PhD has enabled me to explore the topic of self-adaptive software. I also would like to acknowledge the support of IC4 – the Irish Centre for Cloud Computing and Commerce over the last year. The consistent support provided by IC4 and the centre director, Tony McEnroy, has enabled me to apply my ideas to cloud computing, helping to significantly improve the maturity of this thesis.

I will be forever grateful for the lifelong emotional and practical support that my family has provided me throughout my education. Mom, Dad, and Parnian: I can never thank you enough for all you have done for me!

Most of all, I am thankful for Hoorieh, my wife, who accompanied me to Dublin, enduring some hard times with me while we adjusted to a new life. It has been a long yet crazy journey together and as I have told you many times, this accomplishment is also yours. I love you.

Dedication

To Mehri, Ahmad, Parnian, Hoorieh, Bahman and Running!

Propositions

Belonging to the PhD dissertation

A Framework for Robust Control of Uncertainty in Self-Adaptive Software Connectors

1. Software connectors play an important role in increasing the interoperability of software. Software connectors coordinating heterogeneous components support interoperability in ubiquitous environments.
2. The desired behavior of a system in ubiquitous environments considers not only its correct functionality, but also the satisfaction of non-functional properties, i.e., its quality of service.
3. Given the heterogeneity and dynamism characterizing the ubiquitous environments and the need for continuous satisfaction of non-functional properties, self-adaptive solutions appear to be an appropriate approach to achieve interoperability.
4. Self-adaptive connectors adapt the interaction protocols run by connected components to let them communicate timely and with the required level of quality.
5. However, such self-adaptation should be dependable, reliable and resilient to be adopted in such a dynamic environments with different sources of uncertainty.
6. To achieve dependable self-adaptive software connectors, mechanisms to enable the self-adaptation for component connectors should be robust against different sources of uncertainty.
7. For quantitative verifications of non-functional properties, it is required to consider parametric analytical models that can be calibrated at runtime to accurately measure such properties. However, the challenge is that how accurate and trustworthy model calibration can perform given that the input measurement data typically are not complete and contain noise (Chapter 4).
8. Having quantitatively verified non-functional properties and detected a violation, an adaptation is required to change the interaction protocol to let the components communicate with the desired level of quality. However, this adaptation should also be reliable given the input information as well as the adaptation policies may contain uncertainties (Chapter 5).
9. Assuming that dependable model calibration and adaptation reasoning is in place, their integration in the MAPE-K self-adaptation loop to ensure the reliability of the self-adaptation of component connectors in real-world unreliable environment needs to be considered (Chapter 7).

Chapter 1

1. Introduction

"Never promise more than you can perform" – Publilius Syrus (85 - 43 BC).

Contents

1.1. Chapter Overview

Recently, software has become one of the key influential entities of modern society. Consequently, the expectations people place on software systems have quickly changed, even more recently as software systems have become essential for living. With the rise of a new computing paradigm (such as service-oriented and cloud computing), in which the main principle is to move towards independent management of computing entities, applications use and integrate functionality from third-party, potentially distributed services, implemented in different environments and running on different platforms.

Therefore, interoperability and dependability have become fundamental requirements for building software-intensive systems. This raises questions of not only how we coordinate different pieces of software and how we can reason about the properties of the subsequent systems, but also *how we can robustly adapt the coordination architecture at runtime due to the intrinsic dynamics and uncertainties of the environment*. The *robust handling of uncertainty in self-adaptive software connectors* is the topic of this thesis. The term robustness here means the persistence of a system's characteristic behavior under perturbations or unusual conditions of uncertainty. More concretely, we aim at developing mechanisms for enabling a *reliable* (i.e., *dependable*) adaptation of the coordination architecture even if its assumptions that have been made at design-time are somewhat violated by the situations at runtime. In other words, we enable *resilient* self-adaptive software connectors that perform well in uncertain environments. Throughout this thesis, we may use the three key terms in the last two sentences (i.e., reliability, dependability and resiliency) interchangeably, but in the context of this thesis, we mean that we intend to develop a safe and reliable self-adaptation for software connectors in the presence of uncertainty.

The rest of this chapter describes the motivations for this work and provides an overview of the proposed solution. We first give a broad overview of the research context including component-based systems, component connectors as well as self-adaptive software (Section 1.2) and motivate the need for a dependable self-adaptive software connector (Section 1.3). We then elaborate on the problem that this work aims to address by deliberating the hypotheses and research questions (i.e., the *problem space* in Section 1.4). Afterwards, the basic elements of the proposed solution and specific contribution of this thesis are discussed (i.e., the *solution space* in Section 1.5). The details of the research methodology that we have followed throughout this work are then presented (Section 1.6). The thesis of this dissertation and research claims are then stated (Section 1.7). A mapping of the related publications to the individual chapters in the thesis is also provided (Section 1.8). Finally, the structure of the remainder of this document, explaining each chapter's relevance to the stated thesis is presented (Section 0).

1.2. Research Context

In this section, before examining the research motivations of this thesis, we first introduce the most related aspects, which intersect to determine the scope of this research as illustrated in Figure 1.1.

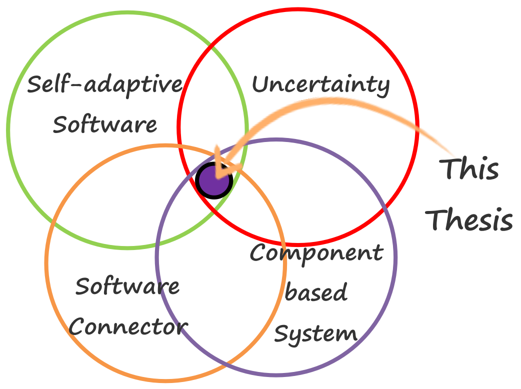


Figure 1.1. Scope of this thesis.

1.2.1. Component-Based Systems

Software intensive systems that are built according to the component-based paradigm are called component-based software systems. A component-based system is described as a composition of components that interact with each other to offer original functionalities resulting from the composition of individual component functionalities. Composition of components not only defines the rules according to which the individual components can interact, but it also describes the interaction between the composed components.

The traditional concept of separation between computation and interaction (Gelernter & Carriero, 1992) is now a necessity for modern large-scale software intensive systems. The separation between computation and interaction becomes more prominent in the component-based paradigm, where reusability, evolution, maintainability and heterogeneity are key principles. In order to realize this separation, the notion of a *connector* has been coined to act as an interaction media. Unfortunately, the majority of programming languages for component-based systems do not provide any mechanism to express the connectors explicitly. This forces the interaction logic to be programmed within components. As a result, the interaction logic becomes entirely hidden inside individual components. This limits the usage of components to the very specific interaction protocol that it contains. Ultimately, this limits the reusability and dynamic interchangeability of software components.

In this thesis, we consider interaction between components as a first-class entity. Here, rather than focusing on the self-adaptation of component-based systems, we focus on the self-adaptation of interaction protocols that describe how the individual components interact and evolve over time to accommodate the changes in the surrounding environment. Interaction protocols are promoted forming a special class of components, the so-called *component connectors*. In this thesis, we intend to build on component connectors by promoting the notion of self-adaptation and form a special class of component connectors, which we call *self-adaptive component connectors*.

1.2.2. Component Connectors

Connectors participate in the design of component-based systems by defining the interaction protocols for composing individual components. Connectors prescribe how the constituting components of a system connect and interact with each other. As opposed to the components, which provide system-specific functionality, connectors have no responsibility for the computation carried out by the overall system. This means that connectors define interaction protocols between components. To form a coherent system that realizes its requirements, connectors are responsible for the coordination of activities realized by components to ensure their correct interaction.

1.2.2.1. A tangible example of component connectors

In order to investigate the details of component connectors, we introduce this notion by a tangible example. Let us use two individual and physical components, a camera and a mini-display. We consider these components as black boxes, with their internal behavior hidden, and they only expose an interface for interaction purposes. The camera interface has a single output port to which it writes a captured data stream. The display module also has an interface to the input data stream to be displayed. The objective is to construct a simple component-based system that enables the user to capture a scene and have the photo of the scene shown on the display. Let us imagine a scenario where there is a mismatch between the rate at which the photo on the display can be refreshed and the rate at which the camera writes the stream to the output port. More specifically, the camera is able to capture the scenes successively one after another at a higher rate than the display allows the photos to be updated on its screen. The situation in this scenario means that we cannot simply *connect* the output port of the camera with the input port of the display. What would happen when the user starts capturing the scene at a pace that exceeds the rate at which the display can show the respective captured photo? The following potential scenarios can be considered:

1. The photographer is forced to wait for when its output port is not busy to capture another photo. This means that the output port of the camera is synchronized with the input port of the display.
2. The extra data stream is buffered and can be displayed in order as they have queued.
3. The photos that are captured are disregarded while the display is busy showing another photo.
4. The second and the third scenarios are combined by offering a limited buffer where a number of photos can be buffered while the display is busy. Once the buffer is full, extra photos are disregarded.
5. The forth scenario is extended by estimating the rate of photo capturing and adapt the buffer by the objective of scaling out the size of the buffer in order not to lose any photos and scaling in the size of the buffer by releasing extra resources.

Let us consider scenario 3 and ignore the extra data stream captured by camera when the display is busy and not ready to show new photos. When someone follows this scenario to construct a composed system, the result would be a system that allows the photographer to capture photos and have the photo shown in the display, but with the issue that if the user captures too quickly, he needs to capture repeatedly until the photo is shown in the display. This example shows a common scenario faced by component-based system designers in which constructing a desirable system is not possible by only wiring the input-output ports together. There is a need for a piece of software, usually called *glue code*, to *coordinate* the interaction of individual components.

Coordination languages (Papadopoulos & Arbab, 1998) are a class of modeling as well as programming languages that offer a solution for the problem of specifying and managing the glue code. More particularly, coordination languages offer mechanisms for composing and controlling systems made of independent and possibly heterogeneous components. There are two classes of coordination languages (Papadopoulos & Arbab, 1998): endogenous and exogenous. In endogenous coordination languages, the coordination can be realized by incorporating the interaction logic with computation. The component-based systems that use endogenous languages for coordination utilize the provided primitives for defining the interaction inside the components. This intermixes the coordination with computation inside components and leads to an implicit coordination logic. In contrast, exogenous coordination languages enable the interactions outside the components as separate entities. This makes the role of coordination explicit in component-based systems and enhances the reusability of individual components.

In the example, we exemplified a connector to enable the interactions between the camera and the display according to the exogenous class of coordination. In exogenous coordination, connectors are separated from components and we can deal with them as a first class entity. The connector acts as a mediator between the camera and the display to enforce the behavior of scenarios that we want for the composed system. This solution retains both the camera and the display independent, reusable, and interchangeable for different contexts. If we need to enforce a new kind of scenario, we could still use the same components, but design a different connector for realizing such behavior. Note that neither camera nor display are aware of the presence of each other. However, the display here dictates the rate at which the resulting composed system is able to display the photos.

1.2.2.2. The language aspect of component connectors

In this thesis, we explicitly work with component connectors designed in the coordination model Reo (Arbab, 2004). Component connectors built using Reo are composed out of primitive channels, with specific behavior, that can be plugged together with the help of nodes. This resembles the arcs and nodes of general graphs in graph theory. The channel based compositions control the dataflow between the components they are coordinating by enforcing well-defined communication protocols among them. This coordination behavior involves a number of different semantics such as (a)synchronous communication, buffering, data manipulation, context dependent behavior and mobility. The Reo coordination language is based on a formal foundation and promotes loose coupling, distribution, mobility, exogenous coordination and dynamic reconfigurability (Arbab, 2004). The formal basis of Reo guarantees verification of quality of service properties and well-defined execution semantics for component compositions. There are also some supporting tools associated with Reo for modeling component connectors, simulating connector behavior, providing formal operational semantic languages and facilities to derive analytical models, which we employ in this thesis.

1.2.3. Self-Adaptive Software

Currently software facilitates many human activities in modern society. With this ubiquitous usage and the expectations that users have, new directions and perspectives have recently been envisaged.

In traditional software development, one of the key concerns is a meticulous requirement analysis in order to avoid costly changes in later stages of the software development process. The situation in recent software engineering processes is different. Software systems are constructed by composing independently developed components that might be offered as third party services. Deployment configuration can be changed quickly thanks to the flexibility of infrastructure in the cloud. Because of the pervasiveness of mobile devices, the use of software becomes ubiquitous and integrated with everyday life in a continuous fashion.

Modern software-intensive systems usually interact with an environment that is not under the control of software itself. Because of the unpredictable occurrence of changes in the environment, a system may not be able to meet the desired requirements. Consequently, software systems need to self-adapt themselves to the occurrence of changes with limited or without any kind of human interventions. Additionally, because software systems are continuously running in their desired environments, they cannot be simply shut down in order to perform the required changes. Therefore, there is a need for a special class of software systems that, while running, is required to recognize the occurrence of changes, analyze the changes and reason about possible reactions to them in a self-managed manner. Systems that fit into this class are called *self-adaptive systems* (SAS).

A number of relevant changes in the environment that affect software systems are:

1. Changes in the components of the systems that are not the core assets of the system itself and are managed by third parties. This set of components is composed in the system under specific considerations and when they change, it causes unexpected behavior of the composed system.
2. Changes in the usage profile of the system's functionality. Users of user-intensive software systems may change their behavior over time by overloading the system at specific time points and cause the system to response slowly.
3. Changes in the deployment infrastructure of the system. For example, the changes for resources that are available for computational purposes may cause the system to violate certain quality of service requirements.

There are also other sources of changes for software-intensive systems comprising the changes in system requirements, which may affect the software system. However, in this thesis, we assume the system requirements are stable over time and we primarily focus on the changes in the *environment* in which software is embedded. The changes in the environment of software cause some non-functional requirements (NFR) such as performance or reliability to be violated. However, users of the systems require a continuous satisfaction of the requirements despite the changes. Most non-functional requirements correspond to quantitative properties (Marta Kwiatkowska, 2007). A convenient adaptation of software can be triggered whenever a quantitative property corresponding to a requirement is violated.

1.3. Research Motivation

In traditional software development, the development processes mostly concentrated on how to carefully analyze the requirements in early phases and avoid costly changes in the latter phases of the development process (L Baresi, 2006). Therefore, in the research context, developing methods and techniques to capture requirements and avoiding changes was one of the central themes for a long period of time, when organizations were centralized and development environments and deployment infrastructure were mostly stable.

However, interestingly, almost none of these assumptions are still valid (Luciano Baresi & Ghezzi, 2010; A Filieri, Tamburrelli, & Ghezzi, 2013). Software development has now become decentralized. Software applications are constructed by composing independent components potentially developed and operated by third parties. The coordination between components and binding to their implementations is delayed until runtime. Infrastructure is often provisioned in the cloud and may change quickly. Accessibility devices are ubiquitous in everyday life, providing continuous interaction with billions of different users through social networks. Communication networks are pervasive and heavily shape software execution.

Today software systems must be designed for change (Luciano Baresi & Ghezzi, 2010) and in the future more software will be required to continuously adapt in response to unpredictable changes in its environment and objectives (Lemos, Giese, & Müller, 2013). In particular, self-adaptation is a key solution to deal with the issues of modern software development (A Filieri et al., 2013):

- *Instability* of requirements, as a consequence of the volatility of user needs
- *Uncertainty* in the environment in which software operates and in the accuracy of design-time parameters
- *Variability* in the behavior of third party components, infrastructure and users.

Another major difference between traditional concerns in software engineering and its current progress is the role of non-functional requirements, such as performance, energy consumption and reliability. Users require the continuous assurance of service level agreements (SLAs), regardless of uncertainties. This requirement has necessitated an important technique typically used in self-adaptive paradigm nowadays (Calinescu, Ghezzi, Kwiatkowska, & Mirandola, 2012). Quantifiable non-functional requirements enable automated verification of specific quantitative properties (Marta Kwiatkowska, 2007). In other words, the realization of this technique enables continuous verification of important properties of software in order to trigger an appropriate adaptation action whenever a requirement is violated.

Self-adaptation is an appropriate approach to deal with the changing dynamics in the surrounding environment of software systems. As in biological systems (Kitano, 2004), when facing external (or internal) perturbations, a self-adaptive software system modifies itself in response to changes in the environment (or requirements). Even though self-adaptive software is beneficial in many application domains, it is not a widely adopted solution (Lemos et al., 2013). It is regarded as a non-dependable solution, which is subject to uncertainty (Lemos et al., 2013). The research community has managed to address the complexities in building self-adaptive software systems (Lemos et al., 2013). However, as reported by other researchers (Esfahani, Kouroshfar, & Malek, 2011; Esfahani & Malek, 2013; Lemos et al., 2013), there is a serious lack of applicable techniques for handling uncertainty in the context of self-adaptive software.

In the field of software engineering in general, the phenomena of uncertainty is considered as a second-class citizen (David Garlan, 2010). Although it is conceivable to decrease the degree of uncertainty, it is not possible to fully eliminate the effect of uncertainty in both real-world physical systems (J. M. Aughenbaugh & Paredis, 2006) and software-intensive systems (David Garlan, 2010). Self-adaptive software is not an exception and uncertainty plays a major role in this area (Esfahani & Malek, 2013). Uncertainty is present in every facet of adaptation, but to varying degrees. As representatives of uncertainty sources, one can refer to the following items:

1. *Uncertainty in monitored parameters of the system*. Sensors employed for monitoring the environment are not usually free of noise. As a result, the monitoring data are rarely a single crisp value. However, they mostly correspond to a distribution of values obtained over an observation period.
2. *Uncertainty in analytical models at runtime*. Analytical models for evaluating system-level quality attributes make simplifying assumptions, which obviously introduces some uncertainty for reasoning purposes. This kinds of analytical models provide only acceptable estimates of the

system quality and since they model based on an underlying theory, they ignore unrelated aspects of the system at a time.

3. *Uncertainty in user preferences*. User preferences for non-functional requirements are imprecise. When users specify their preferences for formulating the utility functions measuring the overall quality of the system, they make subjective decisions. This makes the analysis based on them error prone.

The uncertainty that manifests itself in the aspects that we mentioned above poses critical risks to the adaptive software. Note that we only mentioned a few sources of uncertainty among the several to highlight the problem and motivate the research. We have provided a more detailed discussion of the sources of uncertainty in self-adaptive software in Chapter 3 (mainly motivated by the work of Esfahani et al. (Esfahani & Malek, 2013)). These sources of uncertainty fall into two diverse categories. The first category is associated with the environment surrounding the software. The environment in which software is embedded produces different noises, which is called *external uncertainty* (Esfahani et al., 2011). The impact of adaptation decisions (e.g. replacing a component or reconfiguring a connector) on system quality properties (e.g. response time) cannot be measured precisely at design-time. As a result, this produces a distinct source of uncertainty, which is categorized as *internal uncertainty* (Esfahani et al., 2011). The focus of this thesis is to find a solution for addressing the challenges posed only by external uncertainty.

Some uncertainty is because of the *lack of knowledge* while some other is because of the *variation* in adaptation parameters. Therefore, techniques that are used to mitigate one type of uncertainty are different from the other types. This distinction is related to the location of uncertainty, i.e., the user or the system itself (J. Aughenbaugh, 2006). In other words, variability is the uncertainty inherent in the system under study, while the lack of knowledge is associated with uncertainty of the human being.

The software engineering research community has made progress towards addressing the complexities involved in the construction of self-adaptive software (Lemos et al., 2013). However, despite the fact that uncertainty is predominant in self-adaptive software systems, as reported by a community wide roadmap (Lemos et al., 2013) and reviews of uncertainty handling techniques (Esfahani & Malek, 2013; A. J. Ramirez, Jensen, & Cheng, 2012), there is still a lack of methods and techniques for handling uncertainty in self-adaptive software. These seminal references imply that the issues related to uncertainties are treated in an ad-hoc fashion. For example, (Esfahani & Malek, 2013) hypothesize that this might be related to a lack of understanding and common knowledge about different sources of uncertainties in self-adaptive software, due to the diverse characteristics of each source. In the self-adaptive software community, only a handful of researchers have proposed to address uncertainty issues related to different aspects of software. The main aspects that have been addressed so far are related to 1) requirements specification (Luciano Baresi, Pasquale, & Spoletini, 2010; Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2009). 2) internal quality objectives (S. Cheng & Garlan, 2007; Esfahani et al., 2011; Q.-L. Yang et al., 2013). 3) external environments (Calinescu & Kwiatkowska, 2009; Chan, 2008; Cooray, Malek, Roshandel, & Kilgore, 2010; Epifani, Ghezzi, Mirandola, & Tamburrelli, 2009; Esfahani, Elkhodary, & Malek, 2013; Gmach, Krompass, Scholz, Wimmer, & Kemper, 2008).

All of the mentioned sources of uncertainty challenge the confidence with which decisions are made during the adaptation process. In this thesis, we identify different sources of uncertainty in the self-adaptive loop of component connectors and treat them explicitly in the loop in order to enhance the dependability of self-adaptive component connectors. Therefore, the key objective in this thesis is to robustly handle uncertainty in the self-adaptation of component connectors. Although we constrain the scope of this project to component connectors, in general, the developed techniques can be applied for self-adaptive software systems after some customizations.

1.4. Research Problem

Based on the identified research gap to handle uncertainty in the self-adaptation process of component connectors, in this section, we outline research challenges that we address in this thesis. The primary objective of this thesis is to enable robust control of uncertainty in self-adaptive component connectors to increase the dependability in component-based software systems. The main reason for choosing software connectors in this thesis is their ever-increasing importance to interconnect heterogeneous components in application domains like cloud, where interoperability with an acceptable performance is essential. In this section, we outline the central hypothesis and research questions.

1.4.1. Central Hypothesis

We outline the central hypothesis for this thesis as:

The application of parameter estimation for calibrating models for non-functional requirement verification, in the presence of imprecise monitoring data and fuzzy logic in adaptation reasoning, and the integration of the two in the self-adaptation process, enable component connectors to become robust against uncertainty in the surrounding environment.

We propose that in order to robustly control the uncertainty sources in the self-adaptation process of component connectors, we need to employ appropriate analytical techniques from probability theory for adaptation reasoning in the presence of imprecise (or noisy) monitoring data. By adaptation reasoning here we mean decision making about the changes in the architecture at runtime and this obviously requires adjusting some parameters in the model corresponding to the system architecture. Additionally, we also propose to utilize proper fuzzy reasoning technique in order to plan adaptation in the presence of uncertain measurements. An analysis of the central hypothesis suggests that the research problem that we aim to address in this thesis can be divided:

- How to calibrate analytical models that we employ at runtime for adaptation reasoning of component connectors in the presence of imperfect observations? (see RQ1)
- How to plan the appropriate configuration for component connectors in the presence of imprecise measurements and conflicting objectives? (see RQ2)

1.4.2. Research Questions

In this section, we outline the primary research questions that we aim to address in this thesis. Each of the research questions outlines a key challenge that we identified for this research and an individual aspects of the proposed solution. We validate the proposed solution by evaluating the degree to which each question has been addressed in the proposed solution (Chapter 7).

Research Question 1 (RQ1). How can we estimate the parameters of (i.e., calibrate) the analytical models at runtime that we employ for non-functional requirement verification of component connectors in the presence of noisy monitoring data?

The primary objective of this research question is the development of an appropriate estimation technique based on probability theory for robust model calibration at runtime. Note that the runtime data that has been observed by monitoring facilities and probes contain dynamic noise. However, all existing approaches for model calibration assume that the monitoring data is complete and noise free. As a result, the verification of non-functional requirements based on this assumption is error prone. This question allows us to evaluate the proposed model calibration approach in handling dynamic noises in monitoring data.

Research Question 2 (RQ2). How can we reason about adaptation and derive appropriate configuration for component connectors at runtime in the presence of noisy measurements and imprecise objectives?

The primary objective of this research question is the development of an appropriate adaptation reasoning technique based on fuzzy theory for robust control of uncertainty in reasoning. Note that for reasoning we need to feed measurement data such as workload or response time for reasoning. The reasoning process for deriving an appropriate configuration that is optimal based on the current situation of the connector and its surrounding environment is based on the preferences of stakeholders in terms of adaptation policies. The policies are specified based on imprecise terms such as "if response time is *high* then …". Different stakeholders with different opinions can specify these policies and they might be conflicting with each other. This question allows us to evaluate the proposed adaptation reasoning to derive appropriate configurations in the presence of imprecise and conflicting objectives.

Research Question 3 (RQ3). How well can our approach for model calibration and adaptation reasoning in the feedback control loop ensure the reliability of the self-adaptation of component connectors in a real-world unreliable environment?

The primary objective of this research question is the integration of both robust model calibration and robust adaptation reasoning in the self-adaptation loop of component connectors. Even though we apply model calibration that can handle noisy data, the analytical models themselves are an abstraction of the component connectors. These abstractions as mentioned in (Esfahani & Malek, 2013) are a source of uncertainty in self-adaptive software. This question allows us to evaluate the integration of model calibration with adaptation reasoning to provide an end-to-end mechanism for guaranteeing uncertainty in the self-adaptation process of component connectors.

1.5. Proposed Solution and Contributions

Based on the identified research challenges, an overview of the proposed contribution as an integration of robust model calibration and robust adaptation reasoning is illustrated in Figure 1.2. In contrast to limitations identified in earlier sections, the solution aims to control uncertainty in the self-adaptation process of component connectors to enhance dependability of a component-based system, which consumes such connectors for interaction.

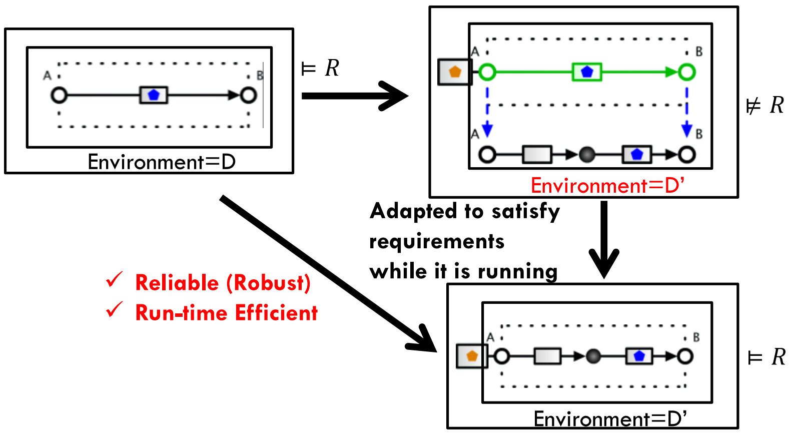


Figure 1.2. An overview of self-adaptive software connectors.

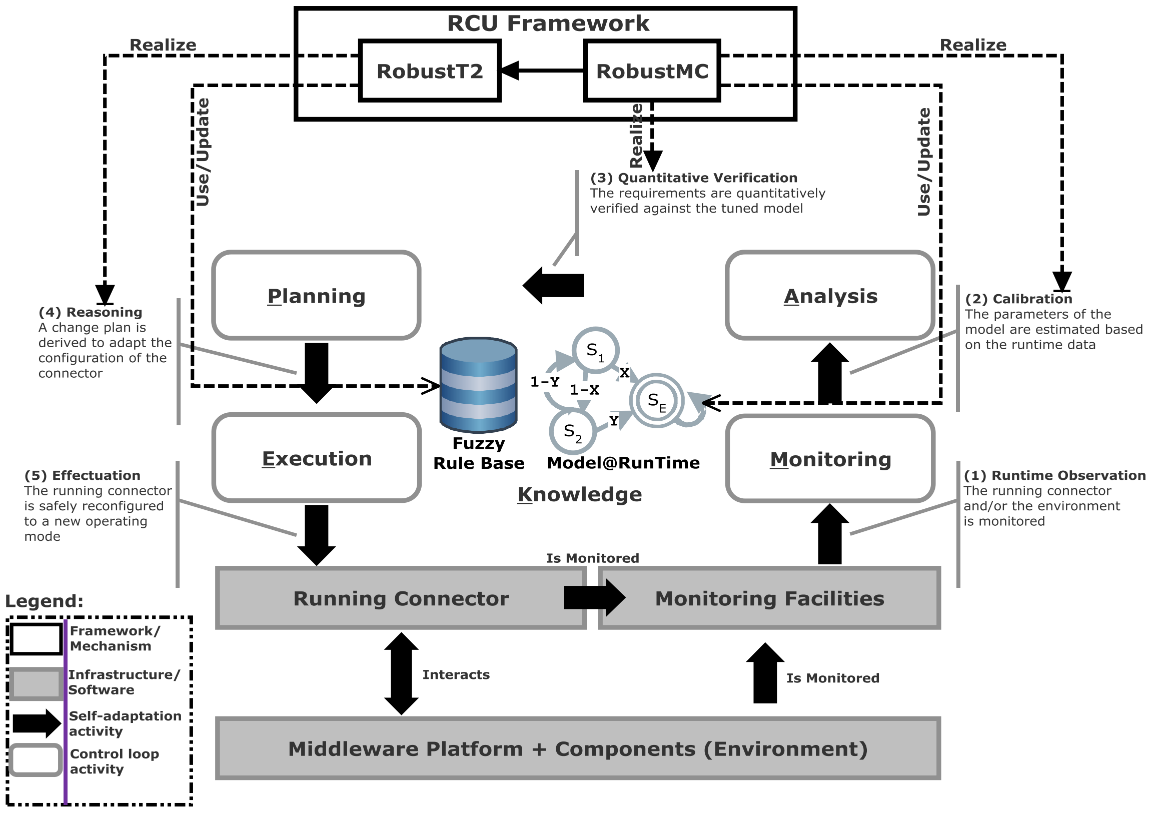
In the context of this thesis, we deal with two types of uncertainty. We provide a solution for robust handling of uncertainty for monitoring and planning in the self-adaptation loop. We utilize probability for updating models that are being kept at runtime for analysis purposes. The techniques that we developed for updating models have the capability to deal with incomplete and noisy data. Probability theory is an obvious choice for dealing with uncertainty, which are related to variability in data rather than lack of knowledge. On the other hand, we adopt fuzzy logic for adaptation reasoning at runtime. The aim of reasoning is to find the appropriate configuration to replace the existing configuration. The kind of uncertainty we deal with for reasoning is related to the users of self-adaptive software. Different users often have different and even conflicting opinions about an adaptation policy that needs to be taken when the software meets a certain condition. This makes the adaptations based on user preferences uncertain and error prone. The sort of uncertainty that we are dealing with here is not related to the variability in the data, but the lack of knowledge. As a result, we chose fuzzy theory for adaptation reasoning.

In Summary, we address two types of uncertainty in this thesis:

1. Uncertainty due to *measurement inaccuracies* (see Chapter 4).
2. Uncertainty in the *adaptation knowledge specification* (see Chapter 5).

1.5.1. Solution Framework

The solution in Figure 1.3 is an integration of two developed mechanisms. The first mechanism, which we call RobustMC, enables the robust model calibration of component connectors. The second mechanism, which we call RobusT2, enables robust adaptation reasoning. We integrate the mechanisms in a framework called *Robust Control of Uncertainty in component connectors* (RCU) as a coherent framework that guarantees reliability in the self-adaptation loop. Note that the process of runtime observation (i.e., monitoring) and execution of change plans (change effectuation) is out of the scope of this thesis. In order to demonstrate the applicability of this approach in real-world domain, we had to implement those two parts as well (see Chapter 7). We use estimation techniques based on time series analysis for model calibration at runtime (see Chapter 4). We also use runtime efficient verification techniques to verify the non-functional requirements at runtime. When a violation is detected, we use fuzzy reasoning for deriving the appropriate configuration that satisfies the requirements (see Chapter 5).



RCU Framework
RobustT2
RobustMC
Realize
Realize
Realize
Use/Update
Use/Update
(3) Quantitative Verification
The requirements are quantitatively verified against the tuned model
Planning
Analysis
(4) Reasoning
A change plan is derived to adapt the configuration of the connector
(2) Calibration
The parameters of the model are estimated based on the runtime data
Fuzzy Rule Base
Model@RunTime
Knowledge
Execution
Monitoring
(5) Effectuation
The running connector is safely reconfigured to a new operating mode
(1) Runtime Observation
The running connector and/or the environment is monitored
Is Monitored
Running Connector
Monitoring Facilities
Interacts
Is Monitored
Middleware Platform + Components (Environment)
Legend:
Framework/ Mechanism
Infrastructure/ Software
Self-adaptation activity
Control loop activity

Figure 1.3. Overview of our solution framework.

1.5.2. Research Contributions

In this thesis, we introduce the RCU framework (cf. Figure 1.3) as the main contribution that enables:

1. **Robust control of uncertainty in runtime verification**.

   The runtime verification task has been split into two sub-tasks – model calibration and quantitative evaluation. The model calibration enables the update of models at runtime. The proposed stochastic approach is able to update the unknown parameters of the models at runtime even in the presence of incomplete and noisy observations. The proposed model calibration technique has been implemented and empirically evaluated, showing a significant precision of parameter estimation and a reasonable overhead at runtime. This contribution will be the main subject of Chapter 4.

2. **Robust control of uncertainty in adaptation reasoning**.

   A general methodology based on fuzzy logic has been defined for the control of adaptation decision that adjusts the configuration of component connectors to the appropriate operating mode. The methodology enables a systematic development of fuzzy logic controllers that can determine the right operating mode for connectors considering that different users with potentially different and even conflicting opinions can specify the adaptation policies. The fuzzy based controller, generated by the methodology, has been shown to be robust against uncertainties in the input data and efficient in terms of runtime overhead, allowing a reliable decision-making in the self-adaptive component control. The control methodology and experimental evaluations are presented in Chapter 5.

3. **Reliable and dependable self-adaptive component connector**.

   The evaluation of the primary contributions – i.e., robust model calibration and adaptation reasoning, is performed by applying the solution to enable self-adaptation of a real-world connector. The approach has been shown to be effective as an end-to-end solution for controlling uncertainty in the feedback control loop of the self-adaptive connectors. This evaluation is dealt with in Chapter 7.

Although in this thesis we claimed for three different contributions as listed above, we consider the second contribution that is reported in **Chapter 5** as the main and core contribution of this thesis. Taming *uncertainty in the adaptation knowledge specification* is the novel contribution that has been proposed for the first time in this thesis.

1.6. Research Methodology

While behavioral science is the appropriate methodology for studying phenomena that are related to human aspects or requirements aspects of software systems, design-science (Hevner, March, Park, & Ram, 2004) is a methodology to study phenomena related to software through building and evaluating artifacts. The synthetic nature of software engineering aligns with the subject of study of the design-science paradigm. Design science is essentially an action-based problem-solving methodology that seeks to create and evaluate artifacts intended to solve identified problems. It concentrates on the usefulness or utility of a method or artifact in practice rather than on its truth, taking into account *real-world constraints* and *practical* considerations. Design-science helps in managing the complicated issues linked to the design of useful artifacts in domain areas in which existing theory or previous knowledge is often not enough. It essentially addresses important unsolved problems in unique or innovative ways, or solved problems in more effective or efficient ways (Hevner et al., 2004).

Since the objectives of this research are synthetic (i.e. robust control of uncertainty in self-adaptive component connectors), we followed a research methodology consistent with the principles of design-science. The artifact that is developed for this thesis is "a framework comprising analytical techniques and mechanisms for controlling uncertainties in component connectors environment" to enable "reliable self-adaptation" of component connectors. The application domains of the research artifacts (i.e., analytical techniques and mechanisms) are in the area of evolving critical systems (e.g., safety-critical, mission-critical, business-critical, or security-critical). The evaluation of the artifacts (i.e., framework and its comprising analytical techniques and mechanisms) are performed through *controlled experiments*. Controlled experiments are frequently used to evaluate and validate research artifact correctness and how precisely the research goals are met through measurement of some criteria. Controlled experiments provide a better understanding of the problem and feedback to improve the mechanisms. Experiments also explain the contributions of the mechanism when compared to existing practices.

While design science provides general guidelines for conducting research, we performed our research according to the model in (Davison, Martinsons, & Kock, 2004), which provides a rigorous step-wise process. Figure 1.4 provides an overview of the research activities performed for this thesis.

The following activities has been followed in this research:

1. *Diagnosis*. In this step, we identified the research problem that needs to be addressed. The general domain of our research is architecture-centric software evolution and more specifically self-adaptive software. In order to identify the research gap in this domain, we performed a systematic literature review reported in (Jamshidi, Ghafari, Ahmad, & Pahl, 2013). This study enabled us to identify the primary research challenge of this thesis, which is controlling the uncertainty in the self-adaptation process of component connectors.
2. *Planning*. In this step, we planned the actions need to be performed according to the research gaps identified in the diagnosis step. A research proposal and a completion plan were prepared. In the research proposal, we described, in detail, the deliverables and ultimate outcome of this research as well as the methods needs to be adopted for evaluating the research outcomes.
3. *Intervention*. In this step, we conducted the activities that were required to develop the solution to solve the identified research problem. The framework that is developed as the primary solution in this thesis concerns runtime model calibration as well as change planning in the self-adaptation loop. The first process includes updating analytical models based on runtime data observation. The techniques that are developed in this process can handle precise model update in the presence of uncertainty such as incomplete data or noisy data. The second process includes planning the right configuration for the component connector based on environmental and internal quality measurements. The techniques that are developed in this process can plan appropriate configuration in the presence of noisy runtime measurements.
4. *Evaluation*. In this step, we evaluated our solution through controlled experiments on individual techniques and mechanisms that we have developed in order to control uncertainty in self-adaptive component connectors.
5. *Reflection*. Reflection consists of activities that involve illustration of the research impact to a specific research community. In the context of this research, the research outcomes were communicated though publications and implementations of the techniques were made available for replicating the results.

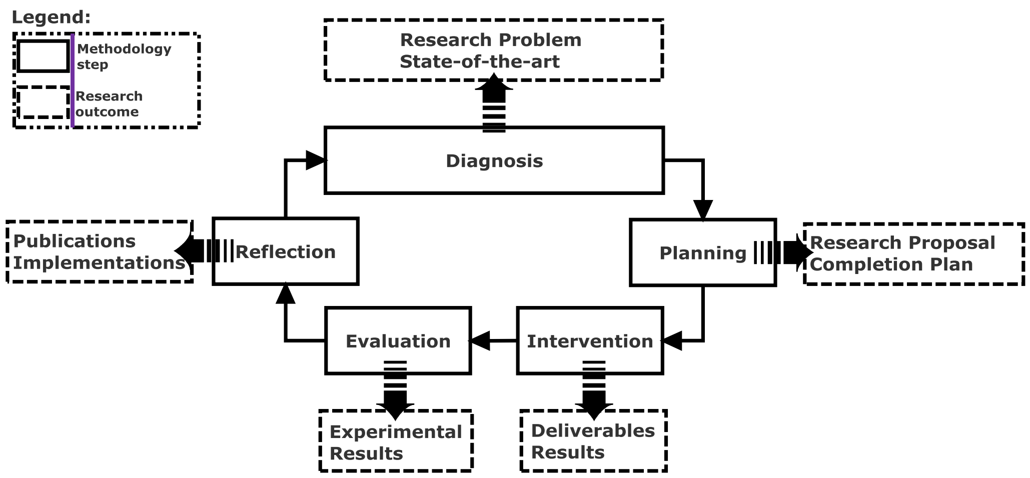


Figure 1.4. Overview of our research methdlogy.

In Table 1.1, we summarize the steps of our research methodology that we have followed throughout this research and their relevance to the individual thesis chapter (represented as Ch*).

Table 1.1. Summary of the research methodology steps and their relevance to the thesis chapters.

Research Methodology Steps	**Thesis Chapters**							
	Ch1	**Ch2**	**Ch3**	**Ch4**	**Ch5**	**Ch6**	**Ch7**	**Ch8**
Diagnosis – Problem Identification	√	√	√					
Planning – Develop Research Plan			√					
Intervention – Execute Research Steps				√	√	√		
Evaluation – Conduct Research Evaluation				√	√		√	
Reflection – Research Impact							√	√

1.7. Thesis

This section first describes the thesis that this work intends to support. This section also explains the research claims of this thesis.

1.7.1. Thesis Statement

In this thesis, we enable reliable and dependable self-adaptation of component connectors in unreliable environments with imperfect monitoring facilities by providing: (a) techniques for robust model calibration, (b) a method for robust adaptation reasoning, and (c) tool support that allows an end-to-end application of the developed techniques.

1.7.2. Research Claim

The thesis statement explains a concise solution to the stated problem being addressed in this thesis. However, it does not explicitly talk about specific claims or the criteria for evaluating the approach. In this section, we consider three research claims and explain an appropriate evaluation method. A summary of these items is shown in Table 1.2.

Research claim 1 (**runtime efficiency**). *The approach for model calibration and adaptation reasoning imposes acceptable overheads and is runtime efficient, satisfying the timing restriction of the self-adaptation loop*.

The activities that need to be integrated in the self-adaptation requires being time efficient. Therefore, as part of ensuring the practicality of the approach, there is a need to evaluate the runtime complexity of the approach.

Research claim 2 (**scalability**). *The approach for model calibration is practical in terms of runtime efficiency with large models. In addition, the approach for adaptation reasoning is applicable even with a large knowledge base*.

It is not sufficient for the approach to be time efficient for small models but it needs to impose an acceptable overhead to large-scale systems, which correspond to complex models. As a result, there is required to ensure the scalability of the approach by scaling out models at runtime for model calibration and fuzzy rule base for adaptation reasoning.

Research claim 3 (**robustness**). *The approaches for model calibration and adaptation reasoning are resilient against dynamic uncertainty in their input*.

The approach needs to be resilient against different amplitude of noises, which resembles the reality of uncertain environments that component connectors are operating. It is required that different levels of noise are injected to the input parameters of the approach and the robustness of the approach is evaluated under dynamic uncertainty.

Research claim 4 (**applicability**). *The approach is applicable to real-world component connectors*.

The approach presented in this thesis develops a set of techniques and methods to control the uncertainty in self-adaptation of component connectors. However, it is not evident that these techniques and methods are actually useful in real world settings. In order to evaluate the applicability of our approach in a real world context, we conducted a case study.

Table 1.2. Research claims, evaluation method and relevant chapters.

Research Claim	Evaluation Method	Associated Chapter (s)
Runtime efficiency	Controlled experiment	Chapter 5, Chapter 7
Scalability	Controlled experiment	Chapter 5, Chapter 7
Robust	Controlled experiment	Chapter 4, Chapter 5, Chapter 7
Applicability	Case study	Chapter 7

1.8. List of Publications

The following publication has been produced during the course of the last three years as part of my PhD research:

Refereed Journal Papers:

[J1] P. Jamshidi, A. Ahmad, C. Pahl, *Taming Knowledge Specification Uncertainty in Self-Adaptive Software,* Elsevier Journal of Systems and Software, 2014, Under Review.

[J2] P. Jamshidi, A. Ahmad, C. Pahl, *Cloud Migration Research: A Systematic Review*, IEEE Transactions on Cloud Computing, 2013, DOI: 10.1109/TCC.2013.10. [Data]

[J3] A. Ahmad, **P. Jamshidi,** C. Pahl, *Classification and Comparison of Architecture Evolution Reuse Knowledge - A Systematic Review*, Journal of Software: Evolution and Process, Wiley, 2014, DOI: 10.1002/smr.1643. [Data]

[J4] A. Ahmad, **P. Jamshidi,** C. Pahl, *A Pattern Language for the Evolution of Component-based Software Architectures*, Electronic Communications of the EASST, Special Issue on Patterns Promotion and Anti-patterns Prevention, 2014.

Refereed Conference Papers:

[C1] P. Jamshidi, C. Pahl, *Orthogonal Variability Modeling to Support Multi-Cloud Application Configuration*, Seamless Adaptive Multi-cloud Management of Service-based Applications (Seaclouds), ESOCC 2014.

[C2] P. Jamshidi, A. Ahmad, C. Pahl, *Autonomic Resource Provisioning for Cloud-Based Software*, in 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), co-located with ICSE'14, 2014.

[C3] P. Jamshidi, M. Ghafari, A. Ahmad, C. Pahl, *A framework for classifying and comparing architecture-centric software engineering research*, in 17th European Conference on Software Maintenance and Reengineering (CSMR), 2013.

[C4] A. Ahmad, **P. Jamshidi**, C. Pahl, F. Khaliq, *PatEvol - A Pattern Language for Evolution in Component-Based Software Architecture*, Workshop on Patterns Promotion and Anti-Patterns Prevention, CSMR, 2013.

[C5] P. Jamshidi, C. Pahl, *Business Process and Software Architecture Model Co-evolution Patterns*, ICSE 2012 Workshop on Modeling in Software Engineering (MiSE 2012).

[C6] A. Ahmad, **P. Jamshidi** and C. Pahl, *Pattern-driven Reuse in Architecture-centric Evolution for Service Software,* 7th International Conference on Software Paradigm Trends ICSOFT 2012, 2012.

[C7] A. Ahmad, **P. Jamshidi**, M. Arshad and C. Pahl, *Graph-based Implicit Knowledge Discovery from Architecture Change Logs*, Seventh Workshop on SHAring and Reusing architectural Knowledge - SHARK 2012 (at WICSA/ECSA'2012), 2012.

[C8] M. Ghafari, **P. Jamshidi**, S. Shahbazi and H. Haghighi, *An architectural approach to ensure globally consistent dynamic reconfiguration of component-based systems*, 15th International ACM SIGSOFT Symposium on Component-based Software Engineering (CBSE'2012), Bertinoro, Italy, June 2012.

[C9] M. Ghafari, **P. Jamshidi**, S. Shahbazi, H. Haghighi, *Safe stopping of running component-based distributed systems: challenges and research gaps*, International Conference on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures, Toulouse, France, June 2012.

[C10] A. Ahmad, **P. Jamshidi**, C. Pahl, *Graph-based Pattern Identification from Architecture Change Logs*, International Workshop on System/Software Architectures IWSSA'2012, CAiSE 2012.

Table 1.3. A mapping of the related publications to the individual thesis chapters.

Chapter	**Primary Publication**	**Secondary Publication**	**Outcome**
Chapter 1	-	C10, C7, C6, C3	Research Questions, Hypothesis
Chapter 2	-	-	Thesis Background
Chapter 3	C3, J3	-	Research Positioning
Chapter 4	-	C3	RobustMC (cf. Figure 1.3)
Chapter 5	J1, C2	-	RobusT2 (cf. Figure 1.3)
Chapter 6	C8, C9	C5, C4, C1, J4	Change Execution Mechanism
Chapter 7	C2, J1	J2	Research Validation (RCU)
Chapter 8	-	-	Conclusions and Outlook

1.9. Thesis Outline

The structural organization of the thesis is illustrated in Figure 1.5. In the remainder of this section, we provide an overview of the contribution of each chapter that follows a summary of the objectives and outcome for each chapter presented in Table 1.3.

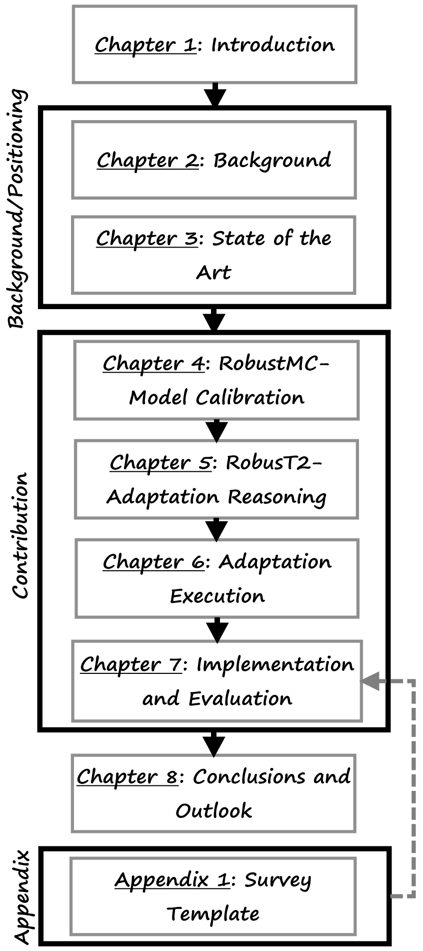

Chapter 1: Introduction
Background/Positioning
Chapter 2: Background
Chapter 3: State of the Art
Contribution
Chapter 4: RobustMC-Model Calibration
Chapter 5: RobusT2-Adaptation Reasoning
Chapter 6: Adaptation Execution
Chapter 7: Implementation and Evaluation
Chapter 8: Conclusions and Outlook
Appendix
Appendix 1: Survey Template

Figure 1.5. An overview of the thesis organization.

Chapter 2 presents the research background and related definitions and concepts that we use throughout this thesis.

We explain some of the fundamental concepts that provide background details before the discussion of thesis contribution. In this chapter, we focus on the role of uncertainty in self-adaptive software, the role of models at runtime in non-functional requirement verification and the central role of fuzzy logic systems in adaptation reasoning.

Chapter 3 critically reviews the state-of-the-art of existing approaches for controlling uncertainty for self-adaptive software systems.

First, a number of comparison criteria of related approaches is given. Then, we provide a demarcation and detailed description of each related research work. Finally, a detailed comparison of the related work according to the comparison criteria is discussed.

Chapter 4 proposes a method for model calibration in the presence of uncertainty.

In this chapter, we present the analytical models to model the component connector behavior. We also propose mechanisms to calibrate the unknown parameters of the models at runtime. The key contribution here is that the mechanisms are capable of carefully determining the parameters even in the presence of uncertainty. The proposed method is comprehensively evaluated with thorough discussions of the results. The results in this chapter provides an answer for research question **RQ1**.

Chapter 5 describes in details the design, implementation and experimental validation of the adaptation reasoning for component connectors.

In this chapter, we first review the state-of-the-art of adaptation reasoning techniques and mechanisms. We then propose the RobusT2 framework to realize the adaptation reasoning using type-2 fuzzy logic system. After presenting the concept of uncertainty and type-2 fuzzy logic systems, this chapter introduces in detail the interval type-2 fuzzy logic systems considered in this research. This chapter presents the application of type-2 fuzzy logic control developed in this research for adaptation reasoning. The chapter explains by using numerical examples the subsystems of the developed framework. This chapter also presents experimental evaluations of the framework. The results in this chapter provide an answer to research question **RQ2**.

Chapter 6 presents a mechanism to enact the transitions from the current connector configuration to the target configuration.

Considering the heterogeneity of models and languages involved in software connectors, this chapter introduces an approach to derive reconfiguration plans using reasoning based on graph theory and feature models. We describe a mechanism for transforming these feature models corresponding to the connector modes to an executable reconfiguration plan.

Chapter 7 reports an end-to-end evaluation of individual research components and overall validation of the proposed framework.

In this chapter, we show how the three key parts of the RCU framework are integrated to enable self-adaptation of component connectors through a real-world case study. To conduct this research, we followed the guidelines of the action research methodology (Chapter 1) that provides a rigorous set of steps focused on planning (Chapter 2, Chapter 3) and conducting the research (Chapter 4, Chapter 5, Chapter 6) along with the evaluations of the research results (Chapter 7). Therefore, in this chapter, we focus on an experimental evaluation of the adaptation management of component connectors in the RCU framework. In summary, we show the validity of the research claims that are discussed in Section 1.7.2. The results in this chapter provide an answer to research question **RQ3**.

Chapter 8 concludes our research contribution in the context of research gaps identified in Section 1.4.

In this chapter, we review the contribution once again. We also discuss limitations, threats to validity and the potential for future research.

Appendix A presents the design of the fuzzy logic system used in Chapter 5.

The type-2 fuzzy logic system used for adaptation reasoning is designed by using expert knowledge. A survey based on a real-world connector was conducted among experts in cloud computing. The survey allows extracting expertise in the form of IF-THEN rules. This appendix chapter mainly presents the template that we adapted from (Solano Martínez, 2012) and extended to use in the survey.

1.10. Chapter Summary

This chapter provides the research motivation based on a brief overview of existing research and its limitations. Based on the identified research challenges, we outlined the central hypothesis that allowed us to identify the research questions. The role of individual research questions is vital in highlighting the solution requirements. We highlighted the adoption of a customized research methodology to plan, conduct the research, evaluate the developed artifacts and reflect on the research implications. We also specified our research claims, which become the main criteria for evaluating the approach.

Finally, we provided an overview of the organization of the thesis. The chapter provides a foundation to present the results of our literature review and to provide an overview of the proposed solution. A summary of the objectives and the outcome for the individual chapters in this thesis is presented in Table 1.3 that allows us discuss the research positioning, contributions and evaluation in subsequent chapters (cf. Figure 1.5).

Chapter 2

2. Background

"We shall not cease from exploration and the end of all our exploring will be to arrive where we started and know the place for the first time." – Thomas Stearns Eliot (1888-1956).

Contents

2.1. Chapter Overview

Several research fields, the most important ones being “Self-Adaptive Software”, “Models at Runtime”, “Fuzzy Logic Systems” and “Component Connectors”, influence this thesis. Each field is divided into many research communities each focusing on special aspects of the respective field. The self-adaptive software community provides the general theme underlying this thesis, and the outcome of this thesis contributes mainly to this community. The “models at runtime” is a closely related technique to realize self-adaptive software, which was a main inspiration for this thesis. Fuzzy logic is a field of research, which has been investigated, tailored and adapted for adaptation reasoning in this thesis. Finally, component connectors play a fundamental role in this thesis as the principal domain to which we apply our solution framework in order to enable dependable self-adaptive architectures.

This chapter serves to ease the understanding of the succeeding chapters by summarizing the most important aspects of the four related research fields to the topic this thesis. Note that readers can skip this chapter as it is not a core contribution chapter and it consists mainly of fundamental definitions that we borrowed from standard literature in order to back up the propositions of this thesis. In other words, none of the definitions that are given in this chapter are originated here and we only include them to make this thesis a self-contained manuscript. In core chapters (i.e., Chapter 4, Chapter 5, Chapter 6 and Chapter 7), wherever necessary, we refer back to the definitions in this chapter.

In the following, Section 2.2 discusses the phenomena of uncertainty in self-adaptive software, followed by an overview of mathematical techniques for controlling uncertainty in Section 2.2.3. Section 2.3 introduces background on analytical models, which are adopted in this thesis for adaptation reasoning. Section 2.4 discusses background on fuzzy logic systems, upon which our adaptation planning is based. Section 2.5 discusses the basic definitions related to the scope of this thesis, which is the notion of component connectors and the adopted language for representing component connectors.

2.2. Uncertainty in Self-Adaptive Software

In self-adaptive software, not all sources of uncertainty have the same features (Esfahani & Malek, 2013). Approaches for modeling different uncertainties are very dissimilar to each other (Esfahani et al., 2011; Esfahani & Malek, 2013). For example, uncertainty for specifying objectives of self-adaptive software is due to lack of knowledge and it is not possible to specify this with a probability distribution suitable for specifying uncertainty due to variability.

2.2.1. Lack of knowledge vs. variability

A distinction that is common in the literature is between aleatory uncertainty and epistemic uncertainty (J. Aughenbaugh, 2006). The former uncertainty comes from the Latin word for gambler “aleatory” and refers to uncertainty rooted in a random process. The later uncertainty arises from the Greek term “episteme” meaning knowledge. Aleatory uncertainty refers to inherent variability in the observed phenomena and is commonly modeled using probability theory. In contrast, epistemic uncertainty is instigated by a lack of knowledge about the observable phenomena. This distinction is because of the *location of uncertainty* – in the state of the analyst, users, decision maker vs. state of the software system under consideration (J. Aughenbaugh, 2006).

The distinction between aleatory and epistemic uncertainty is not always clear (J. Aughenbaugh, 2006). For instance, one may argue that variability observed in the environmental dynamics of software systems is due to the limitations of scientific models and, therefore, lack of knowledge. While these opinions are somehow correct, these distinctions depend on one's specific point of view. More clearly, in one point of view, a phenomenon might be uncertain due to lack of knowledge, but it also might be uncertain due to variability in a different point of view.

2.2.2. Reducibility vs. irreducibility

The uncertainty with respect to unknowable phenomena is often referred to as irreducible uncertainty (J. Aughenbaugh, 2006). Similarly, the uncertainty linked with knowable phenomena, which are currently unknown, is referred to as reducible uncertainty (J. Aughenbaugh, 2006). The distinction between these types of uncertainty is also disputable. One of the main causes behind irreducibility of uncertainty is related to the existing capability of science to mitigate the intractable complexity of phenomena. For example, it is a fact that the physical world is a non-linear system. However, since we do not know enough non-linear mathematics, we model the system through linear mathematics and as a result, models contains irreducible uncertainty.

2.2.3. Mathematical theories for representing and controlling uncertainty

This section provides an overview of two commonly applicable approaches for handling uncertainty in software engineering in general and self-adaptive software in particular. As it is discussed in the state-of-the-art (Esfahani & Malek, 2013), existing work has often relied on one of these approaches to either model or reduce the effects of uncertainty.

2.2.3.1. Probability theory

Probability theory (Bertsekas & Tsitsiklis, 2002) is the most widely used approach for handling uncertainty in the self-adaptive software domain (Esfahani & Malek, 2013). Probability was formerly connected to the classical interpretation of the theory. This interpretation is because the outcome of a phenomenon is equally possible. This classical interpretation of the theory produces inconsistencies when it is used beyond games of chance. Because of this limitation, the frequentist interpretation was conceived. In this interpretation, the probability of an event is delineated as a limit of its relative frequency in large trials.

Bayesian theory (Hoff, 2009) is founded on the subjective explanation of the probability. In this interpretation, the probability is defined as a manifestation of a rational belief of a human about uncertain propositions. This explanation generalizes the frequentist interpretation as it allows probability assignment to a single observation regardless of whether or not it is part of a larger observation. This interpretation is very useful in cases where there is not enough data for frequentists. For instance, by frequentist interpretation we are not be able to analyze a new unknown phenomenon for which enough data is not available, while Bayesians can use subjective information based on related phenomena to analyze the new phenomenon. Bayesian approaches are a unified theory for both data-rich and data-poor problems. Many modern machine-learning methods, including those adopted within this thesis, are based on Bayesian principles.

2.2.3.2. Fuzzy theory

Fuzzy theory (Zadeh, 1965) is an extension of classical set theory. In classical set theory, either an element of a set is a member of the set with membership value 1 or it is not a member of the set with membership value 0. However, in fuzzy theory, a member can be in some degree a member of a set. To that end, the membership value of an element with respect to a set is any value between zero and one. The higher a membership degree is, the more likely that element belongs to that set. As a result, the boundary of a fuzzy set is not evidently defined, while the boundary of a classical set is defined with a crisp value.

Fuzzy theory is useful in domains where information is incomplete or imprecise. For instance, fuzzy sets have been used in linguistics to deal with ambiguity of words. As another example, temperatures that are considered to be cold and warm may be different from person to person. In fact, some temperatures can be considered both cold and warm to some extent. A program that tries to understand written text can use the fuzzified versions of coldness and warmness to deal with uncertainty regarding understanding of the text.

While probability theory deals with quantifying the variability in data, fuzzy theory focuses on quantifying the ambiguity of data. Although, sometimes the two theories can be used interchangeably, it has been shown that the two theories are different. In general, possibility theory is useful when there is little information or imprecise data. However, when more precise information is available, it is better to use probability theory.

2.3. Analytical (Stochastic) Models at Runtime

Runtime modeling of a software system describes the behavior of software, comprising its interaction with users and the environment (cf. Figure 2.1). There are some factors including the dependency of software on physical resources, third party services, and variability of the usage profile that produce uncertainty for the software. All of these changes are not under the control of the system and may occur unpredictably. For instance, the usage load of a cloud-based application may change suddenly during special events like Christmas (Jamshidi, Ahmad, & Pahl, 2014).

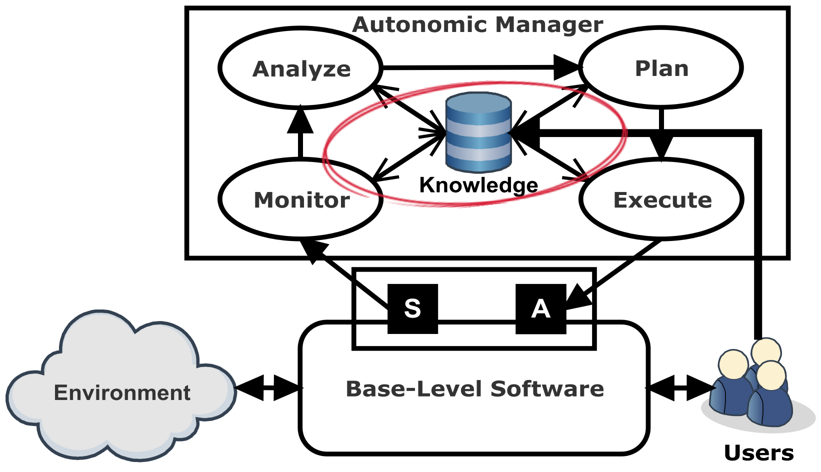


Figure 2.1. The role of models as the K in the MAPE-K loop in self-adaptive software.

The key focus of this thesis is on non-functional requirements (Pohl, 2010), in particular *performance* and *reliability*, that are considerably affected by environmental dynamics. These requirements are often hard to predict at design-time because of their interdependence on environmental factors that are prone to change at runtime. Moreover, even if the predictions were initially made, they are likely to change at runtime. These predictions are often based on human experience, historical data or the observation of similar systems (A Filieri, Ghezzi, & Tamburrelli, 2012). Even when such data is available to make design-time assumptions about the environment, sudden changes in the environment can nullify them.

In order to quantify non-functional requirements, we need to deal with unavoidable uncertainty. By abstracting the behavior of software (component connectors here in this thesis) to a finite and countable set of states, we are able to formally analyze the properties in which we are interested to study finite-state stochastic processes. The reason behind this choice is quite natural. Firstly, since our focus is on non-functional properties of software systems, we specify systems via stochastic models, which support quantitative probabilistic specifications that are particularly suitable to express reliability, performance, and cost concerns. Secondly, reasoning is supported by model checking, which can be utilized to automatically verify a system model against requirements expressed in a suitable logic notation. Consecutively, this may trigger proper adaptation strategies to change system configurations and avoid predicted requirements violations. Conceptually, this framework, usually called models at runtime (Blair, Bencomo, & France, 2009), establishes a feedback control loop between analytical models, here stochastic models, and the running system. At runtime, the system feeds data back to update the analytical model. Models are alive at runtime and they evolve since their parameters are constantly updated by monitoring relevant aspects of the running software, which recognize relevant changes as they occur at runtime and modify the models accordingly (Epifani et al., 2009).

Note that all definitions given in this section are standard definitions in probability theory, stochastic model checking and quantitative verification that we borrowed from standard literature, e.g., (Calinescu et al., 2012; M Kwiatkowska, Norman, & Parker, 2007; Marta Kwiatkowska, 2007; Pinsky & Karlin, 2010).

A stochastic process (Pinsky & Karlin, 2010) is a family of random variables X_t that is intended to model time dependent stochastically evolving dynamic systems, such as software systems. Note that each of the random variables signifies the state of the system at a time point t. More concretely, we describe a stochastic process as:

Definition 1. A ***stochastic process*** is a mapping

$$X: T \times \Omega \rightarrow S \tag{2.1}$$

, where Ω is a probability (sample) space, T is a set of time points and S is the state space of the stochastic process X.

The quantity $X(t, \omega)$ is the value of the stochastic process at time t for the outcome $\omega \in \Omega$. To simplify the definition, the dependence of X on ω can be avoided, we can write the process as $X(t)$, which represents the state of the process at time. One can consider discrete-time processes with $t \in \mathbb{N}$, or continuous-time processes with $t \in \mathbb{R}$. In this thesis, we consider both discrete-time and continuous-time processes.

The temporal evolution of stochastic processes representing a software system is indicated by its previous history, design-time assumptions and random variables capturing the uncertainty about users and environment. Markov processes are a special class of stochastic systems satisfying the Markov property.

Definition 2. A stochastic process $X(t)$ have a ***Markov property*** if for each $t \geq 0$ and a subset $A \subseteq S$:

$$\mathbb{P}(X(t+1) \in A | X(0) = x_0, \dots, X(t) = x_t) = \mathbb{P}(X(t+1) \in A | X(t) = x_t) \qquad (2.2)$$

Therefore, for a Markov process the only information about the past needed to predict the future is the current state of the random variable. On the other hand, knowledge of the values of earlier states do not change the transition probability. Note $\mathbb{P}(X(t+1) \in A | X(t) = x_t)$ is one-step transition probability. Under the assumption of discreteness (finiteness and countability) of state space S, the one-step transition structure of X can be summarized through a square transition matrix P, whose entry $p_{i,j}$ is the value $\mathbb{P}\left(X(t+1) = s_j \middle| X(t) = s_i\right)$ with $s_i, s_j \in S$.

A stochastic process satisfying the Markov property is called *Markov process* (Pinsky & Karlin, 2010). A *Markov chain* refers to a sequence of random variables $(X_0, \dots, X_n)$ generated by a Markov process. Generally, the term Markov chain is used to convey a Markov process which has discrete (finite or countable) state space. More specifically, the possible values of X_i form a countable state space of the chain. There are many alternatives to Markov processes, suitable for representing several aspects of the modeled software systems such as reliability, execution time, or energy consumption (A Filieri et al., 2012). A Markov chain either can be defined for a discrete set of times or can take continuous values $\{X(t): t \geq 0\}$. In the former case, the Markov chain is called Discrete-Time Markov Chain (DTMC) and in the latter case, it is called Continuous-Time Markov Chain (CTMC).

2.3.1. Discrete-Time Markov Chains

Discrete-Time Markov Chains (DTMC) (Pinsky & Karlin, 2010) are widely adopted in software engineering for reliability measurement and analysis (L. Cheung, Roshandel, Medvidovic, & Golubchik, 2008; Pham, 2006; Roshandel, Medvidovic, & Golubchik, 2007; W.-L. Wang, Pan, & Chen, 2006). One common characteristics of these approaches is that they are used for reliability assessment of systems composed by cooperating parts (e.g. component-based software, or service-oriented architectures) at design-time (Immonen & Niemelä, 2007). The most important aspect for adopting DTMCs for reliability analysis is that the system's behavior with some tolerable approximation should meet the Markov property.

Definition 3. A ***Discrete-Time Markov Chain*** is a stochastic process satisfying the Markov property with $T \subseteq \mathbb{N}$ and S is finite and countable. This structure is usually represented by Kripke notation as (S, s_0, P, L), where:

- S is a finite set of states.
- s_0 is the initial state.
- $P: S \times S \to [0,1]$ is the probability of transitions between states. $P(i,j), P\left(s_i, s_j\right), p_{i,j}$ are interchangeably used for representing the transitions.
- $L: S \to 2^{AP}$ is a labeling function that associates to each state the set of atomic propositions (i.e., $L(s)$) that are true in that state.

An element of P such as $p_{i,j}$ signifies the probability that the next state of the process will be s_j given that the current state is s_i. Note that $\sum_{s_j \in S} p_{ij} = 1$, where $\{p_{ij}\}$ is the next state distribution for a state s_i. Usually, $s = s_i, s = i$ are the default atomic propositions of each state s_i. The probability of a path originating from s_i and ending in s_j in precisely 2 steps, i.e., having one intermediate step, is $\sum_{s_x \in S} p_{ix} * p_{xj}$. This summation is the entry (i,j) of matrix P^2. Similarly, the probability of moving from s_i to s_j in exactly k steps is the entry (i,j) of matrix P^k.

Definition 4. An ***execution path*** is a sequence of states $\pi = s_0, s_1, s_2, \ldots$ of a DTMC if for any pair $(s_i, s_{i+1}), p_{i,i+1} > 0$. $\pi[i]$ is used to represent the ith state in the path π. The probability of a finite path to be observed is $\prod_{i=0}^{|\pi|-2} \mathbb{P}(s_i, s_{i+1})$, and 1 when $|\pi| = 1$.

Definition 5. A state s_i is *transient* if:

$$\sum_{n=1}^{Inf.} p_{i,i}^{n} < \infty \tag{2.3}$$

It is recurrent if:

$$\sum_{n=1}^{Inf.} p_{i,i}^{n} = \infty \tag{2.4}$$

And it is absorbing if $p_{i,i} = 1$.

In other words, the number of transitions into state s_i is finite, while recurrent states will be visited infinitely.

Definition 6. A DTMC model is ***well-formed*** if:

- Every state that is recurrent is also absorbing
- All states of the model are reachable from the starting state
- There is a path to at least one absorbing state from every transient state

2.3.1.1. Model Specification with DTMCs

In this section, a few points about the modeling process with DTMCs are provided.

DTMCs have been adopted for modeling different phenomena, e.g. chemical reactions, DNA sequences, financial trading, demographic evolution, human behavior, or business processes (Pinsky & Karlin, 2010). DTMCs can be perceived as state-transition systems with annotated transitions through which non-functional aspects can be specified. State-transition systems are frequently used in practice by software designers and can be used at different levels of abstraction to model software systems.

Several software modeling standards such as UML can be automatically translated into DTMC models by means of automated model transformations, e.g., (Gallotti & Ghezzi, 2008; Carlo Ghezzi & Sharifloo, 2011). Moreover, some integrated design frameworks can automatically transform their design models into corresponding Markov chains in order to provide quality assessment, e.g., (Becker, Koziolek, & Reussner, 2007; Ciancone, Filieri, & Drago, 2011).

Because of its inherent characteristics, DTMCs have been widely used to analyze system reliability (Pham, 2006). The common idea behind the various software reliability analysis approaches is that special states represent software failure condition. The existence of a failure is then represented by a transition toward one of these states. In general, a state in a DTMC model represents an observable condition of the system at runtime that is relevant from the modeling perspective at a specific abstraction level (A Filieri et al., 2012). In this thesis, we assume that it is possible to identify a portion of the execution of the system that we can associated with only one state of the DTMC.

In order to map the execution state of a software system into a DTMC model, there is no general guideline and procedure. In (R. Cheung, 1980) a state of a DTMC model corresponds to a program module and state transitions correspond to control flow between the modules. In (Littlewood, 1975), the very same concept with a different name, i.e., program sub-unit, is used for reliability analysis of a software program. In (Calinescu, Grunske, Kwiatkowska, Mirandola, & Tamburrelli, 2011; A Filieri et al., 2012), a state is referred to as an external service invocation. Moreover, special states such as the initial state of DTMC models represent the users and its outgoing transitions represent user profiles. Similarly, in (A Filieri, Ghezzi, Grassi, & Mirandola, 2010), a state represents service execution, but also contain accumulated errors in the data flow up to the point.

A common practice in the mentioned work is that they classify the states in a way that leads to at least one absorbing state. As we mentioned earlier, when an absorbing state is reached, the model will stay there forever. This characteristic makes absorbing states appropriate for occurrences of failure or successful completion of software. In this thesis, it is assumed that every analytical model that we use here has at least one absorbing state demonstrating the termination of the software execution.

One of the interesting features of Markov models in general and DTMC models in particular is that their transition matrices can handle the temporal changes of the model perfectly (Pinsky & Karlin, 2010). Model change in this type of analytical models is defined as variation in the values of the entries of P. In this thesis, we assume two types of entries exist in the transition matrix of a DTMC model. Some entries in the transition matrix are assumed to be known at design-time and stable at runtime. An example is failure of a particular hardware services, e.g. storage. Some other entries are assumed as either unknown at design-time and/or subject to change at runtime. For instance, usage profiles of users may change due to unexpected events. Note that discovering changes at runtime requires observation of the running system and suitable learning techniques. In this thesis, the latter entries of p are referred to as *parameters* of the model, and a model that has at least one parameter is called *parametric*.

2.3.1.2. Markov assumption verification

According to the Markov property as stated in **Definition 2**, the next state to be executed in a Markov process is independent from the previous history and depends only on the current state. However, it not always easy to verify the satisfaction by looking at the source code of a software. In (R. Cheung, 1980; Ramamoorthy, 1966), several experiments showed that this assumption often holds at architectural level.

In the case that the next action depends on previous history, there are still some cases that can be approximated by a Markov process. The first is the case where a limited number of previous moves, let say k of them, affect the next step. This situation can be modeled by a k-th order Markov process, that is one where the next action depends only on the previous k actions. Nonetheless, a software systems might expose an intrinsically non-Markov behavior (Gokhale, 2007), thus the Markov assumption must be verified before proceeding with the analysis (Billingsley, 1961; Pinsky & Karlin, 2010).

2.3.2. Continuous-Time Markov Chains

Despite the fact that each transition between states in a DTMC model corresponds to a discrete time-step in a Continuous Time Markov Chain (CTMC) (Pinsky & Karlin, 2010), transitions occur in real time. For DTMCs, we assume a fixed set of atomic propositions AP.

Definition 7. A ***Continuous-Time Markov Chain*** is a stochastic process satisfying the Markov property with $T \subseteq \mathbb{R}$ and S is finite and countable. This structure is usually represented by Kripke notation as (S, s_0, R, L), where:

- S is a finite set of states.
- s_0 is the initial state.
- $R: S \times S \to \mathbb{R}_{\geq 0}$ is the transition rate matrix.
- $L: S \to 2^{AP}$ is a labeling function that associates to each state a set of atomic propositions (i.e., $L(s)$) that are true in that state.

The matrix R assigns rates to each pair of states in a CTMC. A transition can only occur between states s_1 and s_2 if $R(s_1, s_2) > 0$ and, in this case, the probability of this transition being triggered within t time-units equals $1 - e^{-R(s_1,s_2)*t}$. Typically, in a state s, there is more than one state s' for which $R(s;\ s') > 0$. This is called *race condition*. The first transition to be triggered determines the next state of the CTMC. The time spent in state s, before any transition occurs, is exponentially distributed with the *exit rate* of state s, i.e., $E(s)$, where:

$$E(s) \overset{\text{def}}{=} \sum_{s' \in S} R(s, s') \tag{2.5}$$

A state s is called absorbing if $E(s) = 0$, i.e., if it has no outgoing transitions. Note that the actual probability of each state s' being the next state, to which a transition from state s is made, is independent of the time at which this occurs. This is defined by the notion of an *embedded DTMC*.

Definition 8. The embedded DTMC of a CTMC $C = (S, s_0, R, L)$ is the DTMC $emb(c) = (S, s_0, P^{emb(c)}, L)$, where:

$$P^{emb(c)}(s, s') = \begin{cases} R(s,s')/E(s)\ \ if\ E(s) \neq 0 \\ 1\ if\ E(s) = 0\ and\ s = s' \\ 0\ otherwise \end{cases} \tag{2.6}$$

By considering the above definition, the behavior of the CTMC can be considered in an alternative way. The model will remain in a state s for a delay, which is exponentially distributed with rate $E(s)$ and then make a transition. The probability that this transition is to state s' is given by $P^{emb(c)}(s, s')$.

Definition 9. The infinitesimal generator matrix for a CTMC $C = (S, s_0, R, L)$ is the matrix $Q: S \times S \to \mathbb{R}$ defined as:

$$Q(s, s') = \begin{cases} R(s, s')\ if\ s \neq s \\ -\sum_{s'' \neq s} R(s, s'')\ otherwise \end{cases} \tag{2.7}$$

Definition 10. An infinite path of a CTMC $C = (S, s_0, R, L)$ is a non-empty sequence $\pi = s_0, t_0, s_1, t_1, s_2, \ldots$ where $R(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$. Similarly, a finite path is $\pi = s_0, t_0, s_1, t_1, \ldots s_{k-1}, t_{k-1}, s_k$, such that s_k is an absorbing state. As with DTMCs, $\pi[i]$ is used to represent the ith state in the path π. $\pi@t$ represents $w(j)$ where j is the smallest index for which $\sum_{i=0}^{j} t_i \geq t$.

There are some traditional properties regarding CTMC models: transient behavior, which relates to the state of the model at a certain instant of time; and steady-state behavior, which describes the state of the CTMC model in the long run. For a CTMC model $C = (S, s_0, R, L)$, the transient probability $\pi_{s,t}^{C}(s')$ is defined as the probability of having the model started in state s and being in state s' at time instant t. Using the definitions of the previous section:

$$\pi^{C}_{s,t}(s') \overset{\text{def}}{=} \mathbb{P}(\pi \in Path^{C}(s) | \pi@t = s') \quad (2.8)$$

On the other hand, the steady-state probability $\pi^{C}_{s}(s')$ is the probability of having the model started in state s and being in state s' in the long run:

$$\pi^{C}_{s,t}(s') \overset{\text{def}}{=} \lim_{t \to Inf} \mathbb{P}(\pi \in Path^{C}(s) | \pi@t = s') \quad (2.9)$$

In this thesis, we only consider CTMC models that are homogeneous and finite-state, i.e., the limit in the above definition always exists.

Uniformization is a method to compute transient solutions of finite state CTMC, by approximating the process by DTMC.

For a CTMC model $C = (S, s_0, R, L)$, Π^{C}_{t} represents the matrix of all transient probabilities, i.e., $\Pi^{C}_{t}(s, s') = \pi^{C}_{s,t}(s')$.

Definition 11. For a CTMC model $C = (S, s_0, R, L)$ with infinitesimal generator Q, the ***uniformized*** DTMC is given by $unif(C) = (S, s_0, P^{unif(C)}, L)$, where $P^{unif(C)} = I + Q/q$ and $q \geq \max\{E(s)\}$.

The uniformization rate q is determined by the state with the shortest mean residence time. All exponential delays in the CTMC C are normalized with respect to q. This means that for each state s with $E(s) = q$, one time period in $unif(C)$ corresponds to a exponentially distributed delay with rate q after which one of its successor states is selected (M Kwiatkowska et al., 2007).

2.3.3. Logics for requirement specification on analytical models

Transient-state and steady-state analysis are two traditional methods for analyzing Markov models in software engineering (Marta Kwiatkowska, Norman, & Parker, 2010). These two classical methods enable the investigation of the probability of the analytical model to be in a particular state at a certain time or in a long run respectively. Although because of their long time use in mathematics they are mature enough, these two types of analysis cannot express behavioral properties, such as the probability of eventually reaching a particular state or never hitting an error before completion.

Probabilistic behavioral properties are appropriate formulations of software requirements, such as invariance, precedence, response, or constrained or unconstrained reachability (Grunske, 2008) that can be interpreted on probabilistic models, like Markov processes. They can be adopted, in general, to specify constraints on the probability that particular (un)desired behaviors may be observed for the system:

- C1 (*Reliability*): "The probability of handling a request successfully must be greater than 0.98"
- C2 (*Complexity*): "At least 50% of the requests must be processed within 7 operations"
- C3 (*Cost*): "The average cost for running a request must be less 0.0005 dollars"
- C4 (*Response time*): "The 95th percentile of response time must be less than 0.02s"

A suitable formal language must specify these informal requirements (i.e., C1 to C4) in order to apply automatic verification techniques. For a more comprehensive examples of such quantitative requirements, we refer to (Antonio Filieri, 2013).

2.3.3.1. Probabilistic Computation Tree Logic

Probabilistic Computation Tree Logic (PCTL) (Hansson & Jonsson, 1994) is a temporal logic, based on CTL (Baier & Katoen, 2008). A PCTL formula expresses conditions on a state of a DTMC, and it is evaluated to either true or false on the model.

The syntax of PCTL is defined by the following rules:

$$\begin{aligned}&\phi ::= true|a|\phi \wedge \phi|\neg\phi|\mathcal{P}_{\bowtie p}(\psi)\\&\psi ::= X\phi|\phi U^{\leq t}\phi\end{aligned} \tag{2.10}$$

, where a represents an atomic proposition and $p \in [0,1], \bowtie \in \{<, \leq, >, \geq\}, t \in \mathbb{N} \cup \{\infty\}$. The temporal operators X and U are called "*Next*" and "*Until*", respectively. Note that formulae that are based on the axiom ϕ are called *state formulae* and those that are originated by ψ are instead called *path formulae*.

Here is some examples of state formulae:

$$\begin{aligned}&s \vDash true\ for\ all\ s \in S\\&s \vDash a\ iff\ a \in L(s)\\&s \vDash \neg\phi\ iff\ s \nvDash \phi\\&s \vDash \phi_1 \wedge \phi_2\ iff\ s \vDash \phi_1\ and\ s \vDash \phi_2\\&s \vDash \mathcal{P}_{\bowtie p}(\psi)\ iff\ \mathbb{P}(\pi \vDash \psi|\pi[0] = s) \bowtie p\end{aligned} \tag{2.11}$$

A path π originating in s satisfies a path formula ψ according to the following rules:

$$\begin{aligned}&\pi \vDash X\phi\ iff\ \pi[1] \vDash \phi\\&\pi \vDash \phi_1 U^{\leq t}\phi_2\ iff\ \exists 0 \leq j \leq t\ (\pi[j] \vDash \phi_2 \wedge (\forall 0 \leq k < j : \pi[k] \vDash \phi_1))\end{aligned} \tag{2.12}$$

As a short form for $trueU^{\leq t}\phi$, we use operator $\diamond$ instead, which means *eventually*: $\diamond^{\leq t}\ \phi$.

PCTL can specify a large number of properties on a Markov model. For example, it can specify constraints on the probability of reaching an absorbing state demonstrating a failure or success, given the initial state. This property is a specific example of the more general class of *reachability properties*. Reachability properties are specified as $\mathcal{P}_{\bowtie p}(\diamond\ \phi)$, which shows that the probability of reaching a state where ϕ is valid meets the constraint $\bowtie\ p$.

2.3.3.2. Continuous Stochastic Logic (CSL)

Continuous Stochastic Logic (CSL) (Aziz, Sanwal, Singhal, & Brayton, 1996) is a temporal logic, where the state formulae are interpreted over states of a CTMC and it is a natural extension of PCTL logic.

The syntax of CSL is defined by the following rules:

$$\begin{aligned}&\phi ::= true|a|\phi \wedge \phi|\neg\phi|\mathcal{S}_{\bowtie p}(\phi)|\mathcal{P}_{\bowtie p}(\psi)\\&\psi ::= X\phi|\phi U^{\leq t}\phi\end{aligned} \tag{2.13}$$

, where a represents an atomic proposition and $p \in [0,1], \bowtie \in \{<, \leq, >, \geq\}, t \in \mathbb{R}_{\geq 0} \cup \{inf.\}$. The temporal operators X and U are called "*Next*" and "*Until*", respectively.

Here is some examples of state formulae:

$$\begin{aligned}
&s \vDash true\ for\ all\ s \in S\\
&s \vDash a\ iff\ a \in L(s)\\
&s \vDash \neg\phi\ iff\ s \nvDash \phi\\
&s \vDash \phi_1 \wedge \phi_2\ iff\ s \vDash \phi_1\ and\ s \vDash \phi_2\\
&s \vDash \mathcal{S}_{\bowtie p}(\phi)\ iff\ \lim_{t\to Inf} \mathbb{P}(\pi@t \vDash \phi|\pi[0] = s) \bowtie p\\
&s \vDash \mathcal{P}_{\bowtie p}(\psi)\ iff\ \mathbb{P}(\pi \vDash \psi|\pi[0] = s) \bowtie p
\end{aligned} \tag{2.14}$$

A path π originating from s satisfies a path formula ψ according to the following rules:

$$\begin{aligned}
&\pi \vDash X\phi\ iff\ \pi[1] \vDash \phi\\
&\pi \vDash \phi_1 U^{\leq t}\phi_2\ iff\ \exists 0 \leq j \leq t\ (\pi[j] \vDash \phi_2 \wedge (\forall 0 \leq k < j: \pi[k] \vDash \phi_1))
\end{aligned} \tag{2.15}$$

2.3.4. Extending Markov models and requirement specification logics with rewards

In this section, we represent an extension for DTMCs with the capability to specify rewards (or costs) and extend the corresponding requirement specification PCTL with the capability to specify over reward structure. A reward can be adopted for specifying additional information about the system that the analytical model describes, e.g., number of messages sent or the number of lost requests, or even the cost for consumptions of energy.

Definition 12. For a DTMC model $D = (S, s_0, P, L)$, a ***reward structure*** (ρ, ι) allows two types of reward. A state reward function $\rho: S \to \mathbb{R}_{\geq 0}$ assigns rewards to states of the model. A transition reward function $\iota: S \times S \to \mathbb{R}_{\geq 0}$ assigns rewards to transitions between states of the model. The state reward $\rho(s)$ is acquired per time-step, while a transition reward $\iota(s, s')$ is incurred each time a transition between the two states s, s' occurs.

The PCTL logic is extended by reward properties by means of the following state formulae (M Kwiatkowska et al., 2007):

$$R_{\bowtie r}(C^{\leq k})|R_{\bowtie r}(I^{=k})|R_{\bowtie r}(F\phi) \tag{2.16}$$

, where $\bowtie \in \{<, \leq, >, \geq\}, r \in \mathbb{R}_{\geq 0}, k \in \mathbb{N}$ and ϕ is a state formula.

Intuitively, the interpretation of the extended structure is as follows:

- $R_{\bowtie r}(C^{\leq k})$ is true in state s, if from the state s, the expected reward cumulated after k time-steps satisfies $\bowtie r$.
- $R_{\bowtie r}(I^{=k})$ is true in state s, if the expected state reward in the state entered at time-step k along the path originating from s meets the bound $\bowtie r$.
- $R_{\bowtie r}(F\phi)$ is true in state s, if the expected reward cumulated before a state satisfying ϕ is reached meets the bound $\bowtie r$. In order to calculate the average cost of a run of the system, we can use this construct by computing the expected cumulated cost until the execution reaches the end state.

For a more detailed description of the reward extension, we refer to (M Kwiatkowska et al., 2007).

2.4. Type-2 Fuzzy Logic

In this section, we only briefly introduce the notion of type-2 fuzzy sets and systems. In the corresponding chapter (i.e., Chapter 5), the details of each key components of such systems are discussed.

2.4.1. Type-2 fuzzy sets

The concept of *type-2* (T2) fuzzy sets (FS) was firstly introduced by Zadeh (Zadeh, 1975) and further elaborated by Mendel (JM Mendel & John, 2002; JM Mendel, 2007). This type of FSs is an extension of the ordinary ones (also known as *type-1* (T1) FS). A T2 FS is characterized by a membership function (MF, cf. Figure 2.2), which associates a FS to each elements of the set, unlike a T1 set where its MF associates a crisp number in $[0,1]$ to each element of the set. Such sets are useful in circumstances where it is infeasible to determine the exact MF. This additional dimension provides new degrees of freedom, which is useful for incorporating uncertainty (Wu, 2012).

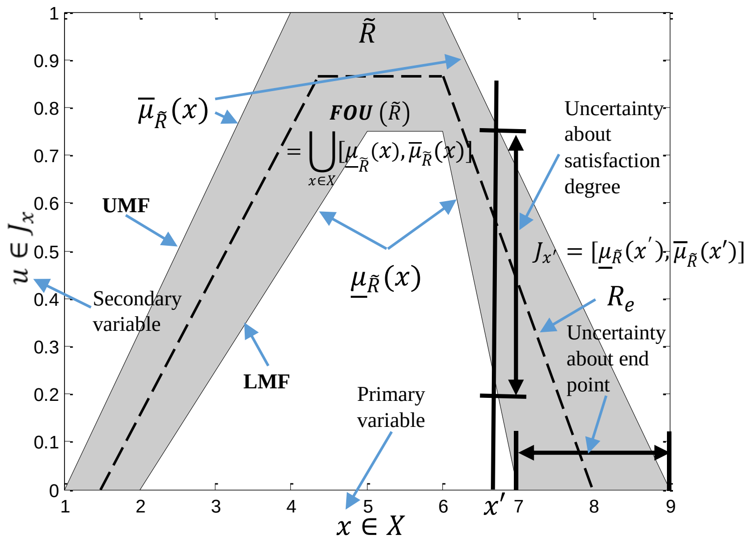


Figure 2.2. An interval type-2 fuzzy set based possibility distribution.

One may consider Figure 2.2 as the blurred version of the T1 MF by shifting the points on the trapezoid either to left or to the right. Therefore, at a specific value, x', there is not a single value, but an interval of values. These values do not necessarily have the same weight. This leads to the definition of a three dimensional MF, a T2 MF, which characterizes a T2 FS. Note all definitions in this section as well as those given in Chapter 5 are standard definitions in fuzzy theory that we borrowed from standard literature (e.g., (JM Mendel & John, 2002; Jerry M. Mendel, John, & Liu, 2006) and more specifically (JM Mendel, Hagras, & John, 2010)).

Definition 13. A ***T2 FS***, denoted by $\tilde{R}$, is characterized by a type-2 membership function $\mu_{\tilde{R}}(x,u)$

$$\tilde{R} = \{((x,u), \mu_{\tilde{R}}(x,u)) | \forall x \in X, \forall u \in J_x, \mu_{\tilde{R}}(x,u) \leq 1\} \tag{2.17}$$

When these values have the same weight, it leads to the definition of a two dimensional MF, which at a specific point, x', has a range $[0,1]$. This type of FSs are called *interval T2 FS* (IT2 FS) (**Definition 14**).

Definition 14. When all $\mu_{\tilde{R}}(x,u) = 1$ in (2.17), then $\tilde{R}$ is an ***interval T2 FS*** (IT2 FS).

Therefore, the MF of IT2 FS can be fully specified by the two T1 MFs (cf. **Definition 16**). The area between the two MFs (the grey region in Figure 2.2) characterizes the uncertainty.

Definition 15. The uncertainty in the membership function of an IT2-FS, $\tilde{R}$, is called ***footprint of uncertainty*** (***FOU***) of $\tilde{R}$, i.e.,

$$FOU(\tilde{R}) = \bigcup_{x \in X} J_x = \{(x,u) | \forall x \in X, \forall\, u \in J_x\} \tag{2.18}$$

Definition 16. The ***upper membership function*** (***UMF***) and ***lower membership function*** (***LMF***) of $\tilde{R}$ are two T1-MFs $\overline{\mu}_{\tilde{R}}(x), \underline{\mu}_{\tilde{R}}(x)$ respectively that bound the FOU.

Definition 17. An ***embedded fuzzy set*** R_e is a T1 FS that is located inside the FOU of $\tilde{R}$.
In Figure 2.2, LMF, UMF and μ_{R_e} are three embedded MFs.

2.4.2. Type-2 fuzzy logic systems

The theory of IT2 FLS is given in (JM Mendel, 2000). Here, we briefly summarize calculating the parameters that we need for adaptation reasoning process. Figure 2.3 represents the architecture of an IT2 FLS (N. Karnik & Mendel, 1999) with a rule base consisting of L rules:

$$R^l: IF\ x_1\ is\ \tilde{F}_1^l\ and \dots and\ x_p\ is\ \tilde{F}_p^l, THEN\ y\ is\ \tilde{G}^l \tag{2.19}$$

, where $\tilde{F}_i^l (i = 1, \dots, p)$ and $\tilde{G}^l$ are IT2 FSs. When a FLS receives input $X' = \{x_1', \dots, x_p'\}$, the inference engine computes the firing degree by performing a meet operation (JM Mendel, 2007):

$$\mu_{\tilde{F}_1^l}(x_1') \sqcap \dots \sqcap \mu_{\tilde{F}_p^l}(x_p') \tag{2.20}$$

Theorem 1. *In an IT2 FLS, the* ***firing interval*** *of the* $\boldsymbol{l}$*th rule is computed as:*

$$\begin{aligned} F^l &= \left[\underline{f}^l, \overline{f}^l\right] \\ \underline{f}^l &= \underline{\mu}_{\tilde{F}_1^l}(x_1') \otimes \dots \otimes \underline{\mu}_{\tilde{F}_p^l}(x_p') \\ \overline{f}^l &= \overline{\mu}_{\tilde{F}_1^l}(x_1') \otimes \dots \otimes \overline{\mu}_{\tilde{F}_p^l}(x_p') \end{aligned} \tag{2.21}$$

The proof is given in (JM Mendel, 2000). Afterwards, the type reducer transforms the fired IT2 FS to a T1 FS, which shows the possible disparity in the crisp output of the FLS. It establishes an interval around the output in the same way that a confidence interval is established for a point estimate. However, this represents linguistic uncertainties.

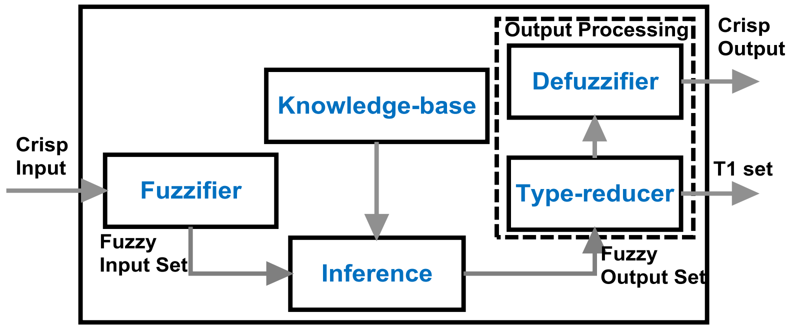


Figure 2.3. The architecture of type-2 fuzzy logic system (*adapted from* (JM Mendel, 2000)).

A general T2 FLS has a high computational complexity (N. Karnik & Mendel, 1999), but the calculations simplify dramatically when we use IT2 FSs in the rules. Therefore, IT2 FLSs are better suited for runtime efficient computation in the adaptation reasoning in self-adaptive software.

2.5. Reo Component Connectors

A component connector, in the context of this research, corresponds to a coordination pattern (Arbab, 2004; N Oliveira & Barbosa, 2013; Nuno Oliveira & Barbosa, 2013) on architectural elements (e.g. components) that performs I/O operations through that connector. In other words, here, the term connector is adopted to name entities that can regulate the interaction of (potentially) heterogeneous components. Thus, connectors must deal with exogenous coordination, handling all those aspects that lie outside the scopes of individual components (Bruni, Melgratti, & Montanari, 2013). A coordination pattern is formally given as a graph of *channels* whose nodes represent the points for interactions between channels. The edges of this graph are represented with channel types and channel identifiers. To provide a concrete illustration of this approach, we utilize Reo coordination model (Arbab, 2004). Therefore, a channel is considered here as a Reo channel (Arbab, 2004).

In the Reo model, channels are primitives out of which more complex and composite component connectors are constructed. A connector channel is directional (except one channel type) with a unique identifier and specific semantics (i.e. coordination protocol). A channel in this model accepts an I/O operation (data flow) on its *source end* and dispenses it from its *sink end*. Figure 6.2 illustrate the basic channel type in the Reo coordination model. Note that Reo support an *open-ended set of channels* (Arbab, 2004), each exhibit a unique behavior with a precise and distinguishing semantic. However, for the purpose of this research, we only consider the construction of component connectors based on the primitive channels, represented in Figure 6.2.

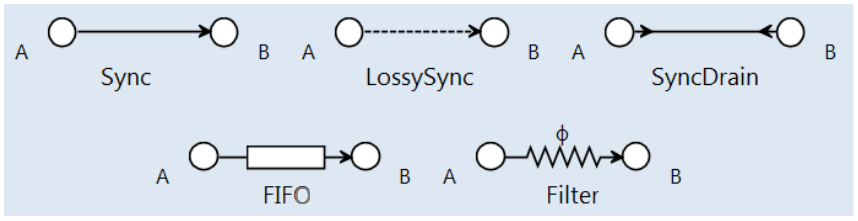


Figure 2.4. Primative connector channels.

2.5.1. Stochastic Reo

Stochastic Reo (Moon, Arbab, & Silva, 2011) extends Reo in a way that channels are annotated with stochastic values representing distributions of their relevant data-flow and request arrival at the channel ends. These distributions are respectively referred to as processing delay rates and request arrival rates. Such stochastic values are non-negative real values and describe the probability of a certain value of a discrete random variable, or similarly intervals that represent continuous random variables. Figure 2.5 shows some primitive channels of Stochastic Reo that correspond to the primitives of Reo in Figure 6.2. In this figure, γa means the arrival rate at node a and similarly, γab means channel delay between two nodes a and b. We describe these concepts later in Chapter 6 in more detail.

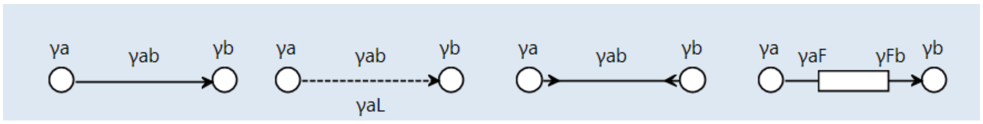


Figure 2.5. Primitive connector channels with stochastic anotations.

Note that the annotations do not change the semantics of Reo connectors, thus, when the rates are ignored, the semantics of Reo connectors and stochastic counterparts are the same. The labels annotating Stochastic Reo channels can be categorized into the following two groups:

Channel delays. A delay rate represents the duration that a channel takes to perform a certain activity such as transferring a data item from one end to the other end. For instance, a $LossySync$ has two associated variables γab and γaL for the stochastic delay rates of, respectively, successful data-flow from node a to node b, and losing the data item at node a when a read request is absent at node b. In a $FIFO$ channel, γaF means the delay for data-flow from its source node a into the buffer, and γFb means the delay for sending the data from the buffer to the sink node b. Similarly, γab of a $Sync$ (and a $SyncDrain$, respectively) indicates the delay for data-flow from its source node a to its sink node b (and losing data at both ends, respectively).

Arrivals at nodes. Arrival rates describe the time between consecutive arrivals of requests at source and sink nodes of Reo channels. For instance, γa and γb in Figure 2.5 represent the associated arrival rates of write/take requests at nodes a and b, respectively. Note that at most one request can wait at a boundary node for acceptance. That is, if a boundary node is occupied by a pending request, then the node is blocked and consequently all further arrivals at that node are lost.

Stochastic Reo supports the same compositional framework of joins of connectors as Reo (Moon, 2011). The nodes in Stochastic Reo have certain quality attributes associated with them. Therefore, joining nodes must accommodate quality property composition. A mixed node delivers data items instantaneously to the source end(s) of its connected channel(s). Therefore, mixed nodes have no associated arrival rates. However, arrival rates on nodes model their interaction with the environment.

Chapter 3

3. State-of-the-art

"If I have seen farther than others, it is because I was standing on the shoulder of giants."– Isaac Newton (1643-1727).

Contents

3.1. Chapter Overview

In this chapter, various approaches covering uncertainty control in self-adaptive systems are investigated. This chapter summarizes related approaches and particularly exposes how the contribution of this thesis for controlling the uncertainty for self-adaptive component connectors makes advances in the state of the art. Note that this chapter only focuses on work that specifically proposes a framework for addressing the issues that uncertainty introduces in self-adaptive software. However, the related work considering the individual contribution chapters (i.e., model adjustment techniques in Chapter 4; adaptation reasoning approaches in Chapter 5; and change execution mechanisms in Chapter 6) is covered in their respective chapters.

Section 3.2 provides a number of comparison criteria of related approaches. Then, Section 3.3 presents a demarcation and detailed description of each related research work. Finally, a systematic comparison of the related work according to the comparison criteria is given in Section 3.4.

3.2. Comparison Criteria

In this section, a set of comparison criteria for an objective comparison of related research is discussed. Figure 3.1 illustrates an overview of a self-adaptive software system consistent with the FORMS reference model (Danny Weyns, Malek, & Andersson, 2010). Based on this reference model, the self-adaptive system can be decoupled into two separate subsystems: *Meta-Level* and *Base-Level*. The base-level subsystem provides the application behavior, while the meta-level subsystem controls the base-level subsystem by adapting its behavior. At the meta-level, we use the IBM reference model for autonomic systems called MAPE-K (JO Kephart & Chess, 2003). There are also two other entities in Figure 3.1. *Users* use the functionalities of the system and specify adaptation logic and the *environment* within which the software system operates.

The different elements as depicted in Figure 3.1 are loosely coupled. The meta-level depends on models of other elements to decide about the adaptation of the base-level system. The loose coupling between the meta-level and the other elements in self-adaptive software are either unavoidable (e.g., system and environment) or essential to provide flexibility, reusability and managing the complexity of constructing self-adaptive software systems. In fact, this loose coupling between the meta-level and the other elements (i.e. Environment, User, and Base level) is the origin of uncertainty in self-adaptive software. A comprehensive number of sources of uncertainty is enumerated and discussed in (Esfahani & Malek, 2013). In this section, we recall them and extend them in order to compare the related work and to identify the research gap in the state-of-the-art.

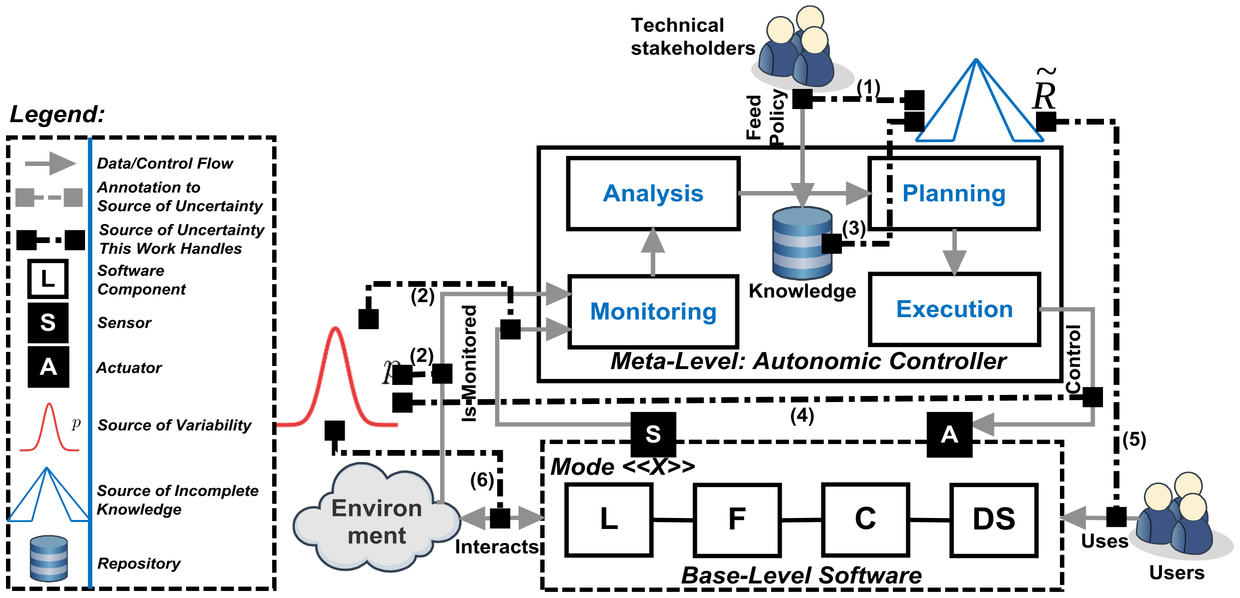


Figure 3.1. Sources of uncertainty in self-adaptive software.

As depicted in Figure 3.1, uncertainty exists in every aspect of the self-adaptation process (Esfahani & Malek, 2013): (1) Stakeholders often have *conflicting preferences* over expressing adaptation policies. (2) Monitoring facilities receive *noisy data* from sensors. (3) Analytical models make *simplifications* for quality assessment. (4) Adaptation facilities may not execute changes correctly or may enact the changes with a latency. (5) Users may not use the system as it is expected. (6) The environment is inherently dynamic and unpredictable.

We now describe these sources of uncertainty referring to the numbers that have been used to annotate different parts of Figure 3.1:

Uncertainty in the expression of adaptation policies [annotation (1)]. This type of uncertainty is related to the "Feed Policy" arrow in Figure 3.1. This uncertainty exists because of the difficulties in expressing requirements and preferences that need to be elicited from users. Users have multiple and sometimes conflicting concerns. The elicitation of user concerns is a challenge (Lemos et al., 2013). Therefore, user preferences in terms of mathematical functions are subjective and analysis based on them is prone to uncertainty.

Uncertainty because of noisy data [annotation (2)]. The data that feeds the meta-level (see the left part of Figure 3.1) is not free of noise because of the errors in the employed sensors. As a result, the input data is not a single value but a distribution of values obtained over time. Therefore, the analysis in the meta-level should explicitly consider this measurement noise. Otherwise, the adaptation decision is prone to uncertainty.

Uncertainty due to simplification of assumptions [annotation (3)]. The analytical models, which are employed for analytical activities (e.g., reasoning based on the impact of adaptations on the system quality attributes) at the meta-level, are mathematical models. These mathematical models can become inaccurate because of the errors in estimation of some parameters in the model. Sometimes the assumptions based on which the model is designed are not upheld at runtime. These inaccuracies, which depend on the circumstances, make the analytical models an inaccurate representation of the real system and as a result, the reasoning based on them becomes error prone.

Uncertainty because of change enactment [annotation (4)]. Some changes that are issued by the execution module may not be enacted exactly as it is requested. Therefore, the models for reasoning in the meta-level become inconsistent representations of the real systems. This makes reasoning based on the inconsistent models prone to uncertainty. In some execution environments, the change enactment is not instant and contains a latency. This time latency may also change due to a number of reasons, leading to a more intricate source of uncertainty (Jamshidi et al., 2014).

Uncertainty because of humans in the control loop [annotation (5)]. Human behaviors are uncertain (David Garlan, 2010). However, modern software systems become ubiquitous and more dependent on user behavior. This creates uncertainty in the software system they use.

Uncertainty in the environment [annotation (6)]. Self-adaptive software systems are used in many different environments. Environments themselves are inherently dynamic and unpredictable events may happen.

Note that in addition to the above-enumerated sources of uncertainty, we also consider the techniques that existing works apply in the feedback control loop (i.e., throughout MAPE-K activities) and their evaluation approach (cf. Table 3.1).

3.3. Existing Frameworks for Controlling Uncertainty

The software engineering research community has made progress towards addressing the complexities involved in the construction of self-adaptive software (Lemos et al., 2013). However, as reported by a community-wide roadmap (Lemos et al., 2013) and reviews of uncertainty handling techniques (Esfahani & Malek, 2013; A. J. Ramirez et al., 2012), there is still a lack of methods and techniques for handling uncertainty in self-adaptive software. In the self-adaptive software community, a few researchers have recently proposed to address uncertainty issues related to different aspects in self-adaptive software. In general, we can categorize these proposals into the following categories: requirements specifications, internal uncertainty, external uncertainty, design-time uncertainty, and control theory. We also structure this section according to this classification.

3.3.1. Requirement specification uncertainty

RELAX (Whittle et al., 2009) is a requirements specification language that incorporates uncertain requirements in self-adaptive systems. RELAX allows designers to explicitly express environmental uncertainty in requirements. More concretely, RELAX defines a number of operators that can be used in defining requirements and making them "disabled" or "relaxed" at runtime depending on the state of the environment. Additionally, the operators are able to capture the uncertainty factor that can initiate the relaxation of requirements.

The RELAX language is extended with goal modeling to specify the uncertainty in the objectives in (B. Cheng, Sawyer, Bencomo, & Whittle, 2009). The sources of uncertainty with the help of threat modeling in goal models can be identified. More specifically, threat modeling helps to identify the environmental elements which can endanger the satisfaction of goals. Once the uncertainty is identified, mitigation tactics are devised. The ultimate tactics for mitigating uncertainty are enabled by relaxing the goal that is prone to uncertainty.

While RELAX is a specification language for identifying and assessing sources of uncertainty, AutoRELAX (E. Fredericks, DeVries, & Cheng, 2014) is an approach that automatically generates RELAXed goal specifications. More concretely, AutoRELAX identifies goals to RELAX through specific operators and by determining the shape of the membership function that establishes the goal satisfaction criteria. AutoRELAX generates solutions by making tradeoffs between minimizing the number of RELAXed goals and maximizing functionality by reducing the number of adaptations triggered by environmental conditions.

FLAGS (Luciano Baresi et al., 2010) uses fuzzy theory to mitigate the environmental uncertainty by enabling the specification of adaptive goals. FLAGS enables the definition of tactics that must be taken if some goals are not satisfied. FLAGS also deals with the uncertainty in goals themselves. More specifically, FLAGS relies on a fuzzy temporal language to specify imprecise goals that some temporary violations are tolerated. It also allows for the specification of crisp goals through linear temporal logic.

Goal-Driven Self-Optimization. Chen et al. (Chen, Peng, Yu, & Zhao, 2014) propose to handle the uncertainties in goal models comprising contribution, preference and effect uncertainty. Taking the quality indicator as the feedback and the estimated earned value as the global indicator of self-optimization, the proposed framework dynamically updates the quantitative contributions from alternative functionalities to quality requirements, tunes the preferences of relevant quality requirements, and determines a proper timing delay for the last adaptation action to take effect. Then, they apply these runtime measures to limit the negative effect of the uncertainty in goal models.

REAssuRE (Kristopher Welsh, Sawyer, & Bencomo, 2011b) enables the specification of the rationale for a choice of alternative goal operationalization when there is uncertainty about the optimal choice by attaching claims to the contribution links. In addition, Ramirez et al. (A. Ramirez & Cheng, 2012) integrated REAssuRE with RELAX to introduce a fuzzy logic layer upon the evaluation criteria of claim validity. When a claim is violated at runtime, its attached contribution link is updated and the goal model is re-evaluated to discover a better solution. Hence, these studies presented a qualitative way to handle the contribution uncertainty.

Bencomo and Belaggoun (N Bencomo & Belaggoun, 2013) proposed to map goal models to dynamic decision networks (DDNs), where each contribution link corresponds to a conditional probability and each configuration is associated with a preference. As soon as the validity of a claim is changed, the relevant conditional probabilities will be updated at runtime and the DDN model will be re-evaluated to find a configuration with the highest utility. Note that experts give the conditional probabilities and preferences and only some of the conditional probabilities will be updated.

In summary, this category of research intends to adopt the concept of partial satisfaction of requirements at design-time and to provide a resolution mechanism at runtime. This category of research, in general, uses the notion of claims and their refinements as the marker for uncertainty at design-time and uncertainty resolution at runtime (Kristopher Welsh, Sawyer, & Bencomo, 2011a). It monitors claims at runtime in order to verify their satisfaction. If a violation is detected, it changes the system's goal model to select an alternative goal realization. This allows for the dynamic adaptation of the system in the presence of uncertainty.

3.3.2. Internal uncertainty

Cheng and Garlan (S. Cheng & Garlan, 2007) proposed high-level uncertainty mitigation strategies for their architecture-based self-adaptation framework, which is called *Rainbow* (D Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004). The proposed mitigation strategies are about three specific sources of uncertainty in the MAPE-K control loop: 1) detecting when there is a violation in the system, 2) determining the right adaptation policy, 3) knowing whether a given adaptation achieved its intended effects. The first one is related to Monitoring and Analysis activities of the MAPE-K loop, whereas the second and third ones are respectively related to Planning and Execution activities. More specifically, they intend to mitigate the uncertainty in the activities of the feedback control loop.

In order to mitigate the uncertainty in the first source, they employ probability theory to determine the running average in monitoring to stand against the variability in the environment. The data is then compared with probabilistic information in the architectural description of the system. Once any problem is detected, a mitigation strategy is then selected to resolve it. The Stitch language helps to mitigate the uncertainty in the strategy selection. Stitch allows explicit modeling of the uncertainty in strategies. As a result, when the Rainbow framework decides a strategy, it can select it based on the expected value, which is a representative of the underlying uncertainty. For the last source of uncertainty, they consider how to deal with it by specifying how long the framework should monitor the implementation of the strategy before committing the change to the running system.

POISED (Esfahani et al., 2011) is a quantitative approach to handling the challenges posed by uncertainty in making adaptation decisions. POISED adopts fuzzy theory for assessing the positive and negative consequences of uncertainty. They proposed a novel approach for finding an optimal solution that has the best range of possible behaviors with regard to the system's utility. POISED aims at improving the quality attributes of software systems through reconfiguration in order to achieve an optimal configuration. However, POISED redefines the traditional definition of optimal adaptation from point estimations to the one that has the best range of behavior. In turn, the selected configuration has the highest chance of satisfying the quality objectives, albeit due to uncertainty, properties cannot be confirmed 100% accurately. POISED uses Possibilistic Linear Programming to make the tradeoff between alternatives. The decision-making problem is based on a coherent representations of uncertainties allows the specification of important aspects of uncertainties in the eye of decision makers. For example, in one scenario one might prefer to have a solution that guarantees certain limits in the worst-case scenario. However, in other scenarios, one may prefer a solution with a higher risk but potential higher quality.

ADC (Anticipatory Dynamic Configuration) (V Poladian, Sousa, Garlan, & Shaw, 2004) facilitates selecting appropriate services to accomplish a task and allocate resources among these services at runtime. This work does not consider environmental uncertainty. In a subsequent work (Vahe Poladian et al., 2007), they extend the original work in order to incorporate anticipatory decisions and considered the inaccuracy of future resource usage. They used the work in (Narayanan & Satyanarayanan, 2003) and utilized profiling data to find requirements regarding resources for different configurations. By incorporating resource availability prediction, the ADC framework chooses a configuration that optimizes the cumulative expected value of utility over time. This significantly reduces the number of changes and the disruptions in the system. For the adaptations, the cost of configuration switching is also considered in the work. If the cost is low, ADC selects a better configuration. On the other hand, if the cost is high, a non-optimum configuration is selected.

Camara et al. (Cámara, Moreno, & Garlan, 2014) proposed a formal analysis technique based on model checking of stochastic multiplayer games that enables quantification of the potential benefit of considering adaptation tactic latency in adaptation mechanisms. They conclude that explicit involvement of this source of uncertainty, i.e., adaptation latency, in adaptation reasoning improves the outcome of adaptation.

In summary, this category of research intends to mitigate the effects of internal uncertainty, which is rooted in the difficulty of determining the impact of system change on the quality properties, e.g., determining the impact of replacing a component on the systems response time, energy usage, etc.

3.3.3. External uncertainty

FUSION (Elkhodary, Esfahani, & Malek, 2010) uses machine learning, Model Tree Learning (MTL), to self-adapt the behavior of the system to unanticipated changes. This approach allows the system to mitigate the uncertainty associated with the change in the environment as it progressively learns the right adaptation in new contexts. The output of learning consists of a number of relationships between the adaptation actions and the quality attributes of the system. The quality attributes can be derived based on measurements by instrumenting the software provided by the underlying runtime platform. The adaptation actions are associated with variation points in the software that can be applied at runtime.

FUSION has two cycles, a learning cycle and an adaptation cycle that complement each other. In the learning cycle, the relationships between quality attributes of the system and the adaptation actions are learned through measurements. In this cycle, the errors in the learned relations are also detected. As soon as the quality factors of the software decreases below a certain threshold, the adaptation cycle make an informed adaptation based on the learned knowledge.

RESIST (Cooray et al., 2010) focuses on the reliability of the system by monitoring internal and external properties, changes in the structure as well as contextual properties to continuously refine reliability measurements at runtime. The updated reliability measures are then used to decide about configuration changing in order to improve its reliability. The target domain of RESIST is mobile, embedded and pervasive software. These systems are highly dynamic and face unknown contexts and fluctuating conditions. They are typically mission-critical systems and require high reliability. RESIST mitigates the uncertainty through constant learning.

RESIST measures component level reliability by learning the unknown parameters of Discrete Time Markov Chains (DTMC). As soon as the reliability measure at the individual component level is measured, a compositional model is adopted to determine the reliability index at the system level. RESIST models the uncertainty in the learning process by probability theory.

ADAM (Carlo Ghezzi, Pinto, Spoletini, & Tamburrelli, 2013) supports adaptation aimed at mitigating non-functional uncertainty. ADAM relies on Markov Decision Processes (MDPs) to model alternative and optional functionality implementations in self-adaptive software. According to the aggregated quality metrics, ADAM can find the execution path with the highest probability to satisfy the non-functional requirements and then enable adaptation by switching to alternative implementations. They do not consider the inaccuracy of the transition probabilities in such MDP models.

KAMI (A Filieri et al., 2012) enables continuous verification of reliability and performance requirements of self-adaptive systems by exploiting analytical models such as Discrete-Time Markov Chains (DTMCs) and

Continuous-Time Markov Chains (CTMCs) respectively. KAMI can update the model parameters of DTMCs and CTMCs through Bayesian estimation based on runtime observations. KAMI employs model checking in order to check the satisfaction of non-functional requirements and triggers adaptations accordingly. Similarly, Sykes et al. (Sykes et al., 2013) proposed to enable self-adaptive systems to cope with incomplete and inaccurate knowledge by updating their behavior models. They use a probabilistic rule learning technique to not only update transition probabilities, but also discover new structures of the model. These studies aim at updating the models used for the knowledge base of planning.

Veritas (E. M. Fredericks, DeVries, & Cheng, 2014)/Loki (A. J. Ramirez, Jensen, Cheng, & Knoester, 2011) uses utility functions to adapt test cases as part of a runtime MAPE-T framework (E. M. Fredericks, Ramirez, & Cheng, 2013). More specifically, Veritas adapts test cases at runtime to ensure that the adaptive software can run reliably in the presence of environmental uncertainty. Veritas monitors an adaptive system, generates an appropriate test plan, verify the test cases, and adapts test cases as necessary. Veritas adopts the Loki framework (A. J. Ramirez et al., 2011) to generate unique system and environmental configurations.

In summary, this category of research aims at mitigating external uncertainty that comes from the environment or domain in which the software is embedded (Esfahani & Malek, 2013). For example, external uncertainty for a software system deployed in a cleaner robot may include the likelihood of colliding with certain objects. Software self-adaptation is one approach in dealing with the effects of external uncertainty, e.g., in a dirty room the cleaner robot navigator component may be replaced with a more conservative navigator to avoid a collision. Therefore, appropriate techniques have been developing to minimize the effects of such external uncertainty in the self-adaptive community.

3.3.4. Design-time uncertainty

The issue of uncertainty control at design-time in requirements engineering has been proposed for requirements elicitation, disambiguation and inconsistency checks. MAVO (Famelis, Salay, & Chechik, 2012; Salay, Chechik, & Horkoff, 2012) uses partial models to manage requirements uncertainty. Yang et al. (H. Yang, De Roeck, Gervasi, Willis, & Nuseibeh, 2012) proposed an approach to detect the uncertainty in natural language requirements. Arora et al. (Arora, Sampath, & Ramesh, 2012) focused on the uncertainty arising from inconsistent feature interactions.

Letier and van Lamsweerde (Letier & van Lamsweerde, 2004) proposed to specify partial degrees of goal satisfaction and quantify the impact of alternative designs on the degree of goal satisfaction for guiding requirements elaboration and design decision making. The partial degree of goal satisfaction is modeled by an objective function on quality variables, and probabilistic models specify the objective function. This study uses a probabilistic technique to tackle goal satisfaction uncertainty at design-time, while we focus on runtime handling of contribution uncertainty, preference uncertainty and effect uncertainty.

GuideArch (Esfahani, Malek, & Razavi, 2013) quantitatively guides the exploration of the architectural solution space, including ranking the candidate architectures, finding the optimal architecture, and identifying the critical decisions, under the uncertain impact of architectural alternatives on properties of interest. The GuideArch framework defines a utility score for each candidate architecture using fuzzy membership function. This study employs fuzzy logic to represent and reason about uncertainty.

EAGLE (M Autili, Cortellessa, & Ruscio, 2012; Marco Autili et al., 2011) is a model-based framework that embrace the incompleteness and inaccuracy of the models with respect to the system goals. EAGLE supports the explore-integrate-validate adaptation loop that will be realized by exploiting model-driven techniques such as statistical inference, machine learning techniques, connector synthesis (Inverardi, Issarny, & Spalazzese, 2010) and goal verification. This integrated framework will support the engineering of software systems that are built by integrating, under uncertainty, existing components and that are dynamically evolving within a changing environment.

Leitier et al. (Letier, Stefan, & Barr, 2014) proposed an approach to apply multi-objective optimization techniques for evaluating uncertainty and its impact on system goals before making critical decisions. They enable software architects to describe uncertainty about the impact of alternatives on system goals; to calculate the consequences of uncertainty; to select candidate architectures; and to assess the value of obtaining additional information before making a decision. Their work is closely aligned to the GuideArch framework (Esfahani, Malek, et al., 2013). Although they differ in their decision analysis techniques, these two approaches reached the same conclusion about the consequences of handling uncertainty: “modelling uncertainty and mathematically analyzing its consequences leads to better decisions than either hiding uncertainty behind point-based estimates” (Letier et al., 2014).

In summary, these design-time approaches more or less require user involvement, making them infeasible to be applied in an unsupervised self-adaptation process. This category of research, although proposing a mechanism for adaptation, does not concentrate on Monitoring or Execution part of MAPE-K loop. However, the self-adaptive community can utilize the insights that have been produced by this category of research to introduce novel approaches for controlling and minimizing the effects of uncertainty.

3.3.5. Control theory for handling uncertainty

Apart from the approaches listed above, there are other approaches targeting uncertainty in order to make dependable self-adaptive software by applying the principles of control theory. The quantitative (or measurement-driven) adaptation has been studied for decades in control theory (Antonio Filieri, Hoffmann, & Maggio, 2014). One major benefit of using control theory in this context is the guarantee of control properties that can be proved mathematically. In this paradigm, adaptive software can be treated as a controllable plant allowing control theory to be applied to enable self-adaptation. Control theory is capturing increasing interest from the software and systems engineering community (Hellerstein, Diao, Parekh, & Tilbury, 2004; Jamshidi et al., 2014; Zhu et al., 2009). The application of control theory in software engineering, however, is still at a very preliminary stage (Patikirikorala, Colman, Han, & Wang, 2012) and is limited to the design of controllers focused on particular ad-hoc solutions that address a specific computing problem. Filieri et al. (Antonio Filieri et al., 2014) developed a general methodology, which reduces the need for strong mathematical background to devise ad-hoc control solutions.

The main difference between the existing control theory approaches and our approach is that the fuzzy logic controller we employed can handle expert knowledge and numerical data in a unified framework, and the fuzzy-based approach, in general, has less computational complexity. The other benefit of our approach is that the fuzzy logic controller does not require the mathematical model of the plant that it controls. In this work, deriving an accurate mathematical model of the underlying software is a very difficult task due the non-linear dynamics of real systems (Esfahani, Elkhodary, et al., 2013; Hellerstein et al., 2004; Lemos et al., 2013; Zhu et al., 2009).

3.4. Discussions and Conclusions

Uncertainty is a critical challenge in the construction of self-adaptive software and it needs to be taken into account specifically when the dependability of the system is important. This hinders the widespread adoption of self-adaptive software in practice. However, as reported by others (Esfahani & Malek, 2013; Lemos et al., 2013; Perez-Palacin & Mirandola, 2014), there is a shortage of applicable techniques for controlling the effects of uncertainty in this setting. As also discussed in (Esfahani & Malek, 2013), there is a need for development of appropriate mechanisms for mitigating the uncertainty underlying self-adaptation of software that is prone to uncertainty. Only a few researchers have recently begun to address uncertainty (Esfahani & Malek, 2013). Table 3.1 summarizes their work with regard to the sources of uncertainty they are dealing with.

According to the approaches positioned in Table 3.1, three areas specifically lack mature mechanisms for controlling the effects of uncertainty: I. noisy data, II. change enactment, and III. user involvement. However, the areas related to I. adaptation policy specification and II. dynamic environments are quite mature with several numbers of mechanisms for controlling the effects of uncertainty.

In the key chapters of this thesis that the core contribution of our work is described (i.e., Chapter 4 on model calibration, Chapter 5 on adaptation reasoning and Chapters 6 and 7 on adaptation execution and real-world applicability), we properly positioned our approach. In those particular chapters, we mentioned some of the concerns that distinguishes our work from existing approaches. However, it is useful to summarize the main characteristics that make this thesis a novel research considering the frameworks that we summarized in Table 3.1. In general terms, the most crucial differences that distinguish our approach from other approaches that have appeared in the literature are:

1. Our approach considers the incomplete and noisy monitoring measurements (*aleatory* uncertainty, cf. Section 2.2.1). Our approach is concerned with calibrating analytical models at runtime in the presence of uncertainty in the input data. This distinguishes it from approaches (cf. seventh column of Table 3.1) that consider only complete data or approaches that consider noise-free data.
2. Our approach captures the uncertainties associated with users' incomplete knowledge regarding system adaptations policies using fuzzy logic (*epistemic* uncertainty, cf. Section 2.2.1). Our approach enhances self-adaptive software with adaptation reasoning that can robustly control the environmental noises. This distinguishes it from approaches (cf. second column of Table 3.1) that assume the problem of conflicting subjective measures from a group of experts is solved elsewhere.

Table 3.1. Literature comparison addressing source of uncertainty and the activities they cover in the feedback control loop.

	Framework	Source of Uncertainty						Feedback Control Loop (MAPE-K)					Evaluation
		Adaptation policy	Noisy data	Simplification	Change enactment	Users in the loop	Dynamic environment	M	A	P	E	K	
Requirement Specification	RELAX	Fuzzy										goal model	Case study
	AutoRELAX	Fuzzy										goal model	Experimental study
	FLAGS	Fuzzy										goal model	Example
	Goal-Driven Self-Optimization	Prob.		Prob.	Prob.				goal reasoning			goal model	Experimental study
	REAssuRE	Fuzzy							goal reasoning			goal model	Example
	(N Bencomo & Belaggoun, 2013)	Prob.							goal reasoning			Decision model	Experimental study
Internal	Rainbow		Prob.	Prob.					√	constraint evaluation	√	Architecture model	Experimental study
	POISED	Fuzzy	Fuzzy	Fuzzy						optimization		Architecture model	Experimental study
	(Cámara et al., 2014)				Prob.					game analysis		Architecture model	Experimental study
	ADC						Prob.		utility reasoning			Utility	Case study
External	FUSION			Prob.			Prob.	√ (learning)	√			Feature model	Experimental study
	RESIST		Prob.	Prob.			Prob.	√ (learning)	√			Markov models	Experimental study
	ADAM						Prob.	√ (learning)	√			Markov models	Experimental study
	KAMI						Prob.	√ (learning)	constraint evaluation			Markov models	Experimental study
	Veritas/Loki						Prob.		test case verification	test plan verification; optimization		Test cases	Experimental study
Control	(Antonio Filieri et al., 2014)						Control			controller synthesis		Regression models	Experimental study
	(Zhu et al., 2009)						Control			integral controller		Regression models	Experimental study
Design-time	GuideArch	Fuzzy (utility)						--		Optimization (arch. selection)	--		Case study
	EAGLE						Prob.	--	Goal verification	Synthesis	--		Example
	MAVO							--	Partial model reasoning		--		Case study
	(H. Yang et al., 2012)							--	Machine learning	Rule reasoning	--		Experimental study
	(Arora et al., 2012)							--	Feature interaction		--		Case study
	(Letier & van Lamsweerde, 2004)						Prob.	--	Partial goal verification		--		
	(Letier et al., 2014)							--	Monte-Carlo simulation	Pareto-based optimization	--		Experimental study
C/E/I	RCU (This Work)	Fuzzy	Prob.				Control	√ (Bayesian learning)	constraint evaluation	Fuzzy reasoning	Mode change	Markov models + Fuzzy rule	Experimental study

Chapter 4

4. Robust Model Calibration for Requirement Verification

"It is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of the subject admits, and not to seek exactness when only an approximation of the truth is possible." Aristotle (384- 322 BC).

Contents

4.1. Introduction

Component connectors are increasingly adopted as a paradigm for building composite connectors facilitating coordination and interaction between functional components of software systems (Arbab, 2004). These composite structures are built by composing and integrating individual coordination channels known as primitive connectors. Consequently, these channels can be executed and managed by third-party providers. The providers can offer channels with different quality of service (QoS), therefore the capabilities and quality of coordination among components will depend on the quality of third-party channels. In other words, component connectors should become robust and resilient against the failure of third-party channels. However, the environment surrounding the connectors, comprising functional components and the amount of requests from them, also affect the quality of coordination.

In order to cope with managing the interaction in highly dynamic and unpredictable environments, fraught with uncertainty, the coordination infrastructure needs to exploit adaptive capabilities. We consider self-adaptive capabilities as a necessary runtime obligation to ensure robustness and resilience against third party channels failure and environmental fluctuations. In order to enable self-adaptive connectors, the feedback control loop, known as MAPE-K loop (cf. Figure 4.1), needs to be realized. One of the tasks involved in the MAPE-K loop that needs to be realized to accommodate the Analysis activity of this loop is continual verification of the non-functional properties (NFPs) (Calinescu et al., 2012) of such connectors. Since the approach of this thesis for self-adaptation is white-box, i.e., using runtime models to enable such adaptation, the challenge of continual verification of NFPs boils down to the estimation of unknown parameters of the analytical models and then formal evaluation of the properties. This choice is motivated by the fact that the current formalisms that are used to specify the underlying behavior of component connectors, e.g., constraint automata, are inherently state-based and there are available tools to transform such formalism to the Markovian models, i.e., CTMC, that we use in this thesis (Moon, 2011). In this thesis, we call the former activity model calibration. The main outcome of this chapter is a model calibration method that is robust against uncertainties regarding input data, comprising noise and incomplete observations. Note that for the non-functional requirement verification, we adopt the runtime efficient approach that is proposed in (Antonio Filieri, 2013).

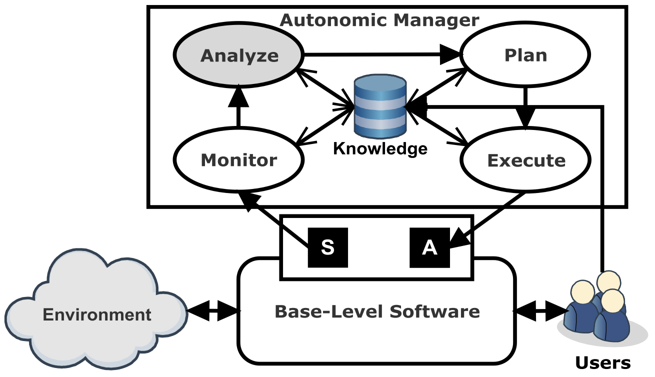


Figure 4.1. Scope of Chapter 4.

Online model calibration in different settings, comprising (I) full observation, (II) partial observations and (III) partial observation with measurement noise, are employed to tune the model at runtime as depicted in Figure 4.2. The updated model is then used to detect the violation of requirements. The detection of a violation may then trigger an adaptation, which should be Planned and Executed to adapt the running connector.

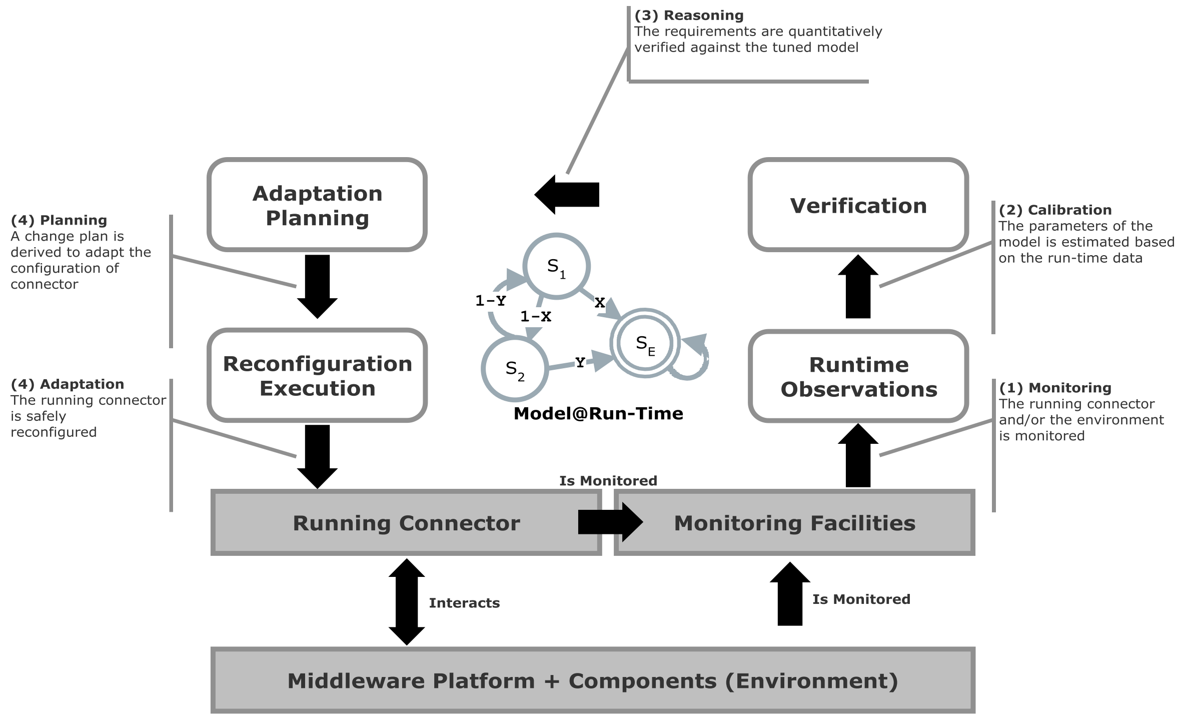

(3) Reasoning
The requirements are quantitatively verified against the tuned model
Adaptation Planning
Verification
(4) Planning
A change plan is derived to adapt the configuration of connector
(2) Calibration
The parameters of the model is estimated based on the run-time data
S1
1-Y
1-X
X
SE
S2
Y
Reconfiguration Execution
Runtime Observations
(4) Adaptation
The running connector is safely reconfigured
(1) Monitoring
The running connector and/or the environment is monitored
Model@Run-Time
Is Monitored
Running Connector
Monitoring Facilities
Interacts
Is Monitored
Middleware Platform + Components (Environment)

Figure 4.2. Overview of the self-reconfigurable component connector.

4.1.1. Problem statement and contributions

Let us consider a situation where the behavior of component connectors is specified by a mathematical model (cf. the model at the heart of Figure 4.2). This mathematical model, which corresponds to a connector, contains some parameters. Now, we can consider some scenarios regarding the parameters. Parameters can be constant over time and known, leading to a *time-invariant model without uncertainty*. Let us also imagine a situation where the parameters are constant over time, but only known as a rough estimations, providing a *time-invariant model with uncertain parameter values*. We can also consider a situation, where parameters can change over time, resulting in a *time-varying model*. In this research, we consider the last two scenarios. More specifically, in the case of uncertain and varying parameters, we develop mechanisms to estimate their current values on the fly, based on the available "uncertain" measurements. The latter part of the last sentence is critical because this is where our contribution lies. Unlike the existing approaches for parameter estimation, we do not assume that available measurements are perfect. We rather assume that they may be incomplete or noisy. While there are different sources of uncertainty in self-adaptive software (Esfahani & Malek, 2013), the incomplete and noisy runtime measurements are the sources that we consider to tackle in this chapter. It is important to understand that the incarnation of uncertainty is different from adaptation reasoning which we address in Chapter 5.

The objective of model calibration is to estimate the unknown parameters of the model based on runtime observations as depicted in Figure 4.3. This estimation of parameters should be accomplished accurately and at the right time. Informally, it means that the estimation should detect as many violations as possible as soon as the actual value of a parameter enters a violation zone. Accurate estimations are important to avoid the execution of unnecessary adaptations. In addition, estimation at the right time is important in order not to miss adaptation opportunities resulting from lagging behind the actual value.

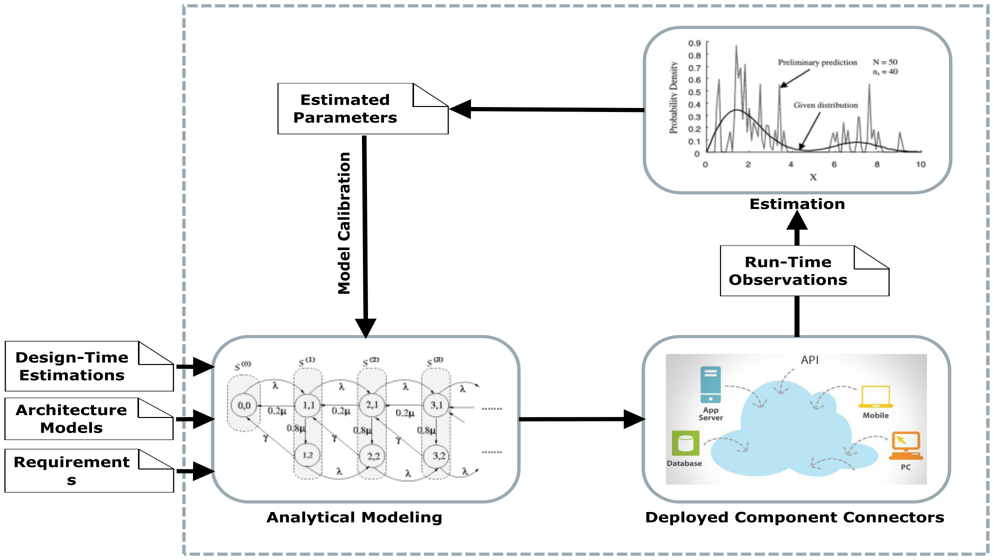


Figure 4.3. Overview of Model Calibration.

As a key contribution of this work, this chapter focuses on the following three important scenarios, which concerns robust model calibrations for reconfigurable component connectors:

1. **Model calibration with full observations**: In this case, the unknown parameters of the model at runtime are estimated based on a full observation of the runtime connector. It means that for the discrete-time models it is assumed that the full discrete time series of data is available, and for the continuous-time models, it is assumed that continuous observations are available.
2. **Model calibration with partial observations**: In this case, incomplete runtime data is monitored and collected for parameter estimation. It informally means that some components of the model are unobserved and we have only partial observations.
3. **Model calibration with incomplete and noisy observations**: In this case, not only a partial observation is available, but also the data is assumed to be perturbed by noise and measured with some controlled errors in order to have a more realistic scenario for robust calibration.

Having considered the above realistic cases of data collection and by ensuring the accuracy of model calibration, the key objective of this work is to enable "*robust model calibration*" to be adopted in the self-adaptive loop of component connectors.

4.1.2. Chapter structure

The outcome of this chapter is a number of parameter estimation techniques that result in accurate parameter estimations given that the runtime measurements contain uncertainty. In this chapter, we aim to address **RQ1** (cf. Chapter 1) which highlights the needs for a model calibration that supports non-functional requirement verification at runtime. Non-functional requirement verification triggers the adaptation reasoning. We discuss the adaptation reasoning in Chapter 5 and the requirement verification in Chapter 7 of this thesis.

The rest of this chapter is structured as follows. Section 4.2 discusses the concept of models at runtime, the type of analytical models that can be adopted as well as the robustness of model calibration. Section 4.3 formally defines the analytical models adopted in this thesis. Section 4.4, as the main section, introduces the proposed method for model calibration in the presence of uncertainty and comprehensively evaluates the adopted techniques with thorough discussions on the results. Section 4.5 reviews the most related work in the literature. Section 4.6 discusses the limitations and future dimensions of this work.

4.2. A Robust Model Calibration

Fundamentally, robustness is the basic organizational principle of dynamic evolving systems. It is attained by some principles, which are observed by well-designed systems. Kitano (Kitano, 2004) discusses four mechanisms that he believes ensure the robustness of biological systems:

- System control
- Alternative mechanisms
- Modularity
- Decoupling

A high-level architectural viewpoint of our model calibration approach in Figure 4.4 shows that our approach also inherits these underlying principles of a robust mechanism. Runtime data are collected, unknown parameters of the analytical models representing the connector are updated, adaptation reasoning based on the adjusted model is performed and the appropriate mode of the connector is derived and enacted on the running system. Each of these modules, though interconnected, are decoupled and perform their own functions. There are also alternative mechanisms to handle input data considering the uncertainty inherent in the observations.

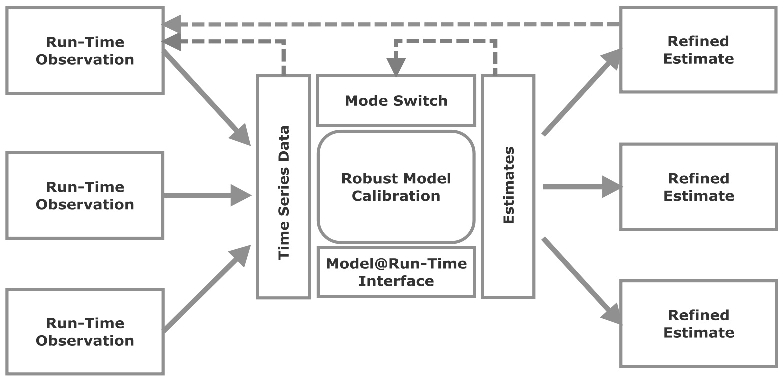


Figure 4.4. Architectural framework of robust model calibration.

4.2.1. Models at runtime: an enabler for self-adaptive behavior and assurance tasks

One of the key usages of models at runtime is to exploit the causal connection between the model and its system under investigation at runtime, see Figure 4.5. The usage of this connection has two different sides (Eder et al., 2013). On the one hand, models and system are in *descriptive causal connection* by which the changes in the system are reflected to the models. This enables analysis techniques to verify high-level models instead of the complex implementation of the application to collect needed information for verification. On the other hand, they are also in *prescriptive causal connection*. This means the models can be changed to originate (or trigger) an adaptation of the application.

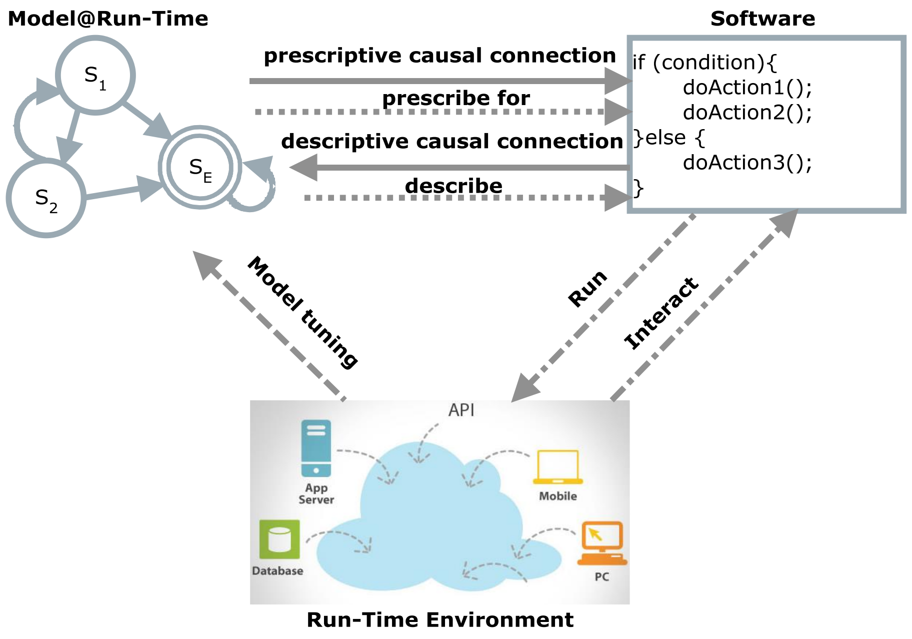


Figure 4.5. Interactions between model at runtime software and its runtime environment.

In the context of assurance of requirements for software systems, models at runtime can play *different roles* for assuring both functional and non-functional requirements of a system. For instance, they may represent requirements to be ensured, the current system state, adaptations that need to be enforced or context that the software uses. They also serve *several facilities* in this context. For example, they may be utilized as information sources for monitoring purposes, or change the system via model manipulations, or model-based analyses such as verification and simulations.

The role of models at runtime specifically as an enabler for self-adaptive behavior of software systems is represented in Figure 4.6. It uses an equivalent description of the architecture of the software system (which here corresponds to the connector mode) that is developed at design-time. This model continues to exist after development time and therefore can be calibrated to monitor the interaction between the software and its runtime environment. On the other hand, the requirements of the system will be verified continuously against the calibrated model. When a violation of a requirement is detected, the software system can be adapted accordingly. The key enabler for this self-adaptive behavior is located at the heart of Figure 4.6, which should be kept alive at runtime to support the tuning of the model. This extension of the lifetime of models in this paradigm enables the autonomic adjustment of the system implementation to tolerate new and possibly unpredictable situations (Luciano Baresi & Ghezzi, 2010).

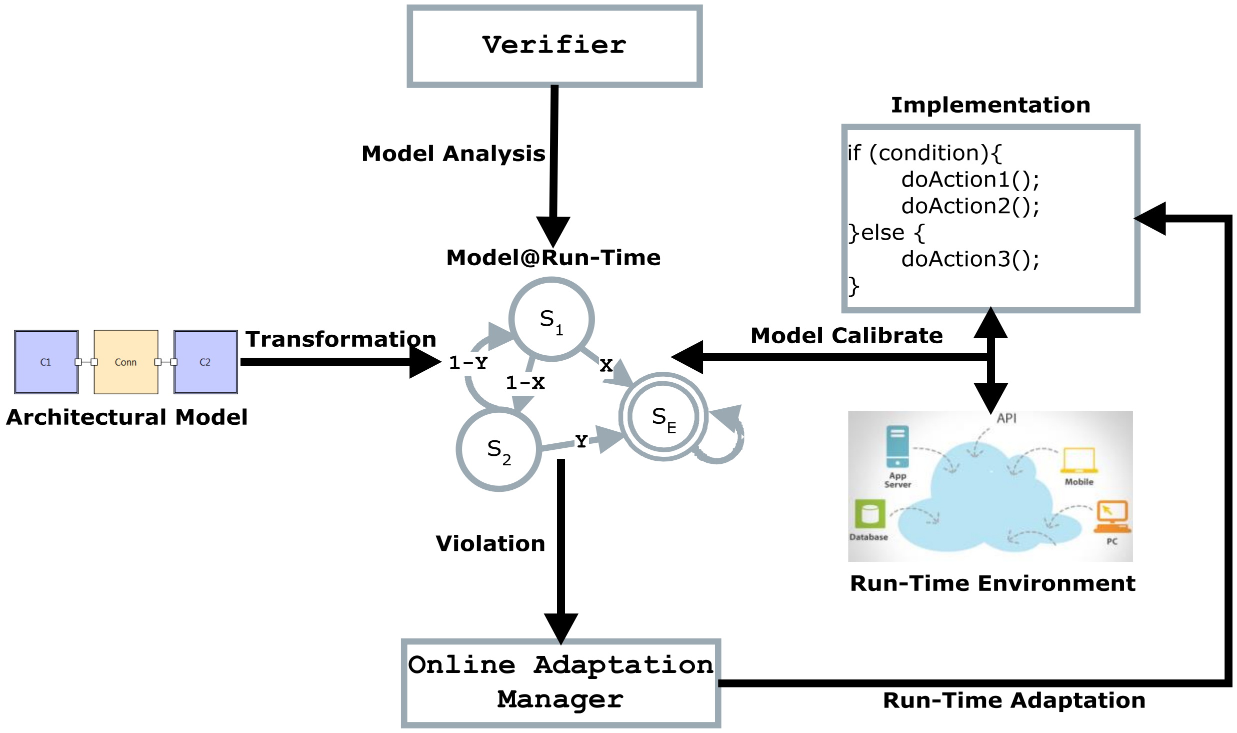


Figure 4.6. The role of models at runtime in self-adaptation loop.

4.2.2. The choice of analytical models

A wealth of models has been proposed over time as models at runtime (Ardagna, Ghezzi, & Mirandola, 2008). They differ in the level of formality and precision, the aspects they are intended to describe, and the types of reasoning they support (Jamshidi et al., 2013).

The type of the models that are employed as models at runtime differs from the more conventional models used by software architects to express their design choices (Ardagna et al., 2008; Blair et al., 2009).

The former type, known as *analytical models*, are mostly used for analysis of non-functional requirements such as reliability or performance (Cortellessa, Marco, & Inverardi, 2007). A list of potential models at runtime is summarized in Table 4.1. For a detailed discussion about such analytical models, we refer to (Ardagna et al., 2008).

Table 4.1. Potential models at runtime and their supports for non-functional requirements (*adapted from* (Ardagna et al., 2008)).

Model Family	**Model Name**	**System Quality**		**Model Characteristic**			
		Performance	Reliability	Adaptability	Cost Effectiveness	Composability	Scalability
Queuing Models	Bound analysis	**X**	**--**	**H**	**H**	**H**	**H**
	Product form	**X**	**--**	**M**	**H**	**H**	**M/H**
	Non-product form	**X**	**--**	**H**	**H**	**M**	**L**
	Layered queuing networks	**X**	**--**	**H**	**H**	**H**	**M**
Markov Models	Discrete-Time Markov Chains	**X**	**X**	**H**	**H**	**L**	**L**
	Continuous-Time Markov Chains	**X**	**X**	**H**	**H**	**L**	**L**
	Markov Decision Processes	**X**	**X**	**H**	**H**	**L**	**L**
	Stochastic Model Checking	**X**	**X**	**H**	**H**	**L**	**L**
Simulation	Simulation Models	**X**	**X**	**H**	**M**	**M/H**	**L/M**
Control-Oriented Models	Linear Time Invariant	**X**	**--**	**M/L**	**L**	**L**	**H**
	Linear Parameter Varying	**X**	**--**	**M/L**	**L**	**L**	**H**

A set of characteristics that can help architects to choose an appropriate model are (Ardagna et al., 2008; Blair et al., 2009; Metzger, Sammodi, & Pohl, 2013) listed in Table 4.1. A qualitative discrete scale is adopted as High (H), Medium (M) and Low (L) to compare the models. The characteristics are as follows:

- *Adaptability*: Estimation techniques should support efficient estimations under software architecture changes.
- *Cost-effectiveness*: The approach should require less effort than measurement at the system level.
- *Composability*: Estimation techniques should be able to estimate the system level parameters based on primitive level values.
- *Scalability*: Estimation techniques should be able to estimate the parameters even in large and complex models.

Regarding *adaptability*, the goal is the capability to revise the model and obtain new estimates, always remaining in the same model family. Markovian models provide a high degree of adaptability. In this type of model, a system change can result in a change in a parameter of the model or a different probability distribution of the state space. Considering *cost-effectiveness*, Markovian models are very cost-effective since modeling requires a small effort comparing to prototyping. With respect to *composability*, the models, which are structured hierarchically, can be composed more easily. Therefore, Markovian models have a low degree of composability. Finally, considering *scalability*, Markovian models require considerable computation time to be analyzed and therefore, have a low level of scalability.

4.2.3. Robustness in model calibration

Robustness is defined as the ability of a system to resist perturbations without adapting its initial key functions (Kitano, 2004). More specifically, in computing, robustness is the ability of an algorithm to continue to operate reliably despite abnormities in input, which is the characteristic of unreliable environments with unreliable components. It is considered the fundamental feature of dynamic adaptive systems (Kitano, 2004).

Robustness is often misinterpreted to mean remaining unchanged irrespective of environmental noises, so that the architecture of the system, and therefore the mode of operation, is unchanged (Kitano, 2004). In fact, it often requires the system to change its mode of operation in a smoothed way. In other words, robustness allows changes in the structure and components of the system due to perturbations, but specific functions are preserved.

Model calibration consists of appropriate estimation mechanisms interacting with models at runtime in a loop as depicted in Figure 4.7. The loop is started with an analytical model, whose structure is determined by the architecture of the software system, and it is specified by initial estimates that are available at design-time. This analytical model with its initial estimates and the parametric values determines the initial model at runtime. These parameters determine the parts of the system that need to be adjusted at runtime. The observed runtime data is collected and then forms a time series, which is passed to the estimation mechanism. The appropriate mechanism is chosen based on the characteristics of the data. The output of the estimation is the refined estimate, which substitutes the previous values in the model at runtime.

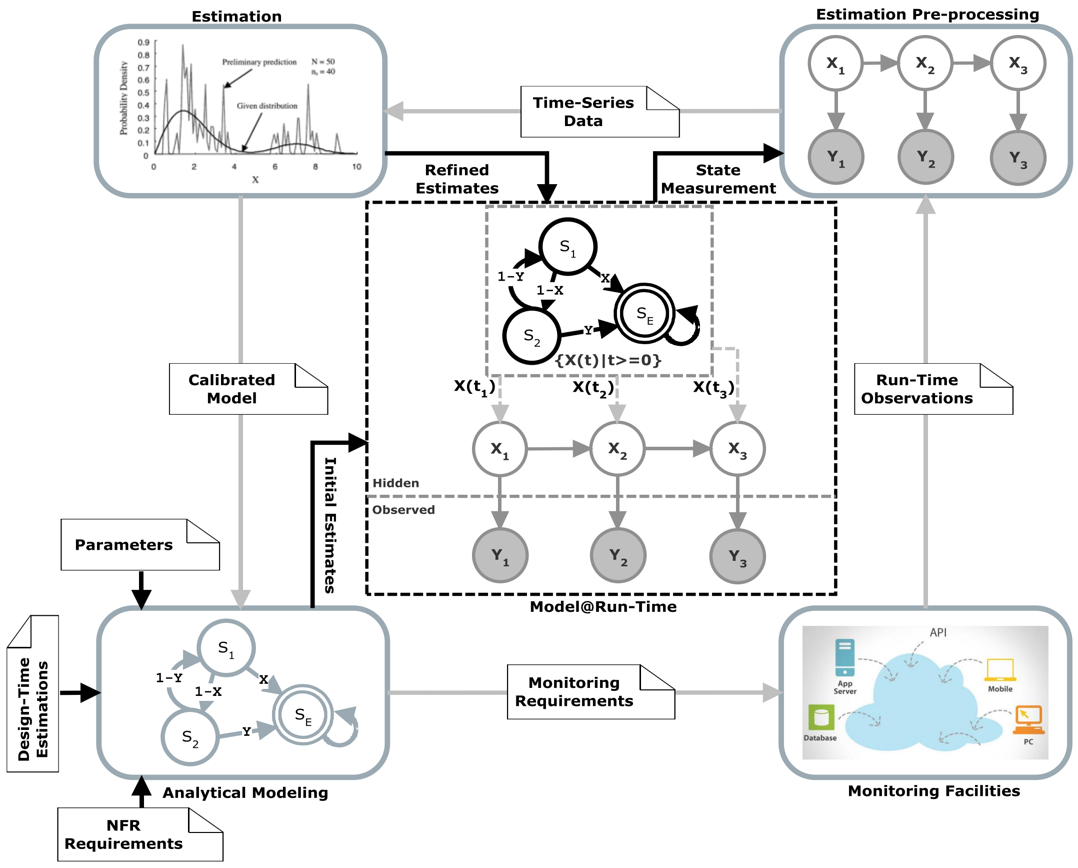


Figure 4.7. Overview of robust model calibration at runtime.

In the next section, we review the adopted analytical models in this thesis. The models will be delineated formally to provide an appropriate foundation for later runtime analyses.

Note most concepts defined and used in this chapter are standard concepts in probability theory, stochastic model checking and quantitative verification that we borrowed from standard literature, e.g., (Calinescu et al., 2012; M Kwiatkowska et al., 2007; Marta Kwiatkowska, 2007; Pinsky & Karlin, 2010).

4.3. The Model Framework

Let us now concisely introduce the analytical models we utilize in this thesis and the automatic analyses we perform on them at run time. Since our focus is on non-functional properties of component connectors, we specify connectors via Markov models, which support quantitative probabilistic specifications that are particularly useful to definite *reliability* and *performance* properties (Ardagna et al., 2008; Glinz, 2005). Markov models have been adopted quite a lot in software engineering (Ardagna et al., 2008) and there are tools available (Moon, 2011) for deriving such models from the architectural design of Reo connectors as this is the principal language for designing and implementing connectors in this work. For a more detailed justification of this choice, refer to the comparison and discussion in Section 4.2.2.

In statistics, a *Markov process* is a stochastic process satisfying a certain property, called the Markov property. A stochastic process satisfies the Markov property if the transition probabilities between different states (i.e., $X_i = x_i$) in the state space depend only on the random variable's current state, i.e.

$$\mathbb{P}\left(X_{n+1}=x_{n+1} | X_0=x_0, X_1=x_1, \ldots, X_n=x_n\right)=\mathbb{P}\left(X_{n+1}=x_{n+1} | X_n=x_n\right) \tag{4.1}$$

Therefore, for a Markov process the only information about the past needed to predict the future is the current state of the random variable. On the other hand, knowledge of the values of earlier states does not change the transition probability.

A *Markov chain* refers to a sequence of random variables $(X_0, \ldots, X_n)$ generated by a Markov process. Generally, the term Markov chain is used to convey a Markov process which has discrete (finite or countable) state space. More specifically, the possible values of X_i form a countable state space of the chain. A Markov chain either can be defined for a discrete set of times or can take continuous values $\{X(t): t \geq 0\}$. In the former case, the Markov chain is called Discrete-Time Markov Chain (DTMC) and in the latter case, it is called Continuous-Time Markov Chain (CTMC).

In the Markov chains (DTMCs and CTMCs), states are directly visible, but in a special class of Markovian models, the states are not directly observed, but a *noisy* version of them can be measured. These models are known as Hidden Markov Models (HMM) or State-Space Models (SSM) in general. This class of models is especially useful when we have incomplete and noisy observations of the system and we need to estimate the parameters of runtime models to enable reliable and on-time adaptations.

4.3.1. DTMC models

The difference between DTMC and CTMC is that rather than transitioning to a new (possibly the same) state at each *time step*, the system will instead remain in the current state for exponentially distributed random time and then change its state to a different state.

DTMC is characterized by *transition probabilities*, $p_{i,j}$ which are the probabilities that a process at state space x_i moves to state x_j in a single step,

$$p_{i,j}=\mathbb{P}(X_j=x_j | X_i=x_i) \tag{4.2}$$

If the state space is finite, the transition probability distribution can be represented by a transition matrix P with the elements $p_{i,j}$. Each row of P adds to one and all elements are non-negative. Therefore, P is a proper stochastic matrix.

The following DTMC model in Figure 4.8 represents a state-based system (here, a connector channel) consisting of 4 states. S represents the "Start" state, T represents a "Temporary" state, L corresponds to the "Lost" state and D is associated to the "Delivery" state. System start at the start state and move to the temporary state, then the message either will be delivered to the target or will be lost by their corresponding probabilities.

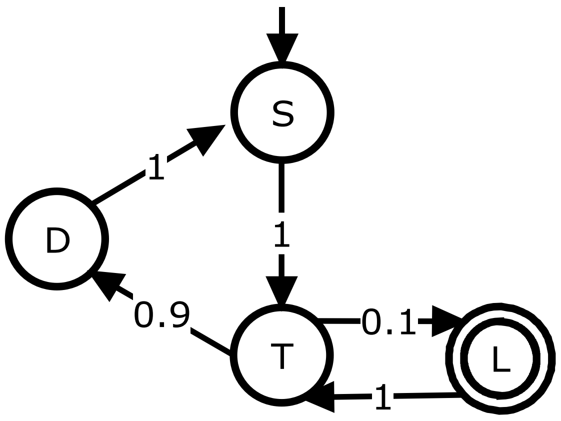


Figure 4.8. A DTMC example.

	S	D	T	L
S	0	0	1	0
D	1	0	0	0
T	0	0.9	0	0.1
L	0	0	1	0

Figure 4.9. Matrix representation of the DTMC example.

4.3.2. CTMC models

CTMC is characterized by *transition rates*, $q_{i,j}$ which measure how quickly transitions x_i to x_j happen. Precisely, after a small amount of time Δt, the probability of the state is now

$$\mathbb{P}\big(X(t+\Delta t) = x_j \big| X(t) = x_i\big) = q_{i,j}\Delta t + o(\Delta t), x_i \neq x_j \tag{4.3}$$

The transition rates $q_{i,j}$ form the transition rate matrix Q. As the transition rates matrix contains rates, the off-diagonal cells indicating the rate of departing from one state to arriving at another should be positive and the diagonal cells indicating the rate at which the system remains in a state should be negative. The rates for a given state should add to zero, resulting in the diagonal element being:

$$q_{i,i} = -\sum_{i \neq j} q_{i,j} \tag{4.4}$$

By considering $p_t = \mathbb{P}(X(t) = x_j)$, the evolution of CTMC is given by the following equation:

$$\frac{\partial}{\partial x}\, p_t = p_t Q \tag{4.5}$$

The probability that no transition happens in some time h is:

$$\mathbb{P}(X(s) = x_i, \forall s \in (t, t+h) | X(t) = x_i) = e^{q_{i,i}h} \tag{4.6}$$

That means the probability distribution of the waiting time until the first move is an exponential distribution with parameter $q_{i,i}$.

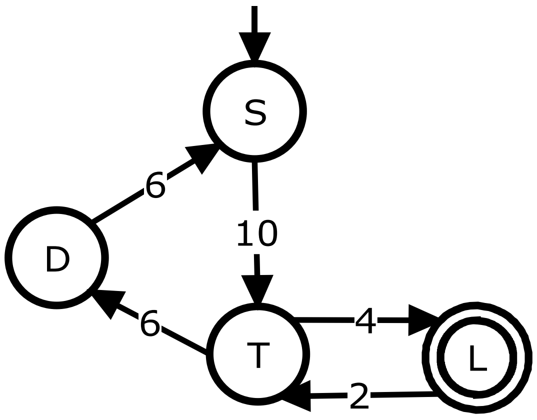


Figure 4.10. A CTMC example.

	S	D	T	L
S	-10	0	10	0
D	6	-6	0	0
T	0	6	-10	4
L	0	0	2	-2

Figure 4.11. Matrix representation of the CTMC example.

4.3.3. HMM models

In HMM, there are basically three involved stochastic process: a hidden one $\{X_i | i = 0,1,\ldots\}$ and an observed one $\{Y_i | i = 1,2,\ldots\}$. In this model, it is assumed that the hidden process $\{X_i | i = 0,1,\ldots\}$ is a Markov chain. Here we assume that the Markov chain is a continuous-time Markov chain $\{Z(t) | t \geq 0\}$. X_i is governed by $Z(t)$ and it has a direct influence on Y_i. Y_i is a noisy version of X_i. In other words, we model runtime data as being noisy observations of some unobserved stochastic process as it is shown in Figure 4.12. The model consists of:

- $\mathbb{P}(X_0)$: Initial distribution
- $\mathbb{P}(X_i | X_{i-1}), i = 1,2,\ldots$: Transitions
- $\mathbb{P}(Y_i | X_i), i = 1,2,\ldots$: Likelihood

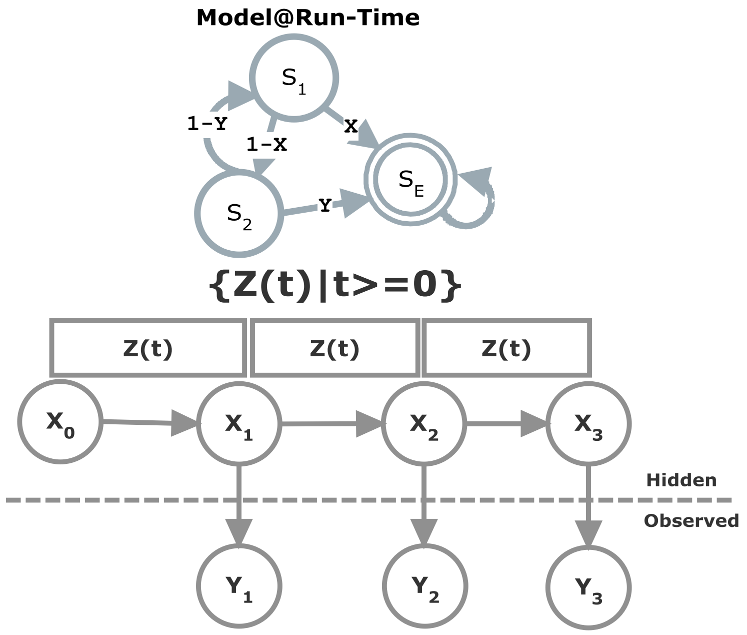


Figure 4.12. Hidden Markov model with hidden CTMC model as runtime model.

4.4. Model Parameter Estimation

The model calibration as depicted in Figure 4.13 consists of two main tasks: 1) data collection (or runtime observations), 2) parameter estimation (or model fitting, model calibration, model adjustment, model update). The observed runtime data is collected and is then processed by the estimation mechanisms. The runtime observations, in general, can be seen as samples of appropriate stochastic processes. The outputs of the estimation methods are the refined estimates, which substitute the initial values in the models at runtime. In this section, we first motivate the need for accurate parameter estimations considering the uncertainties in the runtime observations. We then describe the proposed methods for the estimation of the unknown parameters of the analytical models (i.e., DTMC and CTMC) that we consider in this thesis as the runtime models for connector self-adaptation. As a result of this selection, the problem of parameter estimation of the runtime models, therefore, reduce to the estimation of transition probabilities for DTMC models and transition rates for CTMC models.

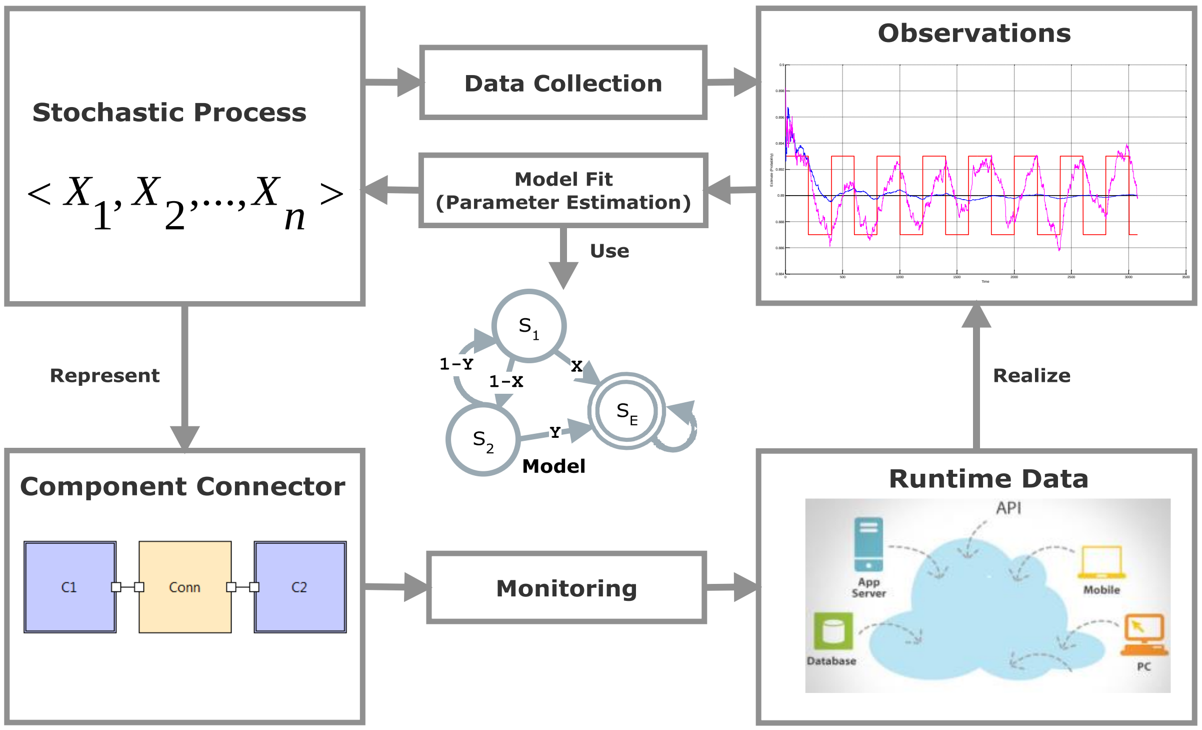


Figure 4.13. Overview of parameter estimation using mathematical model.

4.4.1. The need for an accurate parameter estimation

In order to perform adaptive changes to running component connectors especially in a self-managed manner to respond to the requirement violations, the connectors need to be enhanced with an estimation capability to detect the violations as soon as possible. A key objective of such approaches is to estimate the parameters of the analytical model at runtime (here, DTMC and CTMC models) accurately. By accurate we mean most of the violations of non-functional requirements should be detected, while generating as few false “need for adaptation” decisions as possible. In other words, the more “true” and less “false” violations that can be detected by an estimation approach, the more accurate it would be. In this section, we elaborate on the relevance of accurate estimation for self-adaptive component-connectors. Figure 4.14 illustrates different situations, which may occur when performing parameter estimation of the runtime models and their correlation to accurate self-adaptations.

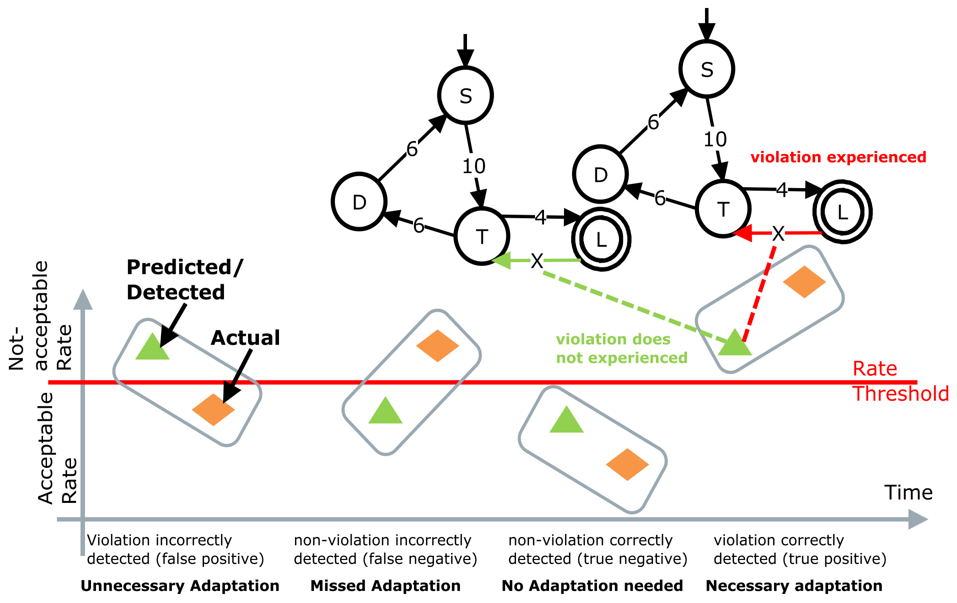

S
D
T
L
6
10
4
X
violation experienced
violation does not experienced
Predicted/ Detected
Actual
Not-acceptable Rate
Acceptable Rate
Rate Threshold
Time
Violation incorrectly detected (false positive)
non-violation incorrectly detected (false negative)
non-violation correctly detected (true negative)
violation correctly detected (true positive)
Unnecessary Adaptation
Missed Adaptation
No Adaptation needed
Necessary adaptation

Figure 4.14. Different scenarios in parameter estimation (adapted from (Metzger et al., 2013)).

We assume here that the monitoring mechanisms are in place and once the running connector changes its state to a different state according to its analytical model, its runtime data including this state change is observed. These observations provide the main input for the estimation of the parameters of the runtime models. The diagram above illustrates this by sketching a CTMC model corresponding to a running connector. The changes in the connector in terms of its runtime state represent the points in time when monitoring is assumed to be performed and thus the state-changes in the model can be observed.

According to Figure 4.14, two important cases may occur during estimation:

- **Unnecessary adaptations**: False positive estimations may trigger the self-adaptation of the component connector although the connector would have in fact worked as expected. Such unnecessary (Metzger et al., 2013) or unrequired (Amin, Colman, & Grunske, 2012) adaptations can have the following consequences: Firstly, the adaptation execution takes time and would leave less time to address the actual violations. Secondly, the replaced connector mode might be unreliable (e.g., if the new channels have bugs) leading to an ultimate failure of the connector.
- **Missed adaptations**: False negative estimations will not trigger a self-adaptation, although the connector will actually violate some requirements and this violation could have been compensated. In this case, where an adaptation opportunity is missed due to an inaccurate parameter estimation of the runtime models, or the need for adaptation is detected very late, it can lead to costly repair strategies or the restart of the connector. This implies that inaccurate estimations would reduce the overall efficiency of the self-adaptation mechanism.

4.4.2. Estimation of transition matrix of a DTMC

Estimating a transition probability matrix of a DTMC with discrete observation data for each model $X = \{X_t | t = 1, \dots, T\}$ is simple. Let us assume that each model has K states and we have a time series of m observations of the model.

4.4.2.1. Parameter estimation with complete data

Let us assume that we want to estimate a two-step transition matrix and the data is from a collection that was followed for four steps with two two-step observation intervals. In this case, the observed two-step intervals coincide with the desired two-step transition matrix. Because the DTMC models are homogeneous, the observed transitions between the first two steps can be summed up with the transition between the second two steps to form an observed two-step transition count matrix as follows.

$$N = \begin{bmatrix} n_{11} & \cdots & n_{1K} \\ \vdots & \ddots & \vdots \\ n_{K1} & \cdots & n_{KK} \end{bmatrix} \tag{4.7}$$

Given the observed count matrix, the maximum likelihood estimate of the transition matrix is the row proportions of N,

$$\hat{P} = \{\hat{p}_{i,j}\}, where\ \hat{p}_{i,j} = n_{i,j} / \sum_{k=1}^{K} n_{i,k} \tag{4.8}$$

Unfortunately, assuming complete runtime data is far from reality. Usually, we can only obtain noisy measurement of a small fraction of the runtime status of connectors, captured at discrete time points.

4.4.2.2. Parameter estimation with incomplete data

In this section, we develop a relationship between the observed data D of size d, the DTMC model X of size K and model parameters P. The objective in estimation (learning) of model parameters (transition probabilities) is to compute the posterior probability density $\mathbb{P}(P|D,X)$. This is the probability of the model parameters, treated as a random variable, given the monitored (observed) data and the model structure. In order to find the posterior distribution, a common approach in statistic is to consider the joint distribution $\mathbb{P}(P,D|X)$ given a certain model structure. This joint distribution can be computed in two ways: $\mathbb{P}(P|D,X)\mathbb{P}(D|X)$ or $\mathbb{P}(D|P,X)\mathbb{P}(P|X)$. This results in the Bayes' theorem as follows:

$$\mathbb{P}(P|D,X) = \frac{\mathbb{P}(D|P,X)\mathbb{P}(P|X)}{\mathbb{P}(D|X)} \tag{4.9}$$

It consists of three components. The prior probability $\mathbb{P}(P|X)$ indicates our assumptions at design-time regarding the model parameters. These assumptions regarding design-time estimations are based on for example previous experience of the designer or empirical data about the connector collected in previous runs. The likelihood $\mathbb{P}(D|P,X)$ specifies the probability of the observations given the model and its parameters. Finally, the evidence or marginal likelihood $\mathbb{P}(D|X)$ is the probability of the observation given the model.

For computing the likelihood, we make a simplifying assumption that the first observation is given. Then the probability of observations is the probability of the second data given the first, times the probability of the third given the second and so on:

$$\mathbb{P}(D|P,X) = \prod_{k=1}^{d} p_{k,k+1} = \prod_{i=1}^{K}\prod_{j=1}^{K} {p_{i,j}}^{N_{i,j}} \tag{4.10}$$

The prior $\mathbb{P}(P|X)$ is utilized to specify assumptions about the model. We use *conjugate distributions* in order to make posterior tractable with respect to the prior. It means that the posterior distribution has the same functional form as the prior. We model each row of P with a Dirichlet distribution. This choice is justified in (Diaconis & Ylvisaker, 1979).

$$(p_{i,1}, p_{i,2}, \dots, p_{i,K}) \sim Dir(\alpha_i^{(0)} p_{i,1}^{(0)}, \alpha_i^{(0)} p_{i,2}^{(0)}, \dots, \alpha_i^{(0)} p_{i,K}^{(0)}) \tag{4.11}$$

Generally, a Dirichlet distribution $Dir(a_1, a_2, \dots, a_K\ )$ with positive parameters is a joint distribution for a vector $X_K = 1 - X_1 - \dots - X_{K-1}$ with density evaluated in $X_1 = x_1, \dots, X_K = x_K$

$$\frac{\Gamma(a)}{\prod_{i=1}^{K}\Gamma(a_i)}\prod_{i=1}^{K} x_i^{a_i-1} \qquad a = \sum_{i=1}^{K} a_i \tag{4.12}$$

With the following properties:

$$E(X_i) = \frac{a_i}{a} \qquad Var(X_i) = \frac{a_i(a-a_i)}{a^2(a+1)} \tag{4.13}$$

Given the likelihood and prior, the evidence $P(D|X)$ is a simple normalization in Bayes' theorem.

$$\mathbb{P}(D|X) = \int \mathbb{P}(D|P,X)\mathbb{P}(P|X)dP \tag{4.14}$$

Now, by using Bayes' theorem, the estimation of parameters boils down to estimation of the posterior.

$$\mathbb{P}(P|D,X) = \prod_{j=1}^{K}\frac{\Gamma(a+N_i)}{\prod_{i=1}^{K}\Gamma(a_i+N_{i,j})}\prod_{i=1}^{K} x_i^{a_i+N_{i,j}-1} \qquad N_i = \sum_{j=1}^{K} N_{i,j} \tag{4.15}$$

Since the Dirichlet distribution is a conjugate distribution, the posterior of the transition matrix P is a product of independent Drichlets with the following properties:

$$E(X_i) = \frac{a_i + N_{i,j}}{a + N_i} \tag{4.16}$$

This is the posterior mean estimate (PME) of the model parameter.

$$E(X_i) = \frac{1}{a + N_i}(a * \frac{a_i}{a} + N_i * \frac{N_{i,j}}{N_i}) \tag{4.17}$$

Therefore, the PME is a weighted sum of prior expectation and maximum likelihood estimate (MLE). As a result, we summarize the estimation as a more intuitive and computationally appealing form as follows:

$$p_{i,j}^{(d)} = \frac{a}{a + N_i} \times p_{i,j}^{(0)} + \frac{N_i}{a + N_i} \times \frac{\sum_{o=1}^{N_i} N_{i,j}^{(o)}}{N_i} \tag{4.18}$$

Note that the full details of the mathematical reasoning above is given in (Strelioff, Crutchfield, & Hübler, 2007). Formula (4.18) is the Bayes rule, which yields the new estimates based on the weighted sum of two terms. The former term is associated with design-time estimates $p_{i,j}^{(0)}$ and represents a priori knowledge about the transition probabilities in the DTMC model. The latter term is related to runtime data, which has been observed from a running system. More specifically, it provides monitoring data about the occurrence of transitions among states of system. The variable a is smoothing parameter, which quantifies our belief in a-priori knowledge. A high value of the smoothing parameter means that we are confident with our estimate at design-time and the runtime data produces a smaller contribution in changing the parameter. The low value of the smoothing parameter highlights the runtime data and as a result, the probability of ever changing the parameters will increase. In case of $a = 0$, the estimator reduce to the MLE estimator. Note that the smoothing parameters need to be treated differently for highly dynamic environments than for fairly stable environments and we can estimate the model parameters with high confidence at design-time. Therefore, in the case of the latter situation, it is better to set a higher number for α.

It is important to mention that the Bayes rule, i.e., Formula (4.18), has been previously applied for estimating unknown parameters of Markovian models in (Calinescu, Johnson, & Rafiq, 2011; Epifani et al., 2009) and we do not claim this as a contribution of this thesis. A minor contribution that we made in this regard is the comprehensive experimental qualitative and quantitative observations of applying this estimation technique to the parameter estimation of component connectors in different settings as reported respectively in Section 4.4.2.6 and Section 4.4.2.7.

4.4.2.3. Failure detection using the Bayes estimator

In the context of the analysis in the MAPE-K loop, a failure is *detected* if the system experiences a nonconformity to the expected behavior described by a requirement. For instance, consider the system described by the DTMC model in Figure 4.8 and a requirement as follows:

$R1$: The probability $P1$ that messages are successfully delivered is greater than 0.89. (4.19)

A failure of $R1$ may be detected by considering the number of successful message deliveries over time. For instance, let us consider a runtime data of length 40 each representing the final state of the message delivery/lost. Suppose that the following trace $\langle d_1, d_2, \ldots, d_{40} \rangle$ represents runtime data with each d_i showing a message delivery/lost. We assume that among these 40 observations, 5 of them $\{d_3, d_5, d_{10}, d_{11}, d_{35}\}$ represent messages lost. It means that the system that corresponds to the DTMC in Figure 4.8, in these moments moves from state T to L and in the other moments goes from T to D. By assuming $a = 20$, after observing the runtime trace and by using Formula (4.18), we have the following parameter update.

$$p_{T,D}^{(d)} = \frac{20}{20 + 40} \times 0.9 + \frac{35}{20 + 40} = 0.88 \tag{4.20}$$

Having calculated the updated parameters associated with transitions from state T to D, we can deduce that based on the observed data, the probability associated with the successful delivery of messages was overestimated at design-time. By using the new calibrated value 0.88 instead of 0.9, the probability of the path from state S to T to D would be $P = 1 * 0.88 = 0.88$, which is lower than 0.89, thus violating $R1$. In this case, the violation of $R1$, which is interpreted as failure of the system, is detected. After detecting the violation, the exception associated to the violated requirement will be fired. This will trigger the adaptation reasoning module, which decides the appropriate mode for the connector. As a result, the current architectural configuration of the connector would be changed and new configuration will be enacted. The details of the adaptation reasoning are given in Chapter 5.

4.4.2.4. *Bayes estimator with exponential smoothing*

Exponential smoothing is a mathematical technique that can be applied to time series data to produce smoothed data (Kalekar, 2004). Time series data are a sequence of observations. For instance, here we can see the runtime data as a time series of random noisy data collected based on monitoring the observed system. In Formula (4.18), the past observations are weighted equally, while exponential smoothing assigns smoothed exponentially decreasing weights over time. More specifically and in a simple way, we want to assign rather different weights to the older observations in a way to differentiate between them by putting more importance on recent observations. This is a logical extension, because as each observation after a while becomes less important compared with more recent runtime data.

In general, the data sequence is represented by $\{x_t\}$ and the smoothed data is written as $\{y_t\}$ which is regarded as an estimate of what the next value of x would be $y_n \sim x_{n+1}$.

4.4.2.4.1. The simple moving average

A simple way to smooth a set of sequential observations is to ignore the old ones and consider the latest k observations.

$$y_t = \frac{1}{k} \sum_{i=0}^{k-1} x_{t-i} = y_{t-1} + \frac{x_t - x_{t-k}}{k}, k > 1 \tag{4.21}$$

, where k is an arbitrary integer higher than one. A small value of k leads to more sudden changes because of recent changes in the data. On the other hand, a larger k will result in a greater smoothing effect. This method cannot be used until the first k observations have been produced.

4.4.2.4.2. The weighted moving average

A more sophisticated method for smoothing the time-series data is to calculate a moving average by choosing a set of weighted factors.

$$y_t = \frac{1}{k}\sum_{i=1}^{k} w_i x_{t-i+1}\,, where \sum_{i=1}^{k} w_i = 1 \qquad (4.22)$$

In general, the weights are often chosen in a way to give more weight to the more recent observations and less weight to the older ones.

4.4.2.4.3. Single exponential smoothing

Both the exponential smoothing techniques weight the history of the workload data by a series of exponentially decreasing factors. An exponential factor close to one gives a large weight to the first samples and rapidly makes old samples negligible. Exponential smoothing is commonly applied in finance, however, it can be applied to any discrete set of sequential observations. Let the sequence of observations begin at time $t = 0$, the simplest form of exponential smoothing is:

$$\begin{aligned} y_0 &= x_0 \\ y_t &= \alpha x_t + (1-\alpha)y_{t-1}, t > 0, 0 < \alpha < 1 \end{aligned} \qquad (4.23)$$

The choice of α is quite important. If it is too close to 1, it has less of a smoothing effect and gives a higher weight to recent changes in the observations and as a result the estimate may fluctuate dramatically. While values of α closer to zero have a better smoothing effect and as a result, the estimate is less responsive to very recent changes.

4.4.2.4.4. Double exponential smoothing

Double exponential smoothing is an extension of the simple version. The output of estimation is now F_{t+m} an estimate of the value of x at time $t + m$.

$$\begin{aligned} y_1 &= x_0 \\ b_1 &= x_1 - x_0 \\ y_t &= \alpha x_t + (1-\alpha)(y_{t-1} + b_{t-1}) \\ b_t &= \beta(y_t - y_{t-1}) + (1-\beta)b_{t-1} \\ F_{t+m} &= s_t + mb_t \end{aligned} \qquad (4.24)$$

4.4.2.5. An extended DTMC estimation algorithm

The Bayes rule as derived in (4.18), as we will show in the experimental evaluations, and is also demonstrated in (Epifani et al., 2009) is effective in scenarios where the runtime estimate $p_{i,j}^{(d)}$ differs from the design-time estimate $p_{i,j}^{(0)}$, but is not changing rapidly after the first estimations and tends to be constant at runtime. However, in real scenarios involving component connectors, $p_{i,j}^{(d)}$ is likely to change dynamically. For such circumstances, Equation (4.18) is not be able to detect requirement violations quickly. This is logical because even the oldest observation is as important as the most recent one. In some situations, this become even more problematic. It cannot detect violations at all specifically when the violation period is quite short. We demonstrate some of these scenarios in Section 4.4.2.6.3. In order to avoid such situations and provide a better estimation mechanism, we can use the idea behind exponential smoothing (see Section 4.4.2.4) as first has been proposed in (Calinescu, Johnson, et al., 2011). In the extended Bayes approach, appropriate weights are assigned to the observations. In order to derive an updated equation for the extended Bayes rule, we first define weights for each observation as:

$$w_o = \frac{1}{a^{(t_o - t_s)}} \tag{4.25}$$

, where $a \geq 1$, t_o is the time that oth observation is made and t_s is the time of the observation that requires to be weighted according to its distance to the latest observations.

Now we can obtain the extended updating rule by multiplying each observations $N_{i,j}^{(o)}$ by its associated weight as defined in Equation (4.25):

$$p_{i,j}^{(d)} = \frac{a}{a + N_i} \times p_{i,j}^{(0)} + \frac{N_i}{a + N_i} \times \frac{\sum_{o=1}^{N_i} w_o N_{i,j}^{(o)}}{\sum_{o=1}^{N_i} w_o} \tag{4.26}$$

Note that if we set $a = 1$, then the Equation (4.26) is then turned back to its original Bayes rule as in Equation (4.18).

4.4.2.6. *Experimental evaluation*

In order to evaluate the appropriateness of the proposed parameter estimator as a calibration method for DTMC models in the context of reconfigurable component connectors, we decided to perform controlled experiments (Pfleeger, 1995). The central factor was the good level of control that we have over the variables, which we describe in Section 4.4.2.6.2. In addition, we need to change the values of controlled variables easily. The other key factor was the high degree to which we needed to replicate the situations we want to investigate. Table 4.2 summarizes the key concerns we consider in choosing the right approach for evaluation.

Table 4.2. Factors related to the choice of evaluation approach.

Empirical concerns	Controlled experiment	Case study
Level of control	High	Low
Difficulty of control	Low	High
Level of replication	High	Low
Generalization	Statistical	Analytic
Place to conduct	In the lab	In context

This section discusses some experimental results and their evaluations through a number of controlled experiments. To be more specific, we simulate the runtime data by using statistical distribution and we apply our estimation algorithm to estimate the parameters of the runtime models.

4.4.2.6.1. Experimental conception

This section contains a subset of the scenarios that involve estimating the probability of successful message delivery in a number of component connectors based on initial design-time estimates and on runtime data obtained through monitoring the connectors. Message delivery was selected because it is a critical semantic in each channel of connectors and influences the performance and reliability of each connector. For instance, in the task queue connector in Figure 4.15, the DTMC model in Figure 4.8 represents the semantics of the channel between nodes $A\ to\ B$.

The objectives of the controlled experiment are:

- $O1$: To show that the Bayesian estimator is an appropriate estimator for model calibration of DTMC models as runtime models for enabling self-adaptive component connectors.
- $O2$: To show that the extended version of Bayes estimator based on exponential smoothing is a superior estimator in terms of identifying the violation of requirements.

Then the objectives are translated into the following hypotheses:

- *Null hypothesis* (H_0): There is no difference in parameter estimation between the estimation derived from the base Bayesian estimator and an extended version.
- *Alternative hypothesis* (H_1): The estimates of the DTMC parameters of the component connectors based on extended version of Bayes estimator is more accurate in terms of detecting the violations and with less errors in terms of point estimations than the base version of it.

The experimental design is a complete plan for applying different experimental conditions to experimental subjects so that one can determine how the conditions affect the result. In particular, the experiment design is to plan how the application of these conditions will help to test hypotheses and answer objective questions.

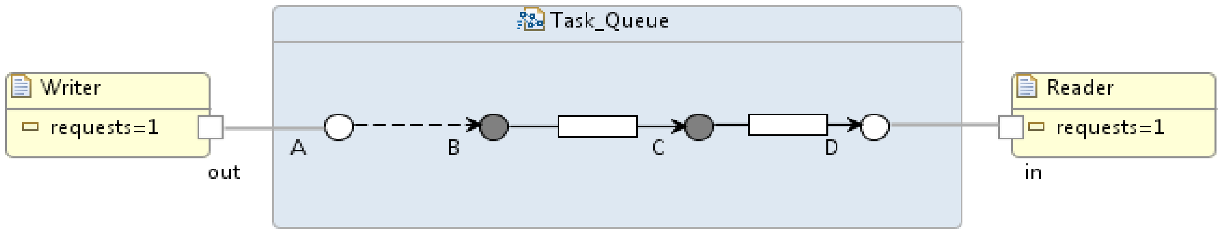


Figure 4.15. Task queue connector.

4.4.2.6.2. Experimental setup

In empirical software engineering, the key discriminator between experiments and case studies is the degree of control over the experimental variables (Pfleeger, 1995). The difference between these two types of evaluation approaches can be stated more rigorously by considering the notion of experimental variables. There are in general three types of variables in the context of controlled experiments: Independent variables, control variables and dependent variables. The *independent variables* influence the application of a treatment and thus results of an experiment. The *dependent variables* are the factors that we expect to change as a result of applying the treatment. The *control variables (controlled variable or extraneous variable)* are specific independent variables, which are kept constant and unchanged in an experiment.

By assuming that the probabilities assigned to the other state transition in the model represented in Figure 4.8 are fixed, it is straightforward to check that the connector satisfies the requirement $R1$ (cf. Statement (4.19)) if and only if the message delivery has a probability of success greater than $p\_Threshold = 0.89$. We assume that, at design-time, the model is specified by an estimation of the successful message delivery. This estimation at design-time determines the starting point of the estimation algorithm and is one of the controlled variable in our experiments. Let us assume that the actual probability is also one of our controlled variables, which might change over time. More specifically, by controlled we mean that it may take different patterns from constant value to a sophisticated change pattern over time. At some times, the actual probability may be above the threshold $p\_Threshold = 0.89$ and satisfies the requirement $R1$ and sometimes it may below the threshold, which leads to a requirement violation. The variance of runtime data is also one of our controlled variables. The variance determines how the runtime observations may vary and fluctuate over time. The smoothing factor is also a controllable variable, which determines the level of confidence with respect to the design-time estimate. The number of simulations for generating the observations in each scenario of experimentation is also a controlled variable. More simulation rounds increase the time for each experiment, but also increase the accuracy of the estimates. For generating runtime observations, we use Bernoulli distributions and their timestamps are generated by exponential distribution with specific parameter λ, which is also one of our controlled variables. In most of the experiments, we put it by default to $\lambda = 1$. However, we consider one medium and one large value for this parameter as well. The reason behind this is that we should have enough samples to represent different environmental conditions. The exponential smoothing coefficient α is also one of our controlled variables, which directly determines the weight of the observations. Finally, the variance in the threshold of acceptance/rejection of requirements is also a controlled variable, which determines the accuracy of estimation by identifying the false positives estimates.

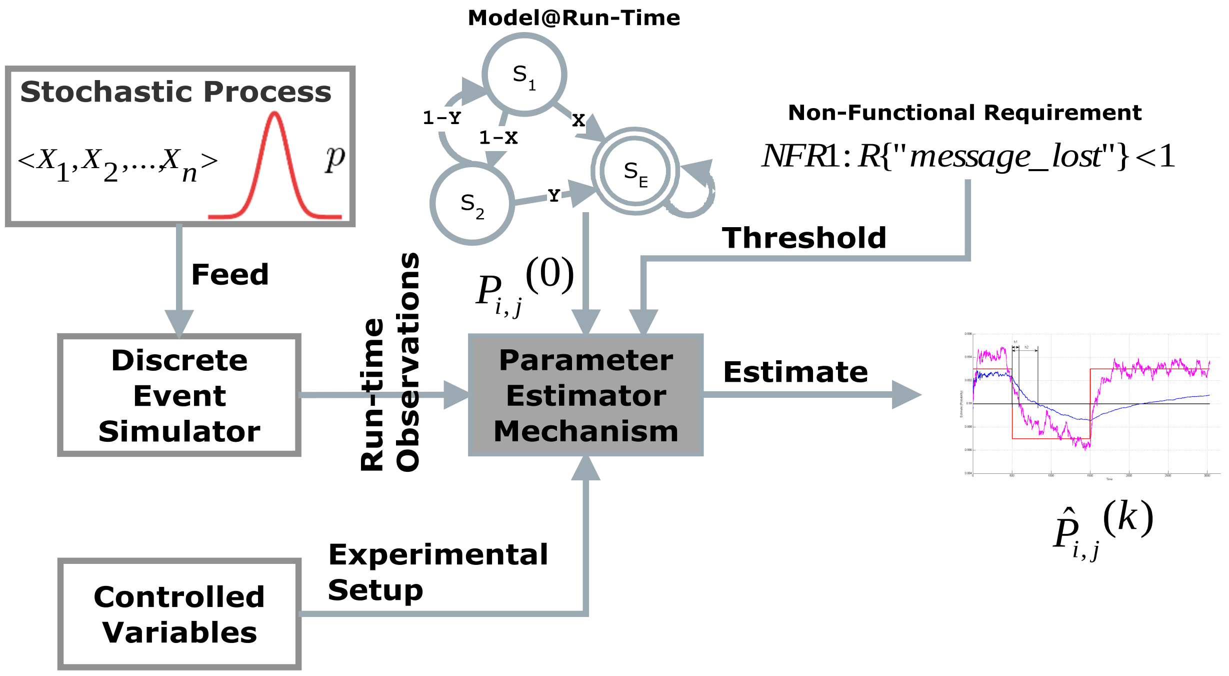


Figure 4.16. Experimental setup overview.

The controlled variables that we use in our experiments are summarized in Table 4.3.

Table 4.3. List of controlled variables and their purpose in our experiments.

Controlled variable	Purpose
$\boldsymbol{p\_Threshold}$	Threshold of violation/satisfaction of requirement
$\boldsymbol{p_{i,j}^{(0)}}$	Design-time estimate
$\boldsymbol{p\_Actual}$	Actual probability of the simulated parameter
$\boldsymbol{var\_Actual}$	Variance in the actual probability of the parameter
$\boldsymbol{a}$	Smoothing parameter
$\boldsymbol{M}$	Number of simulation rounds
$\boldsymbol{\lambda}$	Exponential distribution parameter
$\boldsymbol{\alpha}$	Exponential smoothing coefficient
$\boldsymbol{var\_Threshold}$	Variance in the threshold of violation/satisfaction
$\boldsymbol{N}$	Number of runtime observation (time interval of simulation)

4.4.2.6.3. Running the experiments

The goal of experiments is to assess how the Bayes estimator of the probability of the transition evolves over time, as runtime observations are collected from the running connector by simulation. We generate runtime data that follows *Bernoulli distribution* representing the observation of state transitions between states T to D, which means successful message delivery. We run different experiments to evaluate the Bayes estimator and its extended version. In each experiment, we use the average estimate of the probability of the transition over M number of simulation rounds. In order to have enough statistically sound data we should consider the value of M to be large enough. The result of each experiment is represented by a figure. The horizontal axis of each figure represents the runtime data and the vertical axis represents the estimation value, which starts from a prior value and steadily converge to the actual probability. In each figure, the straight black line represents $p\_Threshold$. The red line represents p_Actual and its changes over time determine the variance of runtime data. The blue line shows the estimated value by the basic Bayes estimator and the purple line is associated with the extended version of Bayes estimator.

In order to have the results of all the combinations of the variables summarized in Table 4.3, we take into account some representative values for each variable. For each experiment, we fix the value of some variables and change a couple of them in comparison with the previous experiment, which are highlighted in each of their settings. In this way, we have a sufficient number of representative experiments that could be interpreted as a sample of all the hypothetical experiments, which could have been otherwise determined by all the combinations of variables, which would have been unfeasible for the purpose of this chapter. Note that all the experiments in this chapter were run on a desktop machine with the specifications as in Table 4.4.

Table 4.4. The platforms used in the controlled experiments.

Platform category	The adopted platform
Hardware	Intel Core i7 CPU, 2.80 GHz, 12 GB memory
Operating system	64-bit Windows 7 Professional OS
Application	MATLAB R2012a

4.4.2.6.4. Experimental results and interpretations

In this section, we present comprehensive experimental results that we have observed by investigating the impact of different settings to the characteristics of the model calibration method. In order to evaluate the effectiveness of the extended model calibration (see Section 4.4.2.5), we carried out a broad range of experiments in which we compared results with those produced by the basic Bayes method. We designed the experiments in a way that covers most of the situations that may happen at runtime in order to make sure that the proposed model calibration performs well in different occasions. For designing the experiments, we considered different degradations in connector reliability at different time points with different change patterns (cf. Table 4.5). We cannot claim that the change patterns that we embed in the experiments cover all possible scenarios at runtime, but it provides sufficient evidence that they cover most of the potential scenarios. These experimental evaluations enable us to claim that the estimation accuracy of the proposed approach is not by chance, under restricted circumstances that may not happen in reality.

Table 4.5. Change pattern in the performed experiments.

Index	Change pattern	Experiment number
1	Normal behavior	1
2	Short degradation	2
3	Early degradation	3
4	Late degradation	4
5	Shallow degradation	5,6
6	Far starting	7
7	Deep degradation	8
8	Irregular degradations	9,10
9	Constant with no degradation	11
10	Balanced degradations	12,13,14

The aim of these relatively comprehensive scenarios was to simulate a degradation in the reliability with which a connector channel that passes a given message within a predefined amount of time, and to test the ability of the two estimation methods to identify this degradation. The 14 designed scenarios considered different types of reliability degradation a shorter (i.e., scenario 2), shallow (i.e., scenarios 5, 6), more significant (i.e., scenario 8), very early (i.e., scenario 3), very late (i.e., scenario 4), uneven, unequal, asymmetrical, and unbalanced (i.e., scenarios 9, 10), balanced degradation (i.e., scenario 12, 13, 14), no degradation (i.e., scenario 11). We also considered a scenario where the design-time estimation is very far from the actual reliability (i.e., scenario 7). Note that in each experimental setting (i.e., Table 4.6 to Table 4.19), we highlighted the change in the controlled variables (corresponding rows in the tables) from the previous setting to the current setting. For each experiment, we also report experimental observations and the interpretation of the results. This helps us to better understand the implications of such stochastic approach for parameter estimations on connectors. Such implications are also part of the contributions that we made in this research.

In other words, we consider different situations in which the target environment changes in its operating conditions and therefore the probability of successful message passing in connector channels evolves over time. In particular, we consider scenarios where initially the probability of successful message passing is varying according to the red lines and the prior guess is equal to $\boldsymbol{p}_{i,j}^{(0)}$. We assume that a sudden change in the running environment shifts the value of the probability according to the changes in the red lines (we embed different patterns of change in different scenarios). The graphs (Figure 4.17 to Figure 4.30) show the results obtained with the two estimation approaches. The figure in each scenario shows how the accuracy of estimation constantly improves until there is sudden change in the red line and after that our simulation starts generating data from the new value. As soon as enough new runtime data are collected, the estimation accuracy of estimation improves again since the estimated parameter begins to converge to the new probability characterizing the new situation.

In the following, we first figuratively illustrate the performance of the two approaches, the setting of the experiment and then discuss the observations and interpretations of the results for each experiment separately.

Experiment number: 1 (Normal behavior)

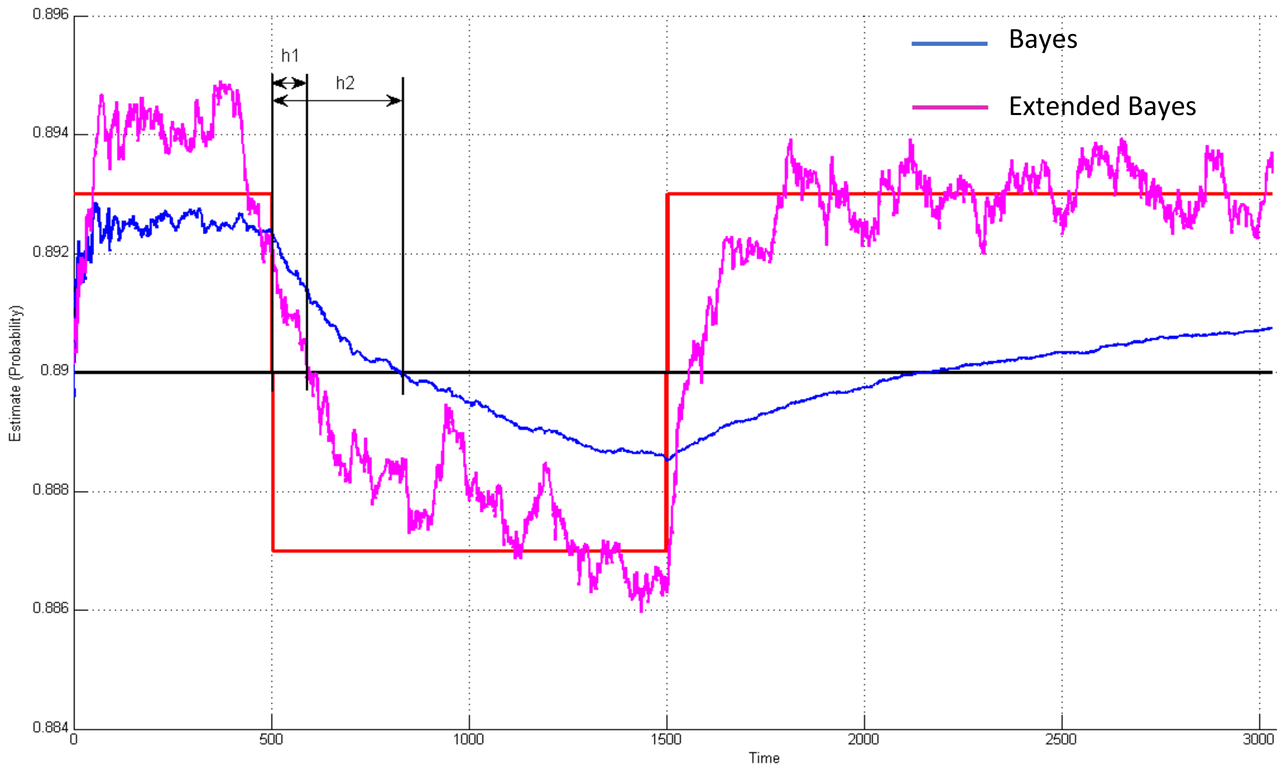


Figure 4.17. Experiment 1's result.

Table 4.6. Setting of the experiment 1 (cf. Table 4.3).

Controlled variable	**Value(s)**
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.887, 0.893 \rangle$
var_Actual	$[0,500,1500]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In the experiment, $h1 < h2$ are time intervals during which the violation of requirements is undetected (erroneous estimation). Both estimations approach quite fast towards p_Actual, but the blue line is more close to the red line before the value of p_Actual drops and a violation happens as a result. Additionally, during the violation and after the actual probability goes to a satisfactory area, the purple line is more close to the actual probability than the blue line.

Experiment number: 2 (Short degradation)

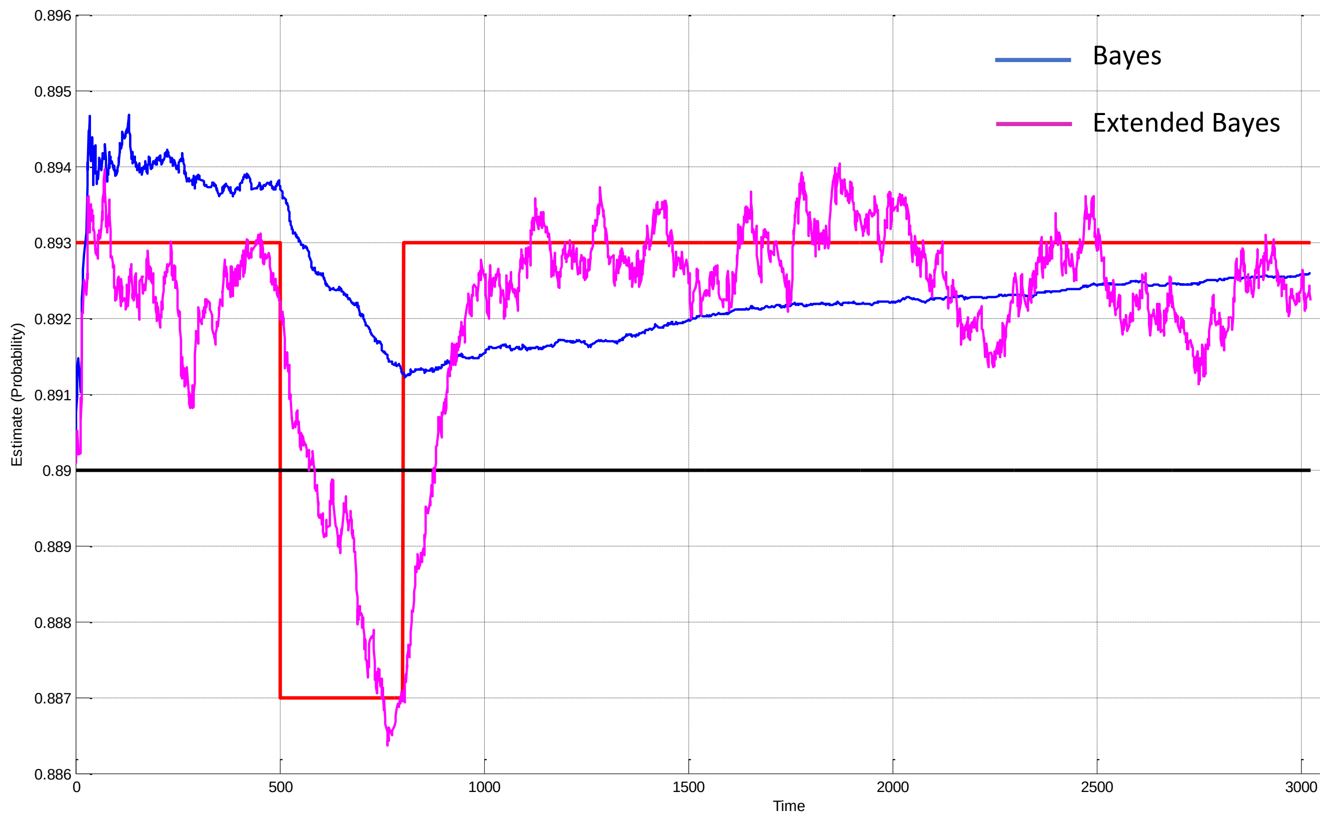


Figure 4.18. Experiment 2's result.

Table 4.7. Setting of the experiment 2 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.887, 0.893 \rangle$
var_Actual	$[0,500,800]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

Although the blue line approaches the actual probability quite fast in the beginning, it could not detect the violation that happens during 300 interval time of simulation.

Experiment number: 3 (Early degradation)

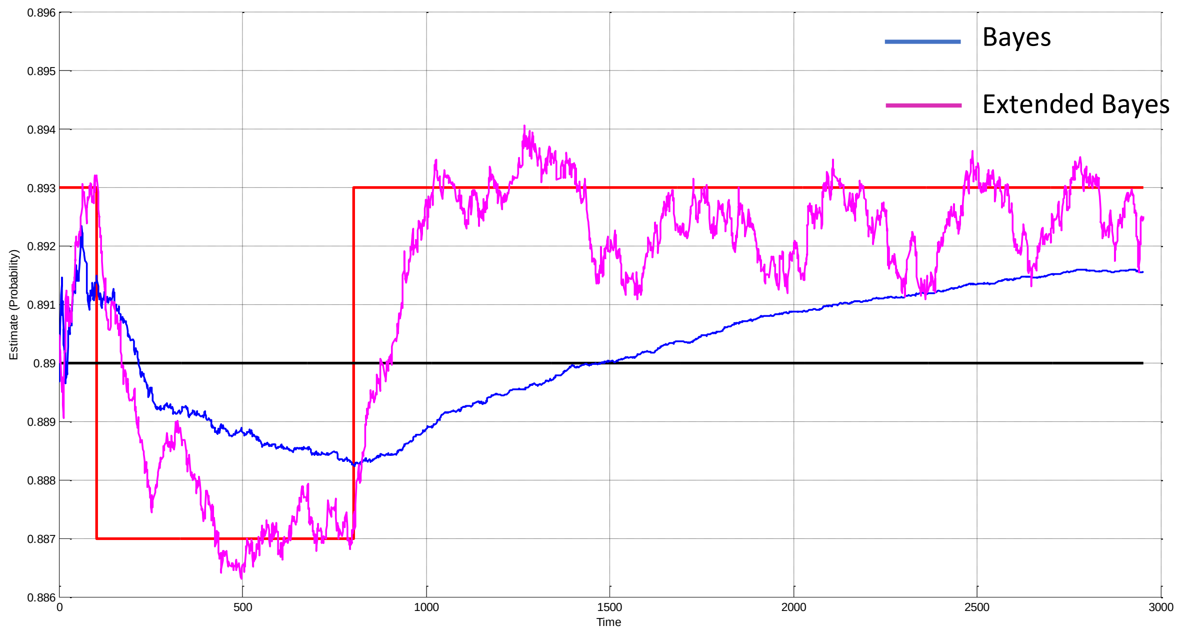


Figure 4.19. Experiment 3's result.

Table 4.8. Setting of the experiment 3 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.887, 0.893 \rangle$
var_Actual	$[0,100,800]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

Although the blue line detects the violation of requirements when the red line drops below the threshold quite close to the purple line, but close to 800 simulation time, the blue line incorrectly classified the observations as violation.

Experiment number: 4 (Late degradation)

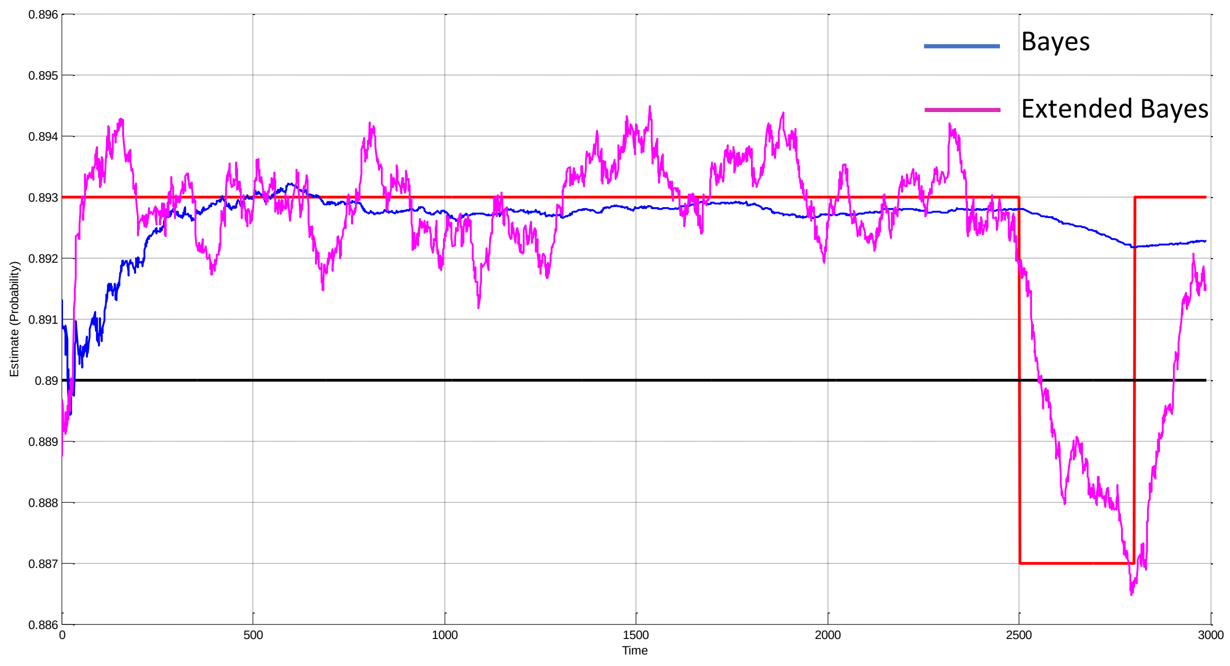


Figure 4.20. Experiment 4's result.

Table 4.9. Setting of the experiment 4 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.887, 0.893 \rangle$
var_Actual	$[0, 2500, 2800]$
a	25
M	1000
λ	1
α	$(1, 1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

Even though the violation has happened when there are enough observations, the blue line again could not detect the violation.

Experiment number: 5 (Shallow degradation)

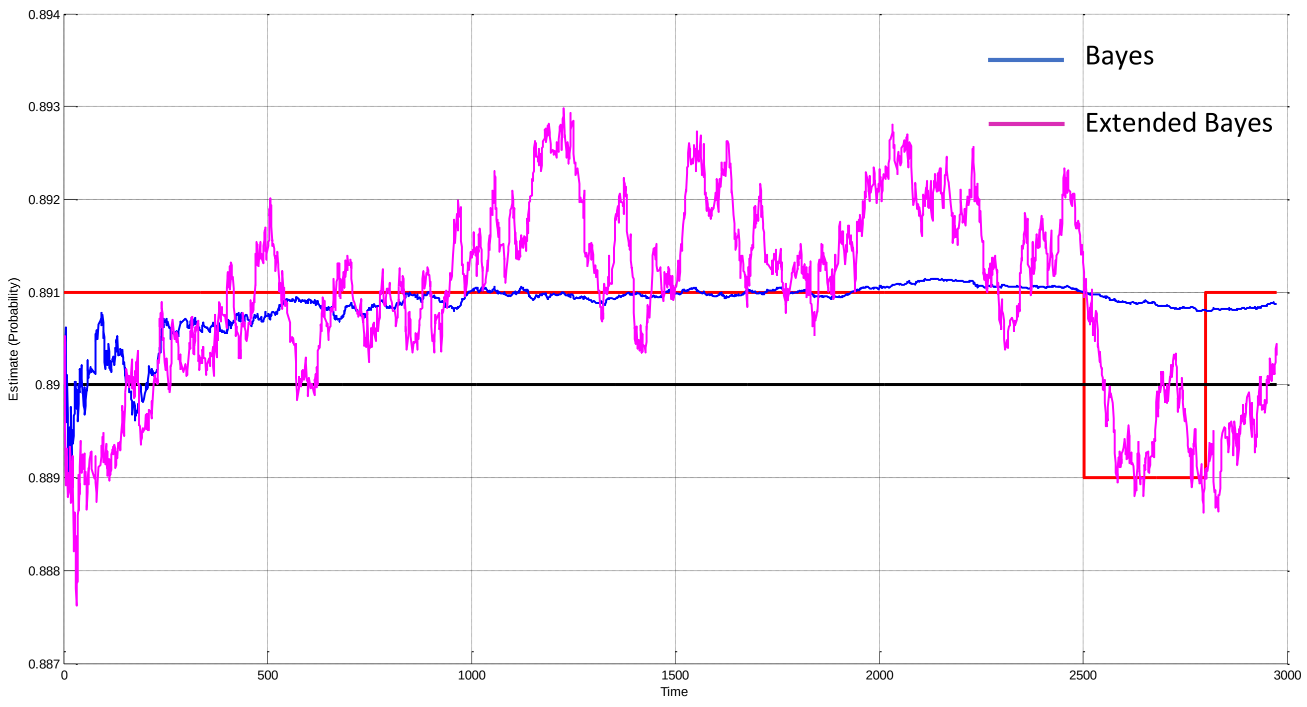


Figure 4.21. Experiment 5's result.

Table 4.10. Setting of the experiment 5 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.891, 0.889, 0.891 \rangle$
var_Actual	$[0, 2500, 2800]$
a	25
M	1000
λ	1
α	$(1, 1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

Even though the actual probability is very close to the threshold, but again the blue line could not identify the violation.

The purple line has some false positives (cf. Figure 4.14) when it crosses the black line both when the requirement is satisfied and when it detects the violation.

Experiment number: 6 (Shallow degradation)

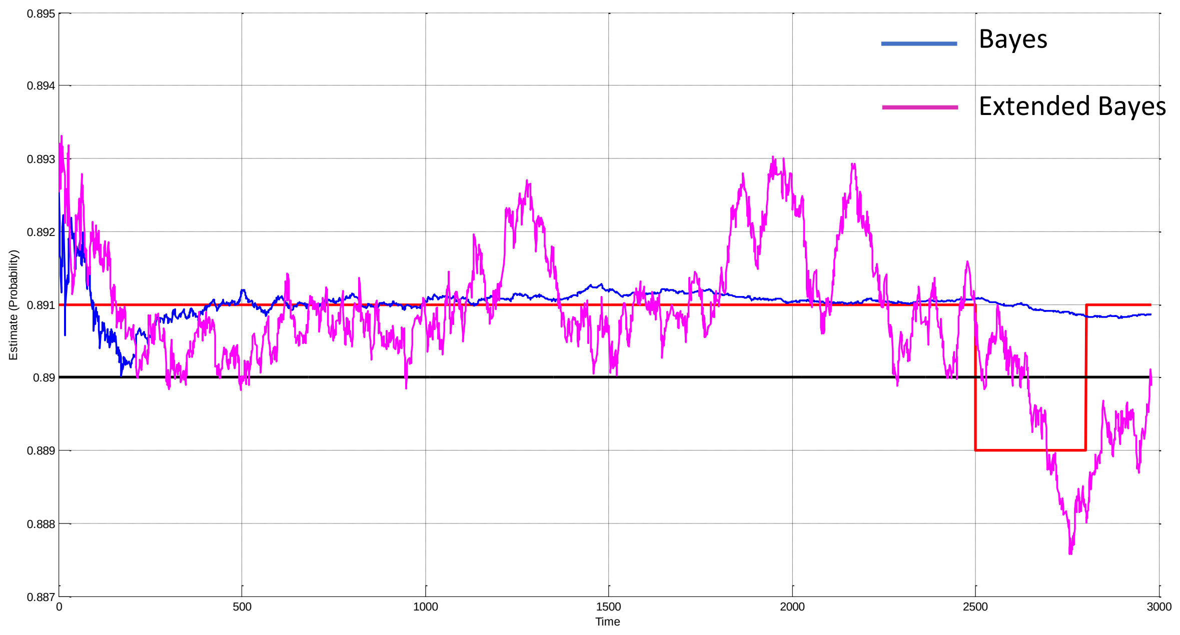


Figure 4.22. Experiment 6's result.

Table 4.11. Setting of the experiment 6 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.893
p_Actual	$\langle 0.891, 0.889, 0.891 \rangle$
var_Actual	$[0,100,800]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

Even when we change the starting point based on estimations at design-time, it did not change the previous observations. We still have unidentified violations and false positives.

Experiment number: 7 (Far starting)

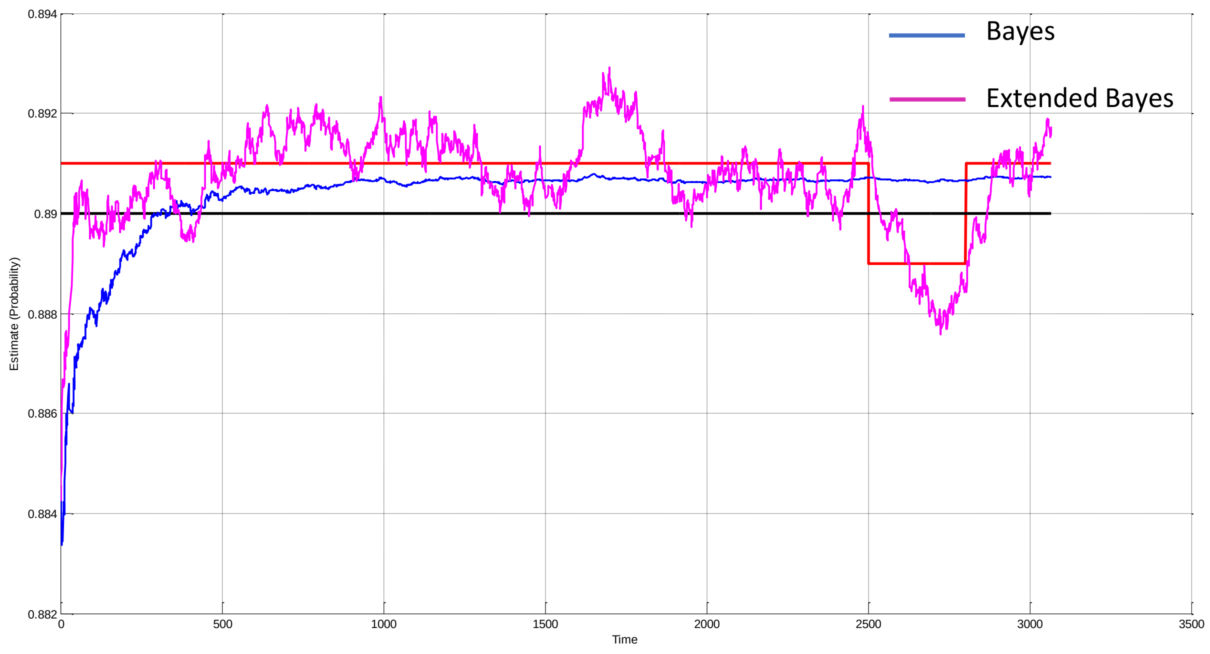


Figure 4.23. Experiment 7's result.

Table 4.12. Setting of the experiment 7 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.884
p_Actual	$\langle 0.893, 0.887, 0.893 \rangle$
var_Actual	$[0,100,800]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting we chose a faraway starting point, but still the same observations of the previous experiments remained.

Experiment number: 8 (Deep degradation)

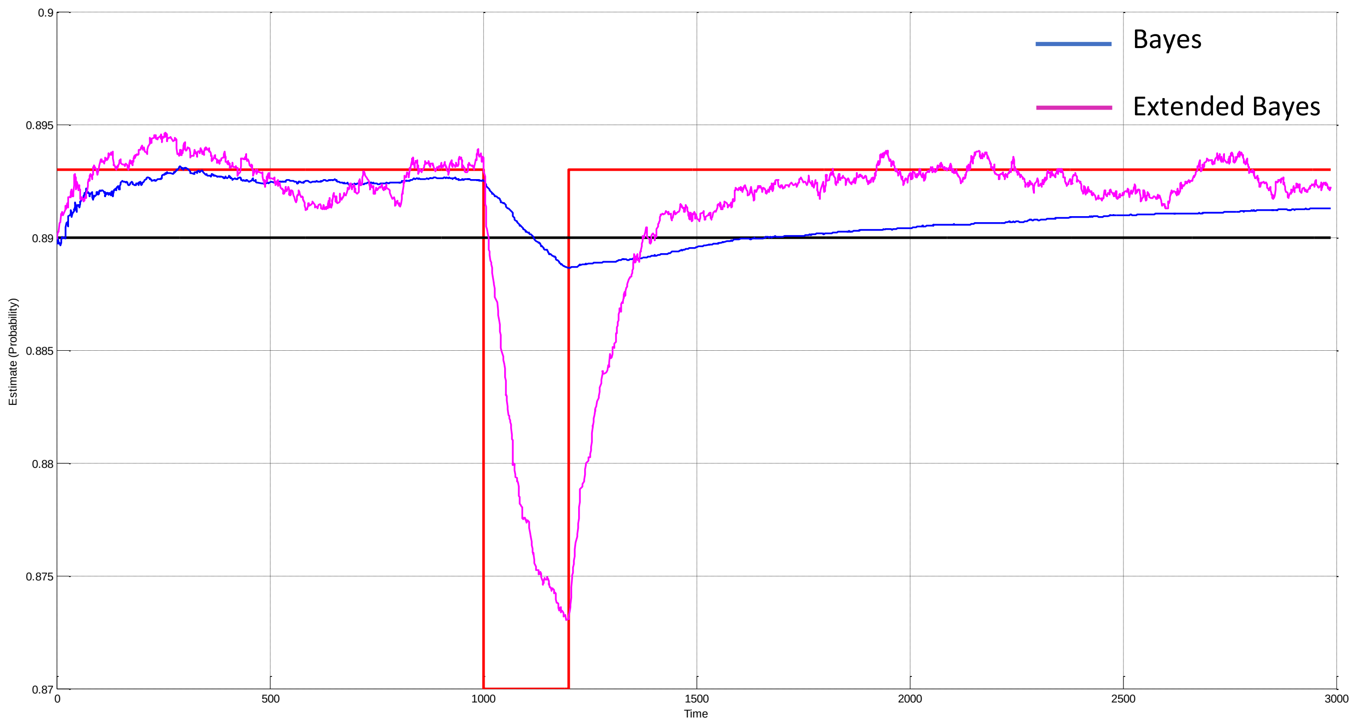


Figure 4.24. Experiment 8's result.

Table 4.13. Setting of the experiment 8 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.87, 0.893 \rangle$
var_Actual	$[0, 1000, 1200]$
a	50
M	1000
λ	1
α	$(1, 1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting, we chose a deep drop in actual probability and the blue line could successfully detect the violation, but in a very slow manner after half of the violation interval is gone.

Experiment number: 9 (Irregular degradations)

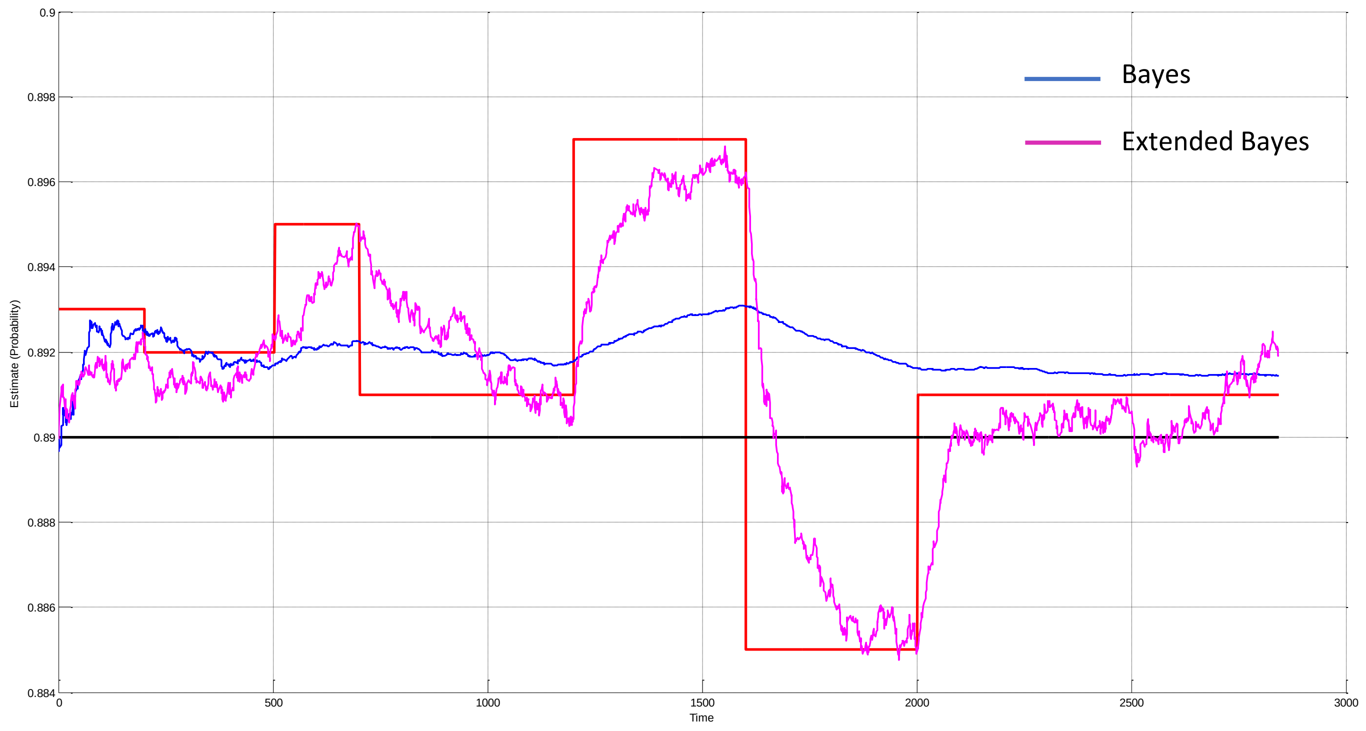


Figure 4.25. Experiment 9's result.

Table 4.14. Setting of the experiment 9 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893,0.892,0.895,0.891,0.897,0.885,0.891\rangle$
var_Actual	$[0,200,500,700,1200,1600,2000]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting, we put a variance into the actual probability which represents one of the features of dynamic environments. The estimation which corresponds to the purple line followed the variation quite well, but most of the time the blue line was insensitive to the variation with a very slow and inefficient approach. Even in this setting the blue line could not detect the violation which injected for 400 simulation time.

Experiment number: 10 (Irregular degradations)

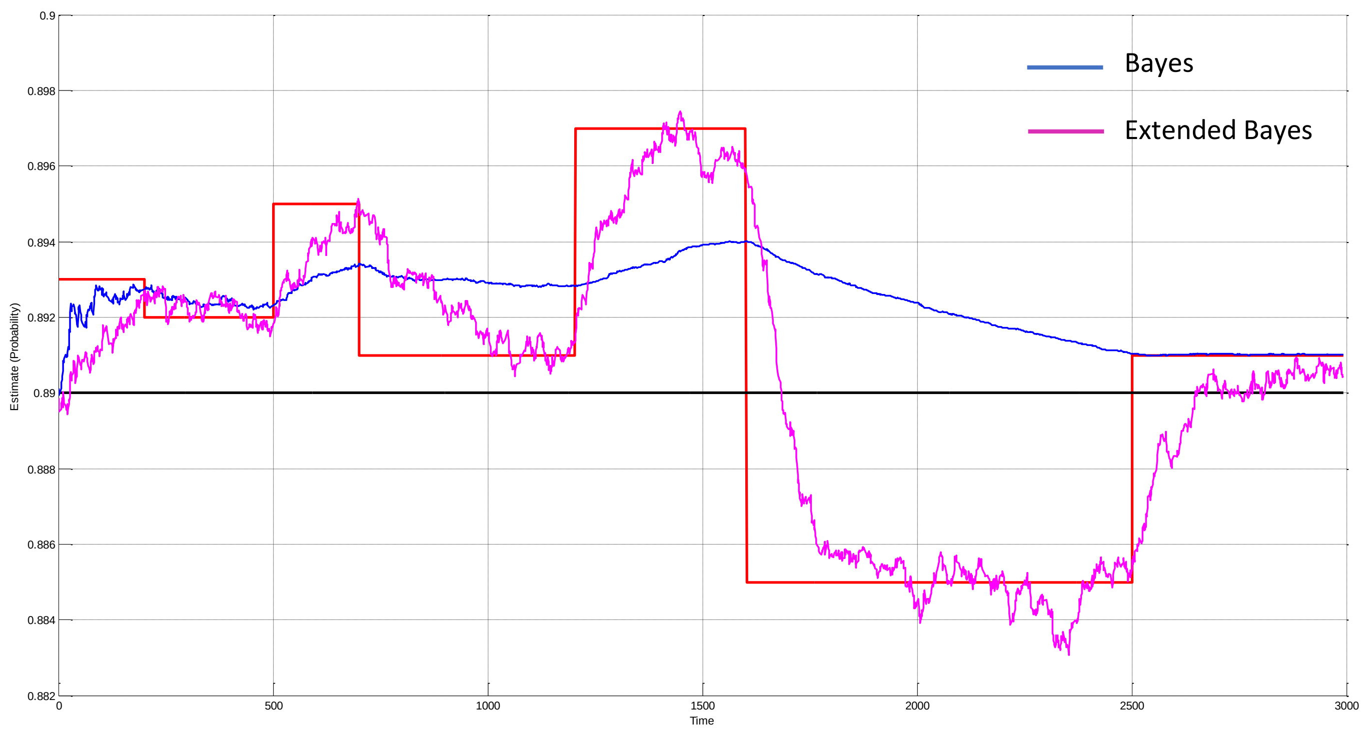


Figure 4.26. Experiment 10's result.

Table 4.15. Setting of the experiment 10 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893, 0.892, 0.895, 0.891, 0.897, 0.885, 0.891 \rangle$
var_Actual	$[0, 200, 500, 700, 1200, 1600, 2500]$
a	25
M	1000
λ	1
α	$(1, 1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting, we have the same variance in the runtime data, but we injected a much longer violation time around 900. Still the blue line could not identify the violation.

Experiment number: 11 (Constant with no degradation)

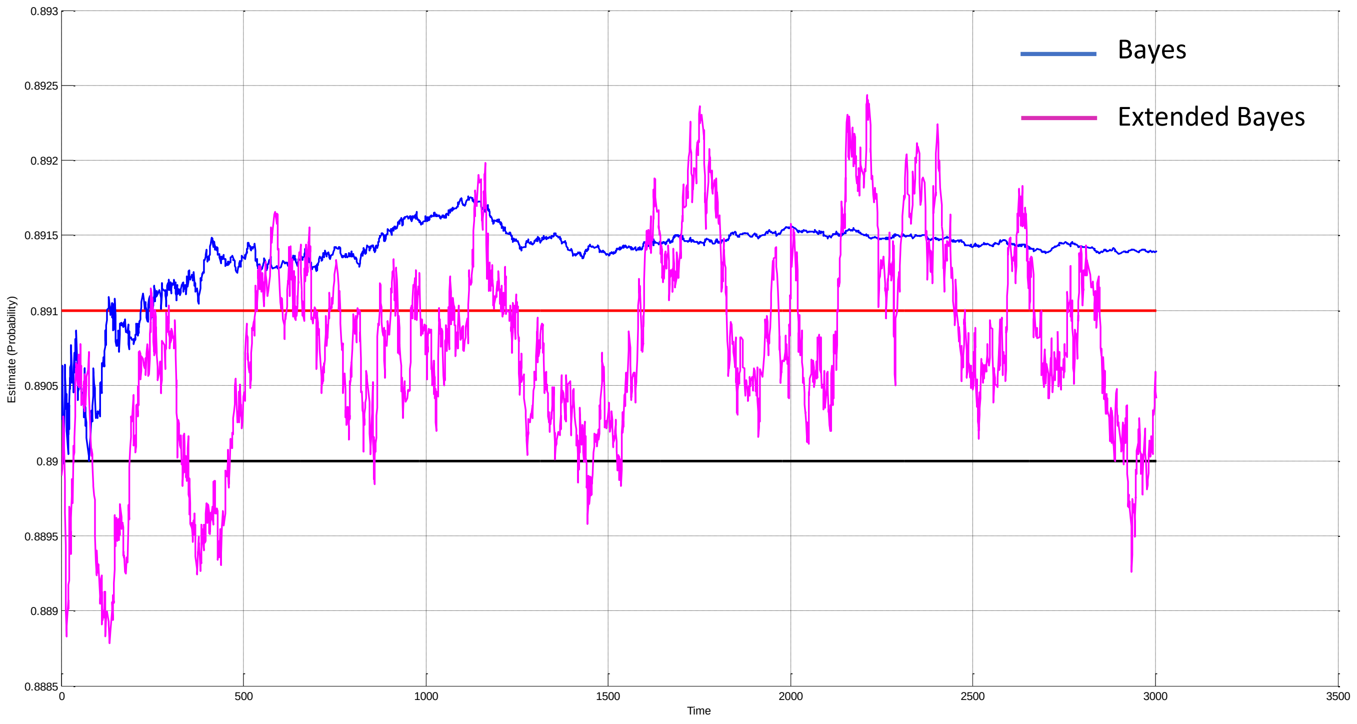


Figure 4.27. Experiment 11's result.

Table 4.16. Setting of the experiment 11 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.891 \rangle$
var_Actual	$[0]$
a	25
M	1000
λ	1
α	$(1,1.01)$
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting we consider a constant actual probability quite near to the threshold. The purple line produced a significant number of false positives in this setting. On the other hand, the blue line had a constant estimation close to the actual probability.

Experiment number: 12 (Balanced degradations)

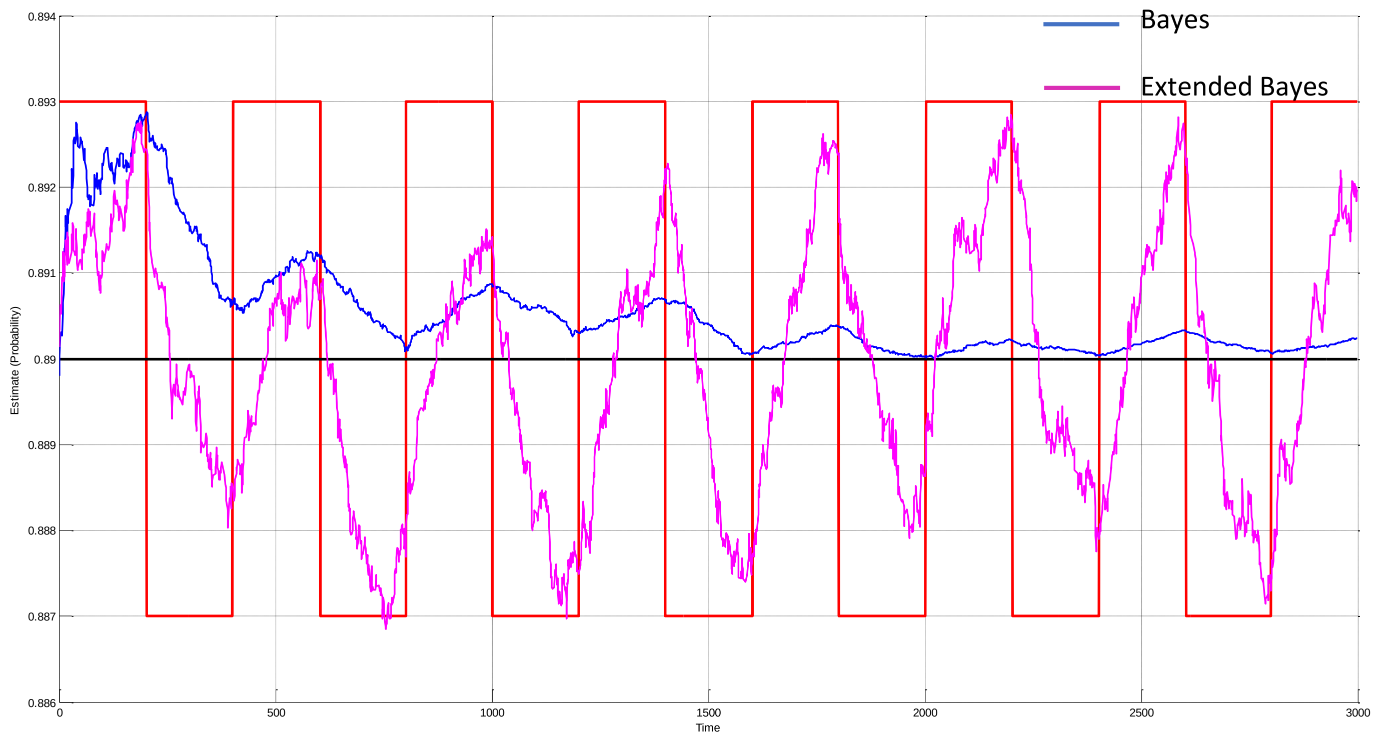


Figure 4.28. Experiment 12's result.

Table 4.17. Setting of the experiment 12 (cf. Table 4.3).

Controlled variable	Value(s)
$\boldsymbol{p\_Threshold}$	0.89
$\boldsymbol{p_{i,j}^{(0)}}$	0.89
$\boldsymbol{p\_Actual}$	$\left\langle \begin{matrix} 0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887, \\ 0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887 \end{matrix} \right\rangle$
$\boldsymbol{var\_Actual}$	$\begin{bmatrix} 0,200,400,600,800,1000,1200,1400,1600 \\ ,1800,2000,2200,2400,2600,2800,3000 \end{bmatrix}$
$\boldsymbol{a}$	25
$\boldsymbol{M}$	1000
$\boldsymbol{\lambda}$	1
$\boldsymbol{\alpha}$	$(1,1.01)$
$\boldsymbol{var\_Threshold}$	Constant
$\boldsymbol{N}$	3000

Experiment observations and interpretations of the results:

In this setting, we put an equivalent satisfaction/violation change pattern in the actual probability and we observed that the purple line could track the changes quite well, but the blue line could identify none of the violations and stayed in the satisfactory area.

Experiment number: 13 (Balanced degradations)

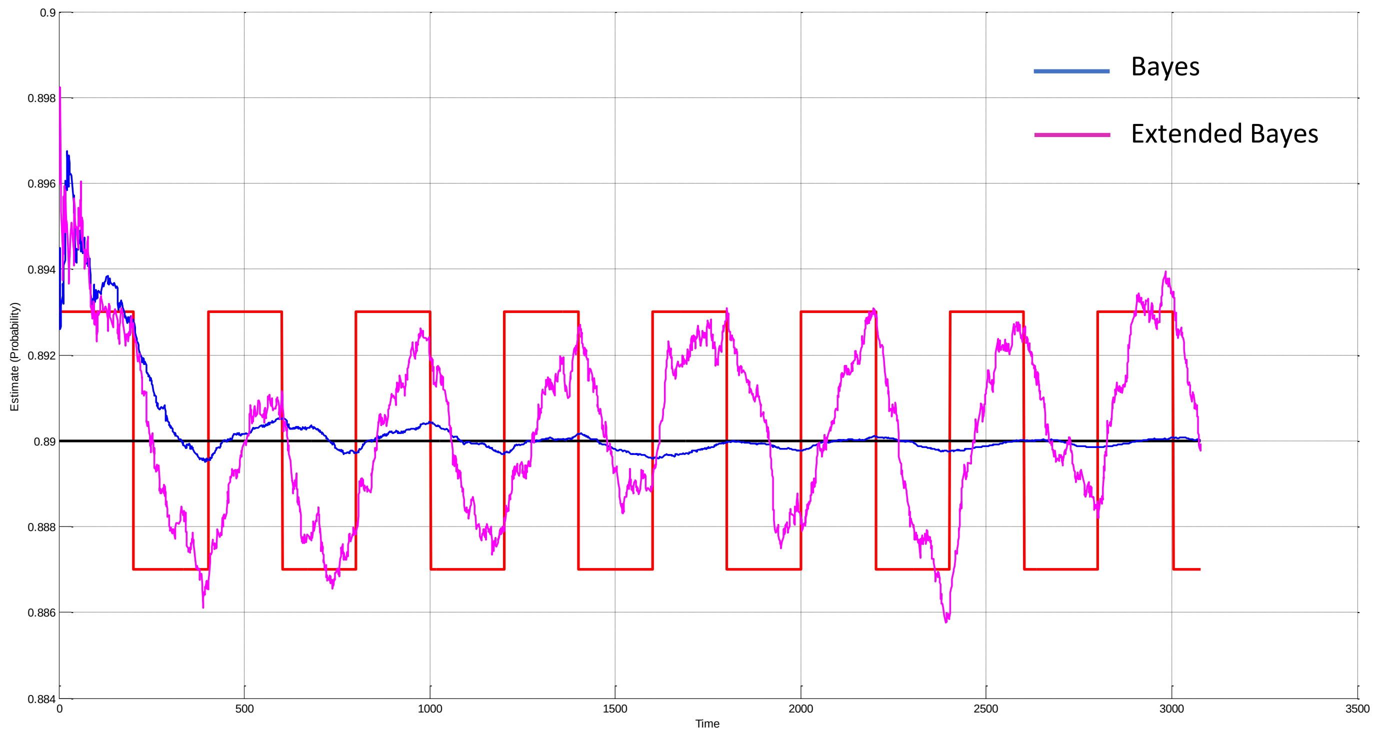


Figure 4.29. Experiment 13's result.

Table 4.18. Setting of the experiment 13 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle 0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887, 0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887 \rangle$
var_Actual	$[0,200,400,600,800,1000,1200,1400,1600, 1800,2000,2200,2400,2600,2800,3000]$
a	1
M	1000
λ	1
α	(1,1.01)
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

We kept the same setting as in Experiment 12, but we considered a very low smoothing parameter and we observed that the blue line could identify the violations, but it could not track the actual probability change pattern very well.

Experiment number: 14 (Balanced degradations)

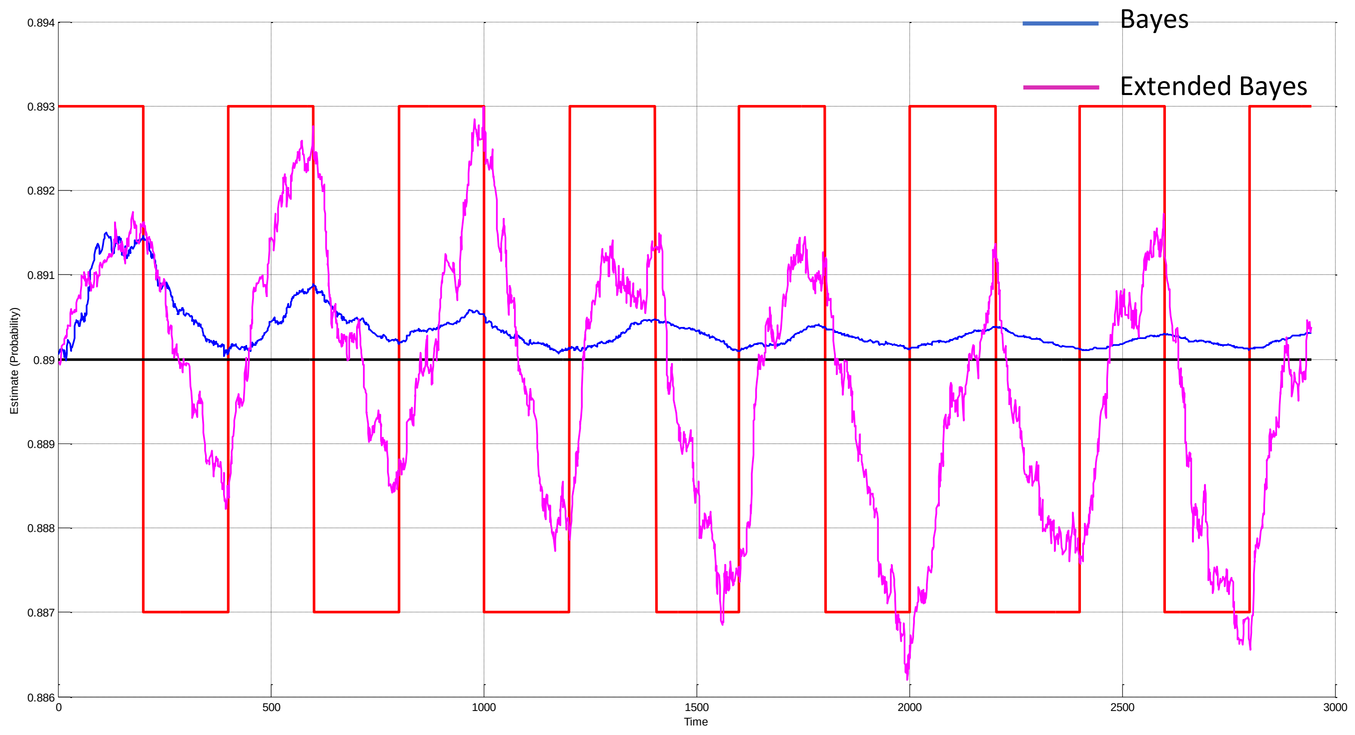


Figure 4.30. Experiment 14's result.

Table 4.19. Setting of the experiment 14 (cf. Table 4.3).

Controlled variable	Value(s)
$p\_Threshold$	0.89
$p_{i,j}^{(0)}$	0.89
p_Actual	$\langle$0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887, 0.893,0.887,0.893,0.887,0.893,0.887,0.893,0.887$\rangle$
var_Actual	[0,200,400,600,800,1000,1200,1400,1600 ,1800,2000,2200,2400,2600,2800,3000]
a	200
M	1000
λ	1
α	(1,1.01)
$var\_Threshold$	Constant
N	3000

Experiment observations and interpretations of the results:

In this setting, we considered a high smoothing parameter and we observed that the blue line could still not identify the violations.

4.4.2.6.5. A qualitative summary of the results

A qualitative analysis of the experimental results in 4.4.2.6.4 shows that the extended Bayes estimation method outperforms the basic Bayes method as summarized in Table 4.20:

- In the experiments, the estimated probability for the extended Bayes approaches the actual probability faster than the basic Bayes. Accordingly, the extended Bayes performs better in detecting the violations.
- In half of the experiments, the basic Bayes method does not produce false positives and even in two experiments (i.e., 4, 5) that produced false positives instances, it performs better than extended Bayes. Both experiments 4 and 5 have one common characteristic. We hypothesize that this outperformance of extended Bayes by its basic counterpart can be attributed to this feature: the lack of change in the actual probability of the parameter, i.e., p_Actual. The only change in the actual probability happens at 300 simulation time between $[2500,2800]$.
- In most of the experiments except 4, 5, 6, 11, the extended Bayes estimates are far more close to the actual probability.
- Although some of the estimates produced by a higher α in the extended Bayes approach much faster than the basic Bayes when a degradation happens, this is achieved at the expense of significant oscillation. Such oscillation is likely to trigger false positives in a real-world scenario, which has the consequences that we described in 4.4.1.

Table 4.20. Overview of the experimental observations; first column: N: could not detect, Y: could detect, Y<: could detect sooner than the other method, Second column: Y: has false positives, Y<: has less false positives than the other method, Third column: Y: is more close to the actual probability.

Experiment	**Detection of violations**		**Appearance of false positives**		**Closeness to actual values**	
	Base	Ext.	Base	Ext.	Base	Ext.
1	Y	Y<	Y	Y<	-	Y
2	N	Y	-	Y	-	Y
3	Y	Y<	Y	Y<	-	Y
4	N	Y	Y<	Y	Y	-
5	N	Y	Y<	Y	Y	-
6	N	Y	-	Y	Y	-
7	N	Y	Y	Y<	-	Y
8	Y	Y	Y	Y<	-	Y
9	N	Y	-	Y	-	Y
10	N	Y	-	Y	-	Y
11	-	-	-	Y	Y	-
12	N	Y	-	Y	-	Y
13	Y	Y	Y	Y<	-	Y
14	N	Y	-	Y	-	Y

Until this point, we reported some observations regarding qualitative evaluation of the proposed estimation approach. In order to claim that the approach provides an accurate parameter estimations for requirement verification, as we claimed in Chapter 1 (see RQ1 and research claim 3), we need to objectively show some evidence of such accuracy. In the following section (i.e., Section 4.4.2.7), we provide quantitative evaluations to assess the accuracy of the proposed approach.

4.4.2.7. *Quantitative evaluations: measuring estimation accuracy*

For evaluating the accuracy of the quantitative estimation technique, in the literature, various quantitative metrics have been utilized. In Section 4.4.2.6.3, we described a number of experiments; we also reported the observations and interpreted the results. In this section, we use some metrics for assessing the estimation methods (basic Bayes and the extended version) that we described in previous sections. More specifically, the effect of different settings for the experiments in estimating the model parameter can be measured more objectively through the error metrics. The estimation of parameters are numerical, but each estimation point have also binary interpretations (satisfaction/violation) as well. This enables us to compare the results through both numerical and binary metrics. An overview of the measurement process is depicted in Figure 4.31.

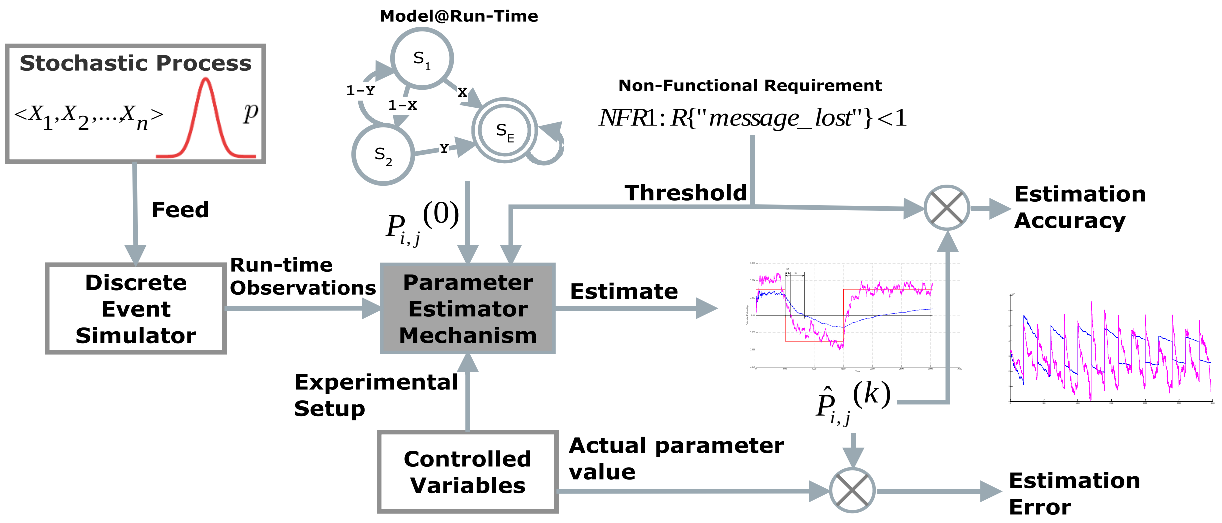


Figure 4.31. Estimation analysis in the context of experimental setup.

4.4.2.7.1. Assessing numeric estimation

Metrics for assessing quantitative estimation measure the size of the error when estimating the value of parameters alongside the runtime data. Table 4.21 lists typical metrics from the literature, which are mostly used in prediction and the data mining domain. Here, we use them for assessing our estimation accuracy. These metrics are meant to quantify the difference between values implied by an estimator and the actual values of the parameter to be estimated.

In statistics, Mean Squared Error (MSE) is the most basic metric, which incorporate both the variance of the estimator and its bias. In an analogy to standard deviation, taking the square root of MSE yields the Root Mean Squared Error (RMSE), which has the same units as the parameter being estimated. Mean Absolute Error (MAE) takes the average of the absolute errors and is known to be more robust against data points that are very much higher or smaller than the next nearest data points. Relative errors including RSE, RRSE and RAE, which are normalized by the error of a naïve estimator (average of the past actual probabilities) to ease comparison. For the above metrics, smaller values indicate a more accurate estimation.

Table 4.21. Estimation error metrics (Witten & Frank, 2005). $\hat{p}$ estimated, p_Actual actual probability, $\overline{p\_Actual}$ average of actual probability.

Metric	Formula
Mean Squared Error (MSE)	$\frac{(\hat{p}^{(1)} - p\_Actual^{(1)})^2 + \cdots + (\hat{p}^{(d)} - p\_Actual^{(d)})^2}{d}$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{(\hat{p}^{(1)} - p\_Actual^{(1)})^2 + \cdots + (\hat{p}^{(d)} - p\_Actual^{(d)})^2}{d}}$
Mean Absolute Error (MAE)	$\frac{\lvert p^{(1)} - p\_Actual^{(1)}\rvert + \cdots + \lvert p^{(d)} - p\_Actual^{(d)}\rvert}{d}$
Relative Squared Error (RSE)	$\frac{(\hat{p}^{(1)} - p\_Actual^{(1)})^2 + \cdots + (\hat{p}^{(d)} - p\_Actual^{(d)})^2}{\left(p\_Actual^{(1)} - \overline{p\_Actual}\right)^2 + \cdots + \left(p\_Actual^{(d)} - \overline{p\_Actual}\right)^2}$
Root Relative Squared Error (RRSE)	$\sqrt{\frac{(\hat{p}^{(1)} - p\_Actual^{(1)})^2 + \cdots + (\hat{p}^{(d)} - p\_Actual^{(d)})^2}{\left(p\_Actual^{(1)} - \overline{p\_Actual}\right)^2 + \cdots + \left(p\_Actual^{(d)} - \overline{p\_Actual}\right)^2}}$
Relative Absolute Error (RAE)	$\frac{\lvert p^{(1)} - p\_Actual^{(1)}\rvert + \cdots + \lvert p^{(d)} - p\_Actual^{(d)}\rvert}{\lvert p\_Actual^{(1)} - \overline{p\_Actual}\rvert + \cdots + \lvert p\_Actual^{(d)} - \overline{p\_Actual}\rvert}$

It is evident that it is not feasible, and no specific value will be added, if we measure and compare the performance of the method quantitatively for *all* the performed experiments. We, therefore, chose experiment number 14 to investigate more carefully the two estimation approaches through the error metrics. We had one specific reason for this choice: the number of variations in p_Actual is quite high in this experiment and this makes it appropriate for quantitative evaluation of the estimation approach. Note that the quantitative results that we report in this section are specific to this experiment. However, all conclusive remarks and interpretations of the results are backed up with the other experiments as well. As we mentioned earlier, it is not practical in terms of space to report them all here in this document.

Figure 4.32 shows the point estimate errors of the basic Bayes (blue line) and the extended Bayes (purple line). The point estimate errors are summarized by boxplot in Figure 4.33. Two observations can be made. First, the mean error for the extended Bayes is smaller than the basic version. Second, there are some estimates (those as outliers and in the third quintiles) that produced higher errors than the highest error produced by a basic Bayes.

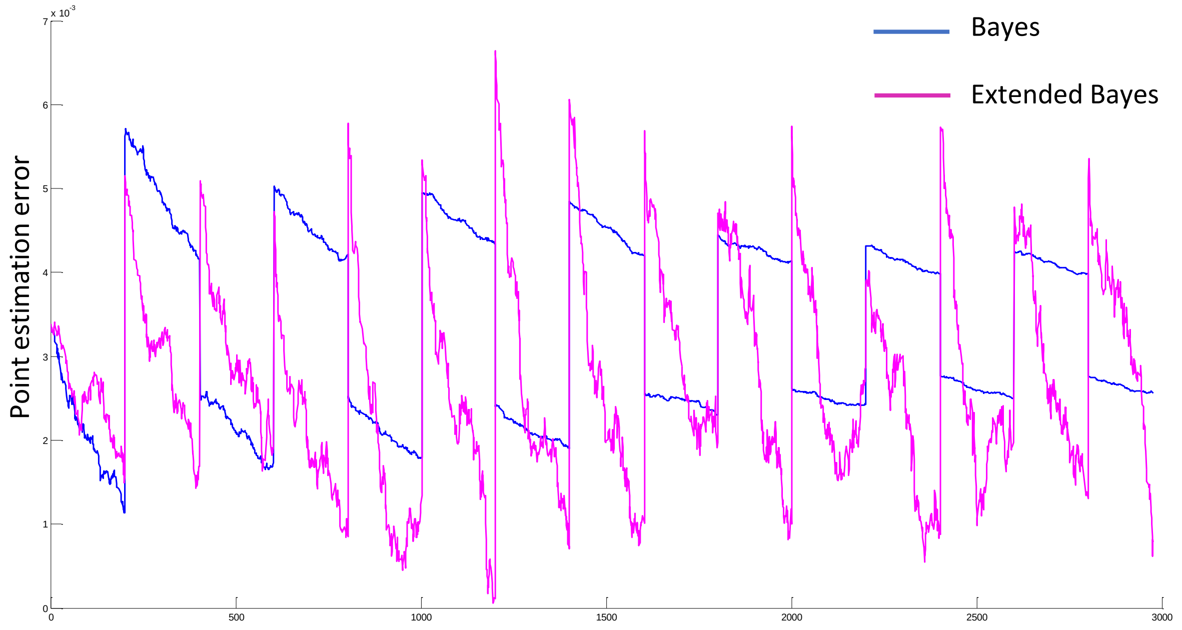


Figure 4.32. Point estimation error for the experiment number 14.

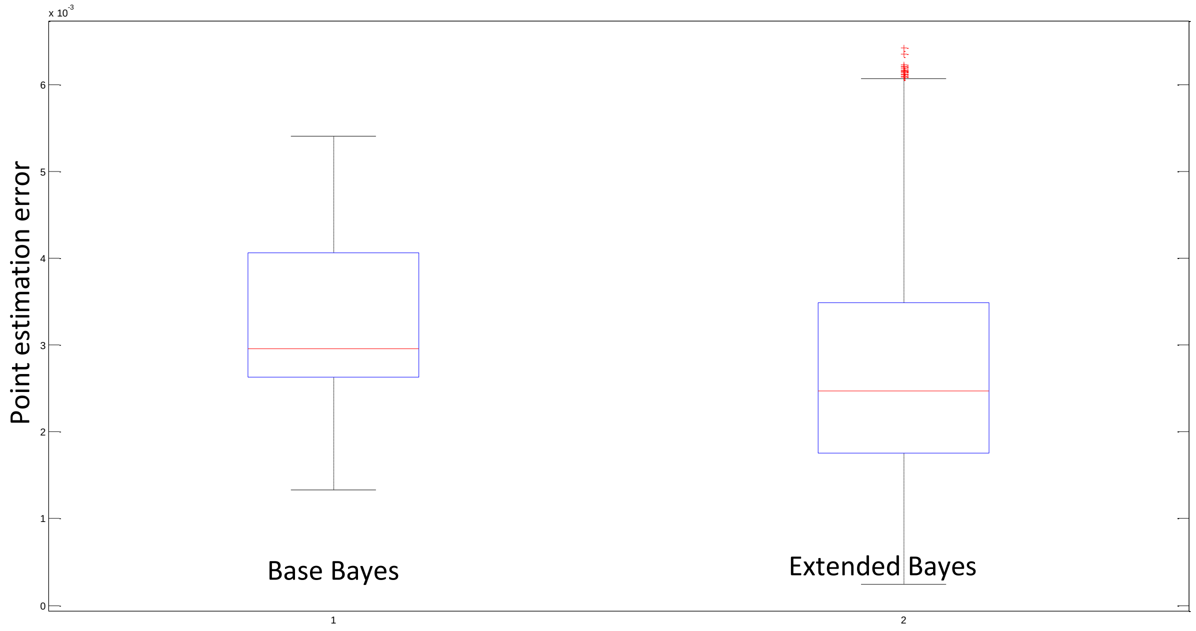


Figure 4.33. Comparison of point estimation errors for the experiment number 14.

We measured the metrics defined in Table 4.21 for experiment number 14 and the results are shown in Table 4.22. Based on this comparison, some observations can be made. Since MSE incorporates both the variance of the estimator and its bias and, as it is evident in Figure 4.33 that there are some noticeable outliers, the basic Bayes rule, according to the MSE metric, performs better than the extended version. However, the values of the other 5 metrics reveal that the extended Bayes performs better than the basic version especially according to RMSE that has the same units as the parameter being estimated.

Table 4.22. Estimation error measurements for the experiment number 14.

Metric	Value ($\alpha = 1$)	Value ($\alpha = 1.01$)
MSE	9.6745e-06	6.9190e-06
RMSE	0.0031	0.0026
MAE	0.0030	0.0024
RSE	1.0780	0.7710
RRSE	1.0383	0.8781
RAE	0.9866	0.8052

4.4.2.7.2. Deficiencies of the error metrics

The estimation error metrics have been utilized for SLA violation prediction in service-oriented systems (Cavallo, Di Penta, & Canfora, 2010; Leitner, Michlmayr, Rosenberg, & Dustdar, 2010). Although the metrics show the accuracy in terms of difference between the estimated value and the actual value, they are not be able to reveal that the violation of requirements actually occurred. More specifically, let us consider the following two scenarios:

1. The difference between the estimated and the actual value is small, but the actual value is just below the threshold and the estimated value is just above as depicted in Figure 4.34 (Experiment number 8). In this scenario, the metrics show relatively low values, but actually, the estimation is incorrect.
2. The difference is quite large, but both the actual and estimated value are above or below the threshold as depicted in Figure 4.34 (Experiment number 8). In this scenario, the metrics show a relatively high value, but the estimation on the other hand is correct.

This shows that the metrics are not comprehensive for evaluating the accuracy of estimation approach. Therefore, we should also consider accuracy metrics that consider violations, not only the numerical values. They are introduced in the next section.

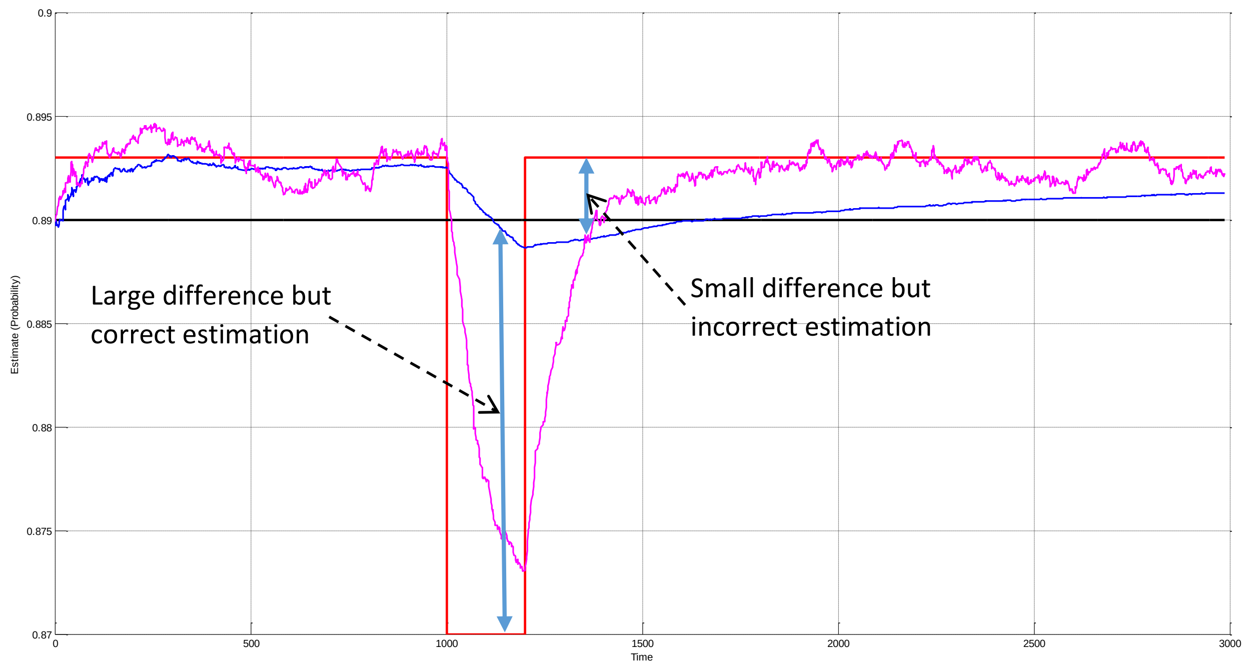


Figure 4.34. Shortcomings of estimation error numerical metrics.

4.4.2.7.3. Assessing binary estimation

The decision to perform adaptation not only depends on the estimated value of interest, but also depends whether or not it actually is a violation of requirements. Therefore, in order to examine how accurately we can estimate the necessity for adaptation, we need to take into account how accurately those violations can be detected.

In this section, we discuss metrics that can be utilized to evaluate the accuracy of such estimation, i.e., estimation of violation or non-violation. These metrics are derived from the well-known contingency table (Table 4.23), which characterizes the four cases, which can result from a binary estimation of violation.

Table 4.23. Contingency table.

		Estimation	
		Violation	Non-violation
Actual	Violation	True Positive (TP)	False Negative (FN)
	Non-violation	False Positive (FP)	True Negative (TN)

A number of metrics derived from the contingency table have been proposed in the literature, e.g., (Salfner, Lenk, & Malek, 2010). Table 4.24 lists a selection of the most commonly used metrics. Precision (P) can be used to evaluate incorrectly detected violations, i.e., unnecessary adaptations. Higher precision means less unnecessary adaptations. On the other hand, recall (R) can be associated with missed adaptations. Higher recall means more actual violations being estimated and therefore less missed ones. Generally, in order to perform well, an estimation algorithm should attain both high precision and recall. However, improving precision may result in worse recall. For instance, if an estimation algorithm detects only 1 true violation, its precision becomes 1. In addition, if an algorithm reaches recall 1 by always detecting violations, its precision becomes low. Therefore, there should be a tradeoff between these two metrics, which is reflected by the F-measure ($\boldsymbol{F_\beta}$). The false positive rate (FPR) is defined as a ratio of incorrectly estimated violations to the number of all non-violations. The smaller it is, the better. On the other hand, the negative predictive value (NPV) is defined as the ratio of incorrectly predicted violations to the number of all non-violations. The higher it is, the better it reflects prediction accuracy. Similarly to precision, specificity (S) can be utilized to evaluate incorrect adaptation needs. Higher specificity implies fewer unnecessary adaptations. Accuracy (A) is the ratio of correct estimation to all estimations performed. In order to have a comprehensive picture of prediction accuracy, we use all of the metrics to measure the accuracy of the estimations in our controlled experiments.

Table 4.24. Contingency table metrics (Salfner et al., 2010).

Metric	Formula	Meaning
Precision (P)	$\frac{TP}{TP+FP}$	How many detected violations were actually a violation?
Recall (R)	$\frac{TP}{TP+FN}$	How many actual violations were correctly detected as violations?
Specificity (SP)	$\frac{TN}{TN+FP}$	How many actual non-violations were correctly detected as non-violations?
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$	How many detected violations were actual non-violations?
Negative Predictive Value (NPV)	$\frac{TN}{TN+TP}$	How many predicted non-violations were actual non-violations?
Accuracy (A)	$\frac{TP+TN}{TP+TN+FP+FN}$	How many detections were correct?
F-measure ($\boldsymbol{F_\beta}$)	$\frac{(1+\beta^2)P*R}{\beta^2P+R}$	Harmonic mean of P and R

Let us now consider a situation in which a violation is detected if the estimated probability $\hat{p}^{(d)}$ (representing the reliability of a system connector channel, as explained above) drops below a threshold value $p\_Threshold$. The threshold value is shown as a horizontal black solid line in all graphs representing the experiments in Section 4.4.2.6.4. Assuming that the estimation methods are used to detect such violations of a reliability threshold, we measured the following properties of the estimated $\hat{p}^{(d)}$ values from Experiment number 14:

- The number of false positives (FP), i.e., instances when $\hat{p}^{(d)}$ drops below $p\_Threshold$ although $p\_Actual^{(d)}$ has its normal value.
- The number of false negatives (FN), i.e., instances when $\hat{p}^{(d)}$ has its normal value while $p\_Actual^{(d)}$ were below $p\_Threshold$.
- The number of true positives (TP) and true negatives (TN), i.e., instances when both $\hat{p}^{(d)}$ and $p\_Actual^{(d)}$ drop below $p\_Threshold$ or both have their normal values.

We then measured the metrics defined in Table 4.24 for experiment number 14 and the results are shown in Table 4.25. Based on this comparison, some observations can be made. Since all the estimations by the basic Bayes were above the threshold in the non-violation area, the precision and F-measure are not defined for this case. Nonetheless, these two measures are quite promising for the extended Bayes approach. In terms of recall and accuracy, the extended version shows its superiority over the basic one.

Table 4.25. Contingency metrics measurement for the experiment number 14.

Metric	Value ($\boldsymbol{\alpha = 1}$)	Value ($\boldsymbol{\alpha = 1.01}$)
P	--	0.7155
R	0	0.3615
S	1	0.8673
FPR	0	0.1327
NPV	1	0.7222
A	0.52	0.6245
F	--	0.4803

4.4.2.7.4. Deficiencies of the contingency table metrics

Although the contingency table metrics enable us to evaluate the accuracy of the need for adaptations, they are heavily sensitive to the threshold values used by the system for determining violations during estimation. Figure 4.35 shows the experiment in the previous section with an additional threshold value with a slight increase. It is apparent that many false negatives now become true positives and a number of true negatives become false positives. The metrics are reevaluated and shown in Table 4.26. Although the difference in threshold values is just 0.04, the differences in the binary metrics are significant. Table 4.26 shows the value of these binary metrics, separately for the basic and extended Bayes. These results indicate that the extended Bayes is suited for identifying the change in the actual probability $p\_Actual^{(d)}$, whereas the basic Bayes method yields probability estimates that follow the changes in $p\_Actual^{(d)}$ with far less accuracy. As a result, it does not perform as well as the extended version in terms of the binary metrics.

In order to have more informed decisions and to avoid missed or unnecessary adaptations, the estimation error metrics should be employed along with the contingency table metrics. In this way, we are able to better understand how reliable the estimation algorithm is in terms of accuracy.

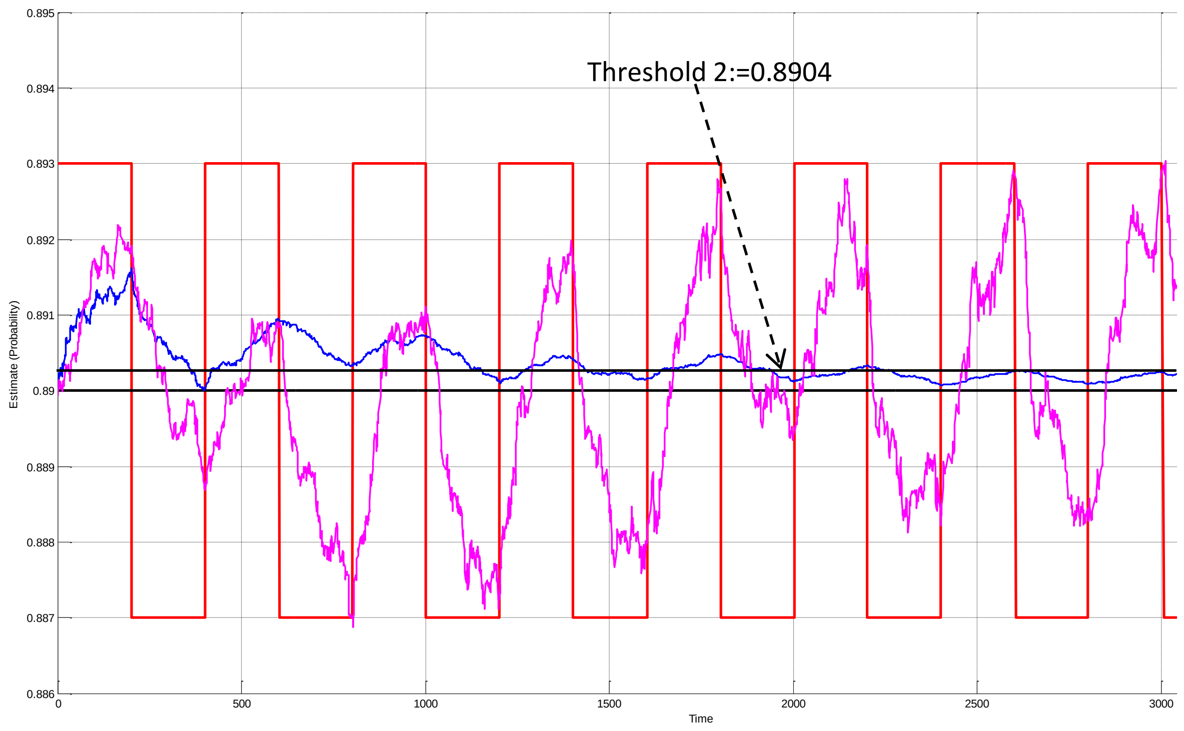


Figure 4.35. Sensitivity of contingency table metrics to the threshold.

Table 4.26. Evaluated contingency metrics for the experiment 14 after changing the threshold value.

Metric	Value ($\boldsymbol{\alpha} = \mathbf{1}$)	Value ($\boldsymbol{\alpha} = \mathbf{1.01}$)
P	0.5430	0.5935
R	0.7117	0.7394
S	0.4290	0.5173
FPR	0.5710	0.4827
NPV	0.3874	0.4233
A	0.5670	0.6257
F	0.6160	0.6585

4.4.2.8. Limitations and Threats to validity

The experiments in Section 4.4.2.6.4 indicate that the effectiveness of the DTMC parameter estimation methods depends on the choice of the smoothing parameter, which is the confidence with initial design-time estimation, and α, which is used in Equation (4.25). Additionally, no combination of values for these two parameters is actually suitable for all potentially infinite runtime scenarios. To address this limitation, one may actually select suitable parameters for these two values depending on the runtime condition.

In Section 4.4.2.6.4, we only showed 14 experiments by combining different values for the controlled variables and the objective was to design enough experiments to demonstrate the effectiveness of the extended Bayes method for estimating the unknown parameters of the runtime models in different situations. We also compared their effectiveness for detecting requirement violations at runtime. However, the comparison is only limited to these 14 settings, but the runtime situations are potentially infinite. Although we reported both qualitative observations in Section 4.4.2.6.5 and quantitative comparison in Section 4.4.2.7, the limitations of the controlled experiments is one of the threats to the validity of this work. The only strategy for mitigating this threat was to extend the experimental setting and perform a case study evaluation. We will report the results regarding this in the evaluation chapter, i.e., Chapter 7.

4.4.3. Estimation of transition matrix of a CTMC

In Section 4.4.2, we proposed an approach based on the principles of Bayes theory to estimate the unknown parameters of DTMC models based on runtime measurements. We experimentally evaluated the performance and effectiveness of the approach for verifying a reliability requirement $R1$ as specified in Statement (4.19). DTMC models are suitable for specifying reliability properties as in the case of $R1$, see Section 4.2.2 and background Chapter 2. However, some other non-functional properties are required to be verified at runtime to trigger an adaptation. For example, performance properties within which we can specify an end-to-end delay of message transmission (i.e., the time that it takes to transmit a message when it enters to the connector from one of its source ends to the time that it leaves the connector from one of its sink ends) are typically used for specifying service level agreements. As a result, such performance properties need to be considered for enabling a self-adaptation of connectors. CTMC models as we discussed earlier in Section 4.2.2 are appropriate models that we can adopt to specify such non-functional properties. In this section, a similar flow of content as in Section 4.4.2 is followed, but we propose an approach to estimate the unknown parameters of CTMC models.

For estimating the parameters of a CTMC model (in statistics, the inference of generator matrix is a more popular term), given continuously observed sample paths of the model is straightforward. In our context, the interpretation of continuously observed sample path refers to fully time-stamped runtime data, which in most cases improbable or infeasible.

In this case, the maximum likelihood estimation is fully tractable if continuous observations for the stochastic process $\{X(t)|0 \leq t \leq T\}$ are available. Let us consider the likelihood of observations with a transition from state i to state j at time τ_1, followed by a subsequent transition from j to k at time τ_2 and so on. Supposing that an initial state probability is known, the likelihood will be:

$$L(Q) = e^{-q_i(\tau_2-\tau_1)} q_{i,j} * e^{-q_j(\tau_3-\tau_2)} q_{j,k} * \dots = \prod_{i=1}^{K} \prod_{i \neq j} e^{-q_i R_i(T)} * q_{i,j}^{N_{i,j}(T)} \tag{4.27}$$

, where $R_i(t)$ is the total time during which the model is in state i by time t and $N_{i,j}(t)$ is the number of transitions between state i to state j by time t. The log-likelihood is:

$$Log\ L(Q) = \sum_{i=1}^{K} \sum_{i \neq j} \log(q_{i,j})\, N_{i,j}(T) - \sum_{i=1}^{K} \sum_{i \neq j} \mathrm{q}_{i,j}\, R_i(T) \tag{4.28}$$

Therefore, the maximum-likelihood estimator for the parameters of CTMC (elements of a generator matrix) is:

$$\hat{q}_{i,j} = \frac{N_{i,j}(T)}{R_i(T)} \tag{4.29}$$

In this ideal case, given continuously observed sample paths, the sufficient statistics are simply the number of transitions between any two states and the total time spent in each state. In a real-world context and especially in our application domain of component connectors, however, a complete runtime observation is not given, but only an *incomplete* and *noisy* observation of the runtime data is available. This level of uncertainty concerning the running system and its environment is due to a number of reasons (Esfahani & Malek, 2013). The reasons may include:

- Different levels of abstractions between the system and runtime models
- Adaptive and not continuous monitoring
- Unobservable phenomena
- Measurement error

We deal with incomplete observations at runtime, which is a realistic approach for continuous-time models in this section. More specifically, the stochastic process $\{X(t)|0 \leq t \leq T\}$ serves as a continuous-time model for data sampled at discrete time points: $t_0 = 0 < t_1 < \cdots < t_N = T$. In this work, we estimate the parameters based on simulation of continuous sample paths from the CTMC conditional on $X(t_0), \ldots, X(t_N)$. By considering the Markov property, knowledge of the runtime data $X(t_0), \ldots, X(t_N)$ partitions the model into independent models $\{X(t)|t_i \leq t \leq t_{i+1}\}$ whose endpoints (i.e., $X(t_i)\ and\ X(t_{i+1})$) are known. Therefore, sampling a realization from the stochastic process $\{X(t)|0 \leq t \leq T\}$ given the observed data is equal to sampling from N independent and identical (i.i.d.) models each conditioned on their endpoints $X(t_i)\ and\ X(t_{i+1})$ between two time points $[t_i, t_{i+1}]$. In the next section, a number of well-known sampling strategies are reviewed.

4.4.3.1. Sampling strategies for endpoint-conditioned CTMC

In this section, we review the strategies for constructing a realization of a finite-state CTMC $\{X(t)|0 \leq t \leq T\}$ conditional on its beginning state $X(t_0) = a$ and ending state $X(t_N) = b$. Sampling is the heart of the estimation approach we propose for estimating the parameters of CTMC models. A CTMC is characterized by its generator matrix Q consisting of transition rates, which are specified at design-time based on the available data.

Simulating a sample path of the CTMC model $\{X(t)|0 \leq t \leq T\}$ is a simple iterative loop. The key point is that the waiting time for the first move (state transition) is exponentially distributed with the mean $\frac{1}{q_{a,a}}$. If $\tau_1 > T$, then there is no move in the interval time $[0, T]$ and the corresponging sample path is constant $X(t) = a$. Otherwise, a new state s_1 is drawn from the discrete random variable with mass $\frac{q_{as_1}}{q_{aa}}$ and the loop is iterated for the time interval of $[\tau_1, T]$.

4.4.3.1.1. Forward sampling

Algorithm 1. Forward Sampling **(adapted from** (Inamura, 2006)**)**

```
1. Sample $\boldsymbol{\tau \sim Exp\ (q_{a,a})}$.
    a. If $\boldsymbol{\tau \geq T}$, the algorithm terminates and $\boldsymbol{X(t) = a}$ for all $\boldsymbol{\tau \in [0, T]}$.
    b. If $\boldsymbol{\tau < T}$, choose a new state $\boldsymbol{s_1 \neq a}$ from the discrete random variable with mass $\boldsymbol{\frac{q_{as_1}}{q_{aa}}}$.
2. Repeat the procedure with the new start state $\boldsymbol{s_1}$ and new time interval $\boldsymbol{[\tau, T]}$
```

Forward sampling proceeds under the assumption that the ending state $X(T)$ is unobserved. However, under the assumption that the end state $X(T) = b$ is observed, conditioning excludes all the paths sampled from the forward sampling that fail to reach the state b at the end.

4.4.3.1.2. Rejection sampling

Naïve rejection sampling uses forward sampling to produce candidate sample paths of the CTMC model and then rejects those paths that cannot end in the observed end state. In particular, when sampling forward the probability of hitting the observed end state $X(T) = b$ is $P_{a,b}(T) = e^{q_{ab}T}$. In the case where $b \neq a$, the time τ to the first move given $\tau < T$ is drawn from:

$$f(\tau, \tau \leq T) = \frac{q_{aa} e^{-q_{aa}\tau}}{1 - e^{-q_{aa}T}}, 0 \leq \tau \leq T \qquad (4.30)$$

The corresponding cumulative distribution function (CDF) is:

$$F(\tau, \tau \leq T) = \frac{1 - q_{aa} e^{-q_{aa}\tau}}{1 - e^{-q_{aa}T}}, 0 \leq \tau \leq T \qquad (4.31)$$

Therefore the inverse of F is:

$$F^{-1}(u) = -\log(1 - u(1 - e^{-q_{aa}T}))/q_{aa} \qquad (4.32)$$

Therefore, sampling from uniform distribution $U(0,1)$ the $F^{-1}(u)$ yields the waiting time to the first move in CTMC model.

Algorithm 2. Modified Rejection Sampling **(adapted from** (Inamura, 2006)**)**

```
If $\boldsymbol{a} = \boldsymbol{b}$:

1. Simulate the CTMC model $\{\boldsymbol{X}(\boldsymbol{t}) | \boldsymbol{0} \le \boldsymbol{t} \le \boldsymbol{T}\}$ using the forward sampling in Algorithm 1.
2. Accept the path if $\boldsymbol{X}(\boldsymbol{T}) = \boldsymbol{a}$; otherwise return to step 1.

If $\boldsymbol{a} \neq \boldsymbol{b}$:

1. Sample $\boldsymbol{\tau}$ from the (4.30) and choose a new state with the probability $\mathbb{P}\ (\boldsymbol{s_1} \neq \boldsymbol{a}) = \frac{\boldsymbol{q_{as_1}}}{\boldsymbol{q_a}}$.
2. Simulate the rest $\{\boldsymbol{X}(\boldsymbol{t}) | \boldsymbol{\tau} \le \boldsymbol{t} \le \boldsymbol{T}\}$ using forwards sampling from the beginning state $\boldsymbol{X}(\boldsymbol{\tau}) = \boldsymbol{s_1}$
3. Accept the simulated path if $\boldsymbol{X}(\boldsymbol{T}) = \boldsymbol{b}$ otherwise return to step 1.
```

The modified rejection sampling, simply avoids simulating constant paths when we know that there will be a move in between. This is particularly beneficial when T is small and the naïve approach in Algorithm 1 will waste time by sampling constant paths.

4.4.3.1.3. Uniformization

This strategy samples from $X(t)$ through the construction of an auxiliary DTMC model $Y(t)$ with the transition probabilities $p_{i,j}$ as follows:

$$p_{i,j} = \begin{cases} \frac{q_{i,j}}{\gamma}, if\ i \neq j \\ 1 - \sum_{i \neq j} \frac{q_{i,j}}{\gamma}\ if\ i = j \end{cases} \gamma \geq \max_i |q_{ii}| \tag{4.33}$$

On the other hand, the matrix representation of the transition probabilities of the auxiliary DTMC is:

$$P = I + \frac{1}{\gamma} Q \tag{4.34}$$

The following calculation of $\mathbb{P}(t)$ shows that a CTMC can be described by a DTMC with a transition probability matrix P where moves can occur according to a Poisson distribution with rate γt.

$$\mathbb{P}(t) = e^{Qt} = e^{\gamma(R-I)t} = e^{-\gamma t} \sum_{n=0}^{\infty} \frac{(\gamma t P)^n}{n!} = \sum_{n=0}^{\infty} e^{-\gamma t} \frac{(\gamma t)^n}{n!} P^n \tag{4.35}$$

This approach is usually called uniformization. The transition matrix of the CTMC in this approach is given by:

$$\mathbb{P}_{ij}(t) = \mathbb{P}(X(t) = j | X(0) = i) = e^{-\gamma t} I_{i=j} + \sum_{n=1}^{\infty} e^{-\gamma t} \frac{(\gamma t)^n}{n!} P_{ij}^n \tag{4.36}$$

Therefore, the number of state changes N including the virtual moves for the conditioned process start at $X(0) = i$ and ends at $X(t) = j$ is given by:

$$\mathbb{P}(N = n | X(0) = i, X(T) = j) = \frac{e^{-\gamma T} \frac{(\gamma T)^n}{n!} P_{ij}^n}{\mathbb{P}_{ij}(T)} \qquad (4.37)$$

Given the number of moves $N = n$, the times $t_1, \dots, t_n$ at which the moves occur are uniformly distributed in $[0, T]$. Moreover, the moves $X(t_0) = a, \dots, X(t_n) = b$ are determined by a DTMC with transition matrix R.

***Algorithm 3. Uniformization Sampling* (adapted from (Inamura, 2006))**

```
1. Simulate the number of state changes $\boldsymbol{n}$ from the distribution (4.37)
2. If $\boldsymbol{n = 0}$, then $\boldsymbol{X(t) = a, 0 \le t \le T}$
3. If $\boldsymbol{n = 1}$ and $\boldsymbol{a = b}$, then $\boldsymbol{X(t) = a, 0 \le t \le T}$
4. If $\boldsymbol{n = 1}$ and $\boldsymbol{a \ne b}$, then simulate $\boldsymbol{t_1}$ from uniform distribution between $\boldsymbol{[0, T]}$ and $\boldsymbol{X(t) = a, 0 \le t \le t_1; X(t) = b, t_1 \le t \le T}$
5. If n>=2, simulate $\boldsymbol{0 < t_1 < \cdots < t_n < T}$ and $\boldsymbol{X(t_1), \dots, X(t_n)}$ from a DTMC with transition matrix $\boldsymbol{P}$ and conditional on starting point $\boldsymbol{a}$ and ending point $\boldsymbol{b}$. Determine which changes are virtual and return the rest.
```

4.4.3.2. The proposed estimation algorithm

The estimation method that we adopted here approximates the posterior distribution for model parameters $\hat{Q}$, given runtime observations $X = x$ and design-time estimations of the parameters $Q^{(0)}$, through samples obtained by generating a sequence of Markov chains $\{Q^{(i)}, X^{(i)}\}$ from the posterior distribution $\mathbb{P}(Q|X)$. As a result, this provides the opportunity to estimate model parameters by summarizing the statistics of these simulated samples. Figure 4.36 illustrates a high-level overview of the proposed approach.

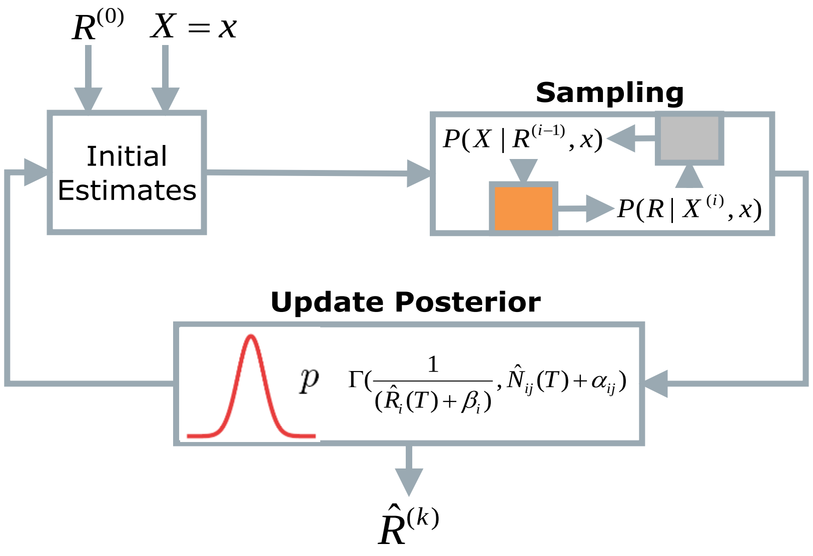


Figure 4.36. Overview of our estimation approach.

Note that by applying the Bayes rule, we can factorize the posterior distribution into the components as:

$$\mathbb{P}(Q|X) = \mathbb{P}(X|Q) \times \mathbb{P}(Q) \tag{4.38}$$

, where $\mathbb{P}(Q)$ is the prior distribution of model parameters.

Here, we assume the presence of a prior distribution. The prior distribution allows us to impose statistical constraints on the parameters estimation. For example, for estimating the unknown parameters of CTMCs, we can choose appropriate $\mathbb{P}(Q)$, in order to have positive rates $q_{i,j}$.

Therefore, one of the key issues in parameter estimation is the choice of a prior distribution $\mathbb{P}(Q)$ and the method used to generate the sequence $\{Q^{(i)}, X^{(i)}\}_{i=1}^{N}$, from the joint posterior distribution on partial observations $X = x$. Here we use Gibbs sampling (Bladt & Sorensen, 2005) for generating the sequence. This sampler, given an initial $Q^{(0)}$, generates sequences as:

1. Draw $X^{(1)} \sim \mathbb{P}\left(X \middle| Q^{(0)}, x\right)$
2. Draw $Q^{(1)} \sim \mathbb{P}\left(Q \middle| X^{(1)}, x\right)$
3. Draw $X^{(2)} \sim \mathbb{P}\left(X \middle| Q^{(1)}, x\right)$
4. …

This sampling process generates a sequence $\{Q^{(i)}, X^{(i)}\}_{i=1}^{N}$, which converges to $\mathbb{P}(Q, X|x)$. The choice of the prior distribution for the unknown parameter $q_{ij} \in Q$ is of crucial importance in a Bayesian framework that we follow here. The selected prior distribution for Q must be a reasonable approximation to the true beliefs about Q. In addition, the prior distribution must be such that the posterior distribution is tractable. For the choice of prior distribution of Q, Bladt and Sørensen (Bladt & Sorensen, 2005) prescribe the Gamma distribution:

$$\mathbb{P}(Q) \propto \prod_{i=1}^{K} \prod_{i \neq j} e^{-q_{ij}\beta_i} * q_{i,j}^{\alpha_{ij}-1} \tag{4.39}$$

, where $\alpha_{ij}, \beta_i > 0$ are constant given values. Here and below, $\propto$ means proportional. By comparing this equation with the likelihood function for complete observations in (4.27), the posterior distribution for Q can be derived as:

$$\begin{aligned} \mathbb{P}(Q|X, x) &= \mathbb{P}(Q|X) \propto \mathbb{P}(X|Q) \times \mathbb{P}(Q) \\ &= \prod_{i=1}^{K} \prod_{i \neq j} e^{-q_{ij}(R_i(T)+\beta_i)} * q_{i,j}^{N_{i,j}(T)+\alpha_{ij}-1} \end{aligned} \tag{4.40}$$

Equation (4.40) shows that the posterior distribution of Q also follows the same distribution as its prior one that is a Gamma distribution. This makes the drawing $Q^{(i)} \sim \mathbb{P}\left(Q \middle| X^{(i)}, x\right)$ tractable. Bladt and Sørensen (Bladt & Sorensen, 2005) suggest that a simple rejection sampling (cf. Section 4.4.3.1.2) for drawing of the Markov process $X^{(i+1)} \sim \mathbb{P}\left(X \middle| Q^{(i)}, x\right)$ is an appropriate sampling strategy. In the estimation algorithm that is proposed in (Inamura, 2006), they have used a similar process.

The estimation mechanism comprising the sampling process is given in Algorithm 4.

Algorithm 4. Parameter estimation algorithm **(adapted from** (Inamura, 2006)**)**

1. A sample of holding time $\boldsymbol{S_k}$, at each observation points $\boldsymbol{t_n}$, is simulated by drawing from $\boldsymbol{q_k e^{-q_k(t_n - t_{n-1})}}$.
2. If $\boldsymbol{t_{n-1} + S_k < t_n}$, then the algorithm lets the CTMC make a transition from current state $\boldsymbol{k}$ to another state $\boldsymbol{j}$ with the probability of $\boldsymbol{q_{kj}/q_k}$. This process will be continued if the CTMC reaches an observed state by $\boldsymbol{t_n}$. If the sample is accepted, the holding times at each state and the transitions between states of the CTMC model are recorded to be added up later for updating the posterior distribution (cf. Figure **4.36**). This process will be continued until all the other transitions in the period $[\boldsymbol{t_{n-1}}, \boldsymbol{t_n}]$ are realized.
3. Repeat steps 1, 2 for the next period, i.e., $[\boldsymbol{t_n}, \boldsymbol{t_{n+1}}]$ again and again until the time associated to the last observation that is $\boldsymbol{T}$.
4. The statistics $\boldsymbol{\widehat{N}_{ij}(T)}, \boldsymbol{\widehat{R}_i(T)}$ regarding the number of transitions from state $\boldsymbol{i}$ to state $\boldsymbol{j}$ in the CTMC model recorded until time $\boldsymbol{T}$ and the time that the model was in state $\boldsymbol{i}$ until $\boldsymbol{T}$ respectively. Note that only accepted samples were recorded in Step 2.
5. A new $\boldsymbol{Q}$ is estimated by drawing the parameters $\boldsymbol{q_{ij}}$ from the Gamma distribution as delineated in Equation (4.40).
6. Steps 1 to 5 are repeated until we derive good enough estimation of the unknown parameters of the CTMC or statistically speaking until the posterior distribution sampling becomes stable. Then we have a number of estimations. Let us assume we iterated $\boldsymbol{N}$ times, $\{\boldsymbol{q_{ij}^{(n)}}\}_{\boldsymbol{n=1}}^{\boldsymbol{N}}$. We can then calculate different statistics from these estimations. The most notable one is the average of these estimations as the final estimation for each parameter.

The estimation mechanism that we proposed in this section is categorized as a class of algorithm that samples from a probability distribution (here we use Gamma distribution for the purpose of tractability) based on constructing a Markov chain that has the desired distribution as the stationary distribution. In this statistics, this class of algorithm is called Markov Chain Monte Carlo (MCMC) (Bolstad, 2011). Note that the method is implemented in MATLAB.

Note that the MCMC algorithm that we presented in this section has been previously applied in different application domains for example in stock price estimation as in (Inamura, 2006) or segmentation of strands in genetics. However, for applying such estimation algorithm for model calibration in self-adaptive software, we need to have a runtime efficient mechanism. We report the runtime overhead of our implementation in Section 4.4.3.3.7.

4.4.3.3. Experimental evaluation

This section discusses some preliminary results regarding CTMC model parameter estimation and its evaluation through a controlled experiment. To be more specific, we simulate the runtime data by using statistical distribution and we apply our estimation algorithm to estimate the parameters of the runtime models. Note that, for the experimental evaluation of the proposed estimation algorithm, i.e., Algorithm 4, we do not assume that the runtime data are complete and this is one of the benefits of our estimation algorithm over existing model estimation approaches in self-adaptive software, such as (Epifani et al., 2009). The main contribution of this work is that the proposed approach can estimate unknown parameters of CTMC models for enabling requirements verification at runtime, even though the runtime measurements may not be perfect and contain uncertainties.

4.4.3.3.1. Experimental conception

This section contains a subset of the scenarios that involve estimating the rate of transitions between the states of a CTMC model corresponding to a component connector based on initial design-time estimates and on runtime data obtained through monitoring the connectors. Here, we use the same case as we selected for the DTMC model calibration in Section 4.4.2.6.1.

4.4.3.3.2. Experimental setup

For the estimation algorithm, 2000 intensity matrices, including a burn-in period of 500 iterations, are drawn for each estimation. Note that the data regarding the burn-in period will be discarded for quantitative analyses. In this experiment, we set both α_{ij} and β_i to 1. We generally use the posterior mean of the distribution from the samples of $\hat{q}_{ij}$ as the point estimate of the unknown model parameters. However, in some circumstances when the values of the parameters are skewed as recommended in (Inamura, 2006), we choose the posterior mode estimate from the samples of $\hat{q}_{ij}$ instead of the mean estimate. The controlled variables regarding this experiment are specified in Table 4.29.

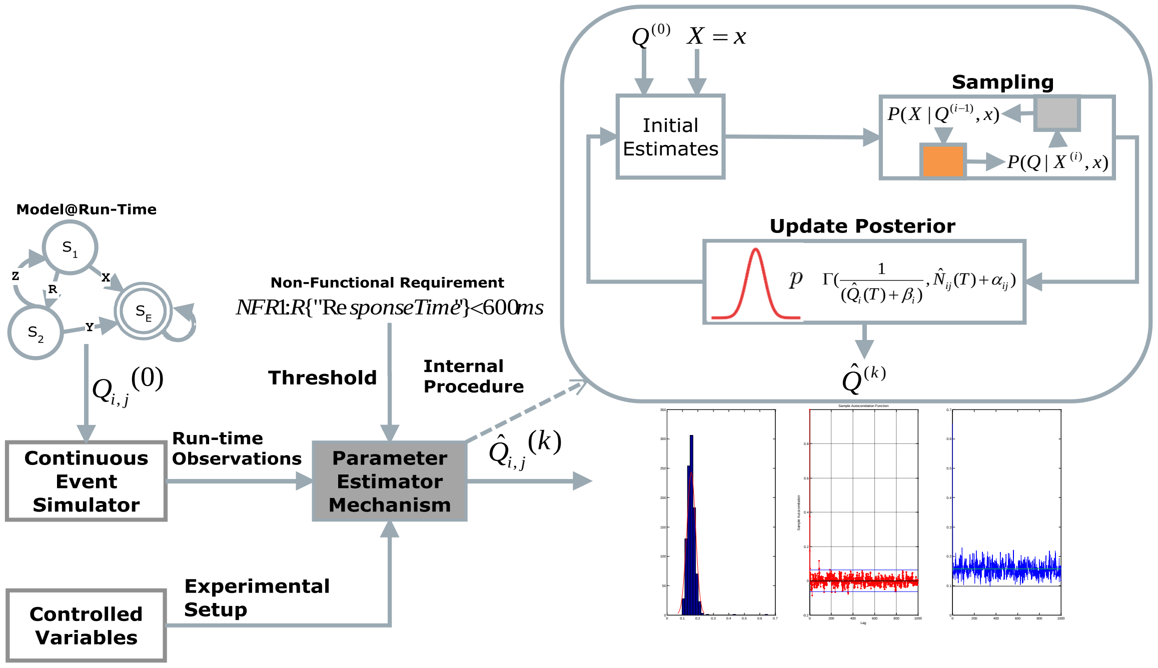


Figure 4.37. Experimental setup overview.

The controlled variables that we use in our experiments are summarized in Table 4.27.

Table 4.27. List of controlled variables and their purpose in our experiments.

Controlled variable	Purpose
$\boldsymbol{q\_Actual}$	Actual transition rates matrix
$\boldsymbol{States}$	Number of non-absorbing states of q_Actual
$\boldsymbol{Abs}$	Number of absorbing states of q_Actual
$\boldsymbol{T}$	Determine the whole observation time period $[0, T]$
$\boldsymbol{\Delta t}$	The observation times (sampling frequencies) in $[0, T]$
$\boldsymbol{Obs}$	The observation size for each parameters in the period $[t_n, t_{n+1}]$
$\boldsymbol{\alpha}$	Shape parameter of Gamma distribution $\Gamma(\alpha, 1/\beta)$
$\boldsymbol{\beta}$	Scale parameter of Gamma distribution $\Gamma(\alpha, 1/\beta)$
$\boldsymbol{M}$	Number of simulation rounds
$\boldsymbol{b}$	Number of burn-in rounds

4.4.3.3.3. Running the experiments

In this experiment, we assume that an actual transition rates of a CTMC model with 7 states is given to us as $\boldsymbol{q\_Actual}$ in Table 4.28. The setting of the experiments in terms of the defined controlled variables is given in Table 4.29.

Table 4.28. The q_Actual matrix in the experiment.

-0.0804761	0.074790	0.00640	0	0	0	0	0
0.007695	-0.012246	0.200742	0	0	0	0	0
0.00130	0.040003	-0.095042	0.10010	0.010093	0	0	0
0.000999	0.000734	0.077212	-0.091115	0.058184	0.005551	0.000621	0
0	0	0.008226	0.200345	-0.199912	0.100043	0.002223	0
0	0	0.00301	0.009131	0.100037	-0.300028	0.200148	0.009954
0	0	0	0	0	0.131351	-0.600318	0.39992
0	0	0	0	0	0	0	0

Table 4.29. Controlled variables in the experiment (cf. Table 4.27).

Controlled variable	Value(s)
$\boldsymbol{States}$	8
$\boldsymbol{Abs}$	1
$\boldsymbol{T}$	7
$\boldsymbol{\Delta t}$	1
$\boldsymbol{Obs}$	100
$\boldsymbol{\alpha}$	1
$\boldsymbol{\beta}$	1
$\boldsymbol{M}$	2000
$\boldsymbol{b}$	500

4.4.3.3.4. Experimental results and interpretations

In the following, we provide for each experiment some graphs for the estimated posterior density, autocorrelation plot and sample path by the estimation algorithm. Figure 4.38, Figure 4.39, and Figure 4.40 illustrate the plots of $\hat{q}_{12}, \hat{q}_{13}, \hat{q}_{24}$ with respect to the same 2000th estimated Q matrix after putting aside the data items regarding the first 500 burn-in rounds.

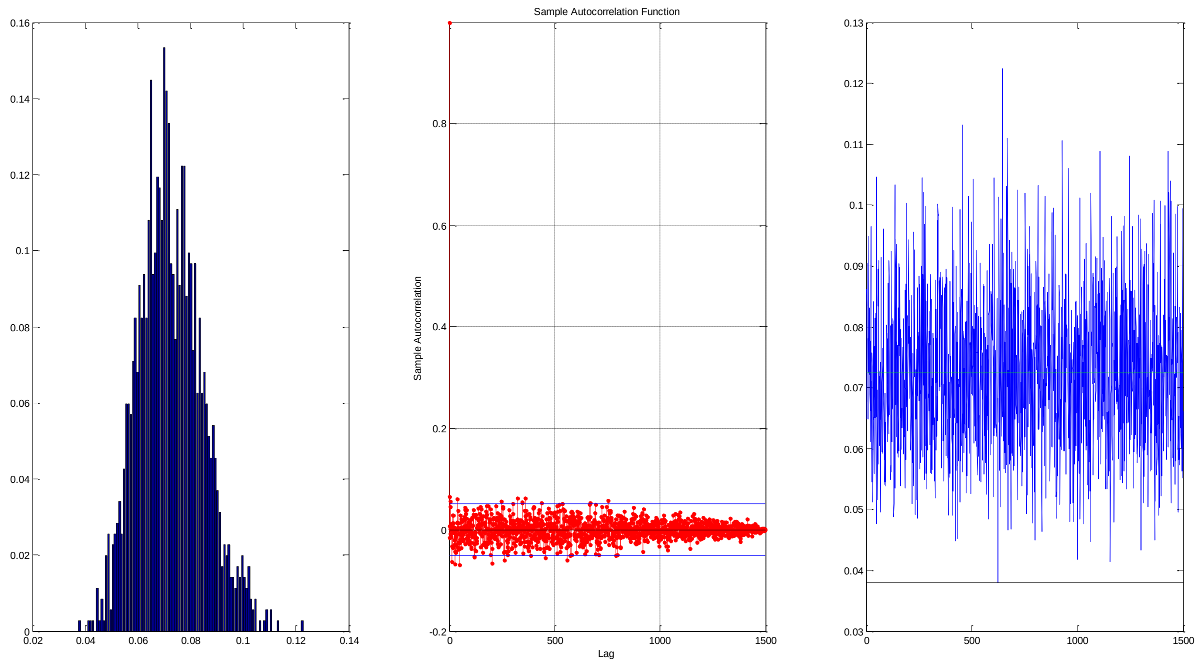


Figure 4.38. Posterior distribution, autocorrelation plot and sample paths of Q(1,2).

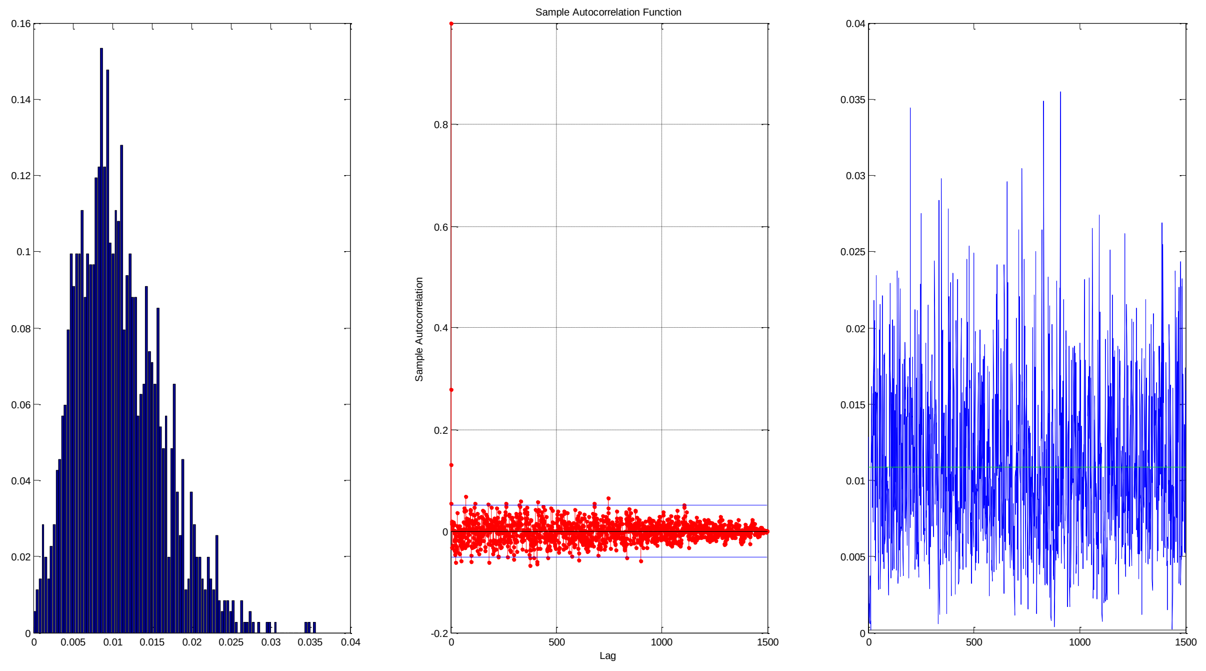


Figure 4.39. Posterior distribution, autocorrelation plot and sample paths of Q(1,3).

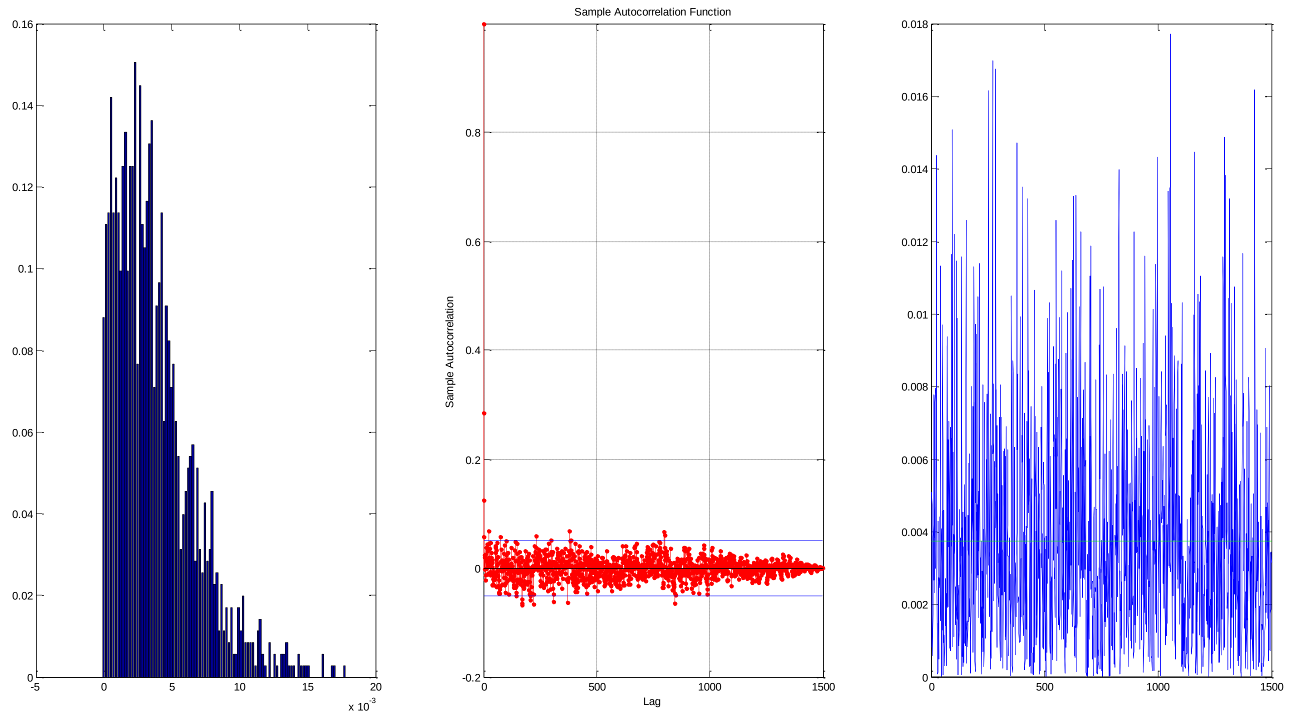


Figure 4.40. Posterior distribution, autocorrelation plot and sample paths of Q(2,4).

Note that none of the figures of autocorrelation plots and sample paths shows any problematic issue in the sampling.

Table 4.30. Mean estimates of default probabilities and their differences with default probabilities of actual matrix.

States	**Default probability of mean estimate**	**Default probability of actual matrix**	**Difference**	**Error**
1	1.4871e-07	1.1294e-08	-1.3742e-07	12.1675
2	2.5178e-06	1.8460e-07	-2.3332e-06	12.6394
3	3.6433e-05	6.7224e-06	-2.9710e-05	4.4196
4	0.0011	2.0873e-04	-9.3313e-04	4.4705
5	0.0069	0.0016	-0.0053	3.3175
6	0.0461	0.0304	-0.0156	0.5135
7	0.3249	0.3262	0.0013	0.0041
$\boldsymbol{|Sum|}$			0.0206	37.5239

To make a more clear evaluation of these estimated default probabilities, we run a bootstrapping algorithm to derive the distribution of the default probabilities. In the bootstrapping, 100,000 rounds of simulations were carried out. The procedures for bootstrapping are summarized as follows:

1. Generate the history of observations according to the CTMC model.
2. Calculate $\widehat{N}_{ij}(T), \widehat{R}_i(T)$ and estimate $\bar{Q}$ from the equation (4.29).
3. Compute $P(Q)$.
4. Iterate 1 to 3 up to the simulation rounds.

Figure 4.41 shows the bootstrapped distribution of default probabilities for 7 states of the CTMC model of Experiment 1. Figure 4.42 and Figure 4.43 provide the resulting boxplot of default probabilities from the bootstrapped distributions. The data demonstrates that the mean default probabilities of the model parameters are all within the confidence intervals of the bootstrapped distribution. This means that the estimation method performs well for estimating the unknown parameters of the analytical model in the given the finite and incomplete noisy observations. To further investigate the performance of the estimation method, we employed quantitative measures for evaluating the estimation accuracy.

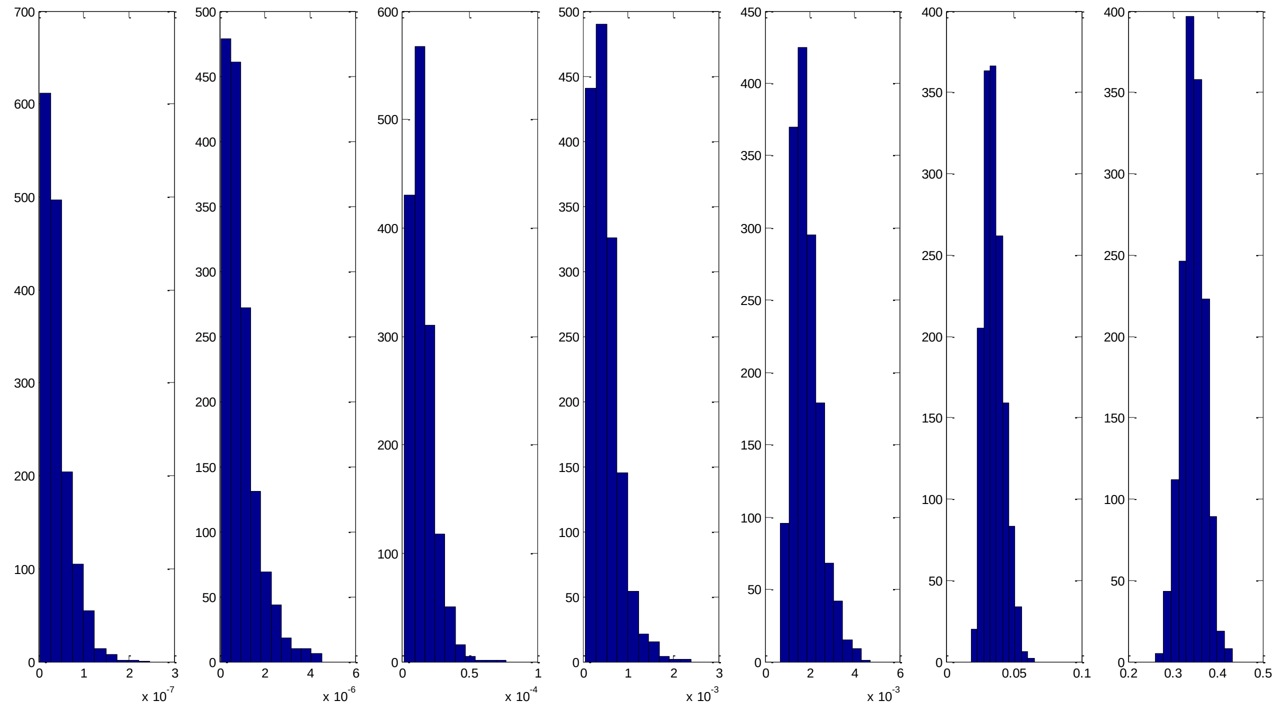


Figure 4.41. Bootstrapped distribution of default probabilities.

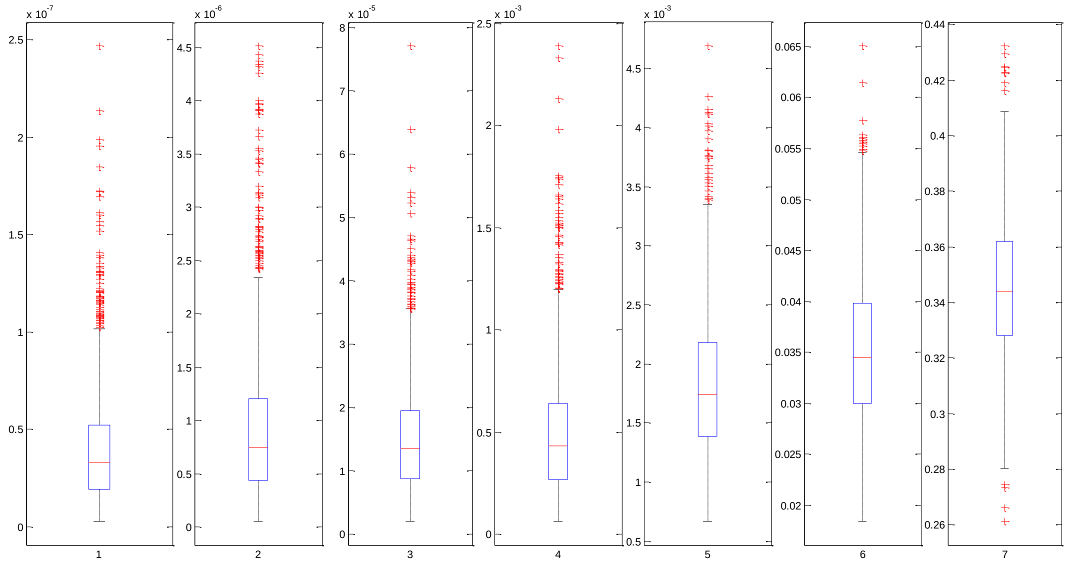


Figure 4.42. Box plot of default probabilities in different scales.

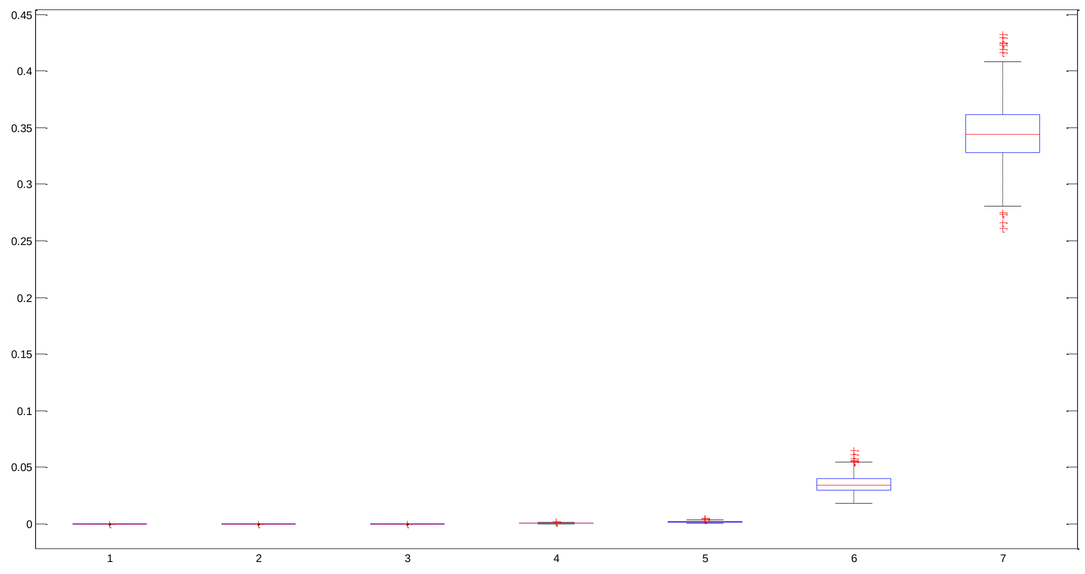


Figure 4.43. Box plots of default probabilities in one scale.

Independency of estimates. To get an idea of how independent the estimates are, we generate scatter plots of the posterior values of (q_{i1j1}, q_{i2j2}) for all 1500 matrices we have generated for each of the 136 combinations of $(i1, j1), (i2, j2), i1 \neq j1$ & $i2 \neq j2$ & $q_{ij} \neq 0$. Most of the plots were similar, so we have chosen 7 typical patterns in Figure 4.44. By analysis of the scatter plots, we can see that the estimates are not very dependent on each other.

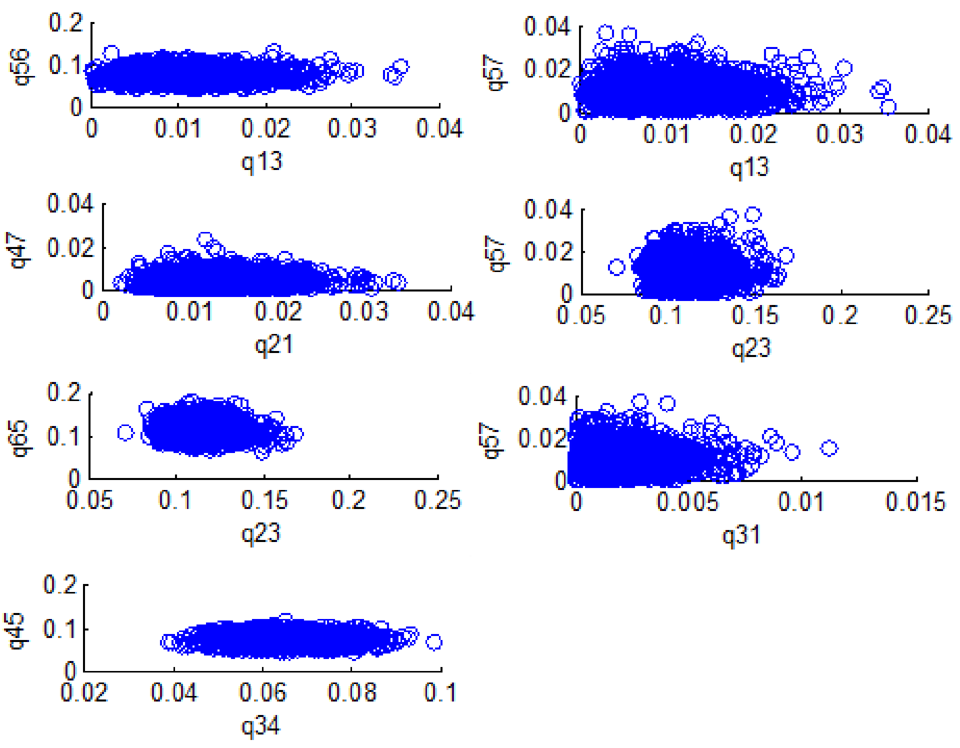


Figure 4.44. Scatter plots of the posterior distribution of estimates that show typical pattern.

4.4.3.3.5. Summary of observations and interpretations of the results

Regarding the proposed parameter estimation approach (see Section 4.4.3.2), a number of observations can be made:

- Based on a visual inspection of the sample paths in Figure 4.38, Figure 4.39 and Figure 4.40, the series settled into a stationary mode after an initial 20 iterations and this is certified with Figure 4.47 as well.
- The auto-correlation diagrams in Figure 4.38, Figure 4.39 and Figure 4.40 were seen to approach to zero very quickly.
- Estimates of the parameters that are not too close to zero were not sensible to the choice of α and β as parameters of the Gamma distribution, while the parameters which are too close to zero will depend on the choice of the prior because $N_{i,j}(T)$ is small.
- The draws in Figure 4.38, Figure 4.39 and Figure 4.40 show that the posterior distribution of the parameters deviate strongly from normal distribution for small parameters. This indicates that the parameter estimates are non-normal.
- The drawings in Figure 4.41 show that the distribution of the default probabilities deviate from normal distribution (for state 7 which is relatively far away from zero (Figure 4.43) were approximately normal).
- According to Table 4.30, the relative errors for states 1 and 2 are too high and for state 7 are small.

4.4.3.3.6. Quantitative evaluations: measuring estimation accuracy

For evaluating the accuracy of the quantitative estimation technique, various quantitative metrics have been utilized in the literature. In Section 4.4.3.3.3, we described a number of experiments; we also reported the observations and interpret the results in Section 4.4.3.3.4. In this section, we use some metrics for assessing the estimation method that we described in Section 4.4.3.2. An overview of the measurement process is depicted in Figure 4.45.

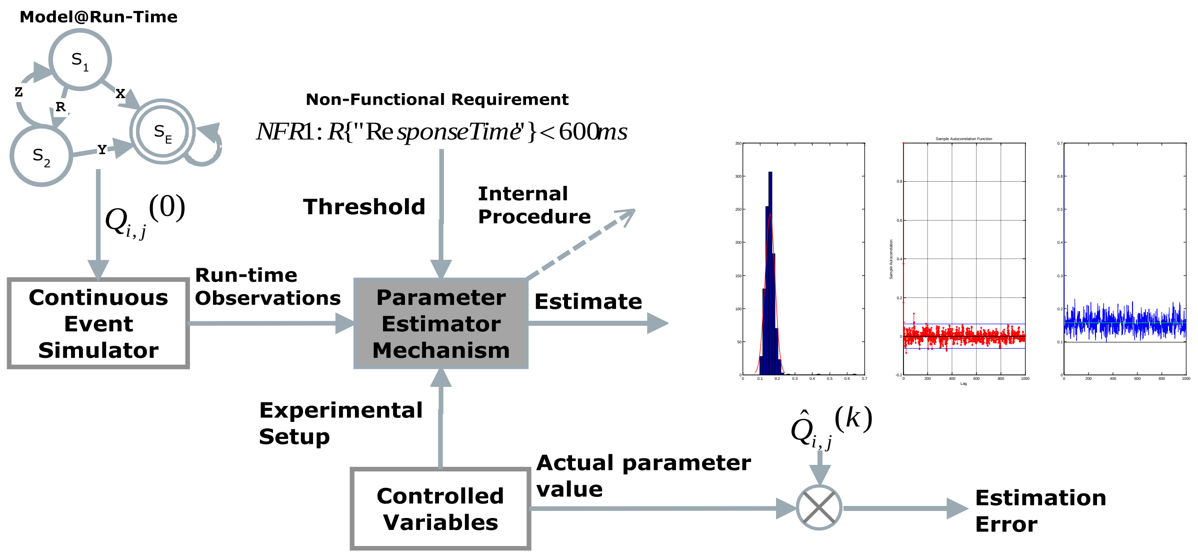


Figure 4.45. Estimation analysis in the context of experimental setup.

Metrics measure the size of the error when estimating the value of parameters alongside the runtime data. Table 4.31 lists typical metrics from the literature and here we use them for assessing our estimation accuracy. These metrics are meant to quantify the difference between values implied by the MCMC-based estimator that we proposed in this chapter and the actual values of the parameter to be estimated.

Table 4.31. Estimation error metrics.

Metric	Formula
$\boldsymbol{D_{L^1}(P,Q)}$	$\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\lvert P_{i,j}-Q_{i,j}\rvert}{N^2}$
$\boldsymbol{D_{L^2}(P,Q)}$	$\frac{\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(P_{i,j}-Q_{i,j}\right)^2}}{N^2}$
$\boldsymbol{\lVert P-Q\rVert_p}$	$\sqrt[P]{\sum_{i=1}^{N}\sum_{j=1}^{N}\lvert P_{i,j}-Q_{i,j}\rvert^p}$
$\boldsymbol{D_{SVD}(P,Q)}$	$D_{SVD}(P,Q)=M_{SVD}(P)-M_{SVD}(Q)$ $M_{SVD}(P)=\frac{\sum_{i=1}^{K}\sqrt{\lambda_i(\overline{P}'\overline{P})}}{K}, \overline{P}=P-I$

Although D_{L^1}, D_{L^2} and $Norm$ are simple to compute, these methods offer no absolute measure for an individual matrix. They only provide a relative comparison between two matrices. For example, if the $\boldsymbol{L_2}$ distance between two matrices turns out to be, let us say, 0.01, it is not clear if this is a ‘‘large’’ or a ‘‘small’’ distance, nor is it possible to infer which matrix is the ‘‘larger’’ of the two. However, $\boldsymbol{D_{SVD}}$ is more appropriate in measuring the difference of the transition matrices than other ordinary metrics since it captures the off-diagonal differences better (Jafry & Schuermann, 2004).

Table 4.32 provides the evaluation of the error metrics of the model parameter matrix Q and default probabilities of Q based on the full set of 1500 estimates. As it is evident from the error measures, the proposed estimation approach produces acceptable estimations.

Table 4.32. Estimation error measurements.

Metric	Experiment	
	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$
$\boldsymbol{D_{L^1}}$	0.0060	0.0046
$\boldsymbol{D_{L^2}}$	0.0014	0.0010
$\boldsymbol{\lVert\hat{Q}-Q\rVert_2}$	0.0615	0.0481
$\boldsymbol{D_{SVD}}$	0.0015	0.1840

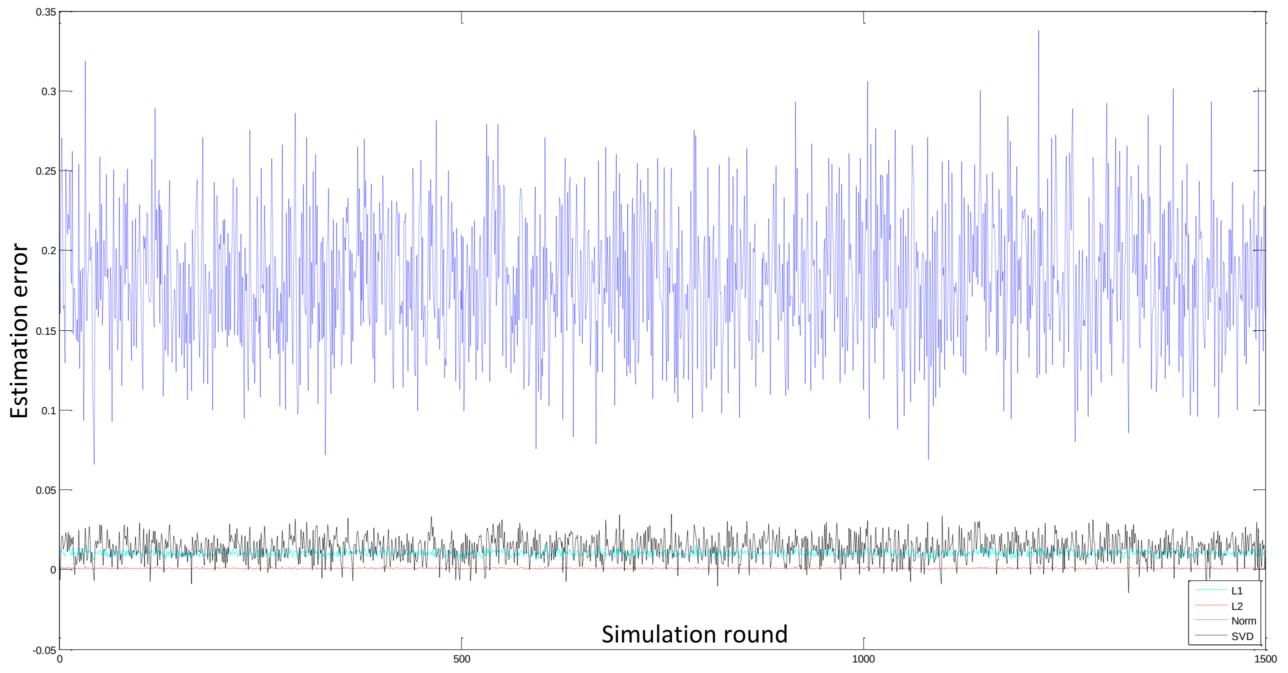


Figure 4.46. Estimation error with different metrics over simulation rounds.

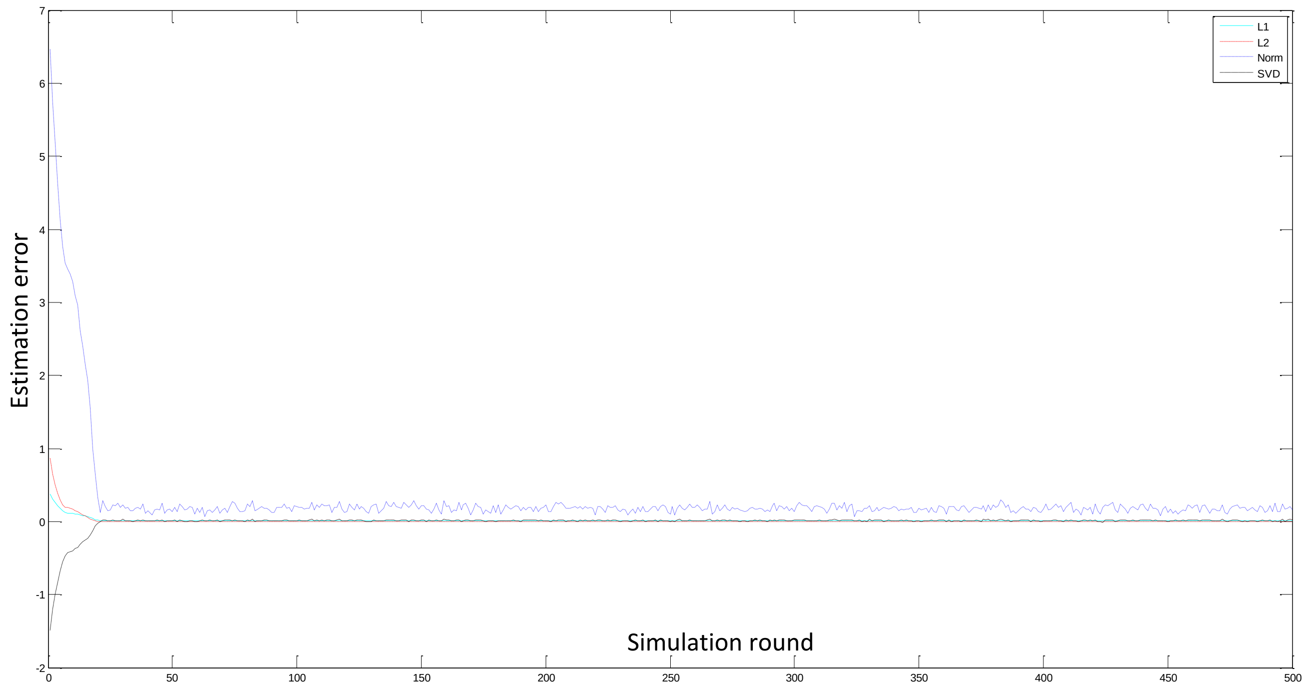


Figure 4.47. Estimation error with different metrics over burn-in rounds.

4.4.3.3.7. Runtime performance

Since the process of requirement verification integrated in a feedback control loop triggers the adaptation of connectors, it needs to be runtime efficient to be applicable in such a setting. In other words, a lengthy requirements verification process hinders the usefulness and, as a result, the adoption of self-adaptive software. Model calibration is an integral part of requirements verification and, consequently, needs to be performant at runtime. In this section, we investigate the runtime overhead of the proposed MCMC-based approach to estimate model parameters.

In order to assess the runtime overhead of the proposed model calibration mechanism (see Section 4.4.3.2), we have conducted a set of experiments using simulations with different settings by changing the controlled variables as listed in Table 4.27. One of the most critical parameters that influence the runtime overhead of the estimation mechanism is the number of observations controlled via Obs (cf. Table 4.27). Therefore, for evaluating the scalability of our approach, we vary the number of observations by changing the values of this variable in our experimental evaluations by an order of magnitude in the range $[1 - 500]$ (cf. Table 4.33). We kept the rest of the controlled variables exactly the same as the previous experiment as in Table 4.29. We performed the experimental evaluations on a dedicated machine for our experiments with the specification as in Table 4.4.

Table 4.33. Experimental settings for runtime performance evaluations (cf. Table 4.27).

variable	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6	Exp. 7	Exp. 8	Exp. 9	Exp. 10	Exp. 11	Exp. 12
$\boldsymbol{States}$	8	8	8	8	8	8	8	8	8	8	8	8
$\boldsymbol{Abs}$	1	1	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{T}$	7	7	7	7	7	7	7	7	7	7	7	7
$\boldsymbol{\Delta t}$	1	1	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{Obs}$	1	2	3	4	5	10	20	30	40	100	200	500
$\boldsymbol{\alpha}$	1	1	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{\beta}$	1	1	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{M}$	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
$\boldsymbol{b}$	500	500	500	500	500	500	500	500	500	500	500	500

Based on the results that are reported in Table 4.34, Figure 4.48, Figure 4.49 and Figure 4.50 regarding the comparison between estimation accuracy and runtime efficiency of a number of experiments, a number of observations can be made:

- Increasing the number of observations (cf. Table 4.33), increases the estimation accuracy (cf. Table 4.34 and visually in Figure 4.48). However, it also increases the runtime overhead of the model calibration.
- Initially increasing the number of observations dramatically decreases the estimation errors and does not cause a high runtime overhead (cf. Exp. 1 to Exp. 2 in Figure 4.48 and Figure 4.50).
- Increasing the number of observations from a certain point only decreases the estimation error up to a certain point, but imposes a large runtime overhead instead (cf. Exp. 9 to Exp. 12 in Figure 4.48 and Figure 4.49).
- The data in Figure 4.49 confirms "10× increase in the number of observations approximately results in ~10× increase in runtime overhead". The data indicate that the model calibration mechanism performs well even when we obtain a large historical observation set (i.e., 40×7 in Exp. 9 for example). It took approximately less than $500ms$ to provide an acceptable estimation for all 32 unknown parameters of the model (see Table 4.28). In other words, the runtime overhead for each parameter is around $15ms$.

Table 4.34. Estimation error comparison between experiments (cf. Table 4.33).

Metric	Exp. 1		Exp. 2		Exp. 3		Exp. 4		Exp. 5	
	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$
D_{L^1}	0.1053	0.0637	0.0399	0.0308	0.0238	0.0179	0.0331	0.0246	0.0261	0.0176
D_{L^2}	0.0362	0.0191	0.0106	0.0079	0.0068	0.0048	0.0088	0.0061	0.0074	0.0048
$\lVert\hat{Q}-Q\rVert_2$	1.9969	1.0113	0.5197	0.3840	0.3524	0.2246	0.3691	0.2198	0.3624	0.2064
D_{SVD}	-0.3370	0.1840	-0.0247	0.1840	-0.0144	0.1840	-0.0531	0.1840	-0.0861	0.1840
Metric	Exp. 6		Exp. 7		Exp. 8		Exp. 9		Exp. 10	
	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$	Q	$P(Q)$
D_{L^1}	0.0137	0.0112	0.0124	0.0092	0.0126	0.0085	0.0080	0.0062	0.0078	0.0058
D_{L^2}	0.0038	0.0031	0.0034	0.0024	0.0037	0.0024	0.0020	0.0014	0.0024	0.0016
$\lVert\hat{Q}-Q\rVert_2$	0.2057	0.1645	0.1516	0.1082	0.1930	0.1253	0.0911	0.0622	0.1330	0.0858
D_{SVD}	-0.0232	0.1840	-0.0172	0.1840	-0.0118	0.1840	-0.0110	0.1840	0.0060	0.1840
Metric	Exp. 11		Exp. 12							
	Q	$P(Q)$	Q	$P(Q)$						
D_{L^1}	0.0059	0.0044	0.0050	0.0034						
D_{L^2}	0.0019	0.0012	0.0022	0.0014						
$\lVert\hat{Q}-Q\rVert_2$	0.1059	0.0654	0.1373	0.0857						
D_{SVD}	0.0096	0.1840	0.0124	0.1840						

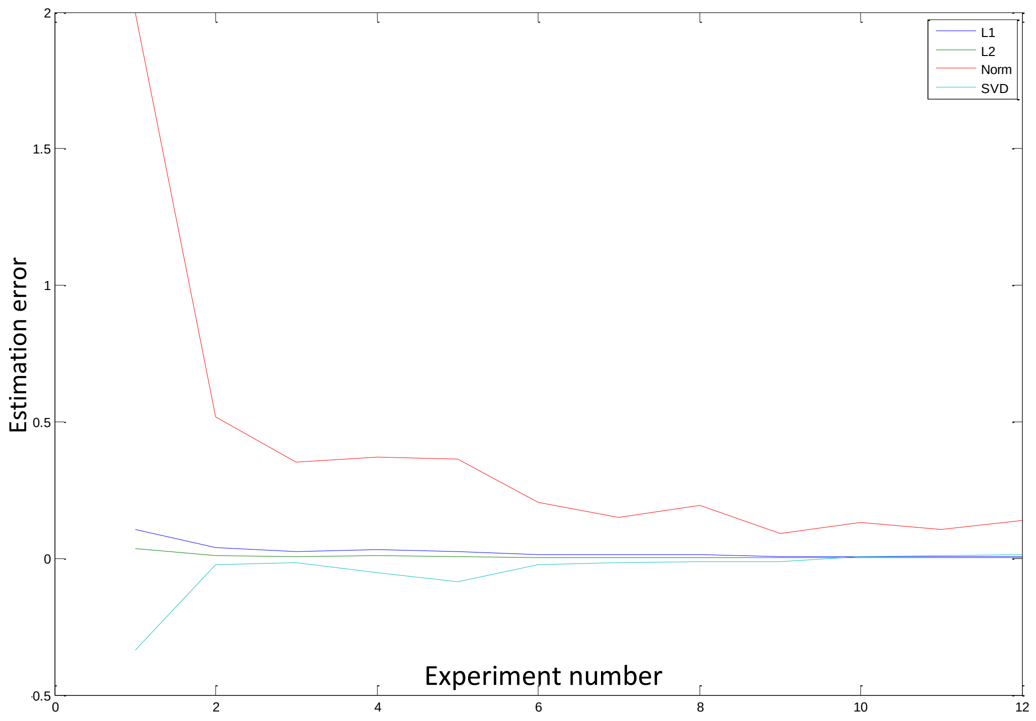


Figure 4.48. Visual comparison of estimation errors between experiments (cf. Table 4.34).

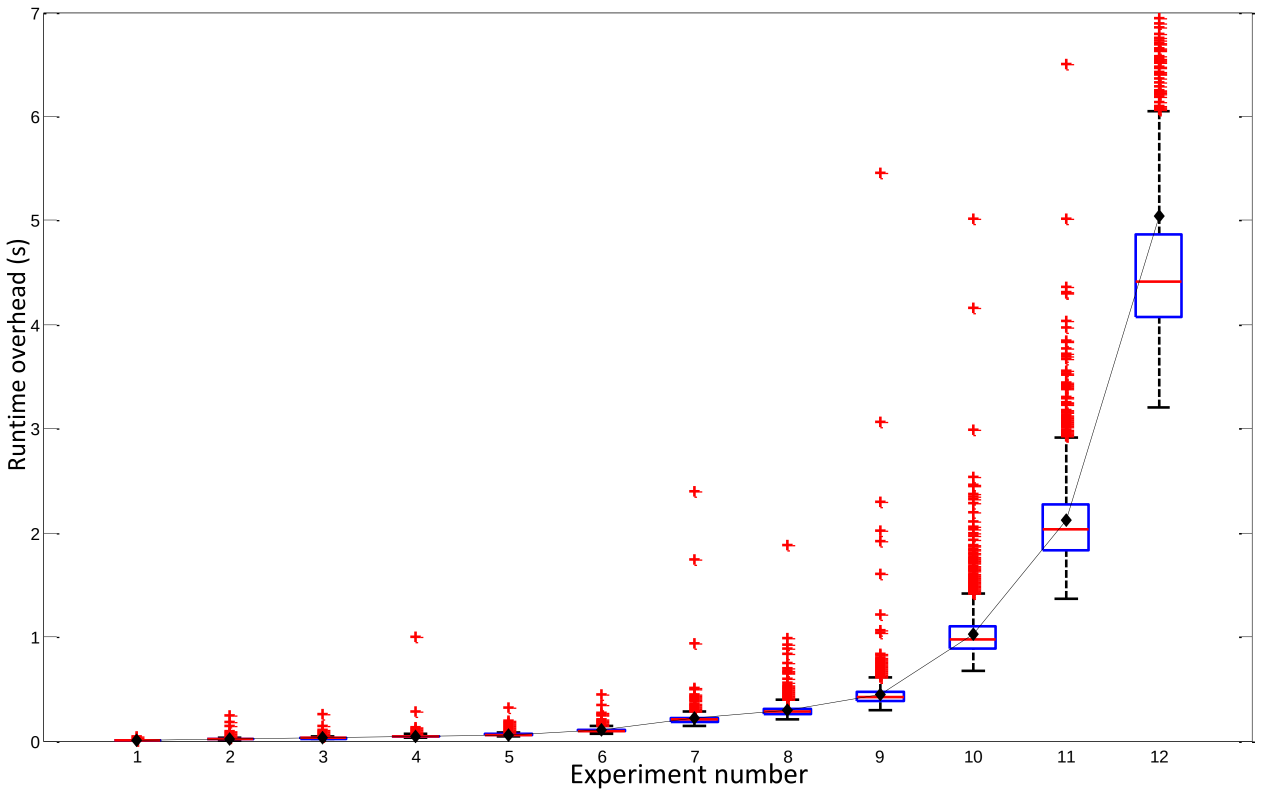


Figure 4.49. Runtime performance w.r.t. observation size (cf. Table 4.27 and Table 4.33).

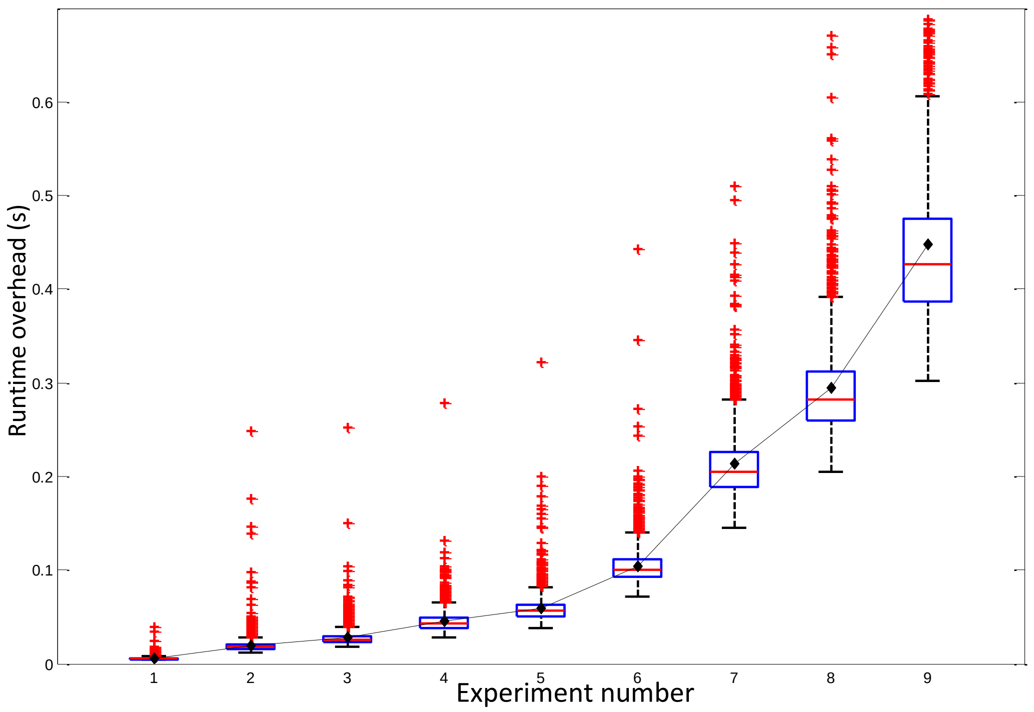


Figure 4.50. Runtime performance w.r.t. observation size (only the first 9 experiments, cf. Table 4.33).

4.4.3.3.8. Sensitivity of the estimation errors to simulation and burn-in rounds

In order to assess the sensitivity of the proposed model calibration mechanism (see Section 4.4.3.2) to the number of simulation rounds, we have conducted a set of experiments using simulation with different settings by changing the simulation and burn-in rounds as listed in Table 4.27. In parameter estimation approaches in statistics, burn-in periods are considered to discard the initial erogenous estimations (Inamura, 2006). However, we claim that our approach converges to the actual value of the parameters quickly in order to avoid a large runtime overhead. In order to evaluate the claim, we increased the number of burn-in periods from Exp. 1 to Exp. 7 in Table 4.35. Based on the results, which are reported in Table 4.36 and visualized in Figure 4.51, the estimation accuracy has not been increased after increasing the burn-in period. These observations based on the experimental results support our claim regarding the quick convergence of our approach. Even increasing in the simulation rounds (cf. Exp. 8 to Exp. 10 in Table 4.35) does not show a decrease in estimation errors necessarily as depicted in Figure 4.51.

Table 4.35. Experimental settings for sensivity analyses (cf. Table 4.27).

variable	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6	Exp. 7	Exp. 8	Exp. 9	Exp. 10
$\boldsymbol{States}$	8	8	8	8	8	8	8	8	8	8
$\boldsymbol{Abs}$	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{T}$	7	7	7	7	7	7	7	7	7	7
$\boldsymbol{\Delta t}$	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{Obs}$	100	100	100	100	100	100	100	100	100	100
$\boldsymbol{\alpha}$	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{\beta}$	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{M}$	2000	2000	2000	2000	2000	2000	2000	200	500	10000
$\boldsymbol{b}$	0	10	20	30	100	200	1000	150	50	1000

Table 4.36. Estimation error comparison between experiments (cf. Table 4.33).

Metric	Exp. 1		Exp. 2		Exp. 3		Exp. 4		Exp. 5	
	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$
$\boldsymbol{D_{L^1}}$	0.0186	0.0142	0.0182	0.0119	0.0146	0.0108	0.0265	0.0176	0.0215	0.0151
$\boldsymbol{D_{L^2}}$	0.0047	0.0034	0.0056	0.0034	0.0041	0.0029	0.0090	0.0053	0.0063	0.0041
$\boldsymbol{\lVert\hat{Q}-Q\rVert_2}$	0.2060	0.1267	0.2945	0.1539	0.2029	0.1398	0.5038	0.2522	0.3153	0.2008
$\boldsymbol{D_{SVD}}$	0.0025	0.1840	-0.0584	0.1840	-0.0332	0.1840	-0.1069	0.1840	-0.0336	0.1840
Metric	Exp. 6		Exp. 7		Exp. 8		Exp. 9		Exp. 10	
	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$	$\boldsymbol{Q}$	$\boldsymbol{P(Q)}$
$\boldsymbol{D_{L^1}}$	0.0231	0.0169	0.0156	0.0120	0.0109	0.0079	0.0168	0.0128	0.0172	0.0132
$\boldsymbol{D_{L^2}}$	0.0074	0.0050	0.0046	0.0032	0.0027	0.0019	0.0041	0.0030	0.0047	0.0034
$\boldsymbol{\lVert\hat{Q}-Q\rVert_2}$	0.3643	0.2325	0.2100	0.1430	0.1351	0.0892	0.1751	0.1168	0.2155	0.1454
$\boldsymbol{D_{SVD}}$	-0.0269	0.1840	-0.0134	0.1840	-0.0167	0.1840	-0.0225	0.1840	-0.0050	0.1840

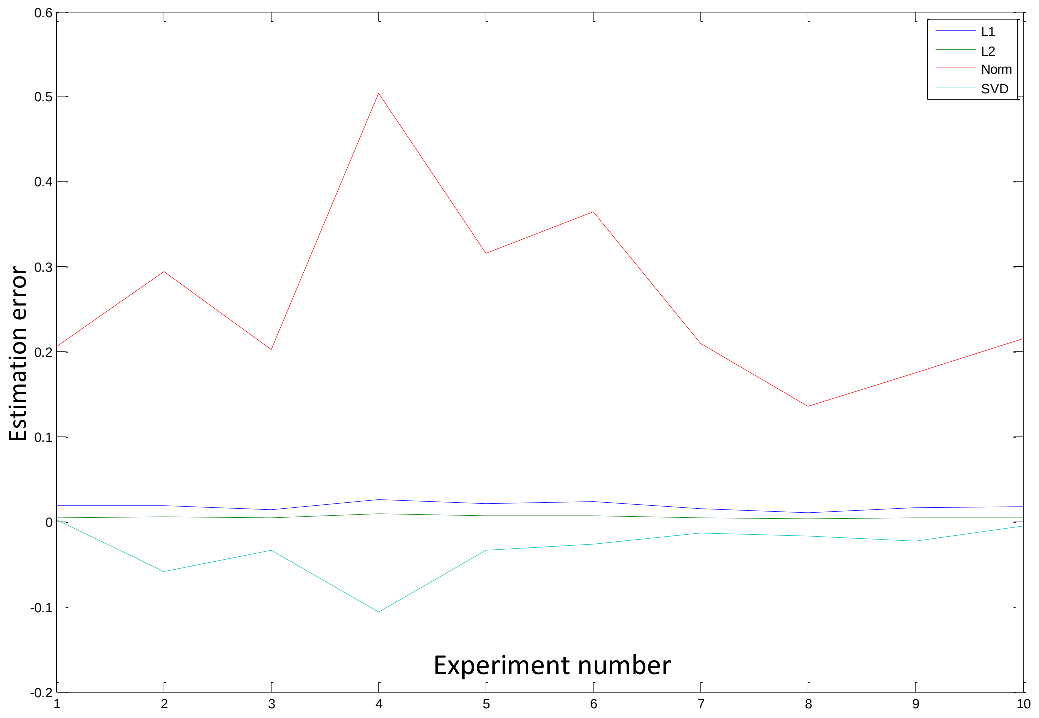


Figure 4.51. Visual comparison of estimation errors between experiments (cf. Table 4.36).

4.4.3.3.9. Limitations and threats to validity

The experiments in this section indicate that the effectiveness of the proposed CTMC parameter estimation approach depends on the number of sampling rounds (cf. Figure 4.36), which directly determine the time overhead of the approach. In other words, the more sampling rounds, the more accurate estimations the approach can provide, but this is less appropriate for runtime requirement verification because of the timing constraints (cf. Sections 4.4.3.3.7 and 4.4.3.3.8). To address this limitation, we found out that the choice of the Gamma distribution for prior distribution enables the estimation approach to quickly converge to the real value of the unknown parameter. However, this does not mean that we can expect to derive an accurate estimation in a couple of rounds. This is one of the limitation of our proposed estimation approach and a fruitful avenue for future improvements in terms of decreasing runtime overhead.

In Section 4.4.3.3, we only evaluated the approach with 1 experiment but for the estimation of 32 non-zero parameters of the q_Actual matrix in Table 4.28. In this experimental evaluation, our aim was to measure the effectiveness of our estimation approach quantitatively as we have done and reported in Section 4.4.3.3.4, Section 4.4.3.3.5 and Section 4.4.3.3.6. However, as opposed to the first experimental evaluations in Section 4.4.2.6 where we combined different values for the controlled variables to demonstrate the effectiveness of the extended Bayes method in different situations, here, we only considered such measurements by changing a limited number of controlled variables (see Sections 4.4.3.3.7 and 4.4.3.3.8). Although we reported both qualitative observations in Sections 4.4.3.3.4 and quantitative comparison in Section 4.4.3.3.5, the limitations of the controlled experiments is one of the threats to the validity of this work. The only strategy for mitigating this threat was to extend the experimental setting and perform a case study. We will report results regarding this in the evaluation chapter, i.e., Chapter 7.

4.5. Related Work

In this section, we review only the most closely related approaches to parameter estimation for requirement verification that have been published in software engineering venues.

Several methods and techniques support measurement or analyses of non-functional properties of running software systems. In general, two approaches are possible: (1) measurement and (2) modeling. The former is established based on direct measurement of the selected requirement of the system through the use of dedicated tools (e.g., profiler, tracer, etc.). For example, JMeter ("JMeter," 2014) can perform load testing for identifying bottlenecks in software applications, specifically cloud-based software. As another example, we can mention Load Runner ("Load Runner," 2014), which performs scalability analyses. Data extracted from these software can help in identifying critical parts of the system that require a modification to achieve the desired quality based on the analyses on non-functional properties. Modeling can solve restrictions of direct measurements specifically in large systems, because it may abstract away from the complexities of systems. In general, measurements and modeling are complementary rather than alternative techniques (Epifani et al., 2009).

The approach that is presented in this thesis for model calibration promises the benefits provided by both approaches based on measurement and those based on modeling. In this thesis, analytical models (i.e., Markov models) are kept alive at runtime and, via measurements, they can become gradually more precise. As a result, the decision for adaptations can become more accurate based on this latest situation of the running system.

In particular, (Woodside & Litoiu, 2008) describe a method for estimating model parameters through a control-based approach. This work is based on a runtime monitoring solution that provides data feeding a Kalman filter, aimed at updating the control-theoretic performance model. This approach differs from the approach presented in this thesis since it explicitly supports uncertainty handling in runtime data. Conversely, the approach is limited to the performance model, while the approach in this thesis is based on both discrete and continuous Markov models, which are appropriate to handle different sorts of properties comprising performance, reliability and so on.

The work in (Sato & Trivedi, 2007) is based on a CTMC formulation of composite services to predict performance and reliability bottlenecks by applying a sensitivity analysis technique. Although this work focuses on design-time, this thesis utilizes CTMCs for run time analyses.

The work in (L. Cheung et al., 2008) presents a framework for predicting component reliability at design-time. The objective of this work is to construct a stochastic reliability model allowing software architects to explore alternative designs. Specifically, the authors address the problem for parameter estimation at architectural level. The problem of correct parameter estimation is also discussed in (Gokhale, 2007) and (Smith & Williams, 2002).

Another work (Calinescu, Johnson, et al., 2011) proposes a novel approach for learning the parameters of DTMC parameters based on runtime observations of the system. This approach learns DTMC parameters through analyzing the system workload over time by adopting time series analysis. A very similar approach is also presented in (Epifani et al., 2009). The difference between the two approaches (i.e. (Calinescu, Johnson, et al., 2011) and (Epifani et al., 2009)) is that the former adopts a smoothing approach, which makes the estimations at runtime more effective in terms of precisions. These two pieces of work are the closest approaches to what we proposed here in this chapter and both were inspiring for the research reported in this thesis. Both approaches are based on Bayes formula (cf. Equation (4.18)) for model update. However, the main difference is that our approach is capable of estimating model parameters in the presence of measurement errors. In general, they (i.e. (Calinescu, Johnson, et al., 2011) and (Epifani et al., 2009)) explicitly assume that the observations based on which they derive the estimations at runtime are complete and contains no measurement errors. The other difference is that we experimentally evaluate the model update mechanisms through which some insights have been revealed. But, in general, in terms of underlying theory that has been adopted in these approaches, there is no critical difference.

There are also some established frameworks for facilitating evolution based on model updates at runtime, specifically in the domain of service-based systems. For example, KAMI (C Ghezzi & Tamburrelli, 2009) is a framework for runtime model updates for service-based systems. It focuses on nonfunctional models, which are typically dependent on some parameters that are provided a-priori by domain experts or extracted from other similar systems. Even if these values are initially correct, they can change during the operating life of the system. Consequently, accurate models must be updated over time. KAMI focuses on keeping nonfunctional models up to date with the current behavior of the modeled system by updating model parameters. Updated models can then be used to recheck at runtime the requirements that were initially verified at design-time to guarantee the correctness of the system. This may trigger appropriate adaptations of the system at runtime in order to adjust the system to satisfy system requirements.

As another example, QoSMOS (Calinescu, Grunske, et al., 2011) is a framework as well as supporting tools for QoS management of service-based systems. QoSMOS implements the full self-adaptation loop that combines formal specification of requirements, model-based QoS evaluation, monitoring and parameter adaptation of the QoS models, and planning and execution of system adaptation. The proposed framework integrates extended versions of existing tools such as KAMI (C Ghezzi & Tamburrelli, 2009) to realize the feedback control loop.

The approach presented in this chapter differs from existing work because we emphasize model calibration without considering that the runtime data are precise and free of noise. In fact, we strongly believe that self-adaptation must be based on explicit assumptions that the runtime data may be incomplete and contain noise.

4.6. Conclusions, Limitations and Future Work

In this chapter, we provided an answer to **RQ1** that requires the development of methods and techniques for estimating model parameters and serves as a pre-requisite for runtime requirement verification in the feedback control loop of self-adaptive component connectors.

Conclusions. In this chapter, our approach to analytical model evolution by runtime adaptation is presented. Our approach exploits Bayesian and Markov Chain Monte Carlo techniques to extend and develop estimators which produce updated model parameters considering the uncertainty in runtime observations. The updated analytical models provide a progressively better representation of the connectors that allows continuous automated verification of requirements at runtime. The analytical models updated at runtime support failure detection and prediction, and contribute to achieving self-adaptive component connector as the key thesis of this work. The main contribution of this work, rather than providing the statistical machineries to perform runtime adaptation of DTMCs and CTMCs, is handling runtime uncertainties by the mechanisms. We provided experimental evidence by conducting extensive simulation campaigns to shed light on issues like the mutual effects of the choice of different values for smoothing parameters, and the distance between design-time estimation and real values of parameters, and different change patterns in runtime data. In our experimental evaluations, the estimates converge to real values of probabilities (in the case of DTMCs) and transition rates (in the case of CTMCs) even in case of uncertain noisy runtime data, but the speed of convergence depends on several factors. For example, we investigated the influence of smoothing parameters and the dependence on the variance of runtime data.

Limitations and threats to validity. There are some threats to the validity of this work. The estimation approaches presented in this chapter can be categorized as time series estimation. The time series estimators in general imply certain limitations when used for short-term prediction of certain quality factors. As observed in the experimental evaluations, there is a conflict between being responsive enough to changes in observation data and being too responsive and thus being perturbed by fluctuations in very recent observations. In other words, if the smoothing parameter is too high, this may lead to a delay in the estimations. On the other hand, too little smoothing may lead to limited precision in highly variable scenarios. Although the value of the parameter can be set at deign-time based on the type of the system, however, the behavior of the system may change dramatically at runtime and the parameter requires to be adapted. We hypothesize that an adaptive filtering such as Kalman filtering approach can be a solution for adjusting the smoothing parameter dynamically at runtime. This can be considered as a potential future direction of such estimation techniques.

The other factor that influences the required time for the estimation to approach to the true value of the unknown parameter is the length of the observations. This means that if monitoring observations occur only very infrequently, the estimators in fact need to estimate based on long runs. Thus, the accuracy of the estimation approach may be severely limited in the setting of the runtime platform that provides observation data.

Threats to construct validity depend on how unknown model parameters were measured. This work considers DTMC and CTMC parameter measurement for self-adaptive component connectors performed through simulations. This can be affected by the load in the real setting of the connectors being deployed in for example internet-wide systems. However, this still reflects a realistic situation, as we generate the data using statistical distributions exactly the same way that simulation embedded in for example the Reo tool sets works (Moon, 2011).

Future directions. The future directions of this work will consist of refining the approach investigating its scalability in real-world noisy data. Currently our approach modifies models through the estimation of their unknown parameters and we do not take into account structural changes in the analytical models. In other words, model structures should be known when the estimation process is started and cannot change during estimations. This would be one of the most challenging future directions of this work mainly because of the difficulties in the theoretical background of structural model evolution and the runtime efficiency which is required in self-adaptive software, see (Antonio Filieri, 2013). However, this direction of research, although challenging, is also promising, as new techniques have been recently coming out, e.g., (Bianculli, Filieri, Ghezzi, & Mandrioli, 2014), based on incremental and compositional techniques, which are relevant to the domain of self-adaptive software.

Chapter 5

5. Adaptation Reasoning

"There are many different styles of composition. I characterize them always as Mozart versus Beethoven. When Mozart began to write at that time he had the composition ready in his mind. He wrote the manuscript and it was 'aus einem Guss' (casted as one). And it was also written very beautiful. Beethoven was an indecisive and a tinkerer and wrote down before he had the composition ready and plastered parts over to change them. There was a certain place where he plastered over nine times and one did remove that carefully to see what happened and it turned out the last version was the same as the first one." – Edsger Dijkstra (1930-2002).

Contents

5.1. Chapter Overview

Adaptation reasoning is the core reasoning process in self-adaptive systems (Lemos et al., 2013). The pieces of software that realize adaptation reasoning make decisions that affect how the base-level system interacts with the environment in which it is embedded and how it satisfies its quantifiable (non-functional) requirements. The decision is made over a reasoning space, which represents a dynamic and situational environment. Such decisions are dependent on domain knowledge and influenced by the historical behavior of the system over time. Note that the entities that reason about adaptation of a software perceive and use historical behavior of the system as a means to support the decision making process.

In the previous chapter, we introduced a number of stochastic methods to calibrate analytical models of component connectors in order to enable the requirement verification in the feedback control loop of self-adaptive connectors. The key challenge that we addressed there is the need to provide acceptable estimations of model parameters given that the input monitoring data are not perfect and contain uncertainties. This ensures that the requirement verification, which triggers connector adaptations, does not trigger unnecessary adaptations or miss necessary ones. In this chapter, the focus is on the reasoning part of the feedback control loop. We introduce a mechanism to select from many connector configurations the one that is most appropriate to obtain some specific performance result based on fuzzy adaptation reasoning. The key challenge that we intend to address here is the uncertainty corresponding to the specification of adaptation policies. More specifically, when specifying adaptation policies, different users have different and even conflicting opinions about adaptation action. We need an adaptation reasoning approach that considers such conflicting policies and decides about the adaptation of connectors considering these sources of uncertainty. Note that the scope of this chapter, as illustrated in Figure 5.1, is to Plan the adaptation (in the context of MAPE-K control loop).

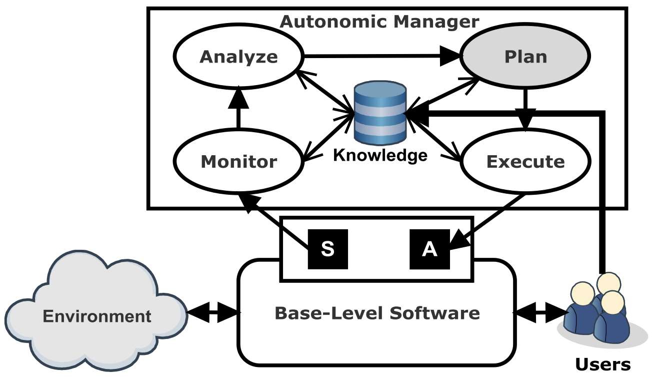


Figure 5.1. Scope of Chapter 5.

The outcome of this chapter is a framework, called RobusT2, for adaptation reasoning using type-2 fuzzy logic systems. In this chapter, we aim to address **RQ2** (cf. Chapter 1) that highlight the need for an adaptation reasoning mechanism that can determine the connector mode appropriate for current situation of the system. We called for a dependable mechanism that performs well in the presence of noisy inputs and imprecise adaptation policy specifications.

The rest of the chapter is organized as follows. Section 5.2 gives some fundamental definitions regarding adaptation reasoning. Section 5.3 reviews existing adaptation reasoning techniques for dynamic adaptive systems and self-adaptive software. Section 5.4 discusses how we specify non-functional requirements for component connectors based on type-2 fuzzy membership functions. Section 5.5 proposes a framework, called RobusT2, for adaptation reasoning using type-2 fuzzy logic. The RobusT2 framework, as the main outcome of this chapter and the main contribution of this thesis, contains the methods for designing the adaptation reasoning engine. This section also contains experimental results and discusses the significance of the main results. Finally, Section 5.6 summarizes the chapter.

5.2. A High-level Overview of Adaptation Reasoning

In this section, we briefly review fundamental concepts regarding adaptation reasoning comprising a definition of basic concepts in self-adaptive software, a high-level overview of autonomous reasoning and types of reasoning.

5.2.1. Environment representation

Environmental situations for software systems in general and self-adaptive software (SAS) in particular can be represented with a number of different techniques. These techniques represent environmental situations through variables as properties of the environment. These variables have well-defined *discrete* or *continuous* domains. For instance, the *battery level* of an autonomous robot or the *reliability* of a software component or the *performance* of a connector might be environmental variables for a software-intensive system. Each variable gets a value of its domain that conveys the current state of environment. For instance, battery level as a variable with a discrete domain space may be quantified as low, medium or high. On the other hand, reliability and performance may be quantified by continuous values within a well-defined range.

Environmental variable values determine the current state of a specific property of an environment at a very particular instant and a set of variables with their values represent the current state of the environment.

Definition 18. An ***environment state*** is a set of pairs each comprising an environmental variable and its current value at a precise time instant.

Definition 19. An ***environment situation*** of a self-adaptive software system is an environment state (**Definition 18**) which is important for the system and needs to be considered for the reasoning process. A set of situations, which obviously is a subset of all environmental states, might be predefined and encoded by domain experts or might be learned during the reasoning process.

5.2.2. Autonomous reasoning

Domain experts of the self-adaptive software know how the system must respond to environmental change. For example, an expert in robotics knows which camera resolution a robot should use when it has low battery level. As another example, an architect knows which communication channel must be used in connectors when the number of requests is high and the messages should not be lost in any circumstance. In adaptive systems in general and self-adaptive software in particular which operate in a specific domain, the reasoning process *encodes* the knowledge of that domain into a mapping between different environmental situations and the actions for system adaptations (Dobson et al., 2006; Müller, Pezzè, & Shaw, 2008).

The reasoning process is typically delegated to a piece of software, which is responsible for linking a particular environmental state to an action, which leads to an adaptation of the underlying software system. For example, in the task queue connector example in Chapter 4, when the rate of receiving messages at the receiving port increases, an environmental change happens. The reasoning mechanism in this case processes the change and based on the environmental information, it decides to use another channel with better quality of service.

The reasoning process typically consists of two critical steps: (i) processing a time-series runtime data, which are collected through monitoring facilities; and, (ii) decision making about the adaptation action to perform when a specific environment state (**Definition 18**) occurs. The first step is covered in the model calibration chapter (i.e., Chapter 4). The second step, which is known as adaptation planning, is the main subject of this chapter.

One important and critical issue is that how this decision making can be performed in self-adaptive software. Let S be a finite set of situations (**Definition 19**) that a software system might encounter based on different environmental states occurs. Also, let C be a finite set of valid architectural configurations that the software system can take. As soon as a specific situation $s \in S$ is detected, the reasoning mechanism chooses a (possibly optimal) architectural configuration $c \in C$ from possible configurations of the software system. This process can be formally delineated by:

$$\begin{aligned} &P: S \rightarrow C \\ &U\big(s_i, P(s_i)\big) = \arg\max \,\{U(c_j, s_i) | \forall c_j \in C: \} \end{aligned} \tag{5.1}$$

Where P in equation (5.1) is a reasoner, which maps each situation $s \in S$ into an appropriate architectural configuration $c \in C$ during the adaptation process, which might be performed in a self-managed manner in particular. The notion of a reasoner here generalizes that of a planner by covering a broader domain of reasoning consisting of analysis and planning altogether. The notation U is a function which quantifies the appropriateness of a configuration $c_j \in C$ for the given situation $s_i \in S$. The main goal here for reasoning is to find the most appropriate and (possibly optimal) configuration given the constraints such as timing or frequency of adaptation in dynamically changing environments.

The problem of building efficient reasoners as a self-adaptive controller has been the target of several projects, which resulted in various approaches that differ regarding the types of reasoning with respect to the required level of flexibility and adaptability, and for the techniques to identify a suitable configuration while reacting to the changes in environment. One of the key requirements for this kind of reaction to the environment is to achieve the adaptation in a timely manner.

5.2.3. Types of reasoning in self-adaptive software

There are two different types of reasoning in general and planning in particular here: offline (design-time) and online (runtime) planning.

5.2.3.1. Offline reasoning

Design-time planning (or offline reasoning) means that decisions to relate environmental situations to architectural configurations (plans) are made statically at design-time. More specifically, whenever a self-adaptive software system encounters a specific environment situation $s_i \in S$, the self-adaptive controller selects one specific architectural configuration $c_j \in C$. The next time when the same situation s_i is sensed, the reasoner chooses exactly the same plan c_j for the software system. These predefined plans, which *freeze* the adaptation space, are referred to as *tactics* (Georgiadis, Magee, & Kramer, 2002) or *strategies* (D Garlan et al., 2004), *change operation* (Oreizy, Medvidovic, & Taylor, 1998), *policy* (Georgas & Taylor, 2008) or even a combination of them (S.-W. Cheng & Garlan, 2012).

These approaches can be effective if it can be demonstrated that the set of frozen change plans are sufficient to deal with any possible environmental situations (Kramer & Magee, 2007). These fixed plans might be sufficient for domains such as embedded software or fault-tolerant architectures. However, it is very difficult to identify a sufficient set of them due to the nature of planning problems (Klein, 2007).

Runtime planning, however, is a more flexible alternative to overcome the shortcoming of offline planning approaches.

5.2.3.2. Online reasoning

Runtime planning (or online reasoning) means that the decisions to relate environmental situations to architectural configurations are made dynamically at runtime. In other words, the planner autonomously generates a configuration, which suits the current situation of the surrounding environment.

Online reasoning, in general, consists of three major steps (Park, 2009): selection, evaluation, accumulation. In the selection step, the planner autonomously chooses a suitable configuration for the current configuration. In the evaluation step, the planner measures the effectiveness of the chosen plan. Finally, in the accumulation, the planner stores the configurations alongside their evaluations for future planning purposes.

In the next section, we review existing techniques that have been applied for enabling adaptation reasoning in open-loop dynamic adaptive software and in closed-loop self-adaptive software.

5.3. Existing Reasoning Techniques in Self-Adaptive Software

There are several techniques (or strategies) to be employed for adaptation planning with different capabilities and characteristics. These techniques comprise rule-based, goal-based, utility-based planning, reactive, heuristic-based, test-based, learning-based, model-based and control-based. Note that among these techniques, rule-based, goal-based and utility function reasoning have been applied a lot in practice and are mature enough to be used by most self-adaptive systems. However, other techniques are still in early formative stages of maturity; for instance, test-based techniques (Metzger et al., 2013) have just recently been proposed in the self-adaptive community.

5.3.1. Rule-based reasoning

In rule-based planning, decisions are made for a set of predefined environmental situations. Generally, they are specified by "$if\ (situation)\ then\ (action)$" statements (Fleurey & Solberg, 2009; Georgas &

Taylor, 2008). When a set of environmental states from the $situation$ occur then the $action$ takes place. An alternative version of rule-based reasoning are event-condition-action (ECA) rules. In this form when a special $event$ occurs, it may activate rules with specific $conditions$ that, if satisfied, activate a set of $actions$ (Georgas, Hoek, & Taylor, 2009; Brice Morin, Barais, Jezequel, Fleurey, & Solberg, 2009). The $conditions$ are usually quantitative values of environmental variables. However, there are some approaches that enable qualitative descriptions, which handle imprecision and variation instead of exact quantitative values (Franck Chauvel, Barais, Borne, & Jezequel, 2008). A special kind of rule-based reasoning considers the notion of reasoning based on some temporal properties. By adopting this type of rule, one can specify $conditions$ over a sequence of the past environmental states.

By adopting this technique, the adaptation behavior can be analyzed and validated at design-time. It is also efficient at runtime. However, it requires the identification and enumeration of all possible configuration variants and adaptation rules at design-time. The number of rules may also become unfeasibly large. For example, although writing new rules is straightforward, it is hard to foresee errors that might be introduced based on conflicts between rules (Serugendo, Fitzgerald, Romanovsky, & Guelfi, 2007).

However, some new advances address some of the issues. In the DiVA project (Brice Morin et al., 2009), the rules can be deduced at runtime by comparing the resulting product model with the model of the running system. The rules in rule-based reasoning can be enhanced by learning new rules, which are derived based on the past facts (Abbas, Andersson, & Weyns, 2011).

5.3.2. Goal-based reasoning

Goal-based planning is inspired by multi-agent planning within which a number of agents cooperate to achieve a specific goal. In goal-based reasoning, high-level goals drive the adaptation process (JO Kephart & Chess, 2003). The goals are typically expressed using a declarative language such as PROLOG. When an environment changes, the controller of the system decides if the system still holds its declared goals. If not, the controller selects a set of adaptation actions in order to lead to the goal.

A number of approaches have explored goal-based planning in self-adaptive software (Eliassen, Gjørven, Eide, & Michaelsen, 2006; Heaven, Sykes, Magee, & Kramer, 2009; Morandini, Penserini, & Perini, 2008; Salehie & Tahvildari, 2012; Sykes, Heaven, Magee, & Kramer, 2007). The general process is as follows. These approaches first define the high-level goals the system should attain. Then a number of primitive change operations are defined. Finally, a number of processes are defined to connect the goals and actions. However, in contrast to rule-based reasoning, the writing of goals is not situational and therefore straightforward. As a result, a number of researchers (Sykes et al., 2007; Sykes, Heaven, Magee, & Kramer, 2008) propose to declare the goals using linear temporal logic (LTL). Then, the plans can be generated automatically to achieve the goals.

In general, adopting goal-based reasoning avoids the problem of fixed configurations, which are enumerated at design-time, but usually at the cost of runtime overhead. This approach suffers from scalability issues. One of the other difficulties of goal-based reasoning is whether all the goals are achievable, because in the system of goals, one special goal might be prevented from being attained because of mutual conflicts between goals.

5.3.3. Utility function reasoning

Utility function planning has its root in functional optimization, which consists of a given mathematical function, and the goal is to find its optimal value with respect to a set of parameters. In self-adaptive software, utility function-based reasoning is used to find and select the most appropriate architectural configuration for a given environmental situation using functional optimization. The function to be optimized is called *utility function* and is defined in terms of the utility that each architectural configuration can provide given a particular state of the environment. More specifically, this function defines a quantitative basis of desirability of a given configuration. The desirability is a mapping such as U in equation (5.20) between each configuration $c_j \in C$ and its worthiness with respect to environmental state $s_i \in S$. Therefore, in the case of environmental changes, the adaptation is decided based on the optimization (maximization or minimization) of the utility function. To do so, there are many approaches in mathematics from linear programming to meta-heuristic approaches that can be applied to find the optimal value.

A number of approaches have investigated the application of utility function optimization for decision making in self-adaptive software (Bennani & Menasce, 2005; S.-W. Cheng, Garlan, & Schmerl, 2006; Georgas et al., 2009; Jeffrey Kephart & Das, 2007; Marzolla & Mirandola, 2010; Sykes, Heaven, Magee, & Kramer, 2010). Typically, the utility function is defined as the sum of the utility value of the adaptable parts of the architectural configuration in relation to the environment. The utility function may be defined by several parameters and even combinations of continuous and discrete functions. Sometimes, the utility functions need to be defined by multiple functions (S.-W. Cheng et al., 2006). Therefore, in this case the problem of optimizing the utility function can be categorized as a multi-objective optimization problem whose goal is to reach the best compromise among different competing functions.

Utility functions can discriminate between different architectural configurations and evaluate them, but without the need for constructing them. Although this makes the reasoning process quite straightforward, constructing the utility function itself is not an easy task.

5.3.4. Reactive planning based reasoning

A reactive planning explores gradually the space of architectural configurations. As opposed to traditional planning which searches for a sequence of actions satisfying predefined objectives, reactive planning searches for a single action, which contributes to a better satisfaction of the objectives, which themselves might change according to environmental changes. A number of approaches have explored reactive planning based in architecture-centric self-adaptive software (F Chauvel, Song, & Chen, 2010; Sykes et al., 2010). These approaches do not need to freeze the adaptation space by ***predefined architecture configurations*** as in *Plastic* (Batista, Joolia, & Coulson, 2005), *C2* (Oreizy et al., 1998), *Genie* (Nelly Bencomo, Grace, Flores, Hughes, & Blair, 2008). Or by ***predefined architecture-change operations*** as in *Rainbow* (S.-W. Cheng et al., 2006; D Garlan et al., 2004), *MADAM* (Floch et al., 2006), *DiVA* (Fleurey & Solberg, 2009), *Sykes et al.* (Sykes et al., 2007, 2008). Instead, they can explore unforeseen architecture configurations in dynamic environments.

5.3.5. Heuristic-based reasoning

As the state space of adaptation grows, the management of adaptation rules becomes difficult. In order to solve this, a number of approaches have investigated heuristic-based approaches such as genetic algorithms (e.g., Plato (A. J. Ramirez, Knoester, Cheng, & McKinley, 2009, 2010)) or hill climbing (C Ghezzi & Sharifloo, 2013) in order to make adaptation plans.

A key benefit of this approach is that there is no need for prescribing adaptation plans beforehand to address environmental situations warranting adaptation. Instead, they use the power of heuristic search to generate suitable adaptation plans at runtime. They also exploit computationally inexpensive fitness functions, which is akin to utility functions in order to evolve the plans in response to changing environmental states. Therefore, they can handle more environmental situations than traditional prescriptive approaches. One potential drawback of this approach is that the heuristic approaches cannot guarantee that the best adaptation plan will be found.

5.3.6. Test-based reasoning

Online testing (E. M. Fredericks et al., 2013; Hielscher & Kazhamiakin, 2008; Metzger et al., 2013) means that the self-adaptive software system is fed with test data in parallel to its normal run in order to detect or predict failures. Whenever a test fails, it can trigger an adaptation in order to prevent the system from actual failure. Since online testing actively engages in collecting runtime data and complement monitoring runtime data, the accuracy of reasoning process can be improved significantly (Metzger et al., 2013). However, there are some limitations including increasing the runtime load and imposing additional costs (Metzger et al., 2013).

5.3.7. Learning-based reasoning

By using learning data, which are accumulated during system execution, the self-adaptive system can determine the best configuration when the environment changes (Park, 2009). Generally, by repeating the process of execution, accumulation, learning and decision-making, the system can make better decisions over time.

Gambi et al. (Gambi, Toffetti, & Pezzè, 2010) use online machine learning to update surrogate models in order to limit violations of SLAs of software applications within data centers. However, their approach does not apply any state space reduction heuristics to improve runtime convergence.

Tesauro et al. (Tesauro, 2007) proposed a hybrid approach to combine white-box analytical modeling (i.e., QN model) with Reinforcement Learning. Online learning uses a black-box model of the running system, while the white-box QN model is used as a training facility. They assume white box QN model of the system is available and can accurately predict its behavior under different adaptation decisions.

Kim and Park (Park, 2009) use a reinforcement learning approach for online planning. Their work is based on behavior improvements of robots by learning from design-time training and by dynamically discovering adaptation plans in response to changes in the environment in which the robots are operating. Similarly, Zhao et al. (Zhao, 2011) combine supervised learning and reinforcement learning to develop an adaptive real-time cruise control system.

FUSION (Elkhodary et al., 2010; Esfahani, Elkhodary, et al., 2013) is a general-purpose framework for self-adaption of the software systems, which is fundamentally different from the above, as it works in a way that it is a general-purpose framework, while the above mentioned works have concentrated on specific problem domains. FUSION combines a number of techniques comprising feature-based modeling, significance testing, and heuristic search to reason about adaptation.

5.3.8. Model-based quantitative reasoning

In general, analytical models can be categorized into two broad groups (Esfahani, Elkhodary, et al., 2013): white-box and black-box. The former requires an explicit model of the internal organization of the software system (i.e., typically an architectural model), while the latter does not need such knowledge.

Queuing Networks (QN) (Ardagna et al., 2008) are mathematical models used for performance analysis of a software system, represented as a collection of Queues (i.e., system resources) and Customers (i.e., user requests). Markov models (Ardagna et al., 2008) are often used for reliability analysis. They are comprised of a stochastic model that captures the state of the system using random variables that change overtime according to the probability distribution of the previous state. These white-box approaches require an explicit model of the internal structure of the software system. Such models are typically used at design-time to analyze the tradeoffs of different architectural decisions before implementation, but recently these models are used at runtime to dynamically analyze the system properties (Ardagna et al., 2008). However, the structure of these models cannot be easily changed at runtime in ways that were not accounted for during their formulation (e.g., addition of new states in a Markov model due to emerging software behavior).

Artificial Neural Networks (ANN) are an effective way of solving a large number of nonlinear estimation problems (Esfahani, Elkhodary, et al., 2013). These approaches do not require knowledge of the internal structure of the system. However, they require enough sampling of the input/output parameters to build a rough calculation of the relationship between the inputs and outputs. A main advantage of black-box approaches is that they can be used to detect changes to the underling properties of a software system over time. FUSION (Esfahani, Elkhodary, et al., 2013) follows the black-box approach.

5.3.9. Control theory based reasoning

Control theory is recently gaining momentum in the software engineering community (Hellerstein et al., 2004; Jamshidi et al., 2014; Zhu et al., 2009). For instance, in the proceedings of the most recent ICSE, Companion Proceedings of the 36th International Conference on Software Engineering (Briand & van der Hoek, 2014), as well as the proceedings of the most recent SEAMS, Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (Engels & Bencomo, 2014), we have seen an increasing number of control theory-based approaches applied to address the challenges in software engineering. The application of control theory in software engineering, however, is still at a preliminary stage (Patikirikorala et al., 2012) and is limited to the design of controllers focused on particular ad-hoc solutions that address a specific computing problem (Antonio Filieri et al., 2014). Filieri et al. (Antonio Filieri et al., 2014) developed a general methodology, which reduces the need for strong mathematical background to devise ad-hoc control solutions.

As a notable control theory-based approach that has been applied for adaptation reasoning, we can mention the work of Filieri et al. (Antonio Filieri, Ghezzi, Leva, & Maggio, 2012). In this work, the bindings among services are dynamically set at runtime. They formulated the dynamic binding problem as a feedback control problem, and solved it with simple controller synthesis.

Yang et al. (Q.-L. Yang et al., 2013) proposed a type-1 fuzzy logic system to adjust system parameters (as opposed to ours on architecture change) in mission-critical systems. They deal with uncertainty in the self-adaptation loop by representing adaptation logic with type-1 fuzzy membership functions. Gmach et al.

(Gmach et al., 2008) and Chuang and Chan (Chan, 2008) proposed the use of type-1 fuzzy logic systems for adaptive service management to remedy exceptional situations and manage quality of services respectively. They applied fuzzy control theory to manage and balance resources of enterprise service. In a recent published work (Jamshidi et al., 2014), we proposed an extended fuzzy controllers for solving similar problem of elasticity reasoning in the context of autonomic resource provisioning for cloud-based applications. The main challenges that we addressed in this work were to control the measurement uncertainties and uncertainty related to specifying elasticity policies (corresponding to the adaptation policies here in this thesis).

5.3.10. Summary of reasoning techniques

Table 5.1 summarizes the adaptation reasoning approaches we reviewed earlier for enabling self-adaptation of software systems. In this table, a check mark (√) indicates where the approach proposes solutions or deals with the criteria, and a blank ( ) in the opposite case.

Note that among the reviewed approaches, rule-based, goal-based and utility function reasoning are mature and other techniques are still in early formative stages of maturity.

All of the reviewed techniques have their own pros and cons and are mostly situational, i.e., they are more appropriate for addressing the challenges in a particular domain. For example, rule-based reasoning is useful in situational contexts, in which predefined responses are triggered by predefined events. Such approaches based on rule-based reasoning can also consider learning to update the rule set at runtime. One of the downsides of such approaches is that, it is difficult to manage rule-based reasoning, especially when thousands of rules are in the rule base. In contrast, goal-based and utility-based reasoning use very different approaches. The first one introduces the concept of goals that enable the reasoning process, and uses a variation of goal models to derive each particular adaptation decision. It enables decision making by accomplishing a set of goals. The later defines a priori mapping in terms of a utility functions between reasoning variables and architectural configurations of the system. The utility-based approaches evaluate a decision and pick one option out of many. More concretely, goal-based reasoning can select the best configuration given the encoding in utility function. However, rule-based and goal-based reasoning cannot ensure such optimum selection.

This domain is not yet mature, and as a result, new adaptation reasoning approaches appear in the software engineering and self-adaptive communities. These techniques use heuristics, models and test cases, machine learning, control theory and reactive techniques to drive the system adaptations and produce promising results. Nevertheless, they are still at a formative stages and subject of current ongoing research. In this thesis, we use fuzzy control as the adaptation reasoning technique. The rationale that motivates this choice is that fuzzy control use rules elicitation from users that properly captures the relationship between particular environmental conditions, historic observations, and decision-making in the reasoning process. The *main difference* between the existing control-based approaches and our approach is that the fuzzy logic controller we employed in this work can handle expert knowledge and numerical data in a unified framework, and the fuzzy reasoning approach, in general, requires less computational complexity. The other benefit of our approach is that fuzzy logic controllers do not require the mathematical model of the system that it controls. In this work, deriving an accurate mathematical model of the underlying software is a very difficult task due to non-linear dynamics of real systems (Esfahani, Elkhodary, et al., 2013; Hellerstein et al., 2004; Lemos et al., 2013; Zhu et al., 2009).

Table 5.1. Classification and comparison of adaptation reasoning approaches.

Reference	Scope				Reasoning Technique									Environ ment		Domain				
	Model	Architecture	Code	Requirement (Spec)	Rule-based	Goal-based	Utility function	Test-based	Learning-based	Model-based	Reactive planning	Heuristic-based	Control-based	Context	NFR	Mobile	Embedded System	Smart-*	Robotic	General-purpose
(Georgas & Taylor, 2008)		√			√										√				√	
(Fleurey & Solberg, 2009)		√			√		√							√					√	
(Brice Morin et al., 2009)		√			√									√		√				
(Georgas et al., 2009)		√			√										√					√
(Franck Chauvel et al., 2008)		√			√		√								√					√
(Serugendo et al., 2007)		√			√		√								√					√
(Eliassen et al., 2006)		√				√	√								√					√
(Heaven et al., 2009)	√	√				√	√								√				√	
(Morandini et al., 2008)				√		√								√						√
(Sykes et al., 2007)		√				√					√				√				√	
(Salehie & Tahvildari, 2012)				√		√	√								√					√
(Jeffrey Kephart & Das, 2007)	√				√		√								√					√
(Bennani & Menasce, 2005)	√						√								√			√		
(S.-W. Cheng et al., 2006)		√					√								√					√
(Sykes et al., 2010)		√				√	√							√	√				√	
(Marzolla & Mirandola, 2010)		√					√								√					√
(F Chauvel et al., 2010)		√					√								√					√
(Batista et al., 2005)		√			√										√					√
(Oreizy et al., 1998)		√			√										√					√
(Floch et al., 2006)		√			√		√							√						√
(A. J. Ramirez et al., 2009)		√					√								√			√		
(C Ghezzi & Sharifloo, 2013)				√			√								√					√
(Hielscher & Kazhamiakin, 2008)				√				√							√					√
(Metzger et al., 2013)				√				√						√	√					√
(Park, 2009)		√					√		√						√					√
(Gambi et al., 2010)	√								√					√	√					√
(Tesauro, 2007)	√								√					√	√					√
(Esfahani, Elkhodary, et al., 2013)	√								√						√					√
(Antonio Filieri et al., 2012)		√								√			√		√					√
(Chan, 2008)		√			√										√					√
(Gmach et al., 2008)		√			√										√					√
(Q.-L. Yang et al., 2013)		√			√								√	√	√			√		
RobusT2 (our approach)		√			√								√	√	√					√

5.4. Non-functional Requirements for Component Connectors

In contrast to traditional requirements, self-adaptive software systems require a new class of requirements (Lemos et al., 2013). We consider two types of quantitative non-functional requirements: 1) ***crisp*** (*hard* (Glinz, 2005)), and 2) ***fuzzy*** (*elastic* (X. (Frank) Liu, Azmoodeh, & Georgalas, 2007), *soft* (Glinz, 2005)). A *crisp* quantitative non-functional requirement imposes rigid constraints on a non-functional property (e.g., end-to-end response time or reliability) of a system or more precisely for component connectors in this work. After verifying the satisfaction of a crisp requirement, it is either satisfied or unsatisfied. While a *fuzzy* non-functional requirement imposes a flexible constraint on a non-functional property of a connector using a membership function of a qualitative term to characterize its satisfaction. Note that a membership function quantifies a degree of membership of a qualitative term in a fuzzy set (Zadeh, 1965).

5.4.1. A specification of non-functional requirements with Type-1 (T1) fuzzy sets (FS)

Below is an example of a crisp quantitative non-functional requirement NFR_1 with satisfaction function in Figure 5.2. In this case, if the coordination time between the two specific points (one source node and one sink node) takes, for example, 1.005 milliseconds, it does not satisfy the requirement and it leads to an adaptation of the connector.

NFR_1: The period of time between receiving a message from a component to dispatching it to another component must be less than a millisecond. (5.2)

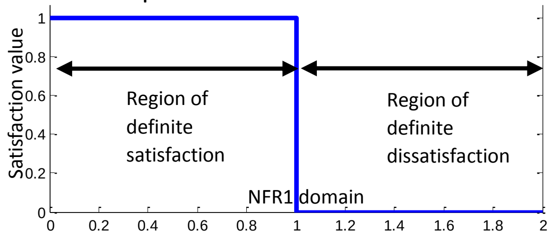


Figure 5.2. Satisfaction function for requirement NFR1.

However, in the case of the following a fuzzy non-functional requirement, i.e., NFR_2 with membership function in Figure 5.3, if the coordination takes 0.8 milliseconds, its satisfaction degree is one, which is the highest. It actually points out that the requirement is fully satisfied by the connector. If it takes, for example, $1.4ms$ its satisfaction degree is around 0.6 and it partially satisfies the requirement though it is acceptable.

NFR_2: The period of time between receiving a message from a component to dispatching it to another component must be SHORT. (5.3)

“SHORT” in (5.3) is a linguistic term, whose membership function (see Figure 5.3) characterizes satisfaction of the requirement NFR_2.

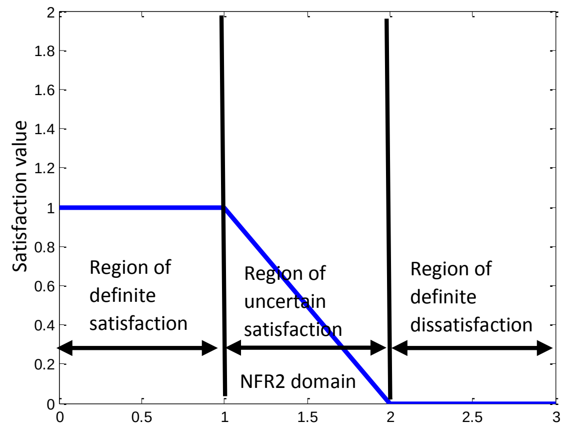


Figure 5.3. Satisfaction function for requirement NFR2.

The constraint imposed by an imprecise or fuzzy requirement R is characterized by a satisfaction function (X. Liu & Yen, 1996).

Definition 20. A ***satisfaction function***, denoted by μ_R, maps an element of R's domain or universe of discourse D to a number in $[0,1]$ that represents the degree to which the requirement R is satisfied.

$$\mu_R: D \rightarrow [0,1] \tag{5.4}$$

Intuitively, the satisfaction function characterizes a fuzzy subset of a requirement's domain, which satisfies the fuzzy requirement. In fuzzy non-functional requirements, a minimum threshold is usually specified. It shows that a connector configuration, whose value is below this threshold, is not acceptable and it should trigger an adaptation to lead to a configuration that has a satisfaction degree of a value higher than zero. For instance, if a coordination takes 2.1 milliseconds, which is greater than the threshold of two, in this case, its satisfaction degree is zero and it is totally unacceptable.

The satisfaction function (cf. membership function in fuzzy set theory (Zadeh, 1965)) for fuzzy requirements must vary in $[0,1]$. The function itself can be of any shape that defines a function, which is simple enough to interpret the satisfaction degree. The simplest satisfaction function is formed using straight lines. Figure 5.4 and Figure 5.5 respectively illustrate a simple triangular and a trapezoidal satisfaction function.

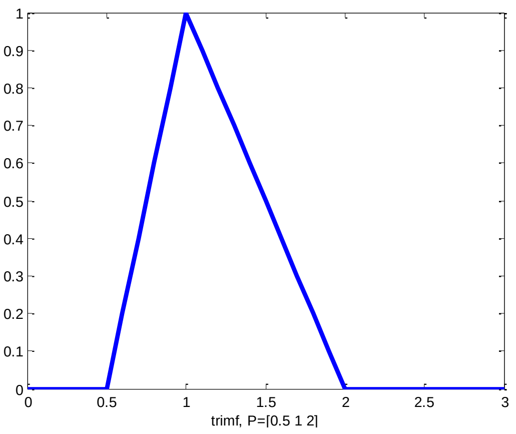


Figure 5.4. A triangular satisfaction function.

A number of key characteristics of satisfaction functions are as follows (cf. Figure 5.5):

The *height* of a satisfaction function is the largest degree, which satisfies the requirement R.

$$height\ (R) = \sup \mu_R(x) \tag{5.5}$$

The support of a satisfaction function is the crisp set containing all elements with non-zero satisfaction degree.

$$support\ (R) = \{x | \mu_R(x) > 0\} \tag{5.6}$$

The core of a satisfaction function is the crisp set containing elements with satisfaction degree equal to one.

$$core\ (R) = \{x | \mu_R(x) = 1\} \tag{5.7}$$

The boundary of a satisfaction function is the crisp set with satisfaction degree higher than zero and lower than 1.

$$boundary\ (R) = \{x | 0 < \mu_R(x) < 1\} \tag{5.8}$$

The $\alpha - cut$ of a satisfaction function is the crisp set that contains all the elements whose satisfaction degree are greater than α.

$$^{\alpha}R = \{x | \mu_R(x) \geq \alpha\} \tag{5.9}$$

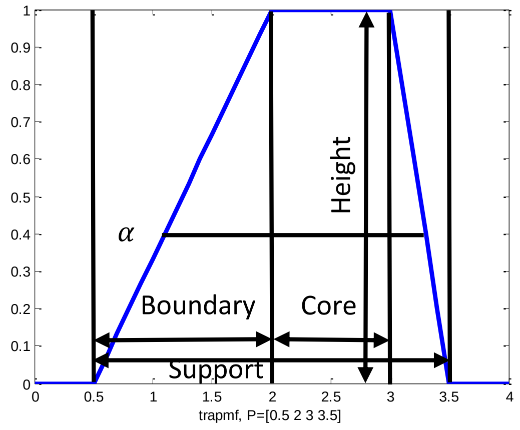


Figure 5.5. A trapezoidal satisfaction function.

Due to their smoothness, Gaussian satisfaction functions are appropriate for specifying fuzzy sets. As illustrated in Figure 5.6 and Figure 5.7, the curve has the advantage of being smooth and non-zero at all points. However, since in self-adaptive software we need to reason about the satisfaction of requirements, and since in this function there is no point with satisfaction degree zero, this type of function is of limited use.

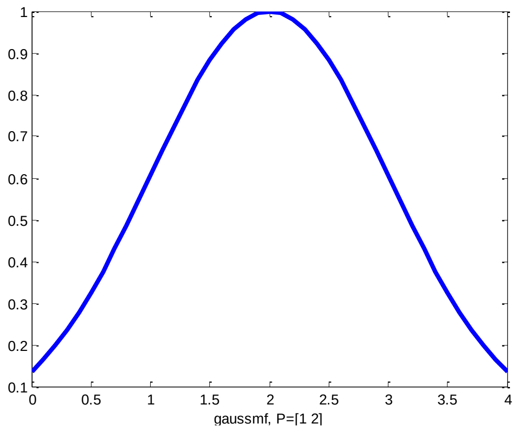


Figure 5.6. A Gaussian satisfaction function.

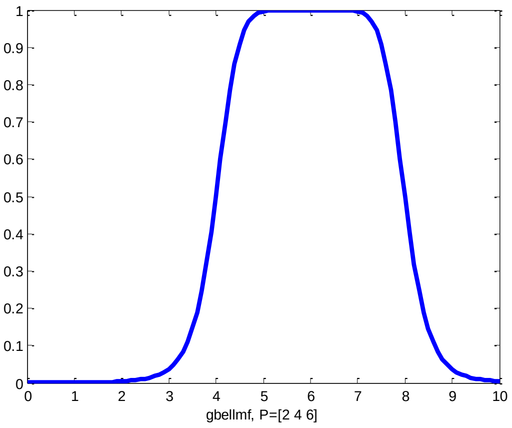


Figure 5.7. A bell satisfaction function.

Although Gaussian and bell functions have the advantage of being smooth, they are unable to specify asymmetric functions, which are critical in some requirement specifications. Figure 5.8 shows a sigmoidal satisfaction function, which is open to the right.

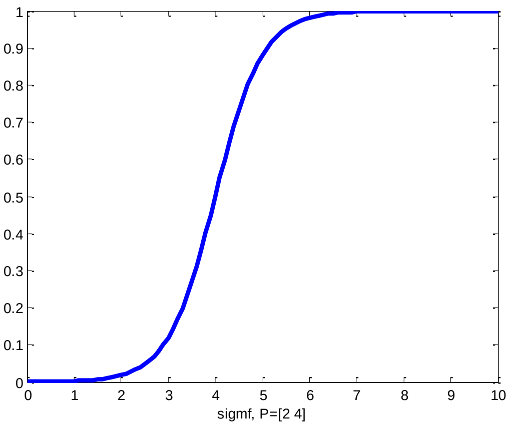


Figure 5.8. A sigmoidal satisfaction function.

One of the main advantages of using fuzzy non-functional requirement specification in self-adaptive software is the avoidance of unnecessary adaptation of connectors due to transient violations of constraints. Let us consider the coordination time in a connector that oscillates between 0.99 and 1.01.

By considering NFR_1, each time the coordination times goes up to 1.01, it violates the requirements and an adaptation will be necessary. However, by considering NFR_2, the satisfaction degree is close to one and far greater than the minimum threshold of satisfaction based on the function illustrated in Figure 5.3. This result is more desirable than the one obtained using crisp function as depicted in Figure 5.2 because it avoids several unnecessary adaptations during runtime execution of the connector.

Such a specification of non-functional requirements through type-1 fuzzy membership function is not new and it has been adopted mainly to specify *flexible* requirements in (Luciano Baresi et al., 2010; X. Liu & Yen, 1996; Whittle et al., 2009; Yen & Tiao, 1997). The challenge is where there is uncertainty about the membership function itself. In the following, we first raise this concern by providing concrete scenarios and we address this challenge by adopting T2 fuzzy logic in order to incorporate uncertainty about the adaptation policies. The RobusT2 framework is proposed in Section 5.5, to address this challenge and provide a mechanism to reason about adaptation handling such sources of uncertainties.

5.4.2. The need for revisiting non-functional requirement specifications

In order to deal with uncertainty, it needs to be quantified. In this section, we discuss why the self-adaptive software community needs to revisit approaches for quantifying uncertainty. Users provide input for different aspects of a self-adaptive software system (Esfahani & Malek, 2013; Lemos et al., 2013). The most prominent user inputs are: 1) Non-functional requirements or quality preferences. 2) Particular system properties that cannot be monitored. 3) Certain environmental properties that can be specified by engineers based on their experiences, hardware specification, or similar systems. 4) Adaptation policies and their effects on the quality factors. However, eliciting user preferences for the aforementioned inputs in terms of a mathematical function or an absolute value is a well-known challenge (Lemos et al., 2013) and introduces subjective and imprecise data to the system. Thus, in order to cease the effects of this uncertainty, we need to adopt an appropriate mathematical theory to quantify them.

There are four different possibilities when extracting a user input: 1) One user provides a crisp value representing an estimation of the expected value for the input. 2) One user estimates a range of uncertainty based on the expected level of variation in the input. 3) A group of users provides a set of crisp values as estimations of the expected value for the input. 4) A group of users provides estimations of the range of uncertainty based on the expected variation in the input. Although it has been commonly used in self-adaptive software domain, we argue that the first three approaches are not scientifically accurate enough to capture the uncertainty regarding adaptation knowledge specification. The first issue is with the crisp estimation of the input (i.e., 1, 3). In this way, the possibility distribution of the input becomes like a step function that takes either "zero" or "one" and the tradeoffs between different quality factors become impossible. The second issue is with the elicitations based on a single user (i.e., 1, 2). Users often have diverse opinions about specific inputs. Therefore, inputs based on just one user is partial and the approaches such as (Luciano Baresi et al., 2010; S. Cheng & Garlan, 2007; Esfahani et al., 2011; Whittle et al., 2009) pursuing this way of knowledge elicitations ignore the uncertainties associated with collecting inputs from different users. This assumption based on one user opinion is, in fact, unrealistic for certain types of applications such as multi-tenant systems. Figure 5.9 shows how a user estimates the range of a requirement with a trapezoidal function.

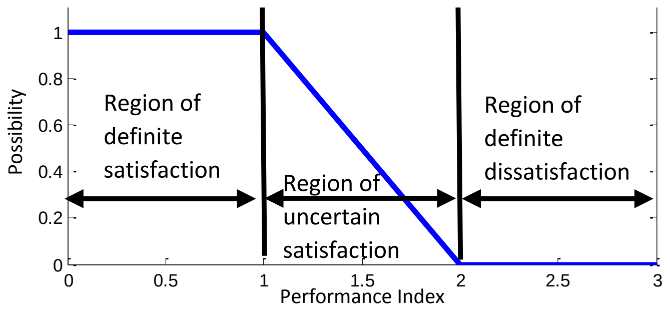


Figure 5.9. A trapezoidal possibility distribution.

The current work for handling uncertainty in self-adaptive software assumes one can accurately specify the possibility distribution. When someone specifies a possibility distribution of an input, as soon as the function is specified, there remains no uncertainty in that function. For instance, as soon as the possibility distribution of the performance of a component is specified in Figure 5.9, the three associated regions are precisely determined. Yet, as we mentioned in the fourth approach for eliciting the input, each user might come up with different distributions (see Figure 5.10). The challenge is how to accommodate all preferences in one coherent distribution.

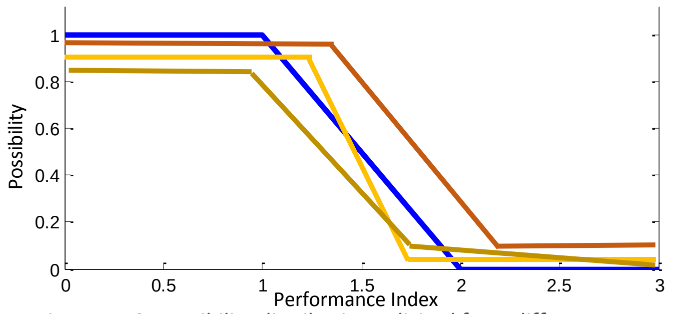


Figure 5.10. Possibility distributions elicited from different users.

We address this challenge by adopting type-2 fuzzy sets (Zadeh, 1975) in order to incorporate uncertainty in the distribution function.

5.4.3. A specification of non-functional requirements with Type-2 (T2) fuzzy sets (FS)

The concept of imprecise requirements (X. Liu & Yen, 1996) is not a new phenomenon. Fuzzy theory is mainly used to specify *uncertain*, *flexible* and *imprecise* requirements (Whittle et al., 2009), to accommodate adaptive goals (Luciano Baresi et al., 2010) and to perform trade-offs among conflicting functional (Yen & Tiao, 1997) and non-functional requirements (Esfahani et al., 2011). In these works, they exploit fuzzy theory to specify at design-time and assess at runtime the satisfaction degree of requirements, which is specified by a membership function. The idea is to prevent violations of requirements by tolerating some small transient deviations. The presumed benefit of such imprecise requirements is that it gives the system room to behave flexibly (Chopra, 2012). More specifically, a system can partially satisfy a requirement depending on circumstances. However, the state-of-the-art in this domain assumes one can specify the satisfaction function precisely. However, this is not practical and realistic in real-world applications. The question is what to do when there is uncertainty about the value of the membership function. In this section, we intent to address this shortcoming by adopting type-2 fuzzy logic (Zadeh, 1975) in order to incorporate uncertainty about the satisfaction function. The fundamental difference between type-1 and type-2 fuzzy logic is in the model of individual fuzzy sets.

Type-2 fuzzy sets employ membership degrees that are not a crisp value, but fuzzy sets themselves. This additional uncertainty dimension provides new degrees of freedom for modeling dynamic uncertainties.

5.4.3.1. Requirement specifications with IT2-FS

Although a number of analytical frameworks (Luciano Baresi et al., 2010; X. Liu & Yen, 1996; A. Ramirez & Cheng, 2012) based on type-1 fuzzy sets have been proposed by researchers in requirements engineering and self-adaptive software, we intent to extend the concepts based on interval type-2 fuzzy sets. We therefore call the previous imprecise requirement specification approaches based on type-1 fuzzy logic traditional approaches. Those traditional approaches based their satisfaction function on **Definition 20**. However, we specify the satisfaction function according to the definition that is given in **Definition 21**.

Definition 21. A ***satisfaction function***, denoted by $\mu_{\tilde{R}}$, maps an element of $\tilde{R}$'s domain or universe of discourse D to an interval $[\underline{\mu}_{\tilde{R}}(x), \overline{\mu}_{\tilde{R}}(x)]$ that represents the spectrum of degrees to which the requirement $\tilde{R}$ is satisfied.

$$\mu_{\tilde{R}}(x): D \rightarrow [\underline{\mu}_{\tilde{R}}(x), \overline{\mu}_{\tilde{R}}(x)] \tag{5.10}$$

By adopting this definition for requirements specification, the satisfaction degree is not a crisp value in the interval $[0,1]$ anymore, but it is a spectrum of values in $[\underline{\mu}_{\tilde{R}}(x), \overline{\mu}_{\tilde{R}}(x)]$. This provides an opportunities to accommodate the scenarios that we have discussed in 5.4.2 and specify a satisfaction function that considers different opinions of different users as shown in Figure 5.10. Figure 5.11 illustrates a type-2 fuzzy membership function that is discussed in detail in Chapter 2.

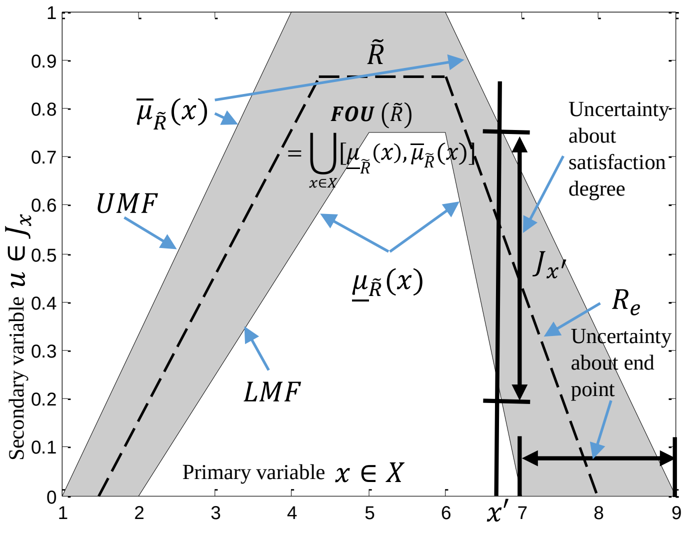


Figure 5.11. An interval type-2 fuzzy set based possibility distribution.

5.4.3.2. Measure of relationships between requirements

Four types of relationships between requirements have been introduced and defined in (X. Liu & Yen, 1996) based on type-1 fuzzy sets. Here we intend to redefine these classes of relationships with regard to interval type-2 fuzzy sets. These relationship types are: 1) conflicting, 2) cooperative, 3) mutually exclusive, 4) irrelevant. This classification is based on the relationships between satisfaction functions (cf. **Definition 20**, **Definition 21**) of the requirements involved in a specific context.

Two imprecise requirements are categorized as *conflicting* if an increase in the mean interval of the satisfaction degree (cf. **Definition 22**) decreases the mean interval of the satisfaction degree of the other (cf. Figure 5.12).

Definition 22. The ***mean interval of satisfaction function***, denoted by $\theta_{\tilde{R}}$, maps an element of $\tilde{R}$'s domain or universe of discourse D to the mean of interval $[\underline{\mu}_{\tilde{R}}(x), \overline{\mu}_{\tilde{R}}(x)]$.

$$\theta_{\tilde{R}}(x): D \to (\overline{\mu}_{\tilde{R}}(x) + \underline{\mu}_{\tilde{R}}(x))/2 \qquad (5.11)$$

On the other hand, two imprecise requirements are classified as *cooperative* if an increase in the mean interval of satisfaction degree often increases the mean interval of satisfaction degree of the other. If the mean intervals of the satisfaction degree of two imprecise requirements cannot be satisfied at the same time, they are categorized as *mutually exclusive* requirements. Finally, if the mean intervals of the satisfaction function of two requirements have no impact to each other, the requirements are called *irrelevant*.

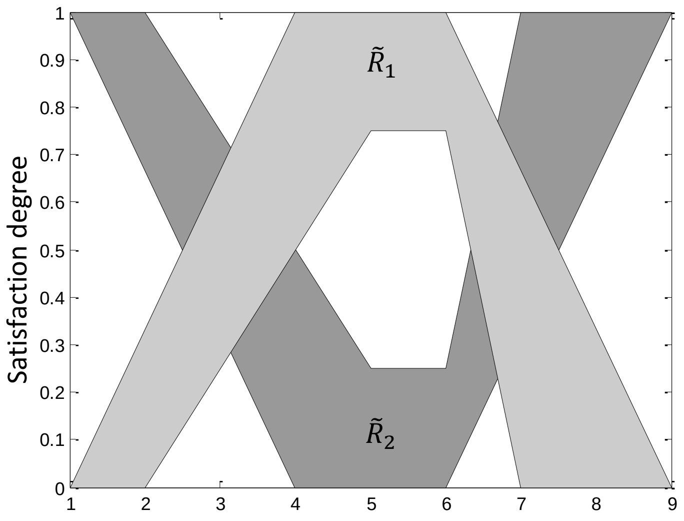


Figure 5.12. Conflicting imprecise non-functional requirements.

5.4.3.3. Non-functional requirements tradeoff analysis

In order to perform tradeoff analysis, multiple non-functional requirements need to be aggregated to formulate an overall aggregated function to be verified. As a result, we introduce and define a number of aggregation operators according to fuzzy logic to enable the tradeoff analyses. The aggregation operations are as follows: union of requirements (cf. Figure 5.13) is given in **Definition 23** and illustrated in Figure 5.14; intersection of requirements is given in **Definition 24** and illustrated in Figure 5.15; complement of requirements (cf. Figure 5.13) is given in **Definition 25**.

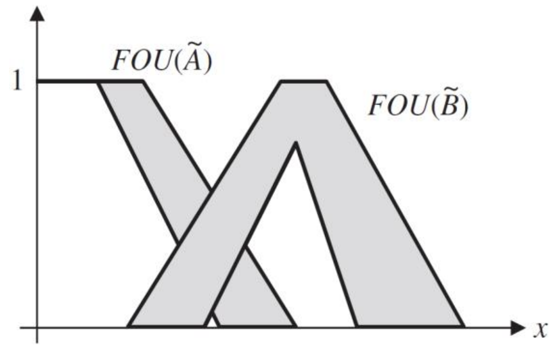


Figure 5.13. Two IT2-FSs, A and B (adapted from (J Mendel & Wu, 2010)*).*

Definition 23. The ***union*** of two imprecise requirements $\tilde{R}_1$ and $\tilde{R}_2$ is

$$\begin{gathered}\mu_{\tilde{R}_1}(x): D \to \left[\underline{\mu}_{\tilde{R}_1}(x), \overline{\mu}_{\tilde{R}_1}(x)\right] \\ \mu_{\tilde{R}_2}(x): D \to \left[\underline{\mu}_{\tilde{R}_2}(x), \overline{\mu}_{\tilde{R}_2}(x)\right] \\ \mu_{\tilde{R}_1 \cup \tilde{R}_2}(x): D \to \left[\underline{\mu}_{\tilde{R}_1}(x) \vee \underline{\mu}_{\tilde{R}_2}(x), \overline{\mu}_{\tilde{R}_1}(x) \vee \overline{\mu}_{\tilde{R}_2}(x)\right]\end{gathered} \tag{5.12}$$

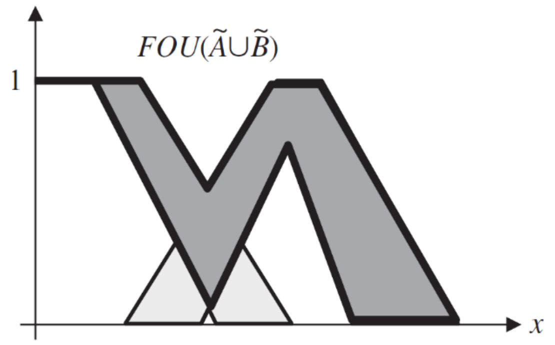


Figure 5.14. Visual representation of union of two IT2-FSs (adapted from (J Mendel & Wu, 2010)*).*

Definition 24. The ***intersection*** of two imprecise requirements $\tilde{R}_1$ and $\tilde{R}_2$ is

$$\begin{gathered}\mu_{\tilde{R}_1}(x): D \to \left[\underline{\mu}_{\tilde{R}_1}(x), \overline{\mu}_{\tilde{R}_1}(x)\right] \\ \mu_{\tilde{R}_2}(x): D \to \left[\underline{\mu}_{\tilde{R}_2}(x), \overline{\mu}_{\tilde{R}_2}(x)\right] \\ \mu_{\tilde{R}_1 \cap \tilde{R}_2}(x): D \to \left[\underline{\mu}_{\tilde{R}_1}(x) \wedge \underline{\mu}_{\tilde{R}_2}(x), \overline{\mu}_{\tilde{R}_1}(x) \wedge \overline{\mu}_{\tilde{R}_2}(x)\right]\end{gathered} \tag{5.13}$$

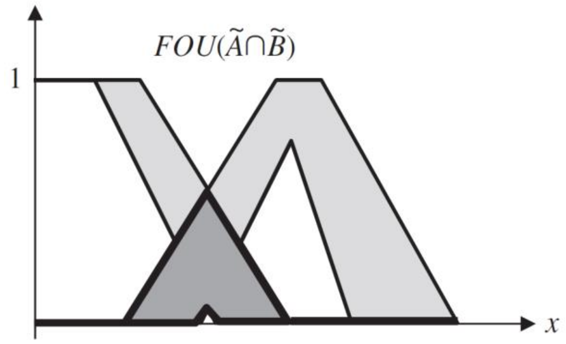


Figure 5.15. Visual representation of intersection of two IT2-FSs (adapted from (J Mendel & Wu, 2010)*).*

Definition 25. The ***complement*** of an imprecise requirement $\tilde{R}$ is

$$\mu_{\tilde{R}}(x): D \to \left[\underline{\mu}_{\tilde{R}}(x), \overline{\mu}_{\tilde{R}}(x)\right]$$
$$\mu_{\overline{\tilde{R}}}(x): D \to \left[1 - \overline{\mu}_{\tilde{R}}(x), 1 - \underline{\mu}_{\tilde{R}}(x)\right] \tag{5.14}$$

Finally, the tradeoff between requirements is defined in **Definition 26**.

Definition 26. Let $\tilde{R}$ be a list of non-functional requirements $\tilde{R} = (\tilde{R}_1, \tilde{R}_2, \dots, \tilde{R}_n)$ that a component connector is supposed to satisfy and $\mu = (\mu_{\tilde{R}_1}(x), \mu_{\tilde{R}_2}(x), \dots, \mu_{\tilde{R}_n}(x))$ be the associated satisfaction degrees. Let $W = (w_1, w_2, \dots, w_n)$ contain a list of real numbers representing the normalized relative importance of the requirements. The overall ***satisfaction degree*** is as follows:

$$U_{\tilde{R}} = \sum_{i=1}^{n} w_i \times \mu_{\tilde{R}_i}(x) \tag{5.15}$$

5.4.3.4. Non-functional requirement change analysis

We consider here three types of change in non-functional requirements and their analysis with type-2 fuzzy sets: 1) relaxing a requirement, 2) strengthening a requirement, and 3) changing the priority of a requirement.

Definition 27. The requirement $\tilde{R}_1$ is considered to be ***relaxed*** to $\tilde{R}_2$ and $\tilde{R}_2$ is considered to be ***strengthened*** to $\tilde{R}_1$ if:

$$\forall x \in D, \theta_{\tilde{R}_1}(x) \leq \theta_{\tilde{R}_2}(x) \tag{5.16}$$

Definition 28. Let $\tilde{R}$ be a requirement, then the feasibility of $\tilde{R}$ in domain D can be defined as

$$Feasibility\left(\tilde{R}\right) = sup_{x \in D} \underline{\mu}_{\tilde{R}}(x) \tag{5.17}$$

Intuitively, according to **Definition 28**, relaxing a requirement improves the feasibility of the system realizing the requirement.

Theorem 2. Let $\tilde{R} = \tilde{R}_1 \otimes \dots \otimes \tilde{R}_i \otimes \dots \otimes \tilde{R}_n$ and $\widetilde{R'} = \tilde{R}_1 \otimes \dots \otimes \widetilde{R'}_i \otimes \dots \otimes \tilde{R}_n$ and assume that $\tilde{R}_i$ is relaxed to $\tilde{R'}_i$. Then,

$$Feasibility\left(\tilde{\mathrm{R}}_{\mathrm{i}}\right) \leq Feasibility\left(\tilde{\mathrm{R}}'_{\mathrm{i}}\right) \tag{5.18}$$

In the next section, we introduce the main outcome of this chapter, i.e., the adaptation reasoning framework.

5.5. RobusT2: A Framework for Autonomous Adaptation Reasoning using Type-2 Fuzzy Logic Systems

In this section, we develop a framework, called RobusT2, to reason about adaptation of component connectors, in which adaptation rules are based on a data collection from a group of users who have potentially conflicting opinions about adaptation policies. As we discussed in earlier sections, we chose to develop a fuzzy controller to give software architects more flexibility to accommodate their preferences even when they are conflicting.

The overall view of the autonomous controller for adaptation reasoning, as the main artifact of the RobusT2 framework, is shown in Figure 5.34. As illustrated, the controller covers both design-time and runtime. During design-time, the aim is to design a fuzzy controller, specify its rule-base, and derive appropriate satisfaction functions. At runtime, while the controller starts operating for connector self-adaptation, it keeps monitoring quality and environmental data that may affect non-functional requirement satisfaction. The requirements are continuously verified with respect to runtime data that may reflect changes in the environment's behavior. In the case of detection of any violations, appropriate adaptation actions (in terms of mode changes here) are generated through the fuzzy logic controller and applied via an execution actuator. More specifically, the controller adjusts the system configuration with respect to runtime data that may affect changes in the connector's behavior. The key mechanism for decision-making at runtime is the fuzzy inference process.

In the following, we discuss each phase in turn and describe the relevant activities. Section 5.5.1 presents basic concepts and phenomena in fuzzy inference and the main entities and processes involved in fuzzy reasoning. We provide concrete examples, in this section, to enable readers to easily grasp the involved intricacies in fuzzy reasoning and prepare them to fully understand the outcome of this chapter, which is the proposed method for designing the fuzzy controller. Section 5.5.2 introduces a running example. Section 5.5.3 provides concrete challenges that motivated us to pursue such solution. Section 5.5.4 gives a high-level overview of the proposed autonomous reasoning providing an abstract overview of the approach. Section 5.5.5 proposes our knowledge elicitation approach for enabling adaptation rule elicitation from different users. Section 5.5.6 proposes a technique for transforming the collected data to design a fuzzy logic controller that acts as the main outcome of this chapter. Section 5.5.7 reviews the benefits of the designed type-2 fuzzy logic controller over traditional type-1 controllers. Section 5.5.8 contains experimental evaluation results regarding the designed and implemented controller. Finally, Section 5.5.9 discusses the significance of the main results.

5.5.1. Fuzzy logic systems and uncertainty control

From a software engineering perspective, fuzzy logic can be interpreted as a theory that allows using linguistic words and human knowledge 1) to represent or model adaptation knowledge and 2) to design their reasoning mechanisms. Fuzzy control has been used in different application areas such as industrial control, mobile robots control and ambient intelligent environments control (Hagras, 2007).

The human brain reasons based on linguistics such as *slow*, *fast*, *near* or *far* and it execute control actions accordingly. Therefore, human activities exemplify the concept of fuzzy control. For instance, people do not need to measure acceleration to be able to safely control the car they are driving.

5.5.1.1. The Concept of Uncertainty in Fuzzy Logic Systems

Uncertainty exists in any situation with a lack of knowledge. For example, knowledge may be *incomplete*, *imprecise*, *noisy*, *patchy*, *not reliable*, *vague*, *contradictory* or *deficient* (Klir & Yuan, 1995). Different authors define and classify different types of uncertainty. The classification proposed in (Esfahani & Malek, 2013) fits to describe and interpret the effects of uncertainty in self-adaptive software. There are some other classification of uncertainty in this domain such as (A. J. Ramirez et al., 2012).

Uncertainty is also exists in fuzzy logic systems as explained in (JM Mendel, 2001; Wu, 2012):

- Uncertainty about the meaning of the words that are used in the rules used for reasoning.
- Uncertainty about the consequence of the rules.
- Uncertainty about the input data that activate the fuzzy logic systems.
- Uncertainty about the data used to tune the design parameters of fuzzy logic system.

As we discussed in the background chapter, T1 MFs are precise and, as a result, T1 FSs as used in T1-FLS cannot capture the uncertainty. This is the reason why Zadeh proposed to represent this uncertainty by using T2 FSs (Zadeh, 1975). T2 FLSs are to some extent different from classical fuzzy logic systems, see the difference in Figure 5.16 and Figure 2.3. The next sections presents these differences by introducing the subsystems of a T2 FLS as presented in Figure 2.3.

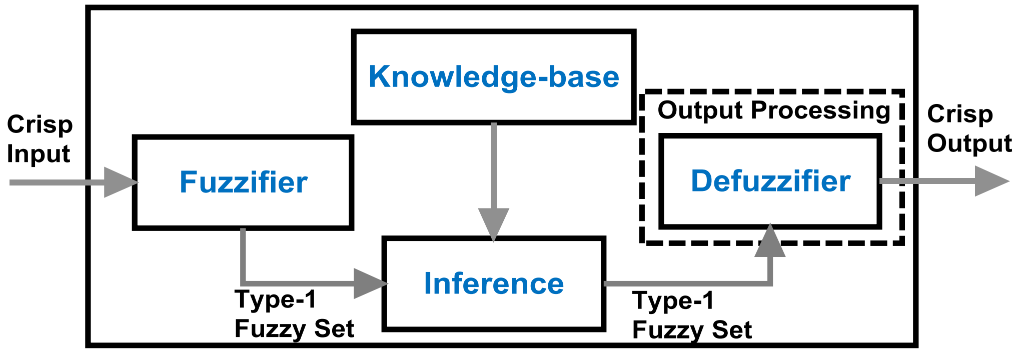


Figure 5.16. The architecture of type-1 fuzzy logic system (adapted from (JM Mendel, 2000)).

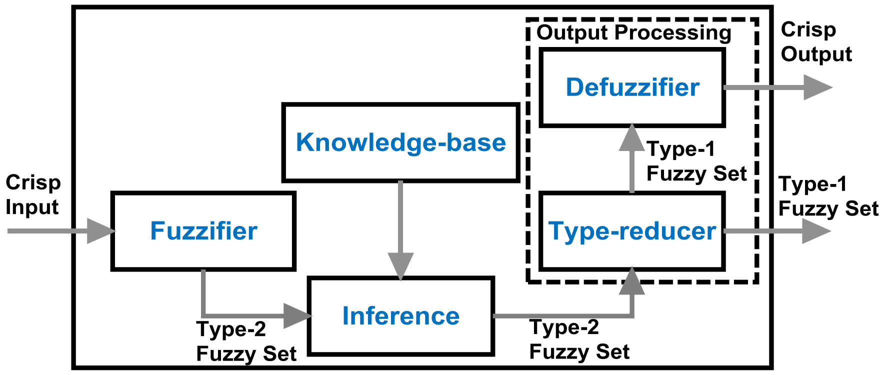


Figure 5.17. The architecture of type-2 fuzzy logic system (adapted from (JM Mendel, 2000)).

5.5.1.2. Knowledge base (Rule base)

The knowledge base (or in some literature it is known as rule base) allows the representation of human knowledge by using linguistic rules. A fuzzy rule is specified with the structure below:

$$\text{IF (some conditions are satisfied) THEN (perform a control action)} \tag{5.19}$$

In general, the fuzzy rules are organized using tables whose objective is to represent all the different combinations of the inputs of the system. The structure of the rule-base are the same in both type-1 and type-2 fuzzy logic systems, except that in the former the linguistics have type-1 MFs, while in the latter the linguistics have type-2 MFs.

5.5.1.3. Membership functions

Membership functions (MFs) enable forming a connection between crisp values and linguistic words. Type-1 fuzzy MFs (T1-MF) are two-dimensional and characterize the membership value μ for a variable $x \in X$. Type-2 fuzzy MFs (T2-MF) are three-dimensional by considering an uncertainty u of the membership value. T1-MFs are a special case of T2-MFs where the uncertainty value is zero. In general, membership functions can be classified as:

- *Singleton MFs*. A membership function that is unity at one particular point and zero everywhere else. See Figure 5.18.

$$\mu = \begin{cases} 1 & x = x_1 \\ 0 & otherwise \end{cases} \tag{5.20}$$

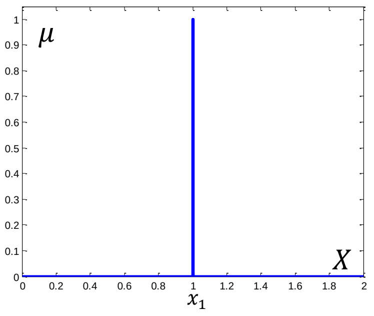


Figure 5.18. Singleton membership function.

- *Interval type-1 MFs*. A membership function that is zero except in the interval defined by its left and right bounds. See Figure 5.19.

$$\mu = \begin{cases} 1 \; l_a \le x \le l_b \\ 0 \; otherwise \end{cases} \tag{5.21}$$

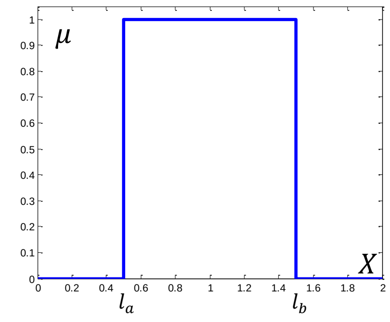


Figure 5.19. Interval type-1 membership function.

- *Type-1 MFs*. A membership function that is a crisp value which vary in the interval $[0,1]$. See Figure 5.2 to Figure 5.8.
- *Type-2 MFs*. A membership function that is characterized by a secondary degree MF $\mu_{\tilde{R}}$. This type of MF can further classified as:
    1. *Interval type-2 MFs.* If $\mu_{\tilde{R}}(x,u)$ is an interval type-1 MF.
    2. *General type-2 MFs.* If $\mu_{\tilde{R}}(x,u)$ is a type-1 MF.

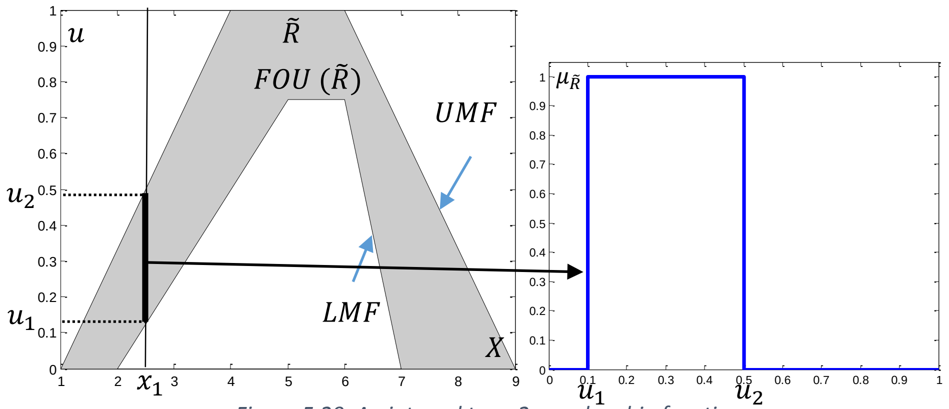


Figure 5.20. An interval type-2 membership function.

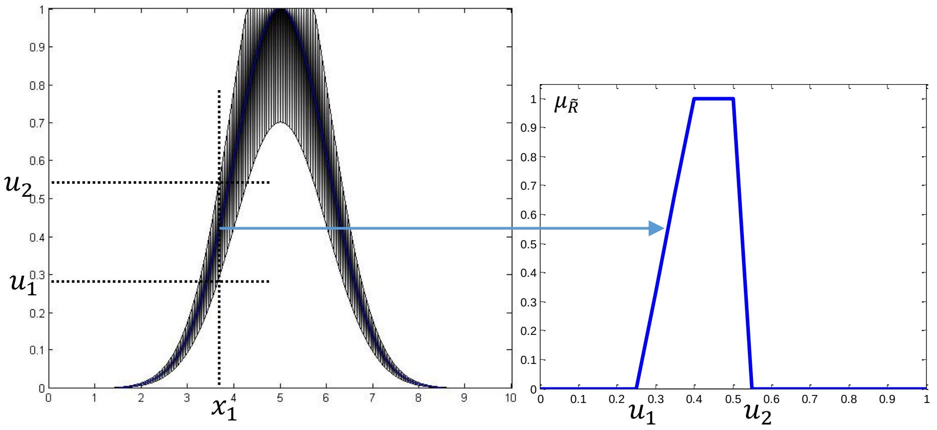


Figure 5.21. General type-2 membership function.

5.5.1.3.1. Interval type-2 membership functions

General T2 MFs are complicated to implement and the computational overhead of processing based on this type of MF is high (N. Karnik & Mendel, 1999) and obviously not appropriate for time-constrained applications such as adaptation reasoning in a self-adaptive control loop. However, interval T2 MFs are straightforward to implement and have been used in almost all works about type-2 fuzzy logic (Hagras, 2007). Therefore, an interval type-2 membership function has been also selected for this research.

An IT2 MF can be created with two T1 MFs. An Upper Membership Function (UMF), which represents the maximum value and a Lower Membership Function (LMF), which represents the minimum value of μ for each x. The uncertainty u is represented by the area between the UMF and the LMF. This region is called Footprint of Uncertainty (FOU) and is illustrated in Figure 2.2. Note that T1 MFs are a particular case of T2-MFs that does not consider the uncertainty; the same MF represents both the UMF and LMF and, therefore, the area of the FOU is zero.

5.5.1.3.2. Membership function creation

In this thesis, for representing the IT2 MFs, trapezoidal and triangular membership functions are used to construct the FOU. IT2 MFs are completely described by 9 points $(a, b, c, d, e, f, g, i, h)$, see Figure 5.22. Note that triangular MFs are a particular case of trapezoidal MFs where the two middle points coincide. For example, in Figure 5.22, if $f = g$ then the IT2 MF $\tilde{X}$ is a trapezoidal MF with trapezoidal UMF and triangular LMF. If both $b = c \ and \ f = g$ then the MF is triangular. An special case of triangular MF, all the middle pints could meet, $b = c = f = g$.

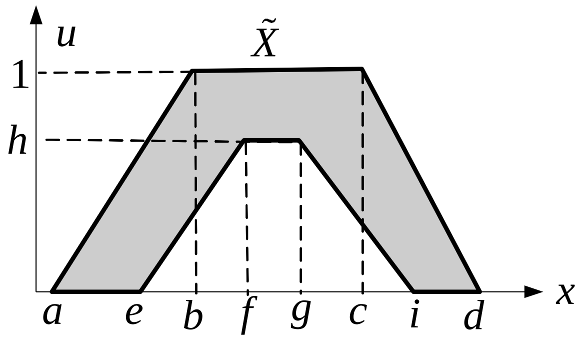


Figure 5.22. The nine points that represent an IT2 FS (adapted from (J Mendel & Wu, 2010)).

5.5.1.3.3. The Concept of centroid of an interval type-2 MF

An IT2 MF can be approximated with a set of N IT1 MFs located at the points x_i and with upper and lower bounds μ_{UMF} and μ_{LMF} as illustrated on Figure 5.23. Then, the centroid of the IT2 MF is calculated as the centroid of the N IT1 MFs. Note that the accuracy of the calculation depends on N.

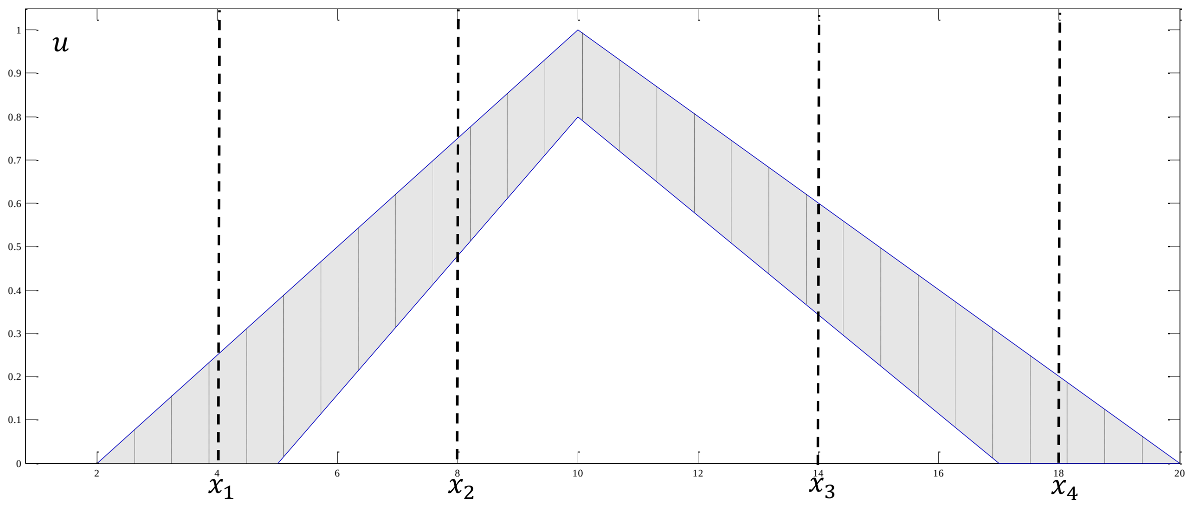


Figure 5.23. A discretized IT2 MF.

The centroid of a T1 MF, R, discretized in N points is located at $x = c$. This point is defined in **Definition 29**.

Definition 29. The centroid of a type-1 fuzzy set R is defined as:

$$c(R) = \frac{\sum_{i=1}^{N} x_i \mu_R(x_i)}{\sum_{i=1}^{N} \mu_R(x_i)} \tag{5.22}$$

Similarly, the centroid of an IT2 MF discretized in N intervals is located at the interval $[c_l, c_r]$. This interval is defined in **Definition 30**.

Definition 30. The centroid of a type-2 fuzzy set $\tilde{R}$ is the union of the centroids of all its embedded type-1 fuzzy sets R_e:

$$\begin{aligned} C_{\tilde{R}} &\equiv \bigcup_{\forall R_e} c(R_e) = \left[c_l(\tilde{R}), c_r(\tilde{R})\right] \\ c_l(\tilde{R}) &= \min_{\forall R_e} c(R_e) \\ c_r(\tilde{R}) &= \max_{\forall R_e} c(R_e) \end{aligned} \tag{5.23}$$

In order to exemplify this concept, the centroid of the IT2 MF in Figure 5.23 is presented. Note that we consider a 4 points discretization of the IT2 MF for simplicity in this example, but a different discretization has been used in our experimental evaluation. The IT2-MF is discretized into the following 4 IT1 MFs:

$$\begin{aligned} x_1 &= 4, \mu = [0,0.25] \\ x_2 &= 8, \mu = [0.48,0.75] \\ x_3 &= 14, \mu = [0.3429,0.6] \\ x_4 &= 18, \mu = [0,0.2] \end{aligned} \tag{5.24}$$

Table 5.2 summarizes all the possible weighted averages defined in Equation (5.22). Since the discretization number is 4 and there are two boundaries for each MF in this example, there are 16 (2^N) weighted averages. Note that the centroid of the IT2 MF in Figure 5.23 can be approximated by finding the minimum and maximum values of the weighted average, c, in the last column in Table 5.2 as defined in Equation (5.23). Therefore, $c = [\min(c), \min(c)] = [8.7874, 12.3750]$. Note that this is only an approximation for the centroid of the IT2 MF and in order to find a better approximation, we need to increase the discretization number.

Table 5.2. Weighted averages of the interval T1 MF.

$\mu(x_1)$	$\mu(x_2)$	$\mu(x_3)$	$\mu(x_4)$	c
0	0.48	0.3429	0	10.5002
0	0.48	0.3429	0.2	11.9666
0	0.48	0.6	0	11.3333
0	0.48	0.6	0.2	12.3750
0	0.75	0.3429	0	9.8825
0	0.75	0.3429	0.2	11.1382
0	0.75	0.6	0	10.6667
0	0.75	0.6	0.2	11.6129
0.25	0.48	0.3429	0	8.9856
0.25	0.48	0.3429	0.2	10.4019
0.25	0.48	0.6	0	9.9549
0.25	0.48	0.6	0.2	11.0065
0.25	0.75	0.3429	0	8.7874
0.25	0.75	0.3429	0.2	9.9816
0.25	0.75	0.6	0	9.6250
0.25	0.75	0.6	0.2	10.5556

However, once the value of N is increased to find better approximations of the centroid, the number of weighted averages grow exponentially, and the computational time will then become unsuitable for the self-adaptation feedback loop application. Karnik and Mendel proposed an iterative algorithm to find the lower and upper bounds of centroid. This algorithm, called KM (N. N. Karnik & Mendel, 2001), dramatically reduces the number of iterations to find the solutions. The KM algorithm is further enhanced in (JM Mendel, 2009).

Table 5.3 presents the results of the centroid of the IT2 MF in Figure 5.23 calculated using the KM algorithm with different values of discretization, N. The number of iterations to find the value of the centroid with respect to a naïve calculation as in Table 5.2, the KM algorithm and its enhanced version. As it is evident in this table, the enhanced KM algorithm enables the efficient calculation of the centroid even with a large N. As a result, the enhanced version of the KM algorithm is adopted here to calculate the centroid of IT2 MF used in the IT2 FLS for adaptation reasoning. For details of the KM algorithm, we refer to (N. N. Karnik & Mendel, 2001; JM Mendel, 2009).

Table 5.3. Computational complexity of centroid calculation for our IT2 MF example.

N	$C_{\tilde{R}}$	KM iterations	Enhanced KM (EKM) iterations	2^N iterations
4	[9.8824,11.3333]	4	1	16
16	[9.4114,11.9426]	6	1	65536
100	[9.3870,11.9615]	6	2	$1.2677e+30$
256	[9.3865,11.9623]	7	2	$1.1579e+77$
1024	[9.3864,11.9623]	8	3	$> 8.9885e+307$

5.5.1.4. Fuzzifier

The functionality of the fuzzifier in IT2 FLS (cf. Figure 2.3) is to map a crisp input $(x_1, x_2 \ldots, x_n) \in X_1 \times X_2 \ldots \times X_n$ into their corresponding IT2 MFs to produce a set of IT1 FSs. This mapping is needed to activate rules that are specified in terms of linguistic words. The inputs to the FLS prior to fuzzification module (cf. Figure 5.16 and Figure 2.3) may be certain (e.g., perfect measurement and noise free) or uncertain (e.g., noisy measurements). IT2 FLSs can handle either kind of measurement (JM Mendel, 2007). Note that the number of sets depends on the number of inputs and the number of MFs. First, we must state how the numeric inputs $u_i \in U_i$ are converted to fuzzy sets (with a process called "fuzzification" (Jerry M. Mendel et al., 2006)) so that they can be used by the FLS, see the input-output of the fuzzifier module in Figure 2.3. A fuzzification can be defined by a transformation $F: U_i \to U_i^*$, where $F(u_i) = \tilde{R}_i$ and U_i^* is a set of all FSs that can be defined on U_i. In this thesis, we use singleton:

$$\mu_{\tilde{R}_i} = \begin{cases} 1 & x = u_i \\ 0 & otherwise \end{cases} \tag{5.25}$$

In order to show the methodology to implement the subsystems of an IT2 FLS, we use a concrete example here. We present the whole process of an IT2 FLS step by step through this example. This process can be viewed as a mapping from crisp inputs to crisps output (cf. the solid path in Figure 2.3): from fuzzification, all the way down to the defuzzification. Note that this mapping can be delineated as $y = f(x)$.

Let us consider a normalized crisp input $(x_1, x_2) = (40,50) \in X_1 \times X_2$ of the IT2 FLS whose MFs with respect to the two elements of the input are illustrated in Figure 5.24 (corresponds to X_1) and Figure 5.25 (corresponds to X_2). Figure 5.24 illustrates the MFs of the first input (i.e., x_1), and Figure 5.25 shows the MFs of the second input (i.e. x_2).

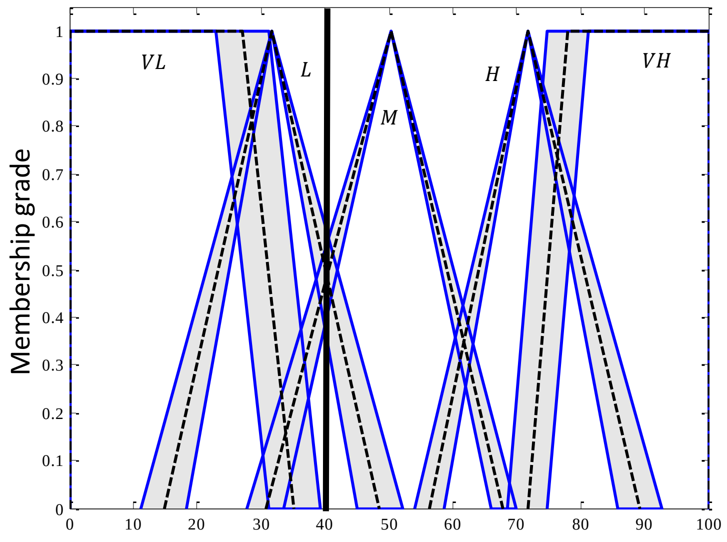


Figure 5.24. IT2 MFs for input x_1 (workload).

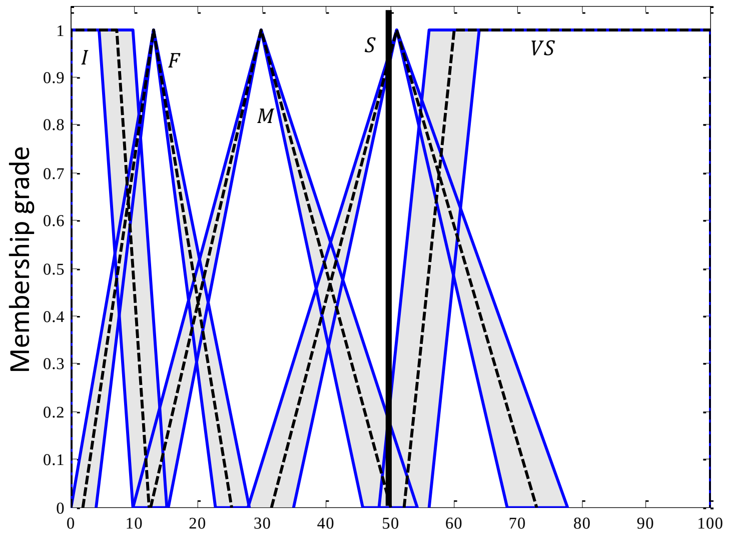


Figure 5.25. IT2 MFs for input x_2 (response time).

First, the input needs to be fuzzified into the MFs. It can be seen in Figure 5.24 and Figure 5.25 that the first element of the input (i.e., x_1) is fuzzified into two MFs (i.e., L, M) and the second element of the input (i.e., x_2) is fuzzified into three MFs (i.e., M, S, VS). Figure 5.26 illustrates the non-null fuzzified sets regarding the former element of the input (i.e., x_1) and Figure 5.27 shows the non-null fuzzified sets with respect to the latter element (i.e., x_2).

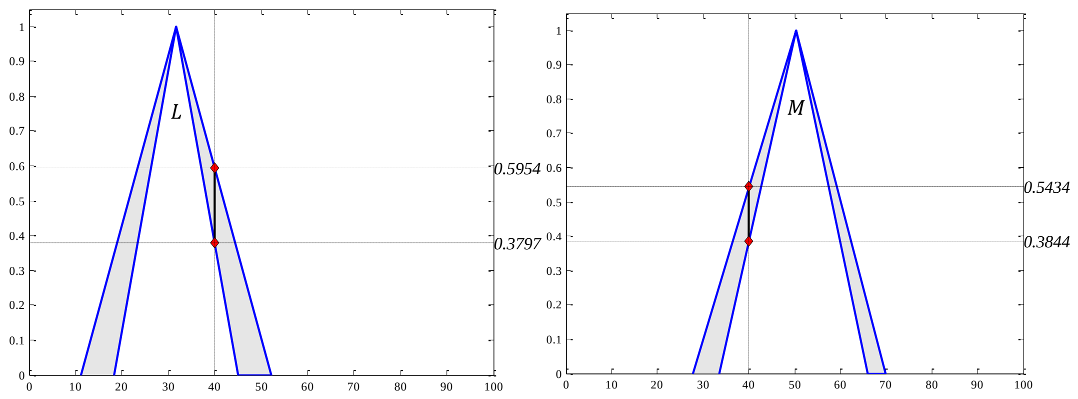


Figure 5.26. Non-null fuzzified sets for x_1.

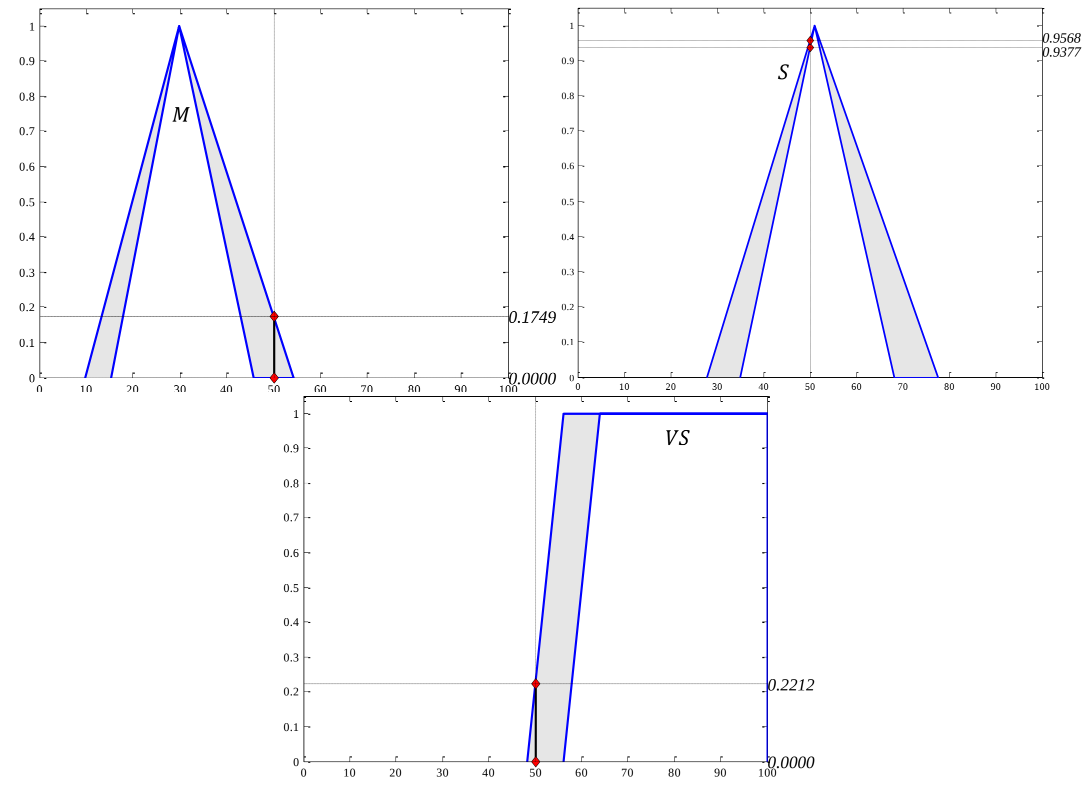


Figure 5.27. Non-null fuzzified sets for x_2.

Figure 5.28 illustrates a black box representation of the fuzzifier module in Figure 2.3. As discussed earlier, the fuzzifier functionality is to transform crisp inputs to fuzzy output in order to use them in the inference process. In the inference process, these fuzzified sets are used to trigger appropriate rules in the rule base. We describe the inference process in the next section.

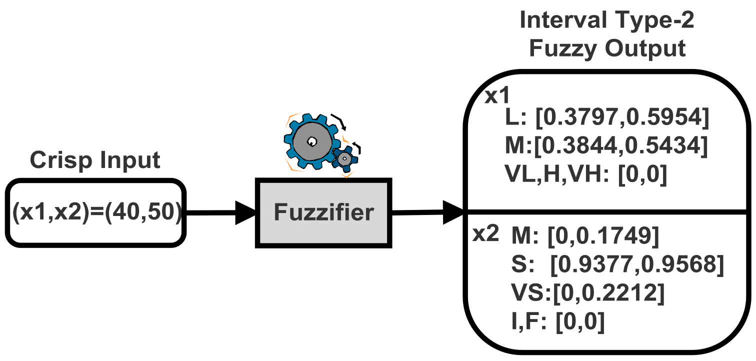


Figure 5.28. The fuzzifier module: maps crisps inputs into interval type-2 fuzzy sets outputs.

5.5.1.5. Inference Engine

The functionality of the inference engine module as in Figure 2.3 is to map the set of IT1 MFs (resulting as output from the fuzzifier) into the consequents of the fired rules. Consequently, the output is a set of IT2 MFs. Note that the number of output MFs is equal to the number of fired rules. Let us continue with our running example. The complete list of rules is summarized in Table 5.4. Note that in this example, the antecedents represent different situations that may happen at runtime and consequent of the rules determine the control action. In the evaluation section, however, we deal with a set of rules with interval type-1 fuzzy output. In this example, as discussed in the previous section, according to the provided input, $(x_1, x_2) = (40{,}50)$, two non-null fuzzified sets for the antecedent x_1 and three non-null fuzzified sets for the antecedent x_2 are derived, see Figure 5.28. As a result, 6 rules are fired (or activated), see highlighted rows in Table 5.4.

Table 5.4. Fuzzy rules with singleton consequent.

Rule (l)	**Antecedents**		**Consequent**
	Workload (x_1)	**Response time (x_2)**	**Nodes (y)**
1	Very low	Instantaneous	-1.6
2	Very low	Fast	-1.4
3	Very low	Medium	0
4	Very low	Slow	0.6
5	Very low	Very slow	1.4
6	Low	Instantaneous	-1.3
7	Low	Fast	-1.1
8	Low	Medium	0.4
9	Low	Slow	1
10	Low	Very slow	1.6
11	Medium	Instantaneous	-1.6
12	Medium	Fast	-0.9
13	Medium	Medium	0.6
14	Medium	Slow	1.1
15	Medium	Very slow	1.5
16	High	Instantaneous	-1.8
17	High	Fast	-1.4
18	High	Medium	0.4
19	High	Slow	1.1
20	High	Very slow	1.4
21	Very high	Instantaneous	-1.9
22	Very high	Fast	-1.2
23	Very high	Medium	0.5
24	Very high	Slow	1
25	Very high	Very slow	1.6

Figure 5.29 illustrates the inference process for the Rule #9. Here, y represents the rule output and F^9 represents its firing value. Figure 5.30 shows a black box representation of the inference module in Figure 2.3 with respect to the 6 activated rules. The functionality of the inference module is to transform the fuzzified inputs to the fuzzy output. However, this output needs to be processed in order to produce a crisp output to be used in the feedback control loop. We describe the output processing in the next section.

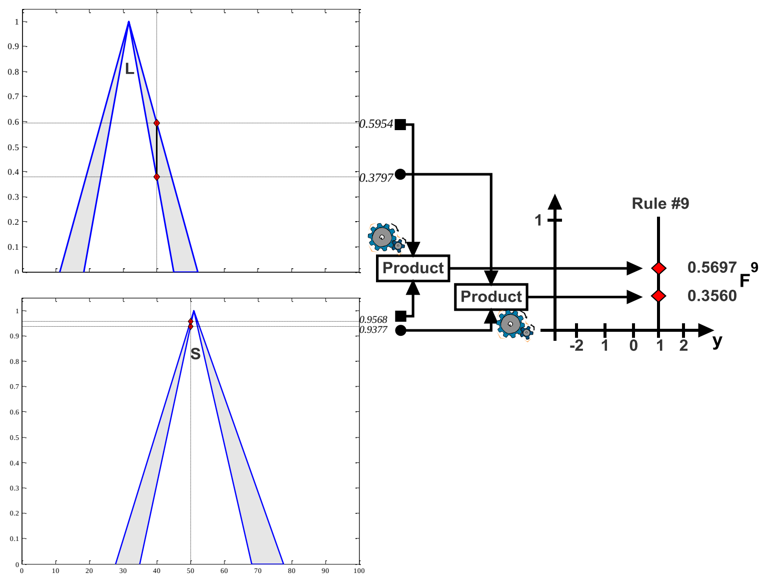
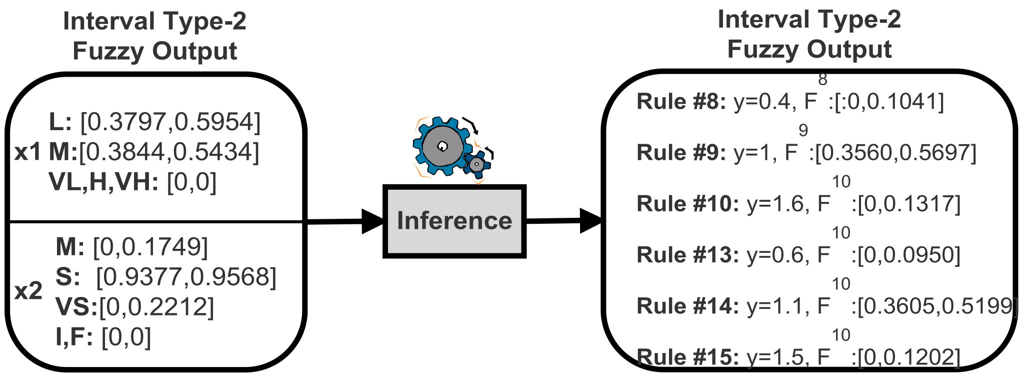


Figure 5.29. The inference engine: calculation of the firing degree for Rule #9 (inference operation: product).



Interval Type-2 Fuzzy Output
x1
L: [0.3797,0.5954]
M:[0.3844,0.5434]
VL,H,VH: [0,0]
x2
M: [0,0.1749]
S: [0.9377,0.9568]
VS:[0,0.2212]
I,F: [0,0]
Inference
Interval Type-2 Fuzzy Output
Rule #8: y=0.4, F^8:[:0,0.1041]
Rule #9: y=1, F^9:[0.3560,0.5697]
Rule #10: y=1.6, F^10:[0,0.1317]
Rule #13: y=0.6, F^10:[0,0.0950]
Rule #14: y=1.1, F^10:[0.3605,0.5199]
Rule #15: y=1.5, F^10:[0,0.1202]

Figure 5.30. The Inference engine module: maps IT2 FSs inputs into IT2 FSs outputs.

5.5.1.6. Output Processor

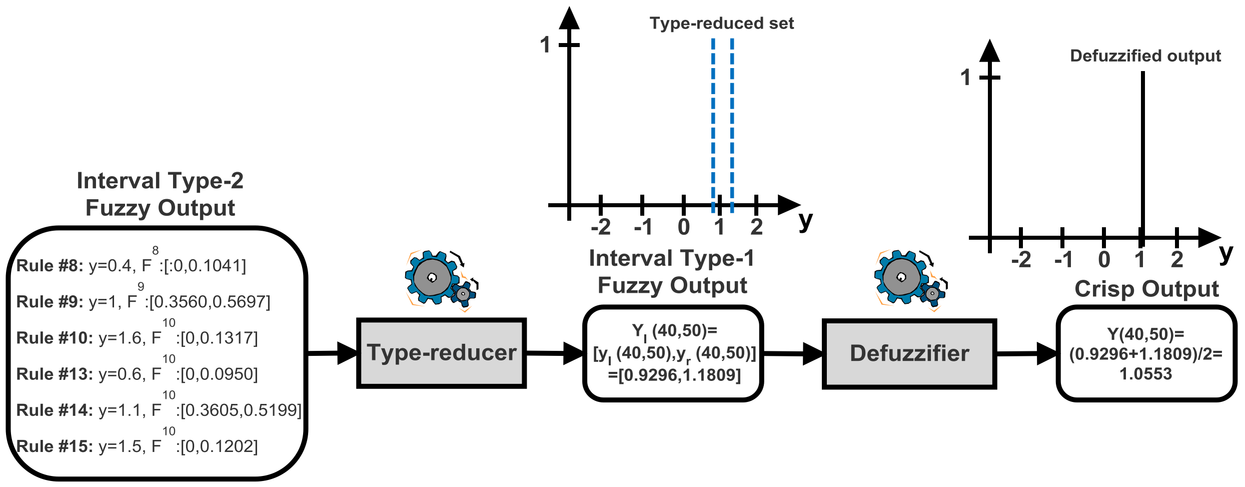
The output-processing module aggregates the IT2 FSs of the fired rules to obtain the crisp output of the T2 FLS. The output-processing module is the main difference between T1 FLSs and T2 FLSs. According to Figure 2.3, the output-processing module is divided into the type-reducer and the defuzzifier sub-modules. However, in T2 FLSs, there is no type-reducer (cf. Figure 5.16) since the output of the inference is of T1 FSs and they only need to be defuzzified.

5.5.1.6.1. Type-reducer

The type-reducer aggregates IT2 FSs into an IT1 FS called the type-reduced set (note the transition between type-2 to type-1). The number of input fuzzy sets matches the number of fired rules, while there is only one output- the type-reduced fuzzy MF. This MF is calculated using the KM algorithm (N. N. Karnik & Mendel, 2001), see Section 5.5.1.3.3. More specifically, the inputs to the algorithm are the fuzzy sets F^l and y^l which are the output of the inference engine. Note that l is the index of the active rules.

5.5.1.6.2. Defuzzifier

Since the processes (in this research they are component connectors) that are under the control of the FLSs can only be controlled with crisp numbers, the output of the FLSs are required to provide crisp numbers. Since the output of the type-reducer is still fuzzy sets (i.e., interval T1 FS), we need another module to transform this fuzzy set to a crisp output. The defuzzifier transforms the type-reduced fuzzy set into a crisp output. It is the simplest subsystem of the FLS in terms of complexity of computation; the crisp output value is calculated as the average of the upper and lower bounds of the type-reduced set.



Interval Type-2 Fuzzy Output
Rule #8: y=0.4, F⁸:[:0,0.1041]
Rule #9: y=1, F⁹:[0.3560,0.5697]
Rule #10: y=1.6, F¹⁰:[0,0.1317]
Rule #13: y=0.6, F¹⁰:[0,0.0950]
Rule #14: y=1.1, F¹⁰:[0.3605,0.5199]
Rule #15: y=1.5, F¹⁰:[0,0.1202]
Type-reducer
Type-reduced set
1
-2 -1 0 1 2 y
Interval Type-1 Fuzzy Output
Y_l (40,50)= [y_l (40,50),y_r (40,50)] =[0.9296,1.1809]
Defuzzifier
Defuzzified output
1
-2 -1 0 1 2 y
Crisp Output
Y(40,50)= (0.9296+1.1809)/2= 1.0553

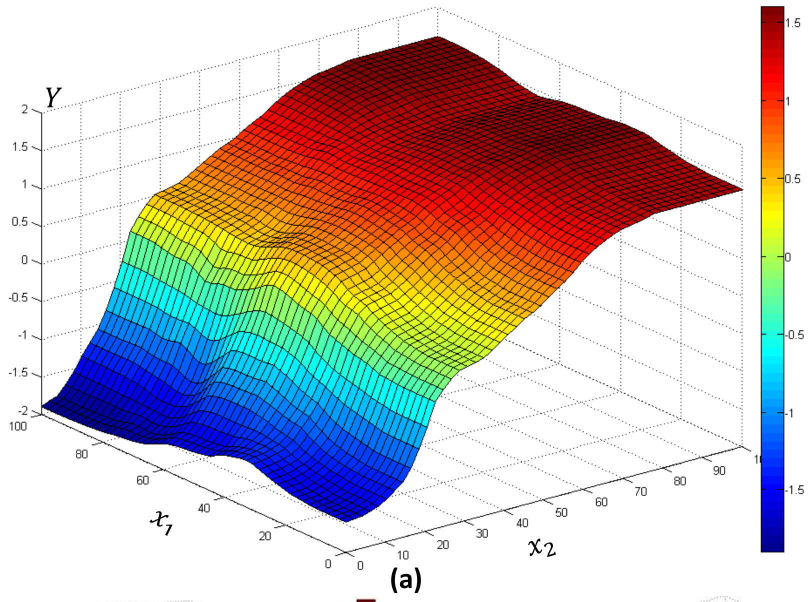
Figure 5.31. The output processing: aggregate interval type-2 fuzzy sets and transforms them into a crisp output.

5.5.1.7. Fuzzy logic control surfaces

Fuzzy logic systems are entirely defined by their fuzzy rules and their corresponding membership functions. The normalized control surfaces regarding the running example is presented in Figure 5.32. Note that this surface shows the output of the following equation:

$$[Y, y_l, y_r] = f(x) \tag{5.26}$$

, for all inputs x throughout the domain of input fuzzy sets.



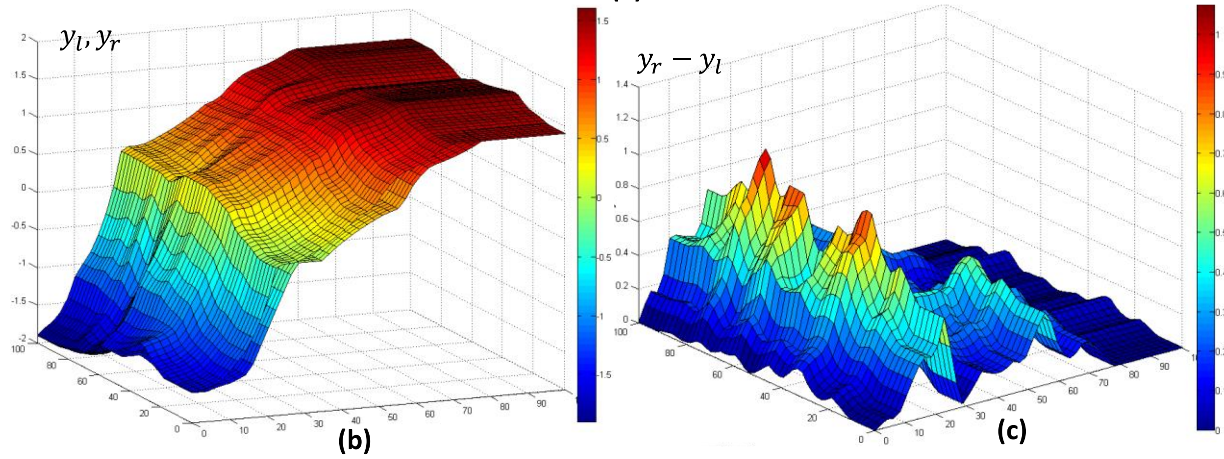


Figure 5.32. Output control surface (a), confidence interval (i.e., y_l, y_r) (b) and their differences (i.e., $y_r - y_l$) (c).

5.5.1.8. *Benefits of Using IT2 FLS over T1 FLS*

As shown in Section 5.5.1.7, the output of IT2 FLS is a boundary and a crisp number rather than a hard-threshold as in T1 FLS (Wu, 2012). Therefore, the control action as the output of the FLS can be more flexible providing a boundary, see the dashed lines in Figure 5.31. For instance, if the system requires a high performance, the decision can be made based on the upper boundary, i.e. $\lceil y_r(40{,}50) = 1.1809 \rceil$. Alternatively, if the system let say requires saving cost, the decision can be made based on the lower boundary, i.e. $\lfloor y_l(40{,}50) = 0.9296 \rfloor$. In addition, if the system needs to achieve a compromise in user experience and cost, the decision can be made based on any value in the boundary. This flexibility and the ability to handle conflicting rules are the key benefits of IT2 FLSs over T1 counterparts that motivated us to choose them for this research.

5.5.2. Running example

We use a running example to highlight the research challenges. Let us consider a Web server (F Chauvel et al., 2010) built from the following components: a listener component (L) reads HTTP requests at a port and transmits them to a data server component (DS) that returns the corresponding HTTP responses; A cache component (C) reduces the response time by caching resolved requests; A filter component (F) detects harmful requests (e.g. SQL injections); and a dispatcher component (D) enables the combination of several data servers. In Figure 5.33, four possible architectural configurations (also known as modes (Hirsch, Kramer, Magee, & Uchitel, 2006)) of the Web server (i.e., *Idle*, *Normal*, *Effort*, and *Best Effort*) are selected to illustrate this mode switching as a result of the tradeoffs between environmental conditions (e.g., request load) and system quality (e.g. performance index). The *Idle* mode as a default mode only includes one listener and one data server to handle the low workload (i.e., $w < L$). As soon as the workload increases to a certain limit (i.e., $L < w < M$), the system switches to the *Normal* mode. If the workload increases even more (i.e. $M < w < H$), the system will switch to the *Effort* mode, where two data servers are both cached and filtered. For heavy loads of request (i.e. $w > H$), the system will switch to the *Best Effort*, where a group of three data servers is cached and filtered.

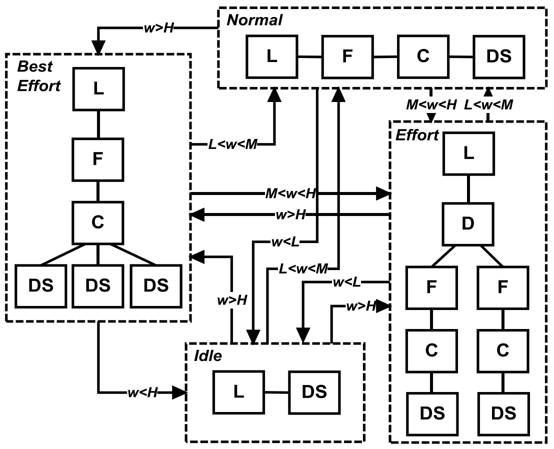


Figure 5.33. Architectural mode switching in Web server (adapted from (F Chauvel et al., 2010)).

5.5.3. Research challenges

As shown in Figure 5.33, environmental conditions (i.e., w) or quality index of the system govern adaptations of the system at runtime. The adaptation logic utilizes some policies (also known as adaptation strategies (S.-W. Cheng & Garlan, 2012), actions (Sykes et al., 2008) and rules (Batista et al., 2005)) to reason about architectural mode switching. These policies are specified in terms of event-condition-action (ECA) rules (as in (Batista et al., 2005; Fleurey & Solberg, 2009; D Garlan et al., 2004; Sykes et al., 2008)). However, these rules are subject to different uncertainties, which makes the adaptation analysis error prone. For instance, a "high" workload (cf. Figure 5.33) could mean one range of values to one person, though possibly a very different range of values to someone else, and this can vary over time. More precisely:

- *Challenge 1*. Different stakeholders often recommend different adaptation policies to the same condition resulting in rules having the same antecedents, but different consequents. As a result, rule application leads to uncertain consequents.
- *Challenge 2*. Qualitative values mean different things to different people (e.g., L, M, H). If we ask users about the parameters of the membership function (e.g., center, spread) representing the imprecise values, we are likely to get different answers. This leads to uncertain antecedents and consequents.

The key challenge with respect to the above approach is the *ignorance of uncertainty in the adaptation reasoning process*. The uncertainty in the "*adaptation rules*" and "*membership functions*" challenges the system ability in making right adaptation decisions. The latter challenge is described, in detail, in the next section.

5.5.4. Overview of autonomous adaptation reasoning

In this thesis, the autonomous reasoning process, discussed in Section 5.2.2, is realized using IT2 FLS. The background chapter (i.e., Chapter 2) contains a more detailed description of IT2 FLSs. Figure 5.34 shows a self-adaptive software within which the reasoning modules are replaced with an IT2 FLS. The reference model we borrowed is FORMS (Danny Weyns et al., 2010). Based on this model, the *base-level* software system is under the control of the meta-level reasoner. In this thesis, we consider *component connectors* that are adapted by a *mode-switching* mechanism (see Chapter 6). In the meta-level, we realized the IBM reference model for autonomic systems called MAPE-K (JO Kephart & Chess, 2003). As depicted in Figure 5.34, *users* specify the adaptation logic in the form of if-then rules and the *environment* comprises the components that interact with the connector. The details of the autonomous reasoning are described in Section 5.2.2.

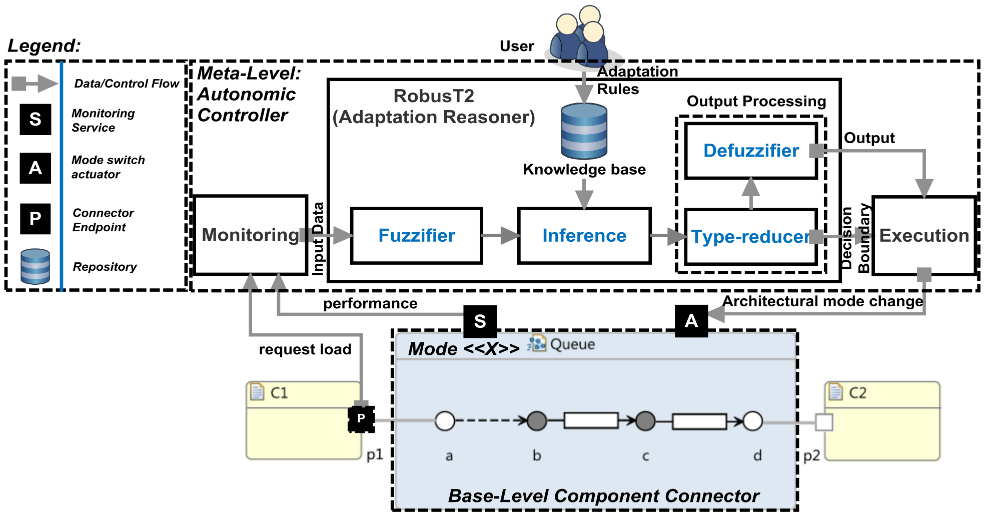


Figure 5.34. High-level view of autonomous fuzzy reasoning.

We now describe the high-level behavior of the autonomic controller that we have realized for connector adaptation. The controller monitors the performance of the connector under control as well as the number of requests at certain observation intervals. These inputs, after smoothing, are fed to the reasoning module (i.e., the FLS). The reasoning module derives the appropriate connector mode and feeds that to the execution module, which enacts the mode physically to the running system by throttling operations executable on the runtime environment. The control loop is closed by starting the next control loop after an appropriate time has been elapsed (i.e., control interval). In Sections 5.5.5 and 5.5.6, we propose a method for constructing the fuzzy-based adaptation reasoning mechanism and in Section 5.5.6.4, the reasoning logic behind it will be described.

In the remainder, we describe a method for designing the adaptation reasoning operating at the heart of the self-adaptation mechanism.

5.5.5. Adaptation knowledge elicitation

In this research, fuzzy membership functions for adaptation rules (i.e., adaptation policies) must be derived from data that were collected from a group of users of the connector under study. The principal stages of the method that we describe here are initially proposed in (J Mendel & Wu, 2010; JM Mendel, Karnik, & Liang, 2000; JM Mendel, 2001) as a generic methodology and adapted for fuzzy knowledge elicitation in several different application domains, e.g., (Jamshidi et al., 2014; Solano Martínez, John, Hissel, & Péra, 2012; Solano Martínez, 2012). In this thesis, we extend and adapt this methodology for adaptation knowledge elicitation.

In Section 5.5.3, we explained that because words mean different things to different people, they are uncertain. For the formulation of adaptation policies, we use linguistic words. As a result, fuzzy sets can be adopted for a word that has the potential to capture its uncertainties. IT2 FSs are characterized by its FOU and, therefore, have the potential to capture word uncertainties. In this section, we explain two methods for designing IT2 FS models for linguistic words in adaptation policies: the first for people who are knowledgeable about fuzzy logic and the second for non-experts in fuzzy logic.

A number of different methodologies for collecting data from a group of experts and mapping that data into the parameters of T1 MFs have been reported in several works, e.g., (Klir & Yuan, 1995). Unfortunately, none of these approaches is able to transfer the uncertainties about collecting word data from a group of experts into a T1 MF, because T1 FSs do not have enough degrees of freedom to represent this uncertainty.

In this chapter, all methods require that:

1. A continuous scale is considered for each variable. For the metrics that we choose for adaptation reasoning most of the times a natural scale exists, e.g. as in workload, response time, end-to-end latency, etc.
2. A vocabulary of qualitative (linguistics) words is produced that covers the entire scale.

A notable issue with the methodology we present here is whether or not data collected on one scale for a specific application can be rescaled on a different scale for (i.e., transferred to) another application (J Mendel & Wu, 2010). The probability elicitation literature (e.g., O'Hagan et al. (2006)) indicates that data collection is sensitive to scale and is application (context) dependent.

For adaptation reasoning, a designer begins by forming a vocabulary of application dependent words, one that is thorough enough to ensure that a person will feel linguistically comfortable interacting with the adaptation reasoner. This vocabulary must include subsets of words that each expert expects together to cover the scale, let us say $[0,10]$. Redundant words and their coverage are not issues in this methodology, although they are important issues when designing an adaptation reasoner. For example, if–then rules are usually only created for a small subset of words that cover the whole scale in this manner, keeping the number of rules as small as possible.

5.5.5.1. Eliciting Adaptation Knowledge from Knowledgeable Experts in Fuzzy

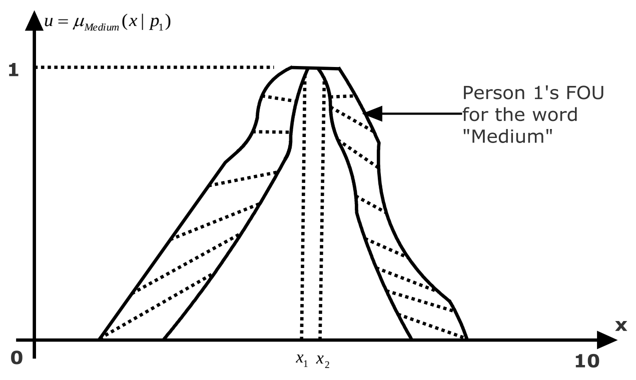
From a high-level perspective, the method we describe here has the following steps:

1. The data reflecting the uncertainties about a word are collected from a group of experts to form the FOUs related to each individual linguistic.
2. An IT2 FS for a word is defined as an aggregation of FOUs that is related to the word;
3. The aggregated FOU is mathematically modeled.

Definition 31. Uncertainty about a qualitative word regarding architecture adaptation is of two kinds: (1) *intra-uncertainty*, which is the uncertainty *a user* has about the qualitative word in an adaptation policy; and (2) *inter-uncertainty*, which is the uncertainty that *a group of users* has about the qualitative word used in an adaptation policy.

It is shown that intra-uncertainty about a qualitative linguistic, W, can be modeled using an IT2 FS, $\widetilde{W}\ (p_i)$, where $i = 1, \dots, n_W$, see (J Mendel & Wu, 2010; JM Mendel, 2001).

An example of such an FOU is depicted in Figure 5.35. The width of the FOU that is roughly provided by a person is associated with how uncertain the person is about a specific qualitative linguistic. A thin FOU means a person has a small amount of uncertainty, while a thick FOU means the person has a large amount of uncertainty regarding the linguistic.



$u = \mu_{Medium}(x \mid p_1)$
1
0
10
x
x_1 x_2
Person 1's FOU for the word "Medium"

Figure 5.35. FOU of person 1 for the linguistic "Medium" regarding the workload.

In principle, the FOUs could be extracted from a group of experts. In practice, and specifically in the domain of self-adaptive software, this may be very difficult to do because such an expert must be a fuzzy expert and understand the concepts of FS, MF, and FOU. In the domain of software and specifically users or administrators, most experts are not knowledgeable about fuzzy concepts. For such a situation, we describe another methodology in the next section to complement this. Here, it is assumed that it is possible to obtain such a FOU.

An IT2 FS captures "first-order uncertainties," whereas a T2 FS that has non-uniform FOU captures first- and second-order uncertainties, see the definitions in the background chapter (i.e., Chapter 2). Based on our experience from collecting information from experts regarding adaptation policies, they like the questions to be as simple as possible in order to provide their opinions. In a number of occasions where we have asked experts to assign a weighting function to their drawn FOU, it was almost impossible to collect appropriate data. The uncertainty that exists about the FOU is categorized as a first-order uncertainty, and the uncertainty about the weight that might be given to each element of the FOU, constructing a three dimensional MF, is considered to be a second-order uncertainty (Jerry M. Mendel & John, 2002). Note in this thesis, the focus is entirely on the first-order uncertainty of a FOU. Even though it is not known how to collect data for second-order uncertainty (J Mendel & Wu, 2010), the reasoning based on T2 FLSs is computationally expensive and not affordable for runtime analysis in self-adaptive software.

The FOUs regarding the adaptation rules are collected from a group of experts. It is important to collect such FOUs from a representative group; for example, a specific software application may only involve users, technical administrators, architects, designers and so on. An example of a FOU that is extracted from three people (i.e., p_1, p_2 and p_3) is depicted in Figure 5.36 for the linguistics term "low" response time. The constraints that each expert must follow when sketching their FOU are that the upper bound cannot exceed 1, the lower bound must not be less than 0, the lower and upper bounds cannot change direction more than one time, and the FOU cannot extend outside of the [0, 10] (or some other normalized boundary) domain for the primary variable. Each FOU models the intra-uncertainty about a word. The collection of FOUs models the inter-uncertainties about a word.

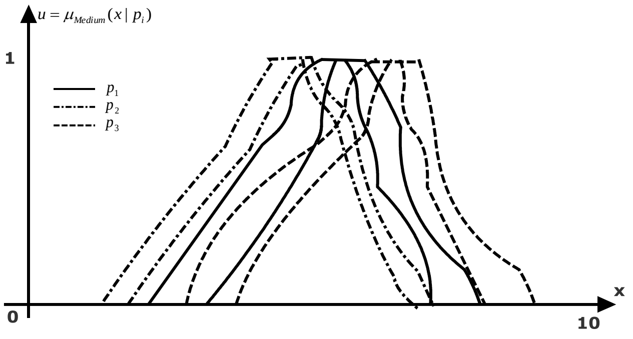


Figure 5.36. FOUs from three experts regarding the linguistic "Medium".

Definition 32. Inter-uncertainty about an adaptation rule linguistic can be modeled by means of an equally weighted aggregation of each person's word FS, $\widetilde{W}(p_i)$ $(i = 1, 2, \ldots, n_W)$, where

$$\begin{gathered}\widetilde{W}(p_i) = \{(x, \mu_{\widetilde{W}}(x|p_i)), i = 1, 2, \ldots, n_W\} \\ \mu_{\widetilde{W}}(x|p_i) = [a_{\widetilde{W}}(x|p_i), b_{\widetilde{W}}(x|p_i)] \subseteq [0,1]\end{gathered} \tag{5.27}$$

Suppose one begins by assuming that inter-uncertainty about a word can be modeled by means of a weighted aggregation of each person's word FOU, where the weight represents a degree of belief associated with each person. This suggests that a degree of belief is known or can be provided for each person, which may or may not be reasonable. Consider the following three possibilities:

1. All experts are equal and the same weight should be assigned to each FOU that is provided by each person.
2. Experts are treated differently, since some experts may have more knowledge about the meaning of a linguistic than others. For instance, an architect may be more knowledgeable about the performance of a system, and a system administrator may know more about the external environment of the system.
3. Experts are treated differently, except now it is possible that a subject's credibility depends on the value of primary variable $x \in X$, that is, some subjects may be more knowledgeable about a word for certain regions of the variable x than other subjects.

Note Scenarios 2 and 3 require additional information as opposed to Scenario 1, and that additional information will itself be uncertain, leading to even further kinds of uncertainty; hence, in this thesis Scenario 1 is focused on exclusively.

There are several ways to aggregate a group of expert's equally weighted FOUs. Mathematical operators such as "union", "intersection", and "addition" are operators to facilitate this aggregation. However, in this thesis, "union" is used for several reasons. First, the union operator preserves the commonalities as well as the differences across FOUs, while the intersection preserves only the commonalities. In this way, the intersection operator abandons a lot of useful information. Aggregation by intersection leads to an FOU that only shows total agreement across all experts regarding the adaptation policies. If a new expert's FOU does not intersect the existing aggregated FOU, then the resulting FOU would be empty. Second, the addition operator destroys the underlying requirement that the FOU of the resulting FS must be in $[0, 1]$.

Therefore, the secondary MFs of W can be expressed as:

$$\begin{aligned}
&\mu_{\widetilde{W}}(x) = \bigcup_{i=1}^{n_W} \mu_{\widetilde{W}}(x|p_i) = \bigcup_{i=1}^{n_W} [a_{\widetilde{W}}(x|p_i), b_{\widetilde{W}}(x|p_i)] \\
&LMF(\widetilde{W}) = \underline{\mu}_{\widetilde{W}}(x) = \min_{i=1,\dots n_W} a_{\widetilde{W}}(x|p_i) \\
&UMF(\widetilde{W}) = \overline{\mu}_{\widetilde{W}}(x) = \max_{i=1,\dots n_W} b_{\widetilde{W}}(x|p_i)
\end{aligned} \tag{5.28}$$

For example, the three FOUs depicted in Figure 5.36 for the linguistic "Medium" can be aggregated as in Figure 5.37.

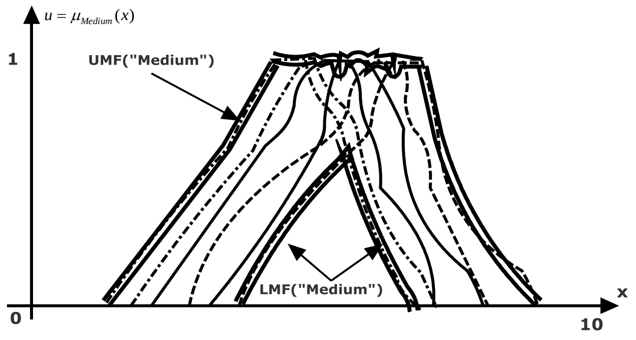


Figure 5.37. Aggregation of the three FOUs regarding the linguistic "Medium".

The aggregated FOU for the example in Figure 5.36 is depicted in Figure 5.38. Note that the aggregated FOU is bounded from above by $UMF("Medium")$ and from below by $LMF("Medium")$, as in Figure 5.38. Regardless of how many experts are asked to participate in data collection, the union of their FOUs have lower and upper bounds, see **Definition 23**. As more experts are added to the data, the shapes of these boundaries may change.

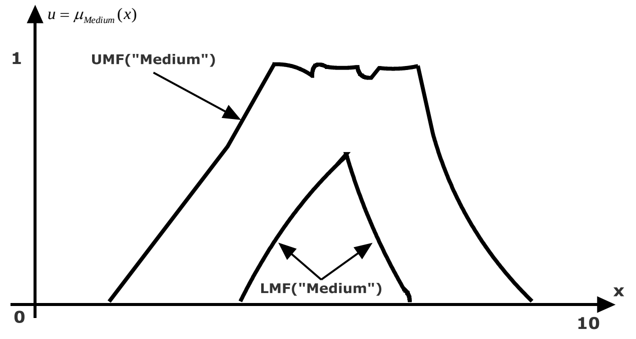


Figure 5.38. Aggregated FOU regarding the linguistic "Medium".

Figure 5.39 depicts a trapezoidal function approximation to the UMF and LMF of the FOU in Figure 5.38. The trapezoidal function is characterized by the four parameters $a, b, c, and\ d$. The triangular function is characterized by the three parameters $e, f, and\ g$. As the aggregated word fuzzy set $\widetilde{W}$ has an FOU associated with it, namely $FOU(\widetilde{W})$, see equation (5.28), we denote the approximated fuzzy set by $\widehat{W}$ and its associated FOU by $FOU(\widehat{W})$:

$$\widehat{W} = \{(x, \mu_{\widehat{W}}(x))\}$$
$$\mu_{\widehat{W}}(x) = \left[\underline{\mu}_{\widehat{W}}(x), \overline{\mu}_{\widehat{W}}(x)\right] \quad (5.29)$$

Note that the closer $\underline{\mu}_{\widehat{W}}(x)$ and $\overline{\mu}_{\widehat{W}}(x)$ are to the $\underline{\mu}_{\widetilde{W}}(x)$ and $\overline{\mu}_{\widetilde{W}}(x)$ over x, the closer $FOU(\widehat{W})$ approximates $FOU(\widetilde{W})$.

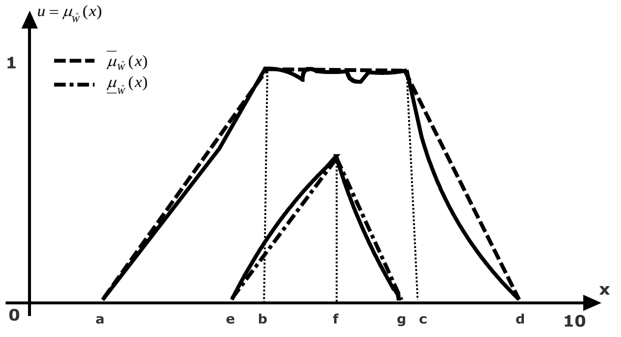


Figure 5.39. Trapezoidal approximation of the UMF, and triangular approximation of the LMF of the linguistic "Medium".

In summary, the methodology starts off by extracting the FOUs from each expert (i.e., $\widetilde{W}(p_i)$), then the union of FOUs is calculated (i.e., $\widetilde{W}$), and finally an approximation of the linguistic fuzzy set (i.e., $\widehat{W}$) is derived. This approximation utilizes data extracted from experts.

The benefits of this methodology are as follows:

- The union of the person FOUs (the data) establishes the shape of the FOU directly.
- All of the data extracted from the experts are used so that no information is lost.
- If all uncertainty disappears (i.e., all experts provide the same MF (i.e., a T1 FS), then the IT2 FS reduces to a T1 FS model.

However, this method has its own disadvantages. This method requires experts to be knowledgeable about fuzzy theory. For example, if the method introduces uncertainties because the experts do not understand what an FOU is, then the method's uncertainties become intertwined with the experts' uncertainties about the word, and this introduces another source of uncertainty. As a result, we introduce a more suitable methodology for extracting adaptation policy knowledge from a group of experts that in general might not understand fuzzy theory.

5.5.5.2. Eliciting Adaptation Knowledge from Experts who are not Knowledgeable in Fuzzy

In this section, we describe a methodology for eliciting adaptation knowledge considering that the experts may not know about the very detail of fuzzy logic and are not be able to suggest FOUs by themselves. From a high-level perspective, the method we describe here has the following steps:

1. Interval end-point data about an adaptation policy linguistic is collected from a group of experts.
2. The mean and standard deviation are established for the collected data.
3. The mean and standard deviation statistics are mapped into a parametric T1 fuzzy set.
4. A blurring parameter is used to transform the T1 FS to the corresponding IT2 FS.

The interesting point here is that this approach is similar to statistical modeling in which first the underlying probability distribution is chosen and then the parameters of that model are fitted using data and a meaningful design method, for example the method of maximum likelihood.

5.5.5.2.1. Methodology for collecting interval end-point data

The methodology is started by determining a normalized scale and creating a vocabulary of words that covers the entire scale. Then the following critical two steps are performed: (1) randomize the linguistics[1] and (2) survey a group of experts to provide end-point data for the linguistics on the normalized scale.

To better describe the methodology, we start by explaining a concrete example of data collection for an adaptation reasoning problem. In this example, two variables need to be specified by qualitative linguistics, i.e., workload and response time. Linguistic variable representing the value of workload was divided into five levels: *very low* (VL), *low* (L), *medium* (M), *high* (H), and *very high* (VH). Similarly, linguistic variable representing the value of response time were divided into five levels: *instantaneous* (I), *fast* (F), *medium* (M), *slow* (S), *very slow* (VS). We also asked 10 experts to locate an interval for each linguistic label for workload and response time in $[0,100]$. For the labels, we received 10 different intervals from the 10 experts. We then calculated the mean and deviations of the two ends in Table 5.5.

Table 5.5. Data regarding Workload and Response time labels.

Linguistic		Means		Standard Deviations	
		Start (a)	End (b)	Start (σ_a)	End (σ_b)
Workload	Very low	0	27	0	8.23
	Low	22	41.5	7.15	7.09
	Medium	36.5	64	5.80	3.94
	High	61	82.5	4.59	6.77
	Very high	78	100	6.32	0
Response time	Instantaneous	0	7.2	0	5.20
	Fast	6.1	20	4.07	5.27
	Medium	18.2	41.5	5.59	8.51
	Slow	38.5	63.5	7.09	9.44
	Very slow	60	100	7.82	0

Because the data that was requested for each linguistic was a range, and each range is defined by the two numbers, i.e., start and end points, the survey boiled down to sample statistics for the two numbers, i.e., their mean and standard deviation. The two end-point standard deviations represent the uncertainties with respect to each linguistic. Note that for each linguistic, standard deviations are not the same for the start and end.

The data regarding workload and response time in Table 5.5 are respectively visualized in Figure 5.40 and Figure 5.41. For each linguistic, there is a heavy solid box between two points. The solid box is located at the mean start and end for each linguistic. The hatched box to the left of the left-hand side of the solid box and to the right of the right-hand of it each is equal to one standard deviation, listed in Table 5.5 for the mean start and ends, respectively.

[1] Note that we reordered the linguistics in order to reduce the threat of ordering effects. More precisely, experts cannot correlate their word-interval ends from one word to the next when it is randomized.

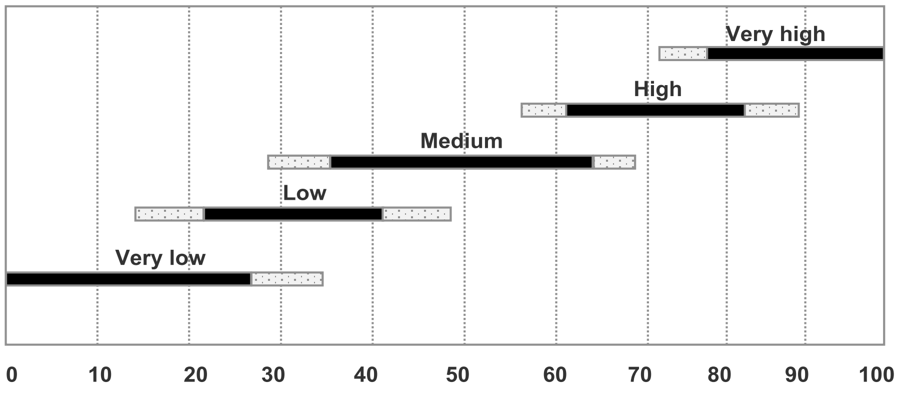


Figure 5.40. Workload linguistics with their intervals and uncertainties.

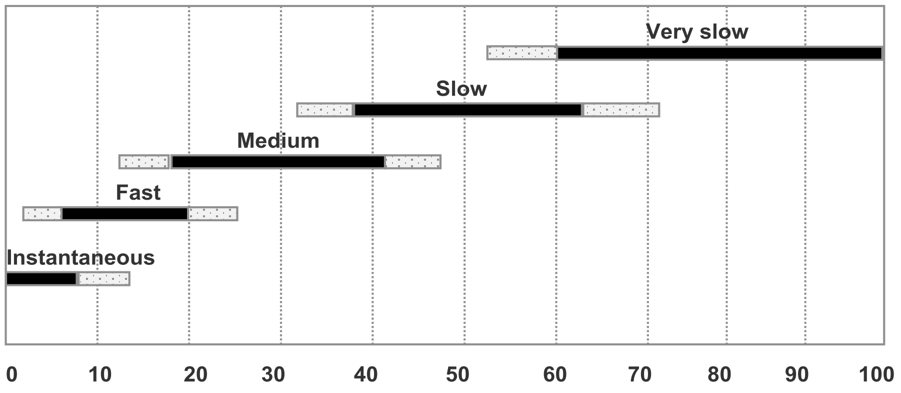


Figure 5.41. Response time linguistics with their intervals and uncertainties.

Based on the visualized data in Figure 5.40, a number of observations can be made:

1. The hatched portions of the intervals for each label represent its uncertainty.
2. Some linguistics such as “Low” (see Figure 5.40) have relatively equal uncertainty for both end points, while most of the linguistics have unequal uncertainty for their end-points.
3. There is no gap between the mean-values of the linguistics, implying that the selection of linguistics for the two parameters were suitable. If for a parameter there was a gap between the mean-value regarding one or more linguistics then either another word should be inserted between them or they should be combined.
4. Experts unanimously agree that “very low” (see Figure 5.40) and “instantaneous” (see Figure 5.41) start at zero, and there is almost no uncertainty about this. The same observation can be made for the linguistics “Very high” and “Very slow”.
5. The 5 linguistics are appropriately covered by the interval $[0,100]$.
6. The intervals between the mean start and ends are not of equal size and there is more (or less) overlap between some linguistics than between other ones.
7. It is possible to cover the $[1,100]$ interval with four labels (by omitting “Low” in Figure 5.40 and “Fast” in Figure 5.41) and this is only possible because of linguistic uncertainties.
8. Linguistic uncertainty is useful because it lets the $[0,100]$ interval be covered with a smaller number of labels than without it.
9. Linguistics mean different things to different experts.

Note that the interval end-point data that are collected from a group of experts are random data. Consequently, for each linguistic interval the following four statistics can be computed: sample mean and standard deviation of the left end, a and σ_a, and sample mean and standard deviation of the right end-point, b and σ_b, see Table 5.5.

5.5.5.2.2. Methodology for data pruning

In this step, data that have been collected from a group of experts (see Section 5.5.5.2.1) are pre-processed in a way that the data are pruned by omitting unnecessary and unwanted data and then some statistics are computed for the remaining intervals. After data intervals $[a_i, b_i]$ have been collected from a group of n subjects ($i = 1, \ldots, n$) for a linguistic, two major steps needs to be done: (1) pruning the n data intervals, and (2) calculating statistics for the data intervals that remain after the pruning step.

Pre-processing the n interval end-point data $[a_i, b_i]$ involves four stages: (1) bad data pruning, (2) outlier pruning, (3) tolerance-limit pruning, and (4) reasonable-interval pruning. Because of the data pruning, some of the n interval data are discarded and there remain $m \leq n$ intervals.

Stage 1. *Bad Data pruning*. During data collection, some experts do not pay enough attention to the instructions and so provide useless results. For pruning this type of useless data, the following constraints are used:

$$\begin{aligned} &0 \leq a_i \leq 100 \\ &0 \leq b_i \leq 100 \\ &a_i \leq b_i \end{aligned} \tag{5.30}$$

If interval ends satisfy the constraints, then an interval is accepted; otherwise, it is rejected. After bad data pruning, there remain $n' \leq n$ data intervals.

Stage 2. *Outlier pruning*. Such processing uses boxplots to identify and then eliminate outliers - outliers are points that are unusually large or small. After outlier pruning, there remain $m' \leq n'$ data intervals with the following data statistics: $a', \sigma_{a'}$ (mean and standard deviation of the m' left end-points), $b', \sigma_{b'}$ (mean and standard deviation of the m' right end-points), and $a_L{}', \sigma_{a_L{}'}$ (mean and standard deviation of the lengths of the m' intervals).

Stage 3. *Tolerance Limit Pruning*. If a data interval $[a_i, b_i]$ and its length L_i satisfy the following conditions (Walpole, Myers, & Myers, 2011), it is accepted, otherwise it is rejected.

$$\begin{aligned} &a_i \in [a' - k\sigma_{a'}, a' + k\sigma_{a'}] \\ &b_i \in [b' - k\sigma_{b'}, b' + k\sigma_{b'}] \\ &L_i \in [a_L{}' - k\sigma_{a_L{}'}, a_L{}' + k\sigma_{a_L{}'}] \end{aligned} \tag{5.31}$$

, where k is tolerance factor, which is determined as explained in (Walpole et al., 2011). For instance, if $k = 3.379$ then we can say that with 95% confidence the given limits contain at least 95% of the expert data intervals regarding adaptation knowledge.

After this stage, there remain $m'' \leq m' \leq n$ data intervals with the following data statistics: $a', \sigma_{a'}$ (mean and standard deviation of the m'' left end-points), $b', \sigma_{b'}$ (mean and standard deviation of the m'' right end-points), and $a_L{}', \sigma_{a_L{}'}$ (mean and standard deviation of the lengths of the m' intervals).

Stage 4. *Reasonable-Interval Pruning*. We only need to keep overlapping intervals. More formally, if there is such a $a' \leq m \leq b'$ that $a_i < m < b_i$ for all $i = 1, \dots, m''$ then the interval is accepted otherwise it is discarded. As a result, there finally remain m data intervals ($1 \leq m \leq n$).

Now we have the pruned data and we need to turn them to useful information for transformation to fuzzy sets. To each of the m remaining data intervals, a probability distribution is assigned. Then statistics are calculated for each interval using the probability distribution and the interval end-points. These statistics will be used later in the next section. According to (Dubois, Foulloy, Mauris, & Prade, 2004), uniform probability distribution is the most appropriate distribution when we only have incomplete knowledge about the underlying data.

If a random variable X is uniformly distributed in $[a, b]$ then (Walpole et al., 2011):

$$\begin{aligned} mean(X) &= (a+b)/2 \\ \sigma(X) &= (b-a)/\sqrt{12} \end{aligned} \tag{5.32}$$

For each of the data interval $[a_i, b_i]$, data statistics $S_1, \dots, S_m$ are calculated as follows:

$$S_i = (mean_i(X), \sigma_i(X)) \tag{5.33}$$

These statistics are then used for probability-to-fuzzy transformation in the next section.

5.5.5.2.3. Methodology for Probability to Fuzzy Transformation

The methodology for transforming the data to type-2 fuzzy sets is originally introduced in (JM Mendel, 2008) and then the enhanced version of it is presented in (J. M. Mendel & Coupland, 2012). This method is known as the interval approach (IA). In this section, we first briefly introduce this methodology and then present a more simplified methodology for the transformation of interval data to IT2 FSs. Note that these methodologies only make use of triangular T1 MF, left shoulder T1 MF, and right-shoulder T1 MF.

1. Transformation using IA Approach

Step 1. *Establish and Compute FS Uncertainty Measures*. Although many choices are available for uncertainty measures of a T1 FS (Klir & Yuan, 1995), the mean and standard deviation of T1 FSs are used for this purpose, see Table 5.6.

Table 5.6. Mean and standard deviation for the T1 MFs (JM Mendel, 2008).

Name	**Mean and Standard Deviation**
Symmetric triangle	$mean_{MF} = (a+b)/2$ $\sigma_{MF} = (b-a)/2\sqrt{6}$
Left-shoulder trapezoid	$mean_{MF} = (2a+b)/3$ $\sigma_{MF} = \left(\frac{1}{6}(a+b)^2 + 2a^2\right) - {mean_{MF}}^2)^{\frac{1}{2}}$
Right-shoulder trapezoid	$mean_{MF} = (2a+b)/3$ $\sigma_{MF} = \left(\frac{1}{6}(a'+b')^2 + 2a'^2\right) - {mean'_{MF}}^2)^{\frac{1}{2}}$ $a' = L - b$ $b' = L - a$ $mean'_{MF} = L - mean_{MF}$

Step 2. *Transformations of the data interval into the parameters of T1 FSs*. The parameters of a T1 FS are calculated by equating the mean and standard deviation of a T1 FS to the mean and standard deviation of a data interval. More specifically, the following equations need to be solved:

$$\begin{gathered} mean^{i}{}_{MF} = mean_i(X) \\ \sigma^{i}{}_{MF} = \sigma_i(X) \end{gathered} \tag{5.34}$$

, where $mean^{i}{}_{MF}, \sigma^{i}{}_{MF}$ are calculated using Table 5.6, and $mean_i(X), \sigma_i(X)$ are computed using (5.32).

The resulting T1 FSs, denoted as $R^{i}{}_{e}$, are called embedded T1 FSs, see Background Chapter.

Step 3. *Compute an IT2 FS Using the Union of Embedded T1 FSs*. The corresponding IT2 FS $\tilde{R}$ can be computed as:

$$\tilde{R} = \cup_{i=1}^{m} R^{i}{}_{e} \tag{5.35}$$

, where $R^{i}{}_{e}$ is the ith embedded T1 FS derived in the previous step.

2. Transformation using the Blurring Parameter

Let us assume the mean values of the interval ends of the linguistic labels are a and b with standard deviations σ_a and σ_b respectively. More specifically:

$$\begin{gathered} a = mean(a_i) \\ b = mean(b_i) \\ \sigma_a = \sigma(a_i) \\ \sigma_b = \sigma(b_i) \end{gathered} \tag{5.36}$$

In this methodology, we only make use of triangular T1 MF, left shoulder T1 MF, and right-shoulder T1 MF. Triangular T1 MFs are constructed by connecting: $l = (a - \sigma_a, 0), m = ((a+b)/2,1), r = (b + \sigma_b, 0)$. Accordingly, trapezoidal MFs are constructed by connecting: $(a - \sigma_a, 0), (a, 1), (b, 1), (b + \sigma_b, 0)$.

As discussed in Section 5.5.5.2.1, there are uncertainties associated with the ends and as a result the locations of the MFs. For instance, one may imagine a triangular T1 MF in: $l' = (a - 0.4 * \sigma_a, 0), m = ((a+b)/2,1), r' = (b + 0.7 * \sigma_b, 0)$. T1 MFs cannot capture these kind of uncertainties, while IT2 MFs can handle them suitably. In IT2 MFs, the FOU can be obtained by specifying the UMF and LMF for each linguistics. Let us consider the blurring parameter $0 \leq \alpha \leq 1$. Then, we are able to construct the FOU. For both the triangular and trapezoidal MFs, the locations of UMF and LMF are indicated in Table 5.7.

Table 5.7. Locations of the main points of IT2 MFs.

Triangular	Trapezoidal
$l_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$	$ll_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$
$m_{UMF} = ((a+b)/2\,, 1)$	$ul_{UMF} = (a - \alpha\sigma_a, 1)$
$r_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$	$ur_{UMF} = (b + \alpha\sigma_b, 1)$
$l_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$	$lr_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$
$m_{LMF} = ((a+b)/2\,, 1)$	$ll_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$
$r_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$	$ul_{LMF} = (a + \alpha\sigma_a, 1)$
	$ur_{LMF} = (b - \alpha\sigma_b, 1)$
	$lr_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$

If we choose $\alpha = 0$, then an IT2 MF will be reduced to a T1 MF. A blurring parameter $\alpha = 1$ implements FSs with a maximum amount of blur and the widest FOUs. One may also imagine different blurring parameters, let say α_L and α_R for each ends, to derive an asymmetric blurring.

5.5.5.3. Evaluation of Adaptation Knowledge Extraction Methodology

In order to evaluate the adaptation knowledge extraction, a dataset was collected from 21 experts[2] for a vocabulary of 5 linguistics. The linguistics were randomized in order to avoid the threats regarding effects. For all linguistics, we asked the following question:

> "What are the ends of an interval that you associate with each linguistic? Please provide the answer on a scale of 0 to 10." (5.37)

All of the data were collected according to 5.5.5.2.1, were pruned according to 5.5.5.2.2 and were processed and transformed into IT2 FSs according to the two presented methodologies in 5.5.5.2.3. Note that the operation details of the data collection including the template for data gathering are presented in detail in Appendix A.

Table 5.8 summarizes the collected raw data from the 21 experts in the survey. Table 5.9 summarizes how many data intervals remained in each of the four pruning stages. Table 5.9 also gives the final left and right end-point statistics that were used to establish the each linguistics' FOU. These statistics are based on the m remaining data intervals after stage 4 of pruning.

Table 5.8. Raw data for 5 linguistics w.r.t. workload collected from 21 experts.

	Linguistics w.r.t. Workload									
Experts	**Very low**		**Low**		**Medium**		**High**		**Very high**	
P1	0	2	1.5	4	3.5	6.5	6	9	8.5	10
P2	0	2.5	2	5	4	6	6	8.5	8	10
P3	0	3	2.5	4.5	4	6.5	6	8	7	10
P4	0	3.5	3	4	4	6	6	9	8.5	10
P5	0	3	2	4	3	7	7	9	8.5	10
P6	0	2	1.5	4	3.5	6.5	6.5	8	7.5	10
P7	0	3	2.5	5	4.5	6.5	6	8	8	10
P8	0	3.5	3	4.5	4	6	5.5	7	7	10
P9	0	1	1	2.5	2.5	6	5.5	8.5	8	10
P10	0	3.5	3	4	3.5	7	6.5	7.5	7	10
P11	0	11	1	2	-1	4	7	8	8	10
P12	0	0.001	1	5	5	6	6	8	100	100
P13	0	2	1	4	3	7	6	8	7	10
P14	0	1	1	2.001	2	4	4	11	9	10

[2] All the experts that we asked for this experiment were PhD students in software engineering whose theses were on topics related to software architecture, software evolution and self-adaptive software. Note that the experts were at different stages of their PhDs and located in different countries, including Australia, Canada, Austria, Italy and United States. To enhance their knowledge about type-2 fuzzy logic, separate training tutorials (each took around 1 hour and through Skype) were carried out before doing experiment. This tutorial consisted of an introduction to fuzzy theory, type-1 and type-2 fuzzy set and membership function, the concept of FOU, LMF, UMF, embedded fuzzy sets and fuzzy logic systems. Note that since they all have experience in web-based application development, they have a good understanding of workload and response time concerns.

P15	0	2	2	4	4	6	6	8	8	10
P16	0	4	3	5	5	7	6	9	8	10
P17	0	5	4	6	5	7	6	9	9	10
P18	0	6	4	7	3	8	6	9	7	10
P19	0	0.5	0.5	1.5	1.5	8.5	8.5	9.5	9.5	10
P20	0	2	1	3	3	7	6	9	9	10
P21	0	1	0.5	4	4	6	6	9	7	10

Table 5.9. Remaining data intervals and their mean and standard deviation.

Linguistic	Pruning stages				Left-end		Right-end	
	n'	m'	m''	m	a	σ_a	b	σ_b
Very low (VL)	20	20	19	18	0	0	2.47	1.19
Low (L)	21	19	19	11	1.59	0.66	4.23	0.61
Medium (M)	20	18	17	17	3.85	0.75	6.47	0.45
High (H)	20	20	18	18	6.14	0.38	8.47	0.53
Very high (VH)	20	20	20	20	7.98	0.80	10	0

Having applied the two transformation approaches introduced in Section 5.5.5.2.3 to the derived statistics of the data collection, two sets of different FOUs for the workload linguistics were derived. Figure 5.42 represents the FOUs regarding the IA approach, while Figure 5.44 illustrates the FOUs as a result of a transformation with blurring parameter. Figure 5.43 illustrates the FOUs regarding the IA approach, but with their embedded T1 MFs. Note that the linguistics are ordered in the figures so that the diagrams start with left-shoulder FOUs throughout interior FOUs ending with right-shoulder FOUs. Note the differences between the interior FOUs, as in Figure 5.42 they are shaped with trapezoidal UMFs and triangular LMFs, while in Figure 5.44 they are shaped with triangular UMFs and LMFs. Note that in this example the number of linguistics for the workloads is quite low (i.e., 5 words) but the interval $[0,10]$ has been covered by them. This shows that the number of linguistics in this example is appropriate and we do not need to add or remove existing words. However, if we need a more efficient reasoner, we can reduce the number of linguistics to minimum level. For example, the FOUs in Figure 5.42 can be reduced to the following sub-vocabularies:

$$(VL, VH), (VL, M, VH), (VL, H, VH), (VL, L, M, VH), (VL, L, H, VH), (VL, M, H, VH) \tag{5.38}$$

Similarly, the FOUs in Figure 5.44 can be reduced to the following sub-vocabularies:

$$(VL, M, VH), (VL, L, M, VH), (VL, M, H, VH) \tag{5.39}$$

More specifically, each of the sub-vocabularies can be substituted as the existing words to cover the interval between $[0,10]$. In the reasoning part, we describe how this reduction will result in a more efficient reasoning procedure.

Moreover, other potential scenarios may happen in different situations. For example, in one situation, the designer may choose a smaller number of linguistics that cannot cover the whole interval. In this case, the designer needs to add more linguistics to the existing vocabulary to accommodate this lack of words. In other situations, the designer may choose more than enough linguistics. As a result, some of them might have very similar FOUs. In this case, the designer needs to discard these redundant FOUs.

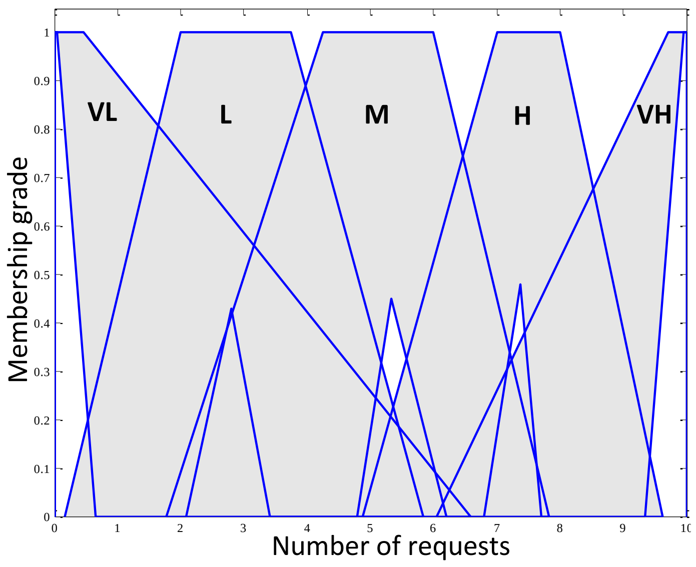


Figure 5.42. IT2 MFs of the workload linguistics resulting from the transformation using IA approach.

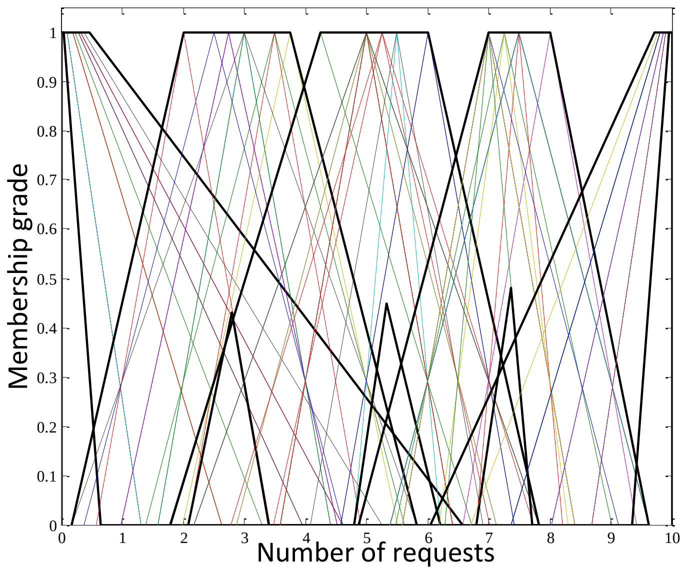


Figure 5.43. IT2 MFs of the workload linguistics after the transformation using IA approach with their embedded T1 MFs.

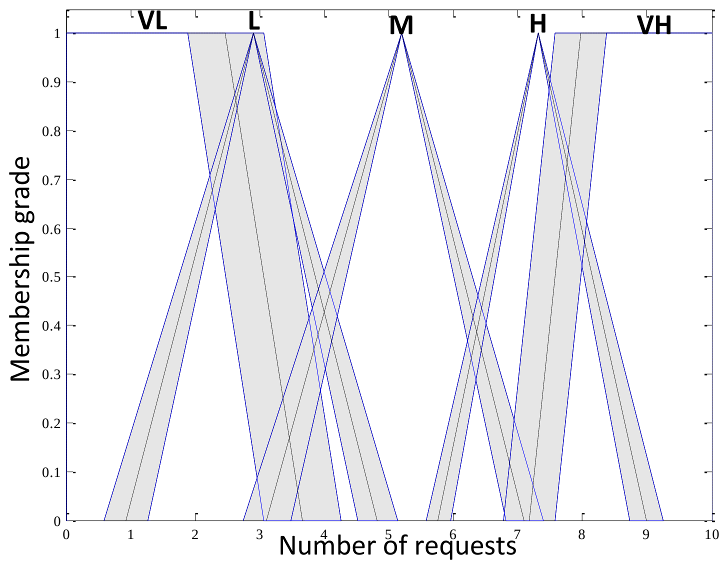


Figure 5.44. IT2 MFs of the workload linguistics resulting from the transformation using blurring parameter (Blurring here is 0.5).

In order to compare the derived FOUs in an objective manner, we adopted some uncertainty measures, which are introduced in (Wu & Mendel, 2007). Intuitively, these uncertainty measures convey the following characteristics of the FOU of a linguistic:

- The *centroid* calculates the center of gravity for the FOU.
- The *fuzziness* (entropy) is used to quantify the amount of vagueness in the word represented by the FOU.
- The *cardinality* measures the average of membership grades in the FOU.
- The *variance* of measures FOU's compactness, i.e. a smaller (larger) variance means the FOU is more (less) compact.
- The *skewness* is an indicator of the FOU's symmetry. This measure is smaller than zero when the FOU skews to the right, and is larger than zero when it skews to the left, and is equal to zero when it is symmetrical.

For a more formal definitions and the formulas for calculating the values of these measure please refer to (Wu & Mendel, 2007).

By an examination of the uncertainty measurement data in Table 5.10 concerning the FOUs created as a result of the transformation methods, several observations can be made. The fuzziness of the FOUs results from an IA approach that is higher than the ones that resulted from the blurring approach. This means that the vagueness of the linguistics represented by the FOU that is derived based on IA approach is higher. Similarly, the values of variance of the linguistics represented by the FOU that is derived by IA approach is higher. This is obvious as the FOUs in Figure 5.44 are more compact than the ones in Figure 5.42. However, the cardinality of the former FOUs are lower than the latter ones. For the other two measures (i.e., centroid and skewness), no specific differences can be observed.

Table 5.10. Uncertainty measures of the IT2 FSs of the workload linguistics w.r.t. the two transformation approaches.

Transformation Method	Linguistic	Uncertainty Measure				
		Centroid	**Fuzziness**	**Cardinality**	**Variance**	**Skewness**
IA	VL	1.7594	0.3650	0.2865	4.0249	8.6027
	L	2.9056	0.4095	0.3489	1.5769	0.4944
	M	5.0643	0.4073	0.3452	1.8448	-1.5161
	H	7.3043	0.3982	0.3228	1.0712	-0.0784
	VH	9.0226	0.3586	0.3045	1.2233	-1.6717
Blurring $\alpha = 0.1$	VL	1.5449	0.1360	0.8069	0.8596	0.1023
	L	2.8883	0.3667	0.4796	0.6363	-0.0086
	M	5.1333	0.3716	0.4794	0.6681	-0.0239
	H	7.3583	0.3701	0.4823	0.4367	0.0131
	VH	8.7825	0.1175	0.8309	0.5261	-0.0367
Blurring $\alpha = 0.5$	VL	1.5659	0.2074	0.7084	0.9640	0.3737
	L	2.8899	0.3672	0.4253	0.6643	-0.0168
	M	5.1349	0.3714	0.4253	0.6946	-0.0489
	H	7.3567	0.3668	0.4346	0.4516	0.0243
	VH	8.7666	0.1850	0.7390	0.5692	-0.1315
Blurring $\alpha = 1.0$	VL	1.5561	0.3040	0.6304	1.4679	1.5235
	L	2.8847	0.3669	0.3731	0.7524	-0.0423
	M	5.1305	0.3638	0.3724	0.7943	-0.1488
	H	7.3590	0.3671	0.3868	0.4966	0.0706
	VH	8.7824	0.2741	0.6663	0.7896	-0.5138

There are also some other characteristics belonging to each of the transformation approaches. One of the most prominent differences is the existence of a design parameter in the blurring approach. The blurring parameter is particularly useful for the designer of adaptive systems to embed more uncertainty into the FOUs of linguistics. Therefore, this parameter can act as one of the design parameters that provide more flexibility to the designer of a fuzzy logic controller in order to design it in a way that better accommodates environmental uncertainties (Sepúlveda, Castillo, Melin, Rodríguez-Díaz, & Montiel, 2007).

In the design of fuzzy logic systems, it is necessary that when all sources of uncertainty disappear, a T2 design must reduce to a T1 (N. Karnik & Mendel, 1998). The blurring parameter provides a straightforward mechanism for this reduction. If we change α to zero, then IT2 MF will be reduced to a T1 MF. On the other hand, a blurring parameter $\alpha = 1$ implements FSs with maximum embedded uncertainty.

Note that forcing experts to understand the concept of a FOU certainly limits the knowledge elicitation method to experts who either already know about fuzzy theory or are trained in fuzzy theory just before the elicitation session. This can introduce methodological uncertainties into the elicitation method, and as a result, linguistic uncertainties are a combination of methodological uncertainties and actual linguistic uncertainties. These cannot be distinguished from each other because no measure for the methodological uncertainties is available as opposed to the linguistic measures of uncertainties that we discussed earlier in this section. Therefore, if a knowledge elicitation methodology does not need that an expert to know anything about the concept of FOU or similar concepts in fuzzy theory, this is considered a favorable point for that method. Note that the knowledge elicitation approach introduced in Section 5.5.5.2 does not require such expertize.

5.5.6. Fuzzy logic system design for adaptation reasoning

In this section, we design and develop three different FLSs to perform the adaptation reasoning of a component-based system introduced in Section 5.5.2. The difference between the FLSs is the level of uncertainty that we have embedded in their membership functions. The first FLS is a T1 FLSs that has an uncertainty of 0%, the second and the third IT2 FLSs encompass an uncertainty of 50% and 70% respectively. The design of the MFs is based on the elicitation approach presented in Section 5.5.5.2. In this thesis, we propose an approach to model and minimize the effects of uncertainties in self-adaptive software by using interval type-2 FLSs.

The objective of this research is to study the feasibility as well as the implications of the use of type-2 fuzzy logic in real world self-adaptive software in general and self-adaptive software connectors in particular. Note that the optimization of the designed FLSs is not considered. However, different levels of uncertainty in the MFs are considered. This choice allows us to evaluate IT2 and T1 fuzzy controllers under comparable conditions. Note that the input and output MFs are defined as trapezoidal and triangular MFs for the two input and one output fuzzy sets.

5.5.6.1. Rule-base design

In self-adaptive systems, quantitative parameters are often classified into two classes: 1) *Environmental variables* (e.g., load), which are not under the control of the application. 2) *Internal quality variables* (e.g., performance), which indicate how well the application is functioning in the environment in which they are embedded. In the running example, linguistic variables representing the value of input parameters were divided into three levels: *low* (L), *medium* (M), *high* (H). The consequent (i.e., adaptation policy) was divided into the architectural modes of the system (i.e., *Idle*, *Normal*, *Effort*, and *Best Effort* as in Figure 5.33). To design the fuzzy rules of the controller, we collected the data by performing a data collection survey among 10 domain experts (see Section 5.5.5 for more details about the methodology and the domain experts background). We used questions as follows to extract knowledge from experts:

IF (workload is *high* **AND** performance is *low*), **THEN** (system must switch to …). (5.40)

These experts were asked to choose a consequent using one of the possible architectural modes. Not surprisingly, different experts chose different modes for the same questions. The questions and conflicting responses are summarized in Table 5.11. Note that in order to reduce the threat of ordering effects, we reordered the questions.

Table 5.11. Questions for adaptation policies and responses.

Rule (l)	Antecedents		Consequent				C_{avg}^{l}	c_{avg}^{l}
	Work Load	Performance	Idle	Normal	Effort	Best Effort		
1	Low	Low	1	7	2	0	[2.12779, 2.57019]	2.34908
2	Low	Medium	2	8	0	0	[1.838, 2.2477]	2.04386
3	Low	High	9	1	0	0	[0.95005, 1.27015]	1.11832
4	Medium	Low	0	2	7	1	[2.87412, 3.34652]	3.10914
5	Medium	Medium	0	4	6	0	[2.58052, 3.07552]	2.8273
6	Medium	High	0	5	5	0	[2.49905, 2.9841]	2.7408
7	High	Low	0	0	2	8	[3.95168, 4.14648]	4.04402
8	High	Medium	0	0	4	6	[3.69036, 3.97016]	3.82634
9	High	High	0	1	5	4	[3.34757, 3.70242]	3.52216

We also asked the experts to locate each linguistic label for both antecedents and consequents in the interval $[0,5]$. For each linguistic labels, we received 10 intervals from the 10 experts. We then calculated the mean and deviations of the two ends in Table 5.12. Note here, for simplicity, we assume that the linguistics for both antecedents have the same quantification.

Table 5.12. Data regarding antecedents and consequent labels.

Linguistic		Means		Standard Deviations	
		Start (a)	End (b)	Start (σ_a)	End (σ_b)
Antecedents	**Low**	0	1.87	0	0.51
	Medium	1.92	3.43	0.98	0.83
	High	3.93	5	0.41	0
Consequent	**Idle**	0	1.64	0	0.62
	Normal	1.32	2.95	0.39	0.91
	Effort	2.37	3.87	0.72	0.88
	Best Effort	3.64	5	0.22	0

5.5.6.2. Input membership functions design

The input fuzzy sets (regarding workload and performance) are composed by the 3 membership functions. The membership functions are distributed in the normalized domain of the fuzzy set (i.e., in the interval $[0,5]$) as illustrated in Figure 5.45 and Figure 5.46 respectively for the uncertainty level 50% and 70% (i.e., blurring parameters $0.5, 0.7$). Note that for transformation of the data presented in Table 5.12 to these MFs, the methodology described in 5.5.5.2.3.2 have been used.

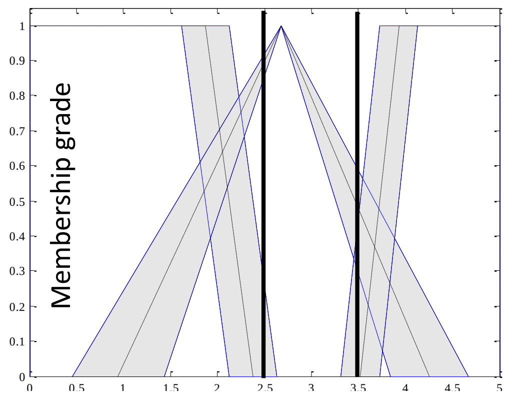


Figure 5.45. IT2 MFs of the antecedents' linguistic labels ($\alpha = 0.5$).

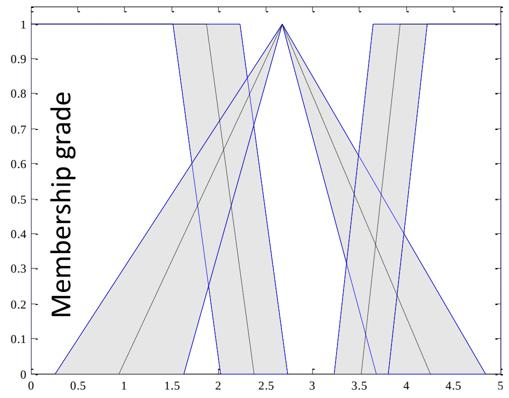


Figure 5.46. IT2 MFs of the antecedents' linguistic labels ($\alpha = 0.7$).

5.5.6.3. Output membership functions design

The output fuzzy set is composed by the 4 membership functions regarding the architectural modes (see Figure 5.33). The membership functions are distributed in the normalized domain of the fuzzy set (i.e., in the interval $[0,5]$) as illustrated in Figure 5.47. Note that for transformation of the data presented in Table 5.12 to these MFs, the methodology described in Section 5.5.5.2.3 have been used.

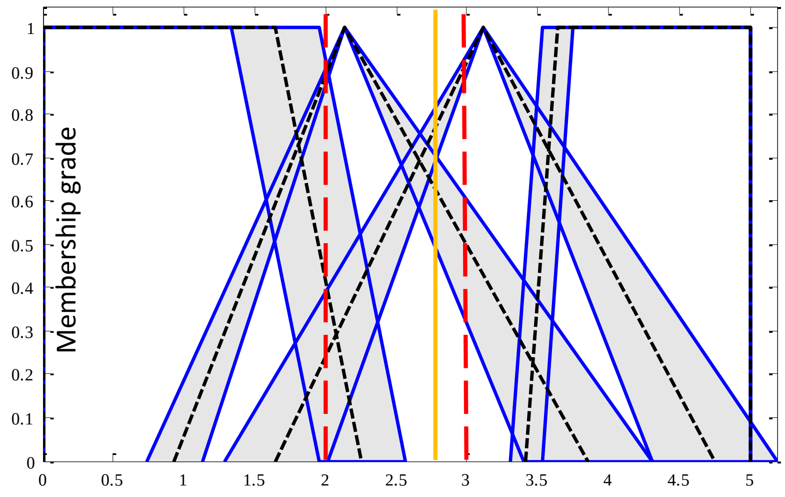


Figure 5.47. IT2 MFs of the consequent's linguistic labels.

5.5.6.4. *Adaptation reasoning process using the designed FLS*

The overall view of the autonomous controller for adaptation reasoning is shown in Figure 5.34. As illustrated, the controller covers both design-time and runtime. During design-time, the aim is to design a fuzzy controller, specify its rule-base (see Section 5.5.6.1), and derive appropriate MFs (see Section 5.5.6.2 and 5.5.6.3). At runtime, while the controller starts operating for connector self-adaptation, it keeps monitoring quality and environmental data that may affect non-functional requirement satisfaction. The controller continuously adjusts the system configuration with respect to runtime data that may affect changes in the connector behavior. The key mechanism for decision making at runtime is the fuzzy inference process. In the following, we discuss each phase in turn and describe the relevant activities.

Design-time. The approach starts at design-time when the architecture of the fuzzy controller is designed through a feedback loop. The key point here is to perform pre-computations of costly calculations to allow a runtime efficient adaptation reasoning based on fuzzy inference. The main reason is that fuzzy controller need depends on a costly calculation of type-reduction algorithm (N. N. Karnik & Mendel, 2001) in order to produce appropriate control actions. Unfortunately, IT2 FLSs, in the traditional design, can hardly satisfy the execution time constraints normally imposed by runtime analyses because of costly centroid calculations, which are proportional to the number of rules in the rule-base. In particular, the excessive use of centroid calculations at runtime leads to unsatisfactory execution time. We will discuss the details concerning the computational complexity in the evaluation section.

The rules in this work are in the form of multi-input single-output:

$$R^l: IF\ x_1\ is\ \tilde{F}_1^l\ and\ \dots and\ x_p\ is\ \tilde{F}_p^l, THEN\ y\ is\ \tilde{G}^l \tag{5.41}$$

Because the preferences of users may not be similar, many adaptation rules in the mind of users may be conflicting. In this step, rules with the same if part are combined into a single rule. For each response that we received from the users, we have:

$$R^l: IF\ x_1\ is\ F_1^l\ and\ \dots and\ x_p\ is\ F_p^l, THEN\ y\ is\ y^{(t_u^l)} \tag{5.42}$$

, where t_u^l is the index for the available responses. In order to combine these conflicting rules, we use the average of all the responses for each rule and use this as the centroid of the rule consequent. Note that as indicated in (5.41), the rule consequents are IT2 FSs. However, when the type reduction is used, these IT2 FSs are replaced by their centroids in the computation, so we represent them as intervals $[\underline{y}^n, \overline{y}^n]$ or crisp values when $\underline{y}^n = \overline{y}^n$. This leads to rules that have the following form:

R^l: IF (workload (x_1) is $\tilde{F}_{i_1}$, AND performance index (x_2) is $\tilde{F}_{i_2}$), THEN (target mode (y) is C_{avg}^l). (5.43)

, where C_{avg}^l is defined as:

$$C_{avg}^l = \frac{\sum_{u=1}^{N_l} w_u^l \times C_{\tilde{F}_u}}{\sum_{u=1}^{N_l} w_u^l} \tag{5.44}$$

, here $C_{\tilde{F}_u}$ is the centroid of IT2 FSs $\tilde{F}_u$, $(u = 1,2,3,4)$, and w_u^l is the weight associated with uth consequent of the lth rule (cf. Table 5.11). The centroids of the four IT2 FSs are as follows:

$$
\begin{aligned}
C_{\tilde{F}_1} &= [0.8232, 1.1305], C_{\tilde{F}_2} = [2.0917, 2.527] \\
C_{\tilde{F}_3} &= [2.9064, 3.4412], C_{\tilde{F}_4} = [4.213, 4.3228]
\end{aligned} \tag{5.45}
$$

Therefore, each C_{avg}^{l} (see Table 5.11) can be computed with the Equation (5.44). For instance, C_{avg}^{4}, which is associated to rule number 4 (cf. Table 5.11) is calculated as:

$$
C_{avg}^{4} = \frac{0 \times C_{\tilde{F}_1} + 2 \times C_{\tilde{F}_2} + 7 \times C_{\tilde{F}_3} + 1 \times C_{\tilde{F}_4}}{0 + 2 + 7 + 1} = [2.87412,3.34652] \tag{5.46}
$$

To summarize, we transform the rule base with IT2 MFs as consequents to a rule base with crisp consequents (cf. Table 5.11) to enable runtime efficient adaptation reasoning.

Runtime. When the approach moves to runtime, its activities are inspired by the MAPE-K loop shown in Figure 5.34. The quality data collected through monitoring must be smoothed and normalized (simply transform to an appropriate scale) that can be used to feed the fuzzy controller. This normalization in general depends on the scale that rule antecedents are specified. An example of such transformation can be found in our recent publication (Jamshidi et al., 2014).

Let us imagine the normalized values regarding the workload and performance index are $x_1 = 2.5$ $x_2 = 3.5$ respectively, see the solid lines in Figure 5.45. For $x_1 = 2.5$, two IT2 FSs regarding the linguistics $\tilde{F}_1 = Low$ and $\tilde{F}_2 = Medium$ with the degrees [0,0.2647] and [0.8594,0.9213] are fired. Similarly, for $x_2 = 3.5$, two IT2 FSs regarding the linguistics $\tilde{F}_2 = Medium$ and $\tilde{F}_3 = High$ with the firing degrees [0.2949,0.5875] and [0,0.4512] are fired. Consequently, four rules are fired: $R^2{:}\left(\tilde{F}_1, \tilde{F}_2\right), R^3{:}\left(\tilde{F}_1, \tilde{F}_3\right)$, $R^5{:}\left(\tilde{F}_2, \tilde{F}_2\right), R^6{:}\left(\tilde{F}_2, \tilde{F}_3\right)$, see Table 5.11. The firing intervals are then computed. For instance, the firing interval (F^5) associated to the rule R^5 is:

$$
\begin{aligned}
\underline{f}^{5} &= \underline{\mu}_{\tilde{F}_1^5}(x_1') \otimes \underline{\mu}_{\tilde{F}_2^5}(x_2') = 0.8594 \times 0.2949 = 0.2534 \\
\overline{f}^{5} &= \overline{\mu}_{\tilde{F}_1^5}(x_1') \otimes \overline{\mu}_{\tilde{F}_2^5}(x_2') = 0.9213 \times 0.5875 = 0.5413
\end{aligned} \tag{5.47}
$$

By following similar procedure, the other firing intervals are: $F^2 = [0,0.1555]$, $F^3 = [0,0.1194]$, $F^6 = [0,0.4157]$. By using a center-of-set type reducer (note that the type-reducer that we use here is called center-of-sets as given in **Definition 33**), the output can be obtained:

$$
Y_l(2.5,3.5) = [y_l(2.5,3.5), y_r(2.5,3.5)] = [1.9934,3.0755] \tag{5.48}
$$

Definition 33. The ***center-of-set type reduction*** is computed as:

$$
Y_{cos} = \bigcup_{\substack{f^l \in F^l \\ y^l \in C_{\tilde{G}^l}}} \frac{\sum_{l=1}^{N} f^l \times y^l}{\sum_{l=1}^{N} f^l} = [y_l, y_r] \tag{5.49}
$$

, where $f^l \in F^l$ is the firing degree of rule l and $y^l \in C_{\tilde{G}^l}$ is the centroid of the IT2 FS $\tilde{G}^l$ (cf. **Definition 30**). Note y_l, y_r are computed by the KM algorithm (N. N. Karnik & Mendel, 2001).

The defuzzified output can then be calculated:

$$Y(2.5,3.5) = \frac{1.9934 + 3.0755}{2} = 2.5345 \qquad (5.50)$$

Similarly, we can compute $Y(x_1, x_2)$ for all the possible normalized values of the input parameters ($x_1, x_2 \in [0,5]$). The resulting hyper-surfaces $Y^{IT2} = f(x_1, x_2), Y^{T1} = f(x_1, x_2)$ corresponding to the output of the designed IT2 and T1 FLS for adaptation reasoning are shown in Figure 5.48 and Figure 5.50 respectively. Note that $Y^{IT2,T1}(x_1, x_2) \subseteq [0,5]$ for any (x_1, x_2). Due to the space limitations, we have not discussed the T1 reasoning process, but the calculations are similar except that in the calculations we use the T1 centroid c^l_{avg} instead of the IT2 centroid C^l_{avg}, see the last two columns in Table 5.11.

5.5.6.5. Fuzzy logic control surfaces

The designed fuzzy logic system for adaptation reasoning is completely defined by its membership functions (see Sections 5.5.6.2 and 5.5.6.3) and fuzzy rules (see Section 5.5.6.1). Having performed the reasoning process for all input values (i.e., throughout the domain of input fuzzy sets), the control surfaces defined by Equation (5.26) are illustrated in Figure 5.48 (for uncertainty level 50%), Figure 5.49 (for uncertainty level 70%) and Figure 5.50 (for uncertainty level 0%, i.e., control surface for T1-FLS). These figures reveal that the higher the uncertainty level is, the larger the confidence interval would be.

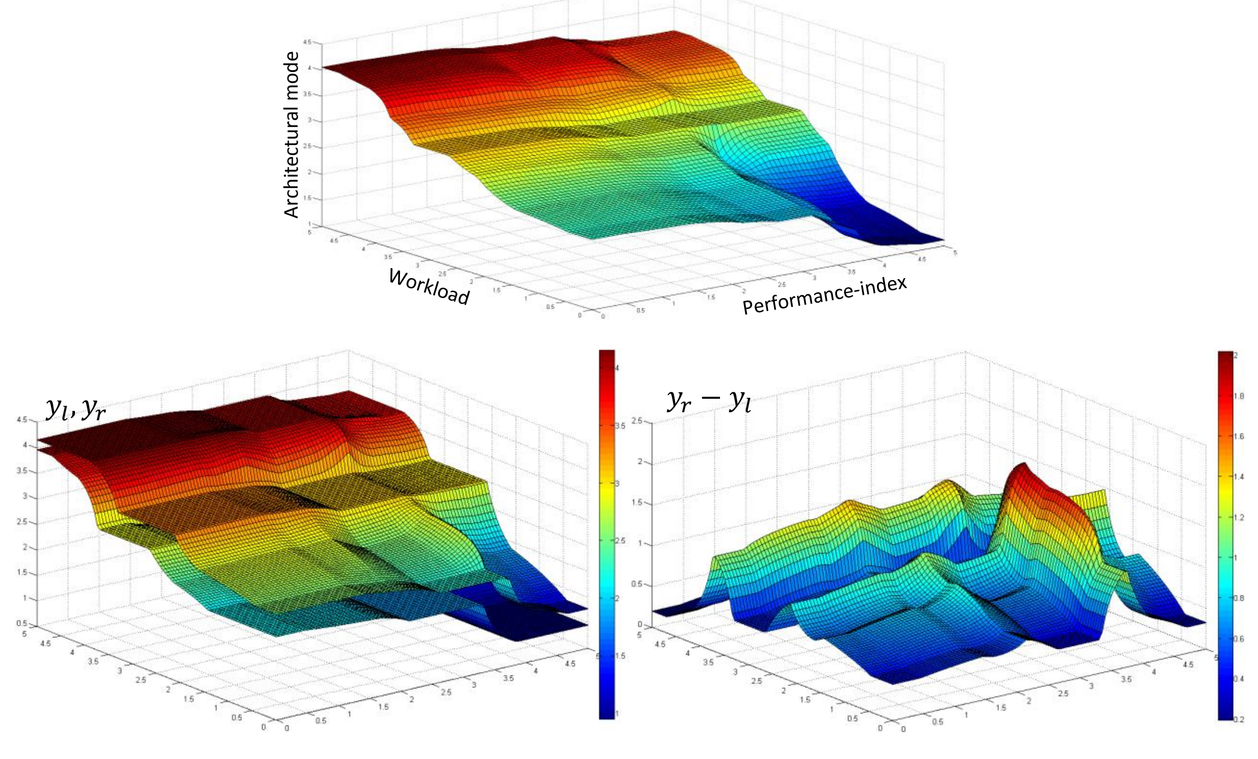


Figure 5.48. Output control surface of the IT2 FLS for adaptation reasoning ($\alpha = 0.5$) (a), confidence interval (i.e., y_l, y_r) (b) and their differences (i.e., $y_r - y_l$) (c).

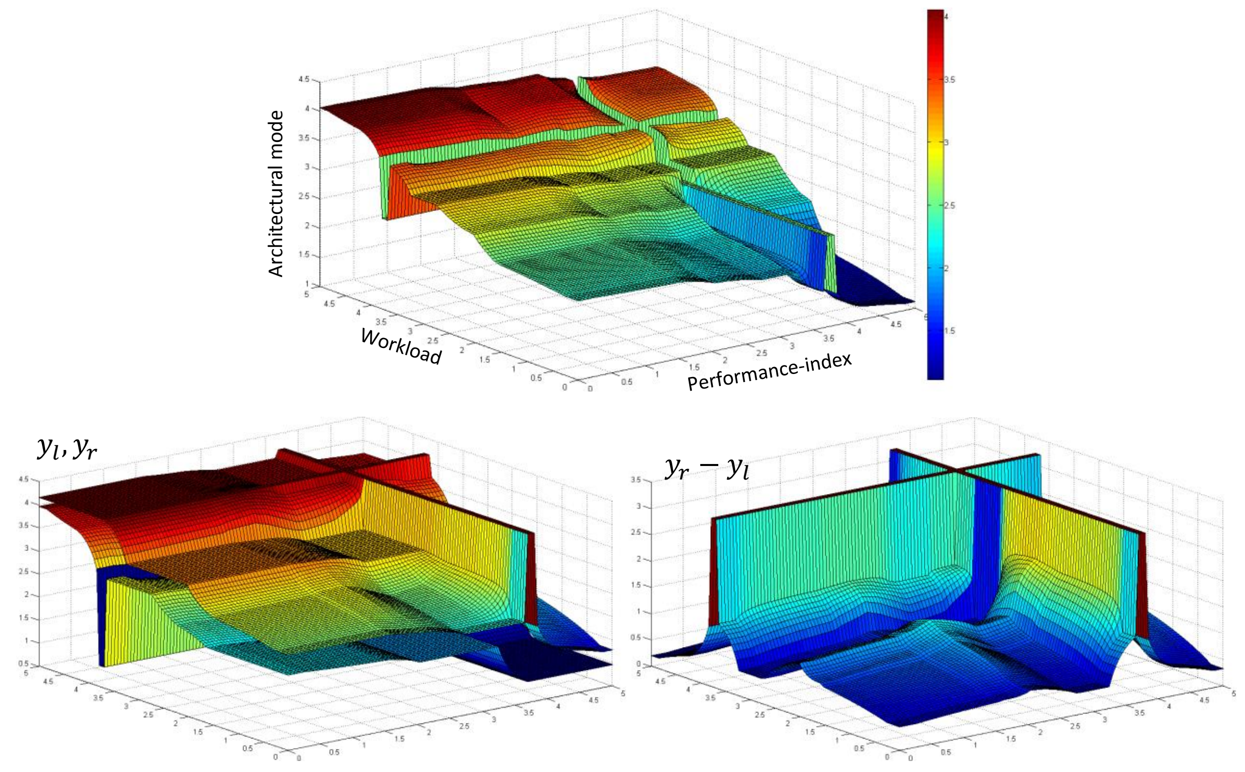


Figure 5.49. Output control surface of the IT2 FLS for adaptation reasoning ($\alpha = 0.7$) (a), confidence interval (i.e., y_l, y_r) (b) and their differences (i.e., $y_r - y_l$) (c).

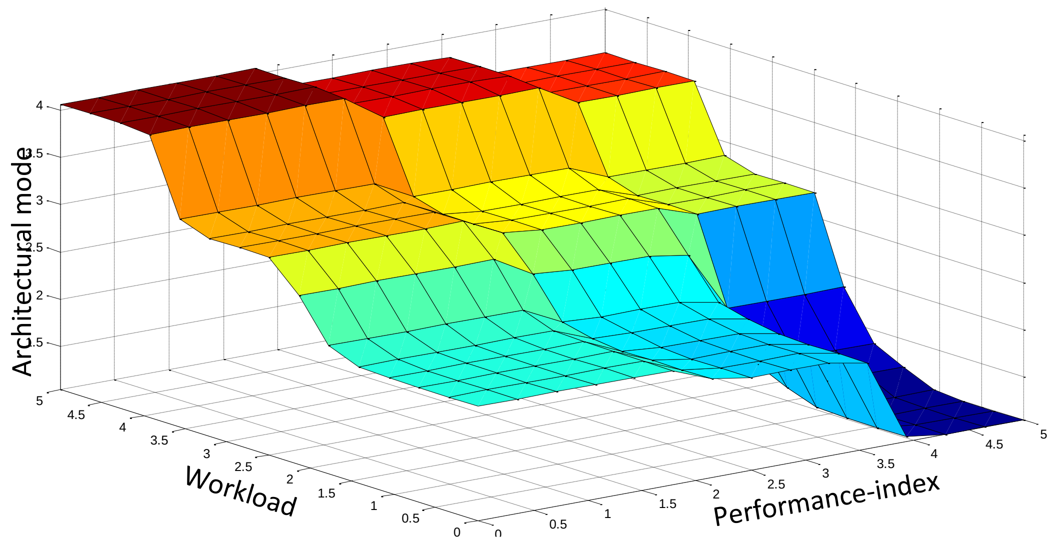


Figure 5.50. Output control surface of the T1 FLS.

5.5.7. Benefits of Using IT2 FLS over T1 FLS

The process of adaptation reasoning is a decision-making problem: choosing an appropriate mode for the running system given the environmental and system situation. As shown in Section 5.5.6.5, the output of the designed IT2 FLS is a boundary instead of a hard-threshold as in T1 FLSs (JM Mendel et al., 2000; Wu, 2012), compare Figure 5.48 and Figure 5.50. Therefore, as two vertical dashed lines in Figure 5.47 indicate, the decision for a mode switch can be more flexible providing a boundary. For instance, if the system requires operating with a high performance, the decision can be made based on the upper decision boundary, i.e. $y_r(2.5,3.5) = 3.0755$. As a result, $M_3 = Effort$ would be chosen. On the other hand, if the system requires saving energy, the decision can be made based on the lower boundary, i.e. $y_l(2.5,3.5) = 1.9934$. Accordingly, $M_1 = Idle$ would be chosen. In addition, if the system needs to achieve a compromise in performance and energy utilization, the decision can be made based on any value between the lower and upper boundaries. Note the lower and upper decision boundary as depicted in Figure 5.48 are for the blurring value of $\alpha = 0.5$. A different confidence interval can be derived by changing α. In addition, IT2 FLSs produce smoother behaviors, see the difference in Figure 5.48 and Figure 5.50. This provides a less disruptive approach (Linda & Manic, 2011). These characteristics and the ability to handle conflicting rules are the key benefits of IT2 FLSs over T1 FLSs that motivated us to choose it for adaptation reasoning of component connectors in this thesis.

Until this point of this chapter, we have described our approach, i.e., RobusT2, for designing a type-2 fuzzy logic controller, which is able to reason about connector adaptation at runtime. We can position the RobusT2 framework in existing adaptation reasoning approaches, see Table 5.1. In the following sections, we first evaluate some characteristics of the framework as we claimed in this thesis. We also discuss some threats to the validity of this work. Note that this is only a primary evaluation of the framework. A comprehensive evaluation of the framework, in a real-world context, is given in Chapter 7.

5.5.8. Experimental evaluations and validation

In this section, we present a number of experimental studies on an adaptive Web server to answer the following research questions:

- Q1 (Effectiveness). *Is it effective for avoiding rule-explosion*?

- Q2 (Robustness). *Is it robust against measurement noises*?

5.5.8.1. Adaptation rule reduction (Q1)

Rule explosion is a major disadvantage of rule-based reasoning approaches in self-adaptive software (Fleurey & Solberg, 2009; D Garlan et al., 2004). More specifically, rule-based reasoning suffers from scalability issues with respect to the management of very large rule sets. In this section, we show the effectiveness of adopting IT2 FLS in eliminating rule explosion.

We applied a rule reduction method, called SVD-QR (Liang & Mendel, 2000), to the FLS designed for autonomous adaptation reasoning. As you may recall from Section 5.5.6.1, the initial number of rules were 9. We applied the rule reduction to both T1 and IT2 FLS. For each of the two T1 and IT2 FLSs, we derived 20 different designs with slightly different parameters before rule reduction. We vary $m_{UMF,LMF}$ in triangular and $ul_{UMF,LMF}, ur_{UMF,LMF}$ in trapezoidal MFs to derive the 20 different designs. Each design of the FLSs was then rule-reduced using the SVD-QR method. Afterwards, we evaluated the performance of the rule reduction by measuring the difference between the outputs of each rule-reduced FLSs with corresponding original designs. We ran the two versions 10,000 times and compared their outputs using root means square metric (RMSE):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{d}(Y(x_1,x_2)^{(i)} - Y_r(x_1,x_2)^{(i)})^2}{d}} \tag{5.51}$$

, where $Y(x_1,x_2)$ is the output of the original design and $Y_r(x_1,x_2)$ is the output of the rule-reduced FLS. In other words, RMSE is a measure of distance between hyper-surfaces of the original and the rule-reduced FLS (cf. Figure 5.48). We summarize the measured RMSEs for the FLSs in Figure 5.51. The ranges of rules after reduction over the 20 realizations are $[3,4], [3,5]$ for T1 and IT2 respectively. We also calculated the range of reduced rules for different FLS designs as summarized in Table 5.13.

Table 5.13. The performance of rule reduction in different scenarios.

Scenario	**Setting (# antecedents, # rules)**	**# Rules after reduction**
1	2, 9	[3,5]
2	3, 27	[7,11]
3	4, 81	[23,31]
4	5, 243	[72,85]
5	6, 729	[221,256]

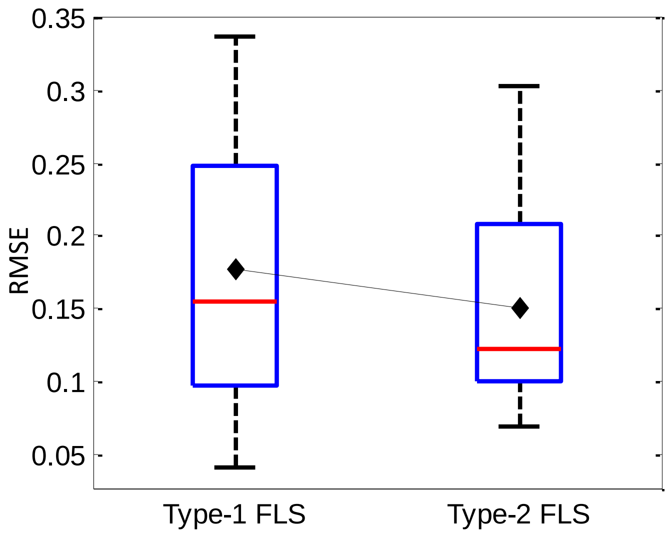


Figure 5.51. The RMSEs for the two FLS types over 20 designs.

Based on Table 5.13 and Figure 5.51, it is observed that:

- The rule reduction reduced the rules quite considerably.
- The rule reduction for adaptation reasoning was successful for both types of FLS without significant error.
- IT2 FLSs are more robust due to the lesser mean error and lesser variation in the estimation error.
- T1 FLSs in some realizations drop more rules in comparison with the IT2 FLSs. However, as it is reported in a number of seminal works (JM Mendel & John, 2002; JM Mendel, 2000), IT2 FLS original designs, i.e. before rule reduction, can be designed with fewer rules.

5.5.8.2. *Robustness testing of the reasoner (Q2)*

In this chapter, we presented a framework, called RobusT2, to handle the uncertainty related to the adaptation process of self-adaptive component connectors and we claim that the systems enhanced by such framework are resilient against the uncertainty leaking to the reasoning engine. In this section, we provide some experimental evidence to support this claim.

In this experiment, the robustness of a group of IT2 FLSs was examined against dynamic noise injected to the input measurement data with amplitudes from 1% to 10%. The injected noise is independent additive white Gaussian samples with zero mean and finite variance. This kind of noise is considered to cover the potential disturbances in self-adaptive software (Esfahani et al., 2011). We injected noises to the both input measurements, i.e. x_1, x_2. We ran RMSE measurements for each level of noise 100 times and for each RMSE measurement, we used 10,000 data items as input. Figure 5.52 shows RMSEs for the 10 levels of uncertainty for the original FLS with a blurring value of $\alpha = 0.5$. Figure 5.53 and Figure 5.54 show the RMSEs for the same FLS design but with blurring values $\alpha = 0.7, 0.95$ respectively. We also measured the RMSEs for the designed T1 FLS, see Figure 5.55.

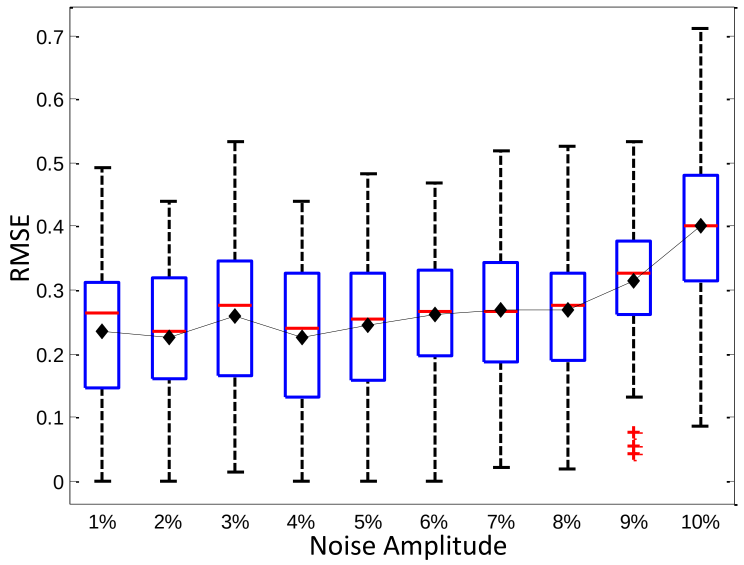


Figure 5.52. The RMSEs for the FLS under noise.

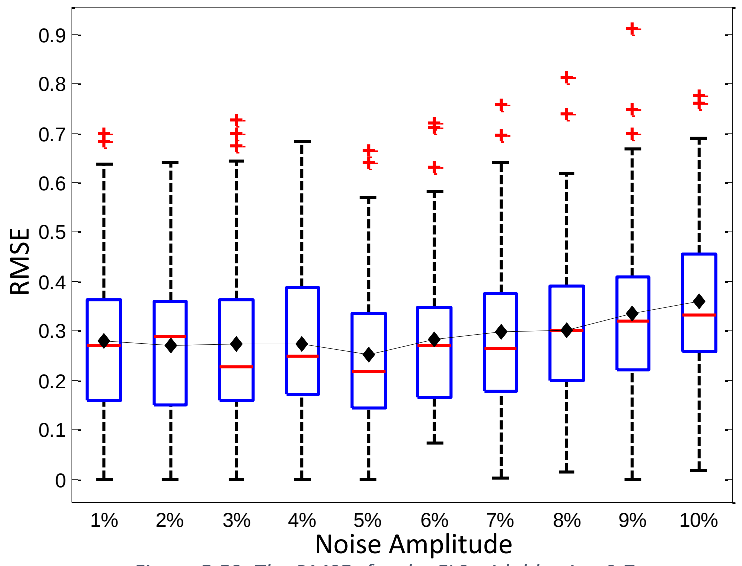


Figure 5.53. The RMSEs for the FLS with blurring 0.7.

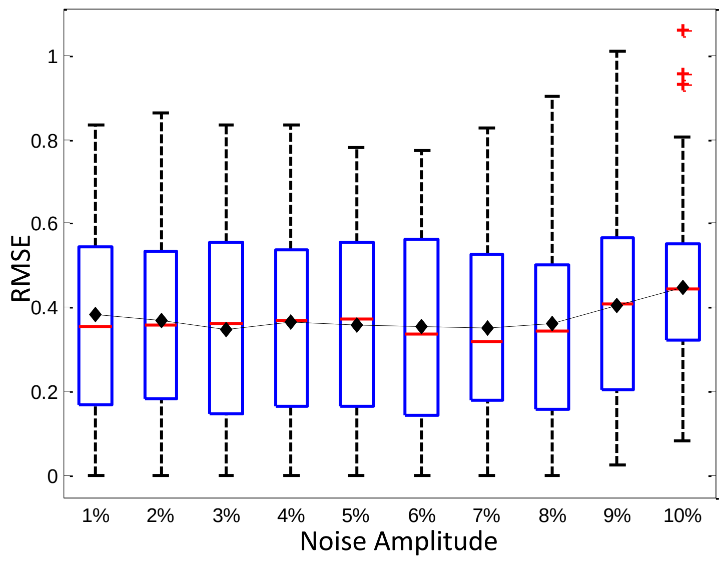


Figure 5.54. The RMSEs for the FLS with blurring 0.95.

Several observations can be drawn from the analysis of the results. The performance deteriorations when increasing the noise level from left to right in Figure 5.52 were negligible. In other words, the distribution of RMSEs exhibits the robustness of the IT2 FLSs when dealing with dynamic input noises. The increase in the value of blurring from 0.5 to 0.7 led to a better performance against noise (cf. Figure 5.52 and Figure 5.53). Interestingly, further increasing of the blurring from 0.7 to 0.95 results in performance degradation, worsening the performance of the original design (cf. Figure 5.52 and Figure 5.54). This is attributed to the overly wide FOUs (Linda & Manic, 2011). Therefore, selecting a proper value for the blurring parameter when designing a controller is critical. However, determining such a value is application-specific and there is no general-purpose benchmark. As another interesting observation, we also noticed a much better performance of the IT2 FLS for handling input noises compared to the T1 FLS, see the steep increase in errors in Figure 5.55.

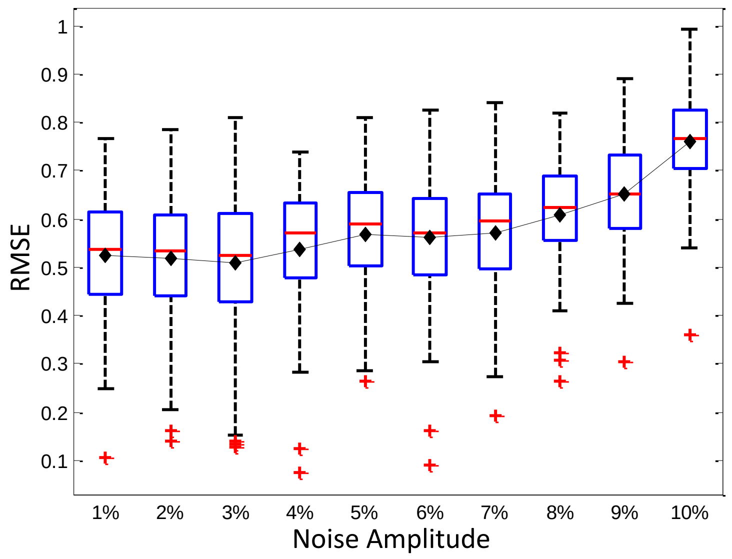


Figure 5.55. The RMSEs for the T1 FLS.

5.5.9. Limitations and future work

In the remainder of this section, we discuss the limitations of the RobusT2 framework and some fruitful avenues as future work. Note that we discuss the future work regarding this thesis (i.e., in the context of RCU framework) in the conclusion chapter (i.e., Chapter 8).

Dynamic update of adaptation rules. Runtime knowledge evolution and sharing is a topic that attracted little attention so far and is considered as an open challenge in self-adaptive software (Abbas et al., 2011). In this research, we have not discussed the dynamic updates to the adaptation mechanism. The inference engine chooses among a set of rules each time an adaptation cycle is performed. Therefore, it would be feasible to add new rules to the rule base at runtime. By adaptation cycle, we refer to the time from receiving input measurements until calculation of the output and sending it to the execution. This allows dynamic incorporation and removal of adaptation rules and indicates another avenue of future work. A promising approach is fuzzy rule learning (L. Wang & Mendel, 1992). Over time, the adaptation outcomes can be captured in a repository. Then by applying runtime efficient fuzzy rule learning, for example the WM method (L. Wang & Mendel, 1992), new rules can be learned and potentially improve the effectiveness of the adaptation mechanism. For instance, this facility can be used to avoid mode switches that have not historically resulted in a better system quality. The rule learning approaches can also be applied at design-time to assist users in rule specifications.

Integration with other uncertainty control approaches. As discussed in the background, there are different sources of uncertainty in the context of self-adaptive software. However, the approach proposed in this chapter can only handle the uncertainties regarding incomplete user knowledge. The integration of this approach with the existing approaches for controlling the uncertainty regarding other sources can be considered as future work. An end-to-end solution for controlling the uncertainties makes self-adaptive systems more resilient against noise and make them more dependable.

5.6. Conclusion

In this chapter, we provided an answer to **RQ2** that requires the development of a framework for reasoning about adaptation of component connectors. In this chapter, we introduced type-2 fuzzy logic for specifying non-functional requirements in self-adaptive software. We explained how type-2 fuzzy logic helps to model the uncertainty and impreciseness in requirements. We introduced a framework, called RobusT2, for autonomous adaptation reasoning of component connectors, using fuzzy logic systems. We explained each subsystem by using concrete examples. We also explained the details of the methodologies, which we have proposed in this thesis, for adaptation knowledge elicitation and transformation of this knowledge into fuzzy membership functions and fuzzy rules. A self-adaptation of a component connector in a component-based system was presented and validated using the RobusT2 framework. Finally, we discussed the results and limitations of this work, some insights and short-term future work.

Experimental results suggest that IT2 FLS can be used in this particular application. Specifically, the results affirms that T2 FLS controllers are better than T1 counterparts in controlling uncertainties involved in self-adaptive software, specifically measurement noise as input data. Future research for this proposed framework would be focused on tuning and optimization of the parameters of RobusT2. The results obtained are an additional motivation to use type-2 fuzzy logic control in other applications such as elasticity reasoning in cloud-based software applications. We presented some primary results of this application that is extracted from one of our papers (Jamshidi et al., 2014) as an example in this chapter.

Note that the templates for data collection as a part of our fuzzy knowledge elicitation and the operation details for data processing to derive appropriate artifacts in FLS design (i.e., the membership function and fuzzy rules) are given in Appendix A and Chapter 7 respectively.

Chapter 6

6. Adaptation Execution Mechanism for Component Connectors

"The art of progress is to preserve order amid change, and to preserve change amid order." – Alfred North Whitehead (1861-1947).

Contents

6.1. INTRODUCTION 192
6.2. ADAPTATION MECHANISMS IN SELF-ADAPTIVE SOFTWARE 192
6.2.1. Reflection-based mechanism 193
6.2.2. Aspect-oriented mechanism 193
6.2.3. Mode-based mechanism 194
6.2.4. Model-based mechanism 194
6.2.4.1. Architecture-based models at runtime 194
6.2.4.2. Variability models at runtime 195
6.2.4.3. Model composition at runtime 195
6.2.4.4. Goal-based requirement models at runtime 195
6.2.5. Summary of adaptation mechanism 196
6.3. DYNAMIC RECONFIGURATION MECHANISM FOR COMPONENT CONNECTORS 198

6.1. Introduction

In the previous chapter, we introduced a method to select from many connector configurations the one that is most appropriate to obtain some specific performance result based on fuzzy adaptation reasoning. In this section, we introduce a mechanism to enact the transitions from the current connector configuration to the target configuration derived from a variability-based reasoning technique that we borrowed from the software product line community. Considering the high heterogeneity of models and languages involved in software connectors, this chapter introduces an approach to derive reconfiguration actions using reasoning based on *graph theory* and *feature models*. We describe a mechanism for transforming the feature models corresponding to the connector modes to an executable reconfiguration plan using the principles of graph theory to derive the required reconfiguration actions. While the contributions of Chapter 4 and Chapter 5 belong to Analysis and Planning phases (in the context of MAPE-K control loop), the scope of this chapter, as illustrated in Figure 6.1, is to Execute the adaptation.

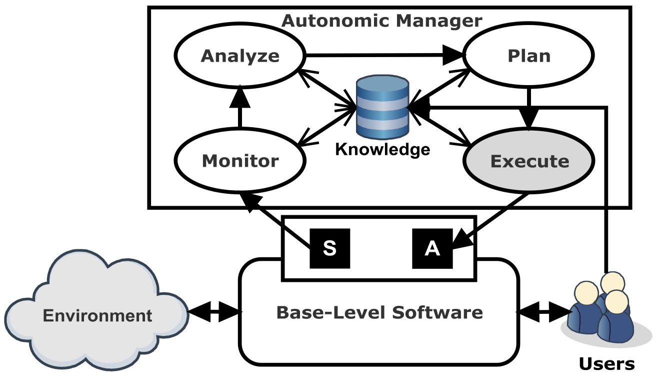


Figure 6.1. Scope of chapter 6.

The rest of the chapter is organized as follows. Section 6.2 reviews existing solutions that control how adaptations are performed in software systems. Section 6.3 defines a theory for representing component connectors based on various constructs in graph theory. Section 6.4 proposes a mechanism based on dynamic software product lines and the theory presented in Section 6.3 to enact the target mode to the current connector configuration at runtime.

6.2. Adaptation Mechanisms in Self-Adaptive Software

An adaptation mechanism in self-adaptive software is a component that controls how the adaptations are actually enacted in the system. For instance, in the case of the task queue connector, the adaptation mechanism is responsible for physically changing the channels in the connector. Note that the reasoner is responsible for *making* a *decision* and the adaptation mechanism *executes* the decision. In this section, we review existing solutions for enabling such adaptation execution.

In general, the adaptation mechanism modifies the changeable parts of the self-adaptive software system. Each changeable part can be replaced with several options. However, the mapping between the adaptation decision and actual reconfiguration is not one-to-one. The translation between high-level adaptation decisions to lower-level executable adaptation actions is the key concern of adaptation mechanism (or adaptation actuator). In this chapter, we use the terms "adaptation mechanism" and “adaptation actuator” interchangeably but basically they convey the same concept of executing (or enacting) the change on the piece of software.

There are different adaptation mechanisms, which are categorized as reflection-based mechanisms, aspect-oriented mechanisms, mode-based and model-based mechanisms.

6.2.1. Reflection-based mechanism

Computational reflection is introduced as a way to reflect the overall architecture of a software-intensive system (Cazzola, Savigni, Sosio, & Tisato, 1998). When this capability was put in forward, the self-adaptive software community started to utilize this capability in order to make use of architectural configuration information at runtime for adaptation reasoning. The critical advantage of this capability is to allow the querying and dynamic reconfiguration of architectural elements in the underlying software. A study shows that this capability conforms to the foundations of self-adaptive software (Andersson, de Lemos, Malek, & Weyns, 2009) and another study reveals that this capability is gaining momentum in architecture-centric software evolution and this trend is also increasing (Jamshidi et al., 2013).

Another advancement in this area was the introduction of component-based technology, which proposed the notion of configurable software systems by adding, removing or replacing their constituent components, connectors or a combination (architectural configuration) of them. A number of reflective component models exist which support dynamic loading and unloading of components such as: FRACTAL (Bruneton, Coupaye, Leclercq, Quéma, & Stefani, 2006), OpenCOM (Coulson, Blair, Clarke, & Parlavantzas, 2002), SOFA 2 (Bures, Hnetynka, & Plasil, 2006), OSGi (“OSGi Alliance,” 2014), EJB and so on (Crnkovic, Sentilles, Vulgarakis, & Chaudron, 2011).

Computational reflection, component models and component-based architectures improve the construction of adaptation mechanisms to achieve dynamic adaptive systems (Coulson et al., 2002). The mechanisms utilize a component architecture to visualize the overall structure of the system comprising components and connectors. These architectural elements are causally connected to actual running components in the system, whereby changes in one result in changes in the other.

Despite the flexibility that is introduced by reflective technology, the adaptations at runtime are still challenging. One challenge is that the reconfiguration of the adaptive software was formerly performed by ad-hoc complex programs. This challenge is addressed by the introduction of a manipulation language on top of component models. For instance, the *FScript* language for FRACTAL component model or the *Plastik* language for OpenCOM component model. By adopting the languages, one can write flexible adaptation mechanisms.

6.2.2. Aspect-oriented mechanism

Aspect-orientation (AO) was introduced to solve the problem of modularizing the crosscutting concerns in software systems (Kiczales, 1996). Although several approaches in terms of language constructs have been proposed to implement aspects, *point-cuts* and *advices* are common. Point-cuts are the placeholder for advices, which are the realization of crosscutting concerns. AO can be considered as complementary with respect to component platforms and computational reflection. Different compositional approaches such as AO to adapt middleware platform are reviewed in (McKinley, Sadjadi, Kasten, & Cheng, 2004).

The mechanisms for composition that enable aspects to be woven into the software systems have been used as adaptation mechanisms for self-adaptive software (David & Ledoux, 2006; Pawlak et al., 2004). The ultimate goal of AO is to weave and unweave adaptation plan in the evolution process (David & Ledoux, 2006). The idea is realized by corresponding each adaptable parts of the software to the aspects. Then these aspects can be woven at runtime when the need for adaptation is raised (Surajbali, Coulson, Greenwood, & Grace, 2007).

AO is an appropriate approach to realize an adaptation mechanism in self-adaptive software. Nevertheless, some shortcomings prevent the full adoption of this mechanism. The first problem is known as the AO *evolution paradox* (Tourwé, Brichau, & Gybels, 2003). This problem occurs when the aspects and the underlying system evolve separately. The second one is known as *aspect interference* (Katz & Katz, 2008) and happens when several advices are woven into the same point-cut or when an advice cancels out other advice effects.

6.2.3. Mode-based mechanism

In early development of self-adaptive software, the mechanisms for change at the architectural level were limited. These mechanisms use predefined architectural configurations (typically called system modes) which are hardcoded using architectural descriptions. A mechanism for adaptations were boiled down to switching between the systems modes by changing some parameters (Hirsch et al., 2006).

A *mode* abstracts a specific set of services that must interact in order to perform a specific functionality of a system (Hirsch et al., 2006). In other words, each mode corresponds to a specific behavior of a system. A mode determines the structural constraints that determine a system configuration at runtime. Therefore, mode switching or change of mode can be considered as a mechanism for adapting software systems. Hirsch et al. (Hirsch et al., 2006) introduce the notion of mode and mode transition as explicit elements of architecture description. They aim for description and verification of complex adaptive systems. Borde et al. (Borde, Haik, & Pautet, 2009) investigate the notion of *operational mode* in component-based systems to specify system behaviors and how to switch from one mode to another one at runtime. A number of studies examine the mode change propagation protocol to enable and formally verify the mode switch at runtime (Bertrand, Déplanche, Faucou, & Roux, 2008; Pop, Plasil, Outly, Malohlava, & Bures, 2012; Yin, Carlson, & Hansson, 2012). The studies implement the mode switch protocols in specific architecture description languages. The main target domain of these approaches is resource constrained embedded systems. This adaptation mechanism, however, cannot handle unforeseen architectural configurations.

6.2.4. Model-based mechanism

The main idea in model-based approaches is to abstract the adaptation mechanism from ad-hoc reconfiguration scripts and reflective platforms.

6.2.4.1. Architecture-based models at runtime

Early approaches proposed the use of architectural models to control the adaptation process (Oreizy et al., 1998). Another architecture-based approach (D Garlan et al., 2004) hardcoded the adaptation mechanism through architecture evolution by primitive change operator and composite adaptation strategies.

Recent work extends the lifetime of architectural models to runtime and uses them to derive the needed adaptations. In this way, models at runtime (Blair et al., 2009) are used as artifacts that control architecture-based adaptations. Moreover, they are used to verify the adaptation at runtime. Note that other models such as behavioral or stochastic descriptions of the systems can also be used at runtime.

6.2.4.2. Variability models at runtime

Variability models in software product line (SPL) are used to model the various parts of a software product. These variable parts are called *variation points* and may represent different architectural elements of a system that may differ from one product to another product in the same family. As a result of this capability to model the changeable parts, some approaches proposed to use SPL to model the self-adaptive software's underlying structure (C Cetina, Haugen, & Zhang, 2009; Fleurey & Solberg, 2009; Perrouin & Chauvel, 2008).

In this case, an ideal association would be "a feature is a component". Therefore, adding, updating or removing a feature would simply lead to an addition, replacement or removal of a component. The approaches that follow this association are problematic. By definition, features are orthogonal to themselves and to the solution space (Jean-Baptiste, Maria-Teresa, Jean-Marie, & Antoine, 2013). Such approaches break this principle. Thereby, they are subject to feature interaction conflicts (Apel & Kästner, 2009).

The real benefit of variability models appear at runtime when dynamic SPL (DSPL) are used to derive new products on the fly (Hallsteinsen, Hinchey, & Schmid, 2008). A DSPL is basically a SPL that is kept alive during runtime, and then when a need for change arises, an adaptation is computed (on the basis of *diff* between architectures) from the variability model that is present at runtime. The changes are then translated into architectural model operations, which assists the real reconfiguration of the adaptive system. However, this translation is not always straightforward and may lead to some limitations of the derived configuration.

6.2.4.3. Model composition at runtime

A more recent approach combines the previous propositions in using architectural and variability models at runtime, aspect oriented mechanisms and causal links (B Morin, Fleurey, & Bencomo, 2008; Parra, Blanc, Cleve, & Duchien, 2011). In general, they use models at runtime to describe the architectural configuration of the system and its varying parts. Dynamic aspects reify variability and model composition transforms a configuration derived at runtime to another adapted configuration. Finally, causal links are used to update the running system.

These approaches such as the ones presented in Section 6.2.4.2 enable model adaptations indirectly based on adaptation plans deduced from structural differences. Thereby, those approaches suffer from the lack of explicit tailoring of the adaptation, as they would be unable to handle the case described in (Jean-Baptiste et al., 2013). They, on the other hand, consider features not as user visible aspects, but as ordered transformations that can be reified at runtime to generate adaptations.

6.2.4.4. Goal-based requirement models at runtime

A majority of the work in self-adaptive software is devoted to the adoption of architectural models that enable flexible adaptations (Jamshidi et al., 2013; D Weyns & Ahmad, 2013; Danny Weyns, Iftikhar, Malek, & Andersson, 2012). On the other hand, much less work has been carried out in utilizing requirements models for self-adaptive software (B. Cheng et al., 2009). Specifically, goal-based modeling is well suited to representing alternative behavior when environmental changes occur (Goldsby, Sawyer, Bencomo, Cheng, & Hughes, 2008; Lapouchnian, Yu, Liaskos, & Mylopoulos, 2006; Yu, Lapouchnian, & Liaskos, 2008). These approaches only enable adaptation with a limited set of alternative behaviors, which should be fixed at design-time. However, there are some more flexible approaches such as RELAX (N Bencomo & Ramirez, 2012).

At runtime, the requirements are assessed to evaluate the conformance of the runtime behavior to the specified requirements. As a result, a violation of requirements may trigger an adaptation. However, the adaptation decisions may not be made based on crisp values “yes” or “no”, but it may have to be made stochastically based on partial (dis)satisfaction of requirements (N Bencomo & Belaggoun, 2013).

6.2.5. Summary of adaptation mechanism

Table 6.1 summarizes the adaptation mechanisms, which we reviewed earlier, for enabling self-adaptation of software systems. In this table, a check mark (√) indicates where the approach proposes solutions or deals with the criteria, and a blank ( ) in the opposite case.

The approaches we have reviewed in this section offer support for both design and runtime adaptations. Some of them use variability modeling and context information as well as models at runtime, reflective platforms, or dynamic aspects that allow them to have both source code manipulations for the design adaptations and dynamic reconfigurations for the runtime adaptations. Some of them also use variability models for modularizing and defining adaptation plans as well as contextualizing annotations to define concrete events at runtime. There are also some other approaches at the code level in programming languages that mainly focus on modularity. In addition, few of the approaches focus on architectural modes that have been derived at design-time and will be used as a means to enable runtime adaptations.

However, there are still two important issues left unaddressed. First of all, the approaches do not offer a process from feature modeling and architectural modes to runtime adaptations. This means that design and runtime adaptation processes do not have many elements in common. Moreover, artifacts used for building applications are treated in a different manner to artifacts used to achieve dynamic adaptations.

Table 6.1. Classification and comparison of adaptation mechanisms.

Reference	Scope				Reasoning Mechanism				Domain				
	Model	Architecture	Code	Requirement (Spec)	Reflection-based	AO	Mode-based	Model-based	Mobile	Embedded System	Smart-*	Robotic	General-purpose
(Cazzola et al., 1998)		√			√								√
(Andersson et al., 2009)		√			√								√
(McKinley et al., 2004)		√			√	√							√
(David & Ledoux, 2006)		√			√	√							√
(Pawlak et al., 2004)		√			√	√							√
(Surajbali et al., 2007)		√			√	√							√
(Hirsch et al., 2006)	√	√					√	√					√
(Borde et al., 2009)				√			√				√		√
(Yin et al., 2012)		√			√		√						√
(Perrouin & Chauvel, 2008)		√						√					√
(C Cetina et al., 2009)		√						√			√		
(Jean-Baptiste et al., 2013)		√						√					√
(Apel & Kästner, 2009)				√				√					√
(B. Cheng et al., 2009)				√				√			√		√
(Goldsby et al., 2008)				√				√					√
(Lapouchnian et al., 2006)				√				√					√
(Yu et al., 2008)				√				√					√
(N Bencomo & Ramirez, 2012)				√				√			√		√
(N Bencomo & Belaggoun, 2013)				√				√					√
Our approach		√					√	√					√

6.3. Dynamic Reconfiguration Mechanism for Component Connectors

While adaptation reasoning (Chapter 5) can be developed abstractly and independently of the language and the platform with which connectors are realized, adaptation enactment has close ties to the underlying connector model. Therefore, we needed to select a target connector model for implementation of the change actuator. We selected Reo (Arbab, 2004), which is a coordination model for realizing exogenous component connectors (Lau, Elizondo, & Wang, 2005), and hence, a perfect fit to this work.

In the remainder of this chapter, we describe the reasoning mechanism for deriving adaptation actions on top of Reo for effecting adaptation decisions. In this section, we describe the structural modeling (Section 6.3.1) of the connector and change reasoning (Section 6.3.2) based on the principle of graph theory to prepare the context to introduce the effectuation mechanism based on the principles of dynamic software product lines in Section 6.4.

6.3.1. Connector configurations

A component connector, in the context of this research, corresponds to a coordination pattern (Arbab, 2004; N Oliveira & Barbosa, 2013; Nuno Oliveira & Barbosa, 2013) on architectural elements (e.g. components) that perform I/O operations through that connector. In other words, here, the term connector is adopted to name entities that can regulate the interaction of (potentially) heterogeneous components. Thus, connectors must deal with exogenous coordination, handling all those aspects that lie outside the scope of individual components (Bruni et al., 2013). This means coordination pattern are without the knowledge of those entities. A coordination pattern is formally given as a graph of *channels* whose nodes represent the points for interactions between channels. The edges of this graph are represented with channel types and channel identifiers. To provide a concrete illustration of this approach, we utilize the Reo coordination model (Arbab, 2004). Therefore, a channel is considered here as a Reo channel (Arbab, 2004).

A number of component connector models exist that we reviewed in Section 6.2.1, but we decided to use Reo, a powerful coordination language introduced by the CWI research group Foundations of Software Engineering (SEN3). The choice for this language is very appropriate, since, for instance, Reo covers different coordination aspects, such as synchronous vs. asynchronous communication, buffering, filtering and data manipulation, context-dependent behavior, and mobility (Arbab, 2004). The various constraints from different functionalities put high demands on the synchronizing aspect, a powerful feature of Reo. Moreover, Reo has a stochastic extension (Moon, 2011) with which we can simulate and develop systems with stochastic behaviors and incorporating performance aspects. For Reo there are also some tools for modeling system architecture, simulating system behavior, formal operational semantic languages and facilities to derive system models which we exploit in our approach. Reo also has a very supporting and active research community. In summary, we believe that this choice offers the following opportunities: (i) Due to its feature-rich models, Reo offers powerful means for describing the coordination that turns a set of components and connectors into a coherent working application. (ii) *Stochastic Reo* is a good choice for representing reliability and performance aspects as our main objectives. (iii) *Proven formal semantics* used in Reo which we can extend and build the formal aspects of our contribution. (iv) *Available open source tools* that we can exploit for our own purposes. (v) *An active and supportive research community* that motivated us to contribute.

In the Reo model, channels are primitives, out of which more complex and composite component connectors are constructed. A connector channel is directional (except one channel type) with a unique identifier and specific semantics (i.e. coordination protocol). A channel in this model accepts an I/O operation (data flow) on its *source end* and dispenses it from its *sink end*. Figure 6.2 illustrate the basic channel type in the Reo coordination model. Note that Reo supports an *extensible set of channels* (Arbab, 2004), each exhibits a unique behavior with a well-defined semantics. However, for the purpose of this research, we only consider the construction of component connectors based on the primitive channels represented in Figure 6.2.

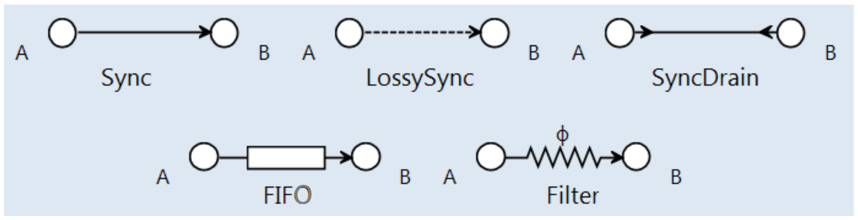


Figure 6.2. Primative connector channels.

Each channel has its own semantics as defined in Table 6.2. The $Sync$ channel transfers data from source end to sink end whenever there is an I/O request at both ends synchronously. The $SyncDrain$ channel accepts data synchronously at both source and sink ends and losing it. The $LossySync$ channel behaves the same, but data may be lost whenever there is a request at the source end but there is no request at the sink end. The $Filter$ channel is similar to $LossySync$, but deterministically only when an item satisfies the filter constraint, the channel delivers it to the sink end. In contrast to the previous channel types, a $FIFO$ channel buffers data inside a memory position and when a request arrives at the sink end, it delivers it to that end. Similarly, $FIFO(F)$ delivers the stored item to its sink end and cleans the buffer.

Table 6.2. Primative channel behavior.

Channel Type	Behavior
$\boldsymbol{Sync}$	Atomically gets an item from its source end A and delivers it to its sink end B.
$\boldsymbol{SyncDrain}$	Atomically gets an item from both source and sink ends A, B and loses it.
$\boldsymbol{LossySync}$	Atomically gets an item from its source end A and non-deterministically, either delivers it to its sink end B or loses it.
$\boldsymbol{Filter}$	Atomically gets an item from its source end A and if the item satisfies the filter constraint ϕ delivers it to its sink end B and loses it otherwise.
$\boldsymbol{FIFO}$	Atomically gets an item from its source end A and stores it in its buffer.
$\boldsymbol{FIFO(F)}$	Atomically fetches the item from its buffer and delivers it to the sink end B.

A component connector is constructed by gluing the channel ends together. In such a composed structure, data items flow through channels and past nodes connecting them. Usually, the interacting parties supply the data that flows through the connectors they are connected to. Note that here the interacting parties coordinated through connectors and connected to them are *components* of a component-based software system. Each connector has an interface. Such an interface comprises of the boundary nodes of a connector: components give and take data only to and from boundary nodes.

Through connecting channel ends, different types of nodes appear as follows: (i) source node, if it connects only source ends; (ii) sink node, if it connects only sink ends; (iii) mixed node, if it connects both source and sink nodes.

Figure 6.3 depicts a sequencer connector, which is a composition of five channels with two different types. It has one source node (A), three sink nodes ($O1, O2, B$) and two mixed nodes ($N1, N2$) differentiated by grey color. Generally, mixed nodes are figuratively internal nodes in connectors and the source and sink types are boundary nodes making the interface of a connector. In this example, the nodes $\{A, O1, O2, B\}$ are the boundary nodes comprising the interface of the sequencer connector depicted in Figure 6.3.

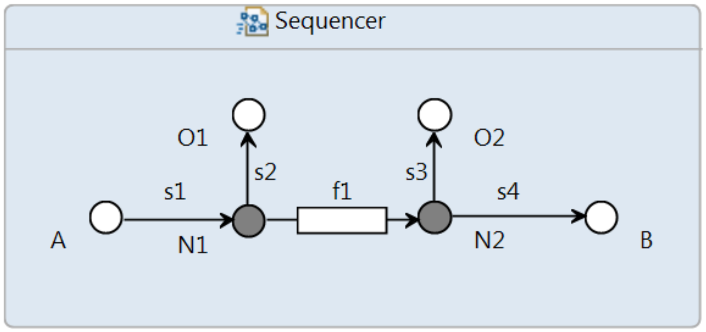


Figure 6.3. The Sequencer connector.

Figure 6.3 represents a sample architectural configuration of a component connector that we intend to formally characterize with the graph-based constructs in the following sections. The main intention of doing this is to provide an appropriate level of abstraction to reason about structural changes in the connectors.

6.3.1.1. Connector configuration

Let N, I, O, Id, T, respectively, denote a set of boundary nodes comprising source (I) and sink (O) nodes, channel identifiers and channel types. We also consider the internal components as a part of a connector architectural configuration (cf. Figure 6.4). Each internal component has a name, type, a set of source ends and sink ends known as ports. Let P, Id, CT denote a set of ports, component identifiers and component types respectively. The connector architectural configuration is represented by the following definition as first introduced in (N Oliveira & Barbosa, 2013; Nuno Oliveira & Barbosa, 2013) as the notion of ***coordination pattern*** and we extend it here as the notion of ***connector configuration*** for the purpose of this research.

Definition 34. A ***connector configuration***, C_{id}, is defined as a triple

$$C_{id} \overset{\text{def}}{=} \langle I, O, R \rangle$$
$$R \subseteq (N \times Id \times T \times N) \cup (N \times (P \times Id \times CT \times P) \times N) \quad (6.1)$$
$$I \neq \emptyset, O \neq \emptyset, I \subseteq N, O \subseteq N$$

, where R is a graph on connector ends N whose edges are instances of primitive channels $id \in Id$ with specific type $t \in T$. I and O are the sets of source and sink ends in graph R. For example, the sequencer can be represented as follows:

$$C_{seq} =< \{A\}, \{O1, O2, B\}, \{(A, s1, Sync, N1), (N1, s2, Sync, O1), (N1, f1, FIFO, N2), (N2, s3, Sync, O2), (N2, s4, Sync, B)\} > \quad (6.2)$$

For representing unidirectional channels, we need to utilize a special notation. For example, $Drain$ has two source ends, but it has no sink ends. We use $\boxdot \in N$ to represent absence of I/O operations. Therefore, a $Drain$ channel can be represented as follows:

$$C_{drain} =< \{I1, I2\}, \emptyset, \{(I1, d, Drain, \boxdot), (I2, d, Drain, \boxdot)\} > \tag{6.3}$$

Since the sets I and O can be inferred based on R by identifying the nodes that appear either as the first or fourth element, the definition of connector configuration can be relaxed by dropping the sets I and O from the triple. However, here, we use the triple as defined in **Definition 34** for specifying component connectors.

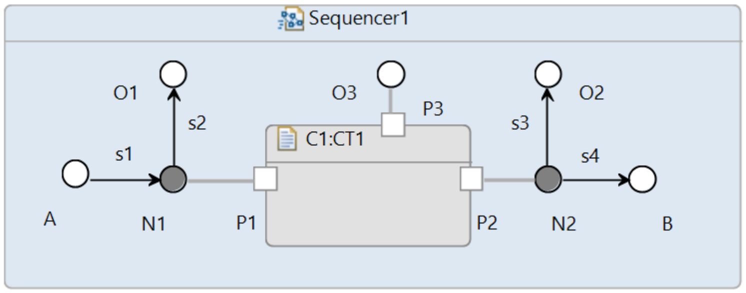


Figure 6.4. A sequencer connector with internal component.

Figure 6.4 illustrates a variation of sequencer connector, which utilizes an internal component $C1$ with type $CT1$. As a result, this connector can be formally defined as follows:

$$\begin{aligned}&C_{seq1} =< \{A\}, \{O1, O2, O3, B\}, \{(A, s1, Sync, N1), (N1, s2, Sync, O1),\\&(N1, P1, C1, CT1, P3, O3), (N1, P1, C1, CT1, P2, N2), (N2, s3, Sync, O2),\\&(N2, s4, Sync, B)\} >\end{aligned} \tag{6.4}$$

As a matter of fact, internal components are architectural elements to abstract away and hide part of its internal structure for reuse purposes. Transforming a connector to a component with the same semantics is straightforward. For example, Figure 6.5 represents the sequencer in the form of a component. This, component representation of a connector abstracts away the details of the connector and provides only four ports $A, B, O1, O2$ for other entities (i.e., connectors or component instances) to write to or read from. The nodes belonging to I are transformed to source ends of the component such as A. Moreover, the nodes belonging to O are transformed to sink ends such as $O1, O2, B$.

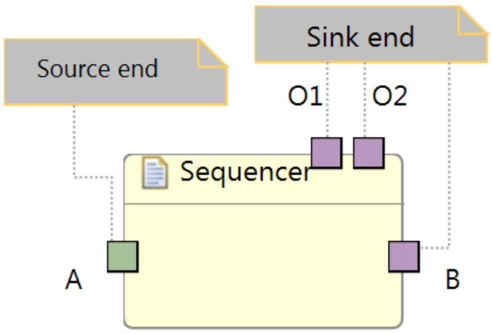


Figure 6.5. Software component corresponding to the sequencer connector.

6.3.1.2. Architectural invariants for connectors

In order to avoid incorrect configurations, we enforce a number of architectural invariants for component connectors as expressed in Table 6.3.

Table 6.3. A list of architectural invariants for connector configurations.

ID	Entity	Invariant description
Inv1	$\boxdot$	This port cannot be connected to other ports.
Inv2	Channel	Only a single channel is allowed to connect two nodes.
Inv3	$\boldsymbol{Id}$	A name can only be associated to a node, port or channel type.
Inv4	$\boldsymbol{Id}, \boldsymbol{R}$	A name can be used at most in two tuples in R.
Inv5	$\boldsymbol{I}, \boldsymbol{O}, \boldsymbol{R}$	The nodes belonging to I, O can only be used once in tuples in R.
Inv6	$\boldsymbol{I}, \boldsymbol{R}$	The nodes in I can only be used as the first element in tuples in R.
Inv7	$\boldsymbol{O}, \boldsymbol{R}$	The nodes in O can only be used as the fourth element (last element) in tuples in R.
Inv8	$\boldsymbol{I}, \boldsymbol{O}, \boldsymbol{R}$	The nodes that not belong to either I or O must be repeated more than one time in tuples in R.

The intention behind such an enforcement of architectural invariants is to preserve well-defined structural properties of connectors and to ensure that suitable architectural principles are maintained as invariants during the evolution of a given connector.

6.3.1.3. Structural constructs of component connectors

In this section, we define a number of structural constructs by adopting the inherent principles of graph theory. We basically define the constructs that we are going to introduce in this section based on the structures that can be formally defined as graphs. This enables us to define composed constructs based on the composition of primary constructs. It also enables us to reason about structural changes based on well-defined mathematical operations applied on the constructs. The structure change reasoning enables us to reason about mode-based adaptation as we propose in Section 6.4.

Definition 35. A ***sub-connector***, SC_{id}, of connector $C_{id} =< I, O, R >$ with reference node N, is defined as a tuple

$$\begin{aligned}
&SC_{id} \overset{\text{def}}{=} \langle I', O', R' \rangle \\
&\mathrm{N}' \subseteq N \\
&R' \subseteq (N' \times Id \times T \times N') \cup (N' \times (P \times Id \times CT \times P) \times N') \\
&R' \subseteq R \\
&I' \subseteq N, O' \subseteq N
\end{aligned} \tag{6.5}$$

, where R' is a partially connected sub-graph of a connector with graph R. Note that $I'\ and\ O'$ can have no intersection with $I\ and\ O$ respectively. In the other direction of **Definition 35**, a ***super-connector*** of $C_{id} =< I, O, R >$ is a connector $SupC_{id}$ of which C_{id} is a sub-connector.

This definition of the sub-connector construct gives a rise to different types of sub-connector with respect to the boundary interfaces of connectors. We can imagine four different sub-connectors as follows:

- ***I-Interface sub-connector***: A sub-connector with $I' \cap I \neq \emptyset$ but $O' \cap O = \emptyset$. For example, $SC_{seq1} =< \{A\}, \{N1\}, \{(A, s1, Sync, N1)\} >$ as a sub-connector of C_{seq} as specified in (6.2).
- ***IO-Interface sub-connector***: A sub-connector with $I' \cap I \neq \emptyset$ and $O' \cap O \neq \emptyset$. For example, $SC_{seq2} = < \{A\}, \{O1\}, \{(A, s1, Sync, N1), (N1, s2, Sync, O1)\} >$ as a sub-connector of C_{seq} as specified in (6.2).
- ***O-Interface sub-connector***: A sub-connector with $I' \cap I = \emptyset$ but $O' \cap O \neq \emptyset$. For example, $SC_{seq3} = < \{N1\}, \{O1\}, \{(N1, s2, Sync, O1)\} >$ as a sub-connector of C_{seq} as specified in (6.2).
- ***Internal sub-connector***: A sub-connector with $I' \cap I = \emptyset$ and $O' \cap O = \emptyset$. For example, $SC_{seq4} =< \{N1\}, \{N2\}, \{(N1, f1, FIFO, N2)\} >$ as a sub-connector of C_{seq} as specified in (6.2).

There are also a number of special variants of sub-connector construct, which preserve different properties of the original connector.

Definition 36. A ***basic construct*** is a special variant of sub-connector (**Definition 35**) that can be replicated in order to grow the capacity of a connector without changing its behavior. This will cause some structural change in the connector configuration by changing either of the sets in the configuration triple (cf. **Definition 34**).

To be more specific, a basic construct (**Definition 36**) may be replicated without influencing $I\ or\ O$. For example, consider a DynamicFIFO connector as illustrated in Figure 6.5. This connector is defined as $C_{DynaFIFO1}$before the change and as $C_{DynaFIFO2}$ after the change. As it is evident, there is no change in either $I\ and\ O$ and just the capability of this connector is increased without changing its behavior, which is exposed by its ports.

$$\begin{aligned} C_{DynaFIFO1} &=< \{A\}, \{B\}, \{(A, f1, FIFO, B)\} \\ C_{DynaFIFO2} &=< \{A\}, \{B\}, \{(A, f1, FIFO, N1), (N1, f2, FIFO, B)\} \end{aligned} \tag{6.6}$$

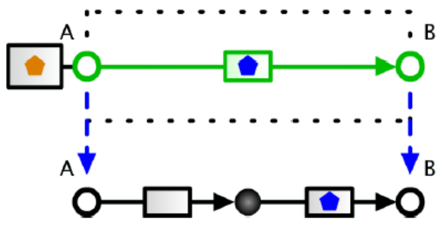


Figure 6.6. A DynamicFIFO connector.

In a more formal way, applying a basic construct to a connector makes the initial connector an embedded structure into the adapted connector.

Definition 37. An ***embedded*** sub-connector $ESC_{id} \overset{\text{def}}{=} \langle I', O', R' \rangle$ of $C_{id} = \langle I, O, R \rangle$ can be determined by an embedding function as a one-to-one function from N' to N such that every channel in R' corresponds to a path in R.

In some cases, we need to define a sub-structure of a connector given that we must preserve all the nodes, but we only need a subset of the channels connecting the nodes. This construct corresponds to a maximum clique ("Maximum clique," 2014) in graph theory.

Definition 38. A ***factor*** $FC_{id} =< I', O', R' >$ is a special kind of sub-connector of a connector $C_{id} =< I, O, R >$ with reference nodes N with the following properties:

$$\begin{aligned} &N = N' \\ &R' \subseteq R \end{aligned} \tag{6.7}$$

In some circumstances, we need a construct to preserve the same channels between nodes that has been defined in the initial connector. A sub-connector FFC_{id} is an induced (or full) sub-connector of C_{id} if it has exactly the channels that appear in C_{id} over the same node set.

Definition 39. A ***full*** (***induced***) sub-connector of $C_{id} =< I, O, R >$ with reference nodes N is a connector $FFC_{id} =< I', O', R' >$ with the following properties:

$$\begin{aligned} &N' \subseteq N \\ &\forall (n1', id', t', n2') \in R' : (n1, id, t, n2) \in R \Leftrightarrow (n1', id', t', n2') \in R' \end{aligned} \tag{6.8}$$

In other words, FFC_{id} is an induced sub-connector of C_{id} if it has exactly the same channels that appear in C_{id} over the same reference nodes N'.

We now need to define the means of traversing the constructs which we have defined for component connectors.

Definition 40 (path in connector). A ***path*** in a connector is a sequence of channels, which connect a sequence of nodes. A path in connectors is *finite* and always has a first node, called *start node*, and a last node, called *end node*. Both of these are called *terminal nodes* and the other nodes are called *internal nodes* of the path. A *cycle* is a path where the start node and end node are the same.

To be more formal, let us consider the concept in **Definition 34** and more specifically Equation (6.1). Given a connector configuration $C_{id} \stackrel{\text{def}}{=} \langle I, O, R \rangle$, a path in this configuration is a triple:

$$\begin{aligned} &P_{C_{id}} \stackrel{\text{def}}{=} \langle I_P, O_P, R_P \rangle \\ &|I_P| = |O_P| = 1 \\ &R_P \subseteq R \end{aligned} \tag{6.9}$$

, where I_P is the start node and O_P is the end node. The elements R_P form an *ordered list* of a subset of channels in R in a way that only two nodes are repeated once in the channel tuples, one as the first element and the other as the fourth element. The rest of the nodes are repeated once as first element and once as the fourth element. If a path is a cycle, then $I_P = O_P$.

Figure 6.7 represents a path in the sequencer connector in Figure 6.3. It can be represented as $< \{A\}, \{B\}, ((A, s1, Sync, N1), (N1, f1, FIFO, N2), (N2, s4, Sync, B)) >$.

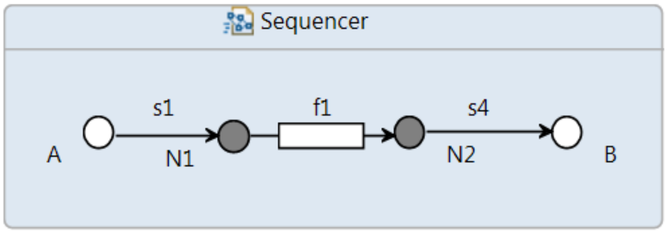


Figure 6.7. A path in Sequencer connector.

Definition 41. The ***size*** of a connector $C_{id} =< I, O, R >$ is the number of channels in it, denoted by $|R|$ or $\|C_{id}\|$.

Definition 42. The ***length*** of a connector $C_{id} =< I, O, R >$ is the size of the longest path $P_{C_{id}}$ in the connector, denoted by $\ell(C_{id})$.

As an example the size and the length of the sequencer connector represented in Figure 6.3 is five and three respectively.

Definition 43. For a node N, the number of channel sink endpoints that meet in the node is called the ***indegree*** (denoted as $deg^{-}(N)$) of the node and the number of source endpoints that meet in the node is its ***outdegree*** ($deg^{+}(N)$).

For all nodes in $i \in I$ (cf. **Definition 34**), $deg^{-}(i) = 0\ and\ deg^{+}(i) > 0$. For all nodes in $o \in O$, $deg^{+}(o) = 0\ and\ deg^{-}(N) > 0$. For the rest of nodes in $n \in N$, $deg^{-}(n) > 0\ and\ deg^{+}(n) > 0$.

Definition 44. A connector $C_{id} =< I, O, R >$ is ***linear*** if

$$\begin{aligned} &|I| = |O| = 1 \\ &for\ n \in N, n \notin I, O, deg^{+}(n) = deg^{-}(n) = 1 \end{aligned} \qquad (6.10)$$

For a linear connector $C_{id} =< I, O, R >$, $\ell(C_{id}) = \|C_{id}\|$.

6.3.1.4. Connector composition

In this section, we define a number of operations on the constructs that we have defined already in the previous sections to produce new connectors from the primary connector constructs. In graph theory, there are some so-called "editing operations" ("Graph operations," 2014) that create a new graph from the original one by a simple, local change, such as addition or deletion of a vertex or an edge, merging and splitting of vertices, edge contraction, etc. However, the main focus of this section is to define compositional operations to derive a composed connector from primary ones.

Two connectors can be composed in different ways. The most intuitive way of composition is setting them in parallel without creating any interconnection between them.

Definition 45. A ***parallel composition*** (juxtaposition) of two connectors $C_{id1} =< I_1, O_1, R_1 >$ $and\ C_{id2} =< I_2, O_2, R_2 >$ is described as follows:

$$\begin{aligned} &C_{id1} \oplus_p C_{id2} =< I, O, R > \\ &I = I_1 \cup I_2 \\ &O = O_1 \cup O_2 \\ &R = R_1 \cup R_2 \end{aligned} \qquad (6.11)$$

Another intuitive way of composition is setting them in sequential order by connecting the output ports of one connector to the input ports of the other connector.

Definition 46. A ***sequential composition*** of two connectors $C_{id1} =< I_1, O_1, R_1 >\ and\ C_{id2} =< I_2, O_2, R_2 >$, if $|O_1| = |I_2|$, is described as follows:

$$\begin{aligned} &C_{id1} \oplus_s C_{id2} =< I, O, R > \\ &I = I_1 \\ &O = O_2 \\ &R = R_1 \cup R_2 \end{aligned} \qquad (6.12)$$

Parallel and sequential composition are two special cases of general definition of composition in **Definition 47**.

Definition 47. A ***composition*** of two connectors $C_{id1} = < I_1, O_1, R_1 >$ with reference nodes N_1 and $C_{id2} = < I_2, O_2, R_2 >$ with reference nodes N_2 is described as follows:

$$\begin{aligned}
&C_{id1} \oplus_L C_{id2} = < I, O, R > \\
&L = (L_1, L_2), L_1 = \left(\{n_1^1, n_2^1\}, \dots, \{n_1^l, n_2^l\}\right), L_2 = \left(n^1, \dots, n^l\right) \\
&R' = R_1 \cup R_2, R = ren_N(R') \\
&ren_N(< q, id, t, s >) = \\
&< \left(q \in L_1^{l_1} \to n^{l_1}, q\right), id, t, \left(s \in L_1^{l_2} \to n^{l_2}, s\right) > \\
&ren_N(< q, p_1, id, ct, p_2, s >) = \\
&< \left(q \in L_1^{l_1} \to n^{l_1}, q\right), p_1, id, t, p_2, \left(s \in L_1^{l_2} \to n^{l_2}, s\right) > \\
&I = (I_1 \cup I_2) \setminus I_3, O = (O_1 \cup O_2) \setminus O_3 \\
&N = ((N_1 \cup N_2) \setminus L_1) \cup L_2
\end{aligned} \tag{6.13}$$

, where L is an ordered list that determines the nodes, which are superimposed, as well as the substituting nodes. L_1 is the ordered list of superimposed nodes and L_2 is the ordered list of substituted nodes. I_3 is a set of input boundary nodes, which belong to L_1 and O_3 is a set of output boundary nodes, which belong to L_1. ren_N is a function that is responsible for changing the labels of source and sink channel ends in R' to produce R. The notation $(\varphi \to s, q)$ corresponds to McCarthy's conditional, returning $s\ or\ q$ if predicate φ evaluates to true or false, respectively.

Note that the parallel composition (**Definition 45**) and the sequential composition (**Definition 46**) are two special cases of composition in **Definition 47**. In the former, $L = (\emptyset, \emptyset)$, and in the latter, $L_1 = (\{o_1^1, i_2^1\}, \dots, \{o_1^l, i_2^l\})$, where $o_1^i \in O_1, i_2^i \in I_2$. As illustrated in Figure 6.8, $S_3 = S_1 \oplus_L S_2, L = ((\{N1, N3\}, \{N2, N4\}), (N5, N6))$.

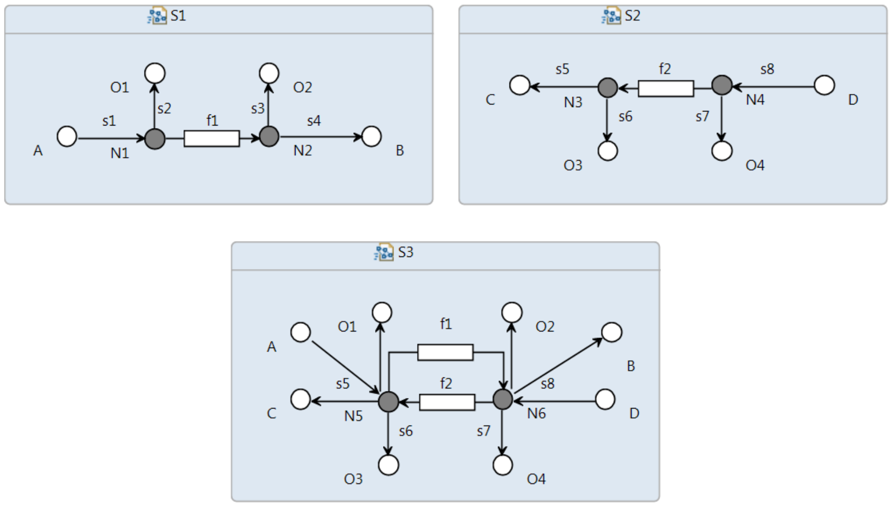


Figure 6.8. Composition of two connectors.

6.3.1.5. Connector sub-structures

In Section 6.3.1.3, we defined intuitive constructs based on component connectors, while in this section, we intend to provide more complicated sub-structures and some properties that are defined based on the verifiability of such structures.

Definition 48. A ***connector cut*** is a partition of the nodes of a connector into two or more sub-connectors. A ***cut-set*** of the cut is the set of channels whose channel ends reside in nodes, which belong to different subset of the partition. Channels are marked to be crossing the cut if they are in its cut-set. Formally speaking, a cut $C\_Cut = (N_1, \dots, N_n)$ is a partition of reference nodes N of a connector $C_{id} = < I, O, R >$. As a result, n sub-connectors $SC_{id1} = < I_1, O_1, R_1 >, \dots, SC_{idn} = < I_n, O_n, R_n >$ are formed. Note that $N = N_1 \cup \dots \cup N_n$ and $Cut\_Set = R \setminus (R_1 \cup \dots \cup R_n)$ is the channels that are cut to form the sub-connectors. The ***size*** of a cut is $|Cut\_Set|$.
Figure 6.9 illustrates two sub-connectors, which resulted from cutting $S1$ in Figure 6.8. Cut-set in this example is $Cut_{Set} = (N1, f1, FIFO, N2)$ and the size of the cut is one.

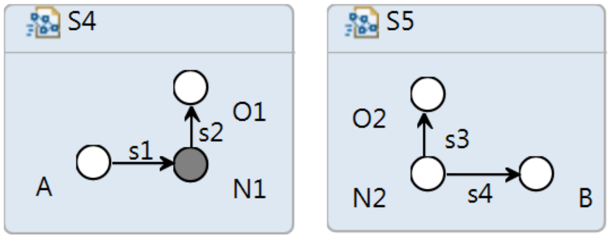


Figure 6.9. Sub-connectors resulting from cutting S1.

Definition 49. A connector $C_{id} =< I, O, R >$ is called ***connected*** if every pair of nodes in N is connected through one or a set of channels in R without considering their directions.
For example, the connectors in Figure 6.3 and Figure 6.4 are connected, but the connector in Figure 6.10 is not.

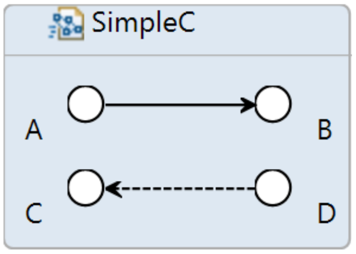


Figure 6.10. A disconnected connector.

Definition 50. In a connector $C_{id} =< I, O, R >$, a ***connector partition*** is a set of nodes and channels $P =< N', R' >$ with the following properties:

$$\begin{aligned} N' &\subseteq N \\ R' &\subseteq R \end{aligned} \tag{6.14}$$

For example, for the connector $S1$ in Figure 6.8, $P1 =< \{A, B\}, \{(A, s1, Sync, N1), (N2, s4, Sync, B)\} >$. Note that this partition can be disconnected as the $P1$ partition demonstrates. Also, note that none of the nodes involved in the definitions of channel set R' is part of N'. For connected partition, we need another concept to be defined.

Definition 51. In a disconnected connector $C_{id} =< I, O, R >$, a ***connected partition*** is a minimum number of connected sub-connectors $SC_{id1} =< I_1, O_1, R_1 >, \dots, SC_{id1} =< I_n, O_n, R_n >$ in a way that

$$\begin{aligned} R &= R_1 \cup \dots \cup R_n \\ I &= I_1 \cup \dots \cup I_n \\ O &= O_1 \cup \dots \cup O_n \end{aligned} \tag{6.15}$$

For example, the connector in Figure 6.10 has two connected partitions.

Definition 52. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. A ***structural variant*** for C_{id} is a connector $C_{id'} =< I', O', R' >$ with the following properties:

$$\begin{aligned} I' &= I \\ O' &= O \end{aligned} \tag{6.16}$$

By considering **Definition 52**, when $R \subset R'$, the connector is *expanded* (such as Figure 6.11) and, when $R' \subset R$, the connector is *shrunk* (such as Figure 6.12). A special case happens when $R \cap R' = \emptyset$. This is the case when the connector is *replaced* with a new set of channels, but with the same boundary nodes.

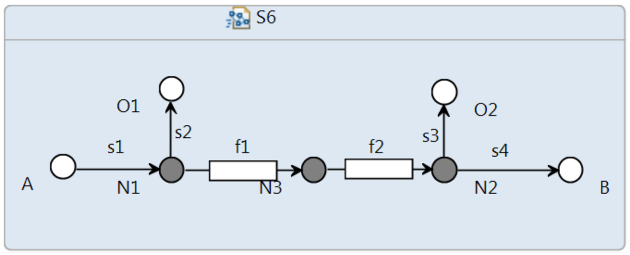


Figure 6.11. A structural variant of Sequencer connector.

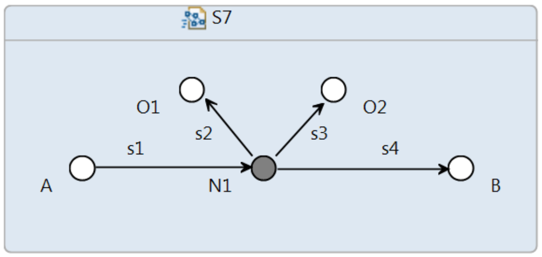


Figure 6.12. A structural variant of Sequencer connector.

Definition 53. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. An ***equivalent variant*** for C_{id} is a connector $C_{id'} =< I', O', R' >$ with reference nodes N' that hold the following properties:

$$\begin{aligned} I' &= I \\ O' &= O \\ R' &\cong_{Id} R \\ N' &= N \end{aligned} \tag{6.17}$$

, where $R' \cong_{Id} R$ means the tuples in both sets are identical, but their $id \in Id$ might be different.

Definition 54. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. An ***structurally equivalent variant*** for C_{id} is a connector $C_{id'} =< I', O', R' >$ with reference nodes N' that hold the following properties:

$$\begin{aligned} I' &= I \\ O' &= O \\ R' &\cong_{T} R \\ N' &= N \end{aligned} \tag{6.18}$$

, where $R' \cong_{T} R$ means the tuples in both sets are identical, but their channel types $t \in T$ might be different.

6.3.2. Connector reconfigurations

The main focus of traditional architecture-centric software evolution (Ahmad, Jamshidi, & Pahl, 2014; Jamshidi et al., 2013; A. J. Ramirez & Cheng, 2010) is the addition or removal of individual components, rather than the reconfiguration of underlying interaction protocols. This section, however, discusses architectural adaptations (we use reconfiguration interchangeably) of component connectors introduced in Section 6.3.1. The focus of this section is to provide the foundation to define reconfigurations, which affect significant parts of the connector by adopting the structures that are defined in Section 6.3.1.

We follow a more general perspective of reconfiguration of connectors here. We consider a reconfiguration to be any transformation from a connector $C_{id} =< I, O, R >$ to another connector $C_{id'} =< I', O', R' >$ through a sequence of elementary change operations. Our aim is to build a foundation for enabling runtime adaptation of component connectors through generic and reusable adaptations. Later on, we define different categories of reconfiguration based on this general interpretation of adaptation. As a specific case, we can restrict reconfigurations by ruling out the ones that do not preserve some specific properties. For example, one can only consider structural restrictions such as preserving boundary interfaces $I\ or\ O$. As another example, one may only be interested in reconfigurations that preserve the initial behavior of the connector. This requires an underlying semantic model of the component connector.

Definition 55. A ***connector homomorphism*** f from a connector $C_{id} =< I, O, R >$ to a connector $C_{id'} =< I', O', R' >$, written $f: C_{id} \to C_{id}'$, is a mapping $f: N \to N'$ from reference nodes N of C_{id} to N' of $C_{id'}$ such that $(n1, id, t, n2) \in R$ implies $(f(n1), id', t', f(n2)) \in R'$. C_{id} is said to be homomorphic to $C_{id'}$. If $f: N \to N'$ is a one-to-one function whose inverse is also a homomorphism, then f is a ***isomorphism*** of the connectors.

Definition 56. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. A ***structure preserving reconfiguration*** r is a connector homomorphism (**Definition 55**) when applied to C_{id}, denoted by $C_{id} \cdot r$, yields a structural variant (**Definition 52**) $C_{id'} =< I', O', R' >$ of it.

Definition 57. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. A ***removal*** from a connector $remove$ is a reconfiguration that cuts a partition $P_1 =< N_1, R_1 >$ out of connector C_{id} and removes the orphaned nodes. Application of $remove$ to C_{id} is represented as $C_{id} \cdot remove(P_1)$ and yields a new connector $C_{id'} =< I', O', R' >$ with reference node N'.

$$\begin{aligned} & N' \subseteq N \\ & I' \subseteq I \\ & O' \subseteq O \\ & R' = R \setminus R_1 \end{aligned} \qquad (6.19)$$

Let consider connector $S6$ in Figure 6.11 and the connector partition

$P_{S6} =< \{N_3\}, \{(N1, f1, FIFO, N3), (N3, f2, FIFO, N2)\} >$.

The application of the removal operation $S6.remove(P_{S6})$ will result in connector $S6'$ as it is depicted in Figure 6.13. Note that after the removal of the two channels, the node $N3$ becomes orphaned and as a result it should be removed.

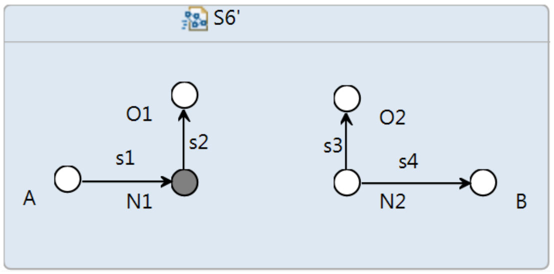


Figure 6.13. A reconfigured version of sequencer connector after application of removal.

Definition 58. Let connectors $C_{id1} = < I_1, O_1, R_1 >$ and $C_{id2} = < I_2, O_2, R_2 >$ be two connectors with reference nodes N_1 *and* N_2 respectively. An ***insertion*** of C_{id2} into C_{id1} is an special case of composition of the two connectors in a way that a node connecting two or more channels in C_{id1} will be disconnected and C_{id2} is inserted into the room created by that separation by stitching nodes in $J \subseteq N_1 \times N_2 \times N$. This reconfiguration is not structure preserving because the boundary nodes may be changed by this reconfiguration. Application of $insert$ to C_{id1} is represented as $C_{id1} \cdot insert(C_{id2}, J)$ and yields a new connector $C_{id3} =< I_3, O_3, R_3 >$ with reference node N_3.

$$\begin{aligned} &I_1 \subseteq I_3 \\ &O_1 \subseteq O_3 \\ &R_3 = ren_{N_3}(R_1 \cup R_2) \\ &|N_3| = |N_1| + |N_2| - |J| + 1 \end{aligned} \tag{6.20}$$

Let us consider the sequencer connector in Figure 6.3, a simple connector $I1$ as in Figure 6.14 and stitching nodes $J = \{(N2, A, N3), (N2, B, N4)\}$. The application of insertion $Sequencer.insert(I1, J)$ would result in a new connector $Sequencer'$, which is a variation of sequencer connector called proactive waiting sequencer as depicted in Figure 6.15. This insertion is not structure-preserving and adds a new node $i1$ to the boundary nodes of the sequencer. Moreover, as in (6.20), $8 = 6 + 3 - 2 + 1$ holds for this reconfiguration.

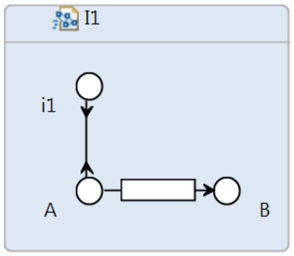


Figure 6.14. A simple connector.

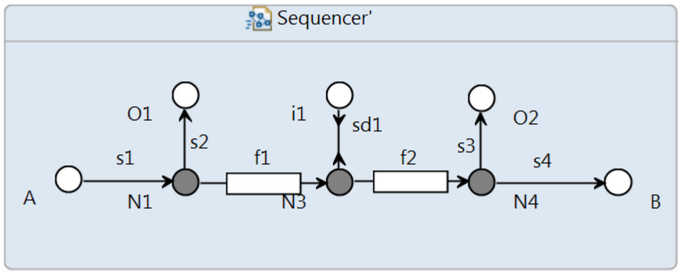


Figure 6.15. The proactive waiting sequencer.

Definition 59. Let connector $C_{id} =< I, O, R >$ be a connector with reference nodes N. A ***replacement*** is a structure-preserving (**Definition 56**) reconfiguration $replace$ that replaces a partition $P_1 =< N_1, R_1 >$ (cf. **Definition 50**) of connector C_{id} with another connector partition $P_2 =< N_2, R_2 >$ by stitching nodes in $J \subseteq N_1 \times N_2 \times N_3$. The application of $replace$ to C_{id} is represented as $C_{id} \cdot replace(P_1, P_2, J)$ and yields a new connector $C_{id'} =< I', O', R' >$ with reference nodes N'.

$$\begin{aligned} & N_1 \subseteq N \\ & R_1 \subseteq R \\ & I' = I, O' = O \\ & R' = ren_{N'}((R \cup R_2) \setminus R_1) \\ & N' = (N \cup N_3) \setminus N_1 \end{aligned} \tag{6.21}$$

, where each element in triple J respectively indicates which pair of nodes from P_1 and P_2 are to be joined to form a new set of nodes as in N'.

Let us consider the proactive waiting sequencer in Figure 6.15,

$P_1 =< \{O1, O2\}, \{(N1, s2, Sync, O1), (N4, s3, Sync, O2)\} >$,

$P_2 =< \{O3, O4\}, \{(N5, ls1, LossySync, O3), (N6, ls2, LossySync, O4)\} >$,

$J = \{(N1, N5, N7), (N4, N6, N8)\}$

The application of the replacement operation $Sequencer'.replace(P_1, P_2, J)$ will result in connector $Sequencer''$ as depicted in Figure 6.16.

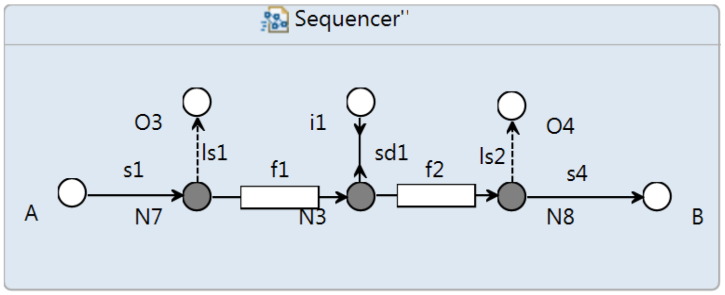


Figure 6.16. The proactive waiting weak sequencer.

6.4. Adaptation Effectuation through Dynamic Software Product Lines

In this section, we first review existing work that addresses dynamic adaptation of software systems through the concept of variability in the underlying software, which is considered as a relevant work in the software product line community. We then use the concept of feature models to define connector modes with the structural construct that we defined in Section 6.3.1. Finally, we propose our mode-based adaptation mechanism based on a reasoning mechanism on feature models corresponding to the connector mode configurations.

6.4.1. Runtime adaptation and dynamic software product line

According to the approaches reviewed in Section 6.2, a limited set of architectural configurations is determined and associated with environmental situations. As a result, they restrict the variability space of self-adaptive software in a dramatic way. However, the relationship between self-adaptive software and its surrounding environment is not that straightforward. This relationship affects application functionality, non-functional requirements and the inherent capability of the platform on which the software system is running. Capturing this complex relationship with a limited set of architectural configurations imposes a risk of overlooking important environmental situations and missing architectural configurations (Perrouin & Chauvel, 2008).

Software Product Line (SPL) engineering is a way to deal with varying user requirements that lead to the derivation of customized product variants. Once the product has been created, they tend to keep their structure and behavior throughout their lifetime. However, Dynamic Software Product Lines (DSPL) embrace software systems that are capable of modifying their own structure or behavior with respect to environmental situations by using runtime adaptation (C Cetina, Giner, Fons, & Pelechano, 2010). This capability of changing the structure and behavior is enabled by a traditional notion of variability in SPL. However, as opposed to the traditional perspective, in which variants are decided for the variation points at design-time, the variability in DSPLs is bound or unbound at runtime. Moreover, the binding decisions on the variations may change several times in its lifetime (Hallsteinsen et al., 2008). Therefore, DSPL is regarded as an efficient approach to build dynamic adaptive software (Hallsteinsen et al., 2008).

There are different works focusing on adaptation of software systems based on the different kinds of software artifacts driven by the variability bindings. As a result, the relationship between variation points and variants are also different. Trinidad et al. (Trinidad, Cortés, Peña, & Benavides, 2007) associate feature models to component architecture for building a DSPL. The mapping is one-to-one and adaptation can be realized by dynamic connections between specific components. However, this one-to-one mapping contradicts the clear separation between functional and architectural dimensions. Wolfinger et al. (Wolfinger, Reiter, Dhungana, Grunbacher, & Prahofer, 2008) propose the same sort of mechanism, but combining it with a plug-in technique. The adaptation is enabled by loading and unloading the plug-ins at runtime. Lee et al. (Kotonya, 2010) present their work on service-based systems. Therefore, the adaptation is mapped to service selection with the right quality level. Perrouin et al. (Perrouin & Chauvel, 2008) clearly separate the variability space into the three dimensions functional, platform and topological. Feature model, component repository and collaboration diagram manage the three variability spaces respectively. Hallsteinsen et al. (Floch et al., 2006; Hallsteinsen, Stav, Solberg, & Floch, 2006) define variability directly in the reference architecture. The architecture constitutes component, which realize component types as variation points. Lee and Kang (Lee & Kang, 2006) introduce the notion of a binding unit, which are used to identify architectural components by grouping features. Montero et al. (Montero, Pena, & Ruiz-Cortes, 2008) focus on managing variability in business processes. Cetina et al. (Carlos Cetina, Giner, Fons, & Pelechano, 2009) focus on reconfiguration of architectural models based on reasoning on feature models.

These works are based on the clear mapping between the features and the software artifacts whether it is a component, service, plugin or process variant. Shen et al. (Shen, Peng, Liu, & Zhao, 2011) propose a solution of managing complex relationships between variability model and variants. They introduce a role model to clarify this complex mapping.

Table 6.4. Variability binding in existing approaches.

Approach	Variability	Variant	Relationship
(Trinidad et al., 2007)	Feature model	Component	1-to-1
(Wolfinger et al., 2008)	Feature model	Plug-in	1-to-1
(Kotonya, 2010)	Feature model	Service	1-to-N
(Perrouin & Chauvel, 2008)	Feature model	Component	1-to-N
(Hallsteinsen et al., 2006)	Reference architecture	Component	1-to-N
(Floch et al., 2006)	Reference architecture	Component	1-to-N
(Lee & Kang, 2006)	Feature model	Component	1-to-N
(Montero et al., 2008)	Feature model	Process variant	1-to-N
(C Cetina et al., 2010)	Feature model	Service configuration	M-to-N
(Shen et al., 2011)	Feature model	Code	M-to-N

6.4.2. Feature models for component connectors

In general, a feature is an increment in functionality of a system (Czarnecki, Helsen, & Eisenecker, 2004). The features are typically related through a hierarchical tree structure, called feature model. This structure has top-down optional/mandatory relationships, cross-node alternative/or relationships and crosstree requires/excludes constraints. However, the features in feature models are inherently symbolic. Therefore, in order to give them a precise semantics, we need to map features to other models corresponding to the structure or behavior of a software system. In this section, we use the concepts of connector configuration, as formally defined in Section 6.3.1, for mapping feature models to concise representations of variability in connector configurations. Therefore, the feature models that we consider throughout this research express connector variability, meaning that feature models are devoted to the modeling of the variation points and their relationships in a given component connector.

Definition 60. A ***feature model*** $FM = (F, \phi)$ is defined as a finite set of features $F = \{f_1, f_2, \dots, f_n\}$ and ϕ as a propositional logic formula over F.

An example of a feature model of the sequencer connector is represented in Figure 6.17. $Data\_In$ and $Data\_Out$ are features that correspond to the capability of the sequencer to have ports for accepting data and releasing the data out of the connector. The feature $Output$ determines the output ports, which facilitate sequential delivery of data to the corresponding entities. The feature $MiddleLayer$ determines the protocol of interactions between the output ports. $Data\_In, Data\_Out$ and $Output$ are mandatory features. The $MiddleLayer$ feature can be optionally selected to enhance the output interaction with one of its alternative sub-features. The $Output$ feature can be optionally refined with more than one output instances maximum of five.

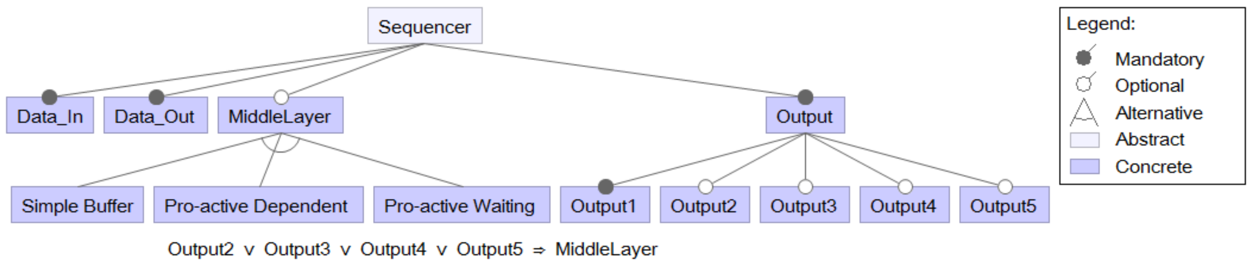


Figure 6.17. Sequencer connector feature model.

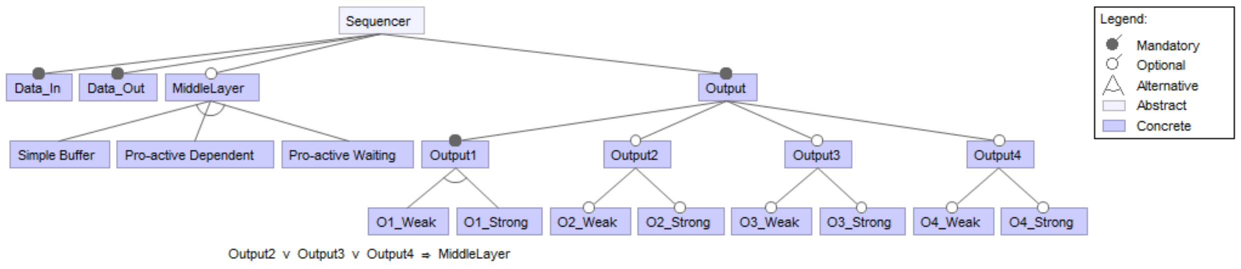


Figure 6.18. Refined sequencer connector feature model.

According to **Definition 60**, $F = \{Sequencer, Data_{In}, Data_{Out}, MiddleLayer, Simple\ Buffer, Proactive\ Dependent, Proactive\ waiting, Output, Output1, Output2, Output3, Output4, Output5$.

With respect to **Definition 60**, we now define the concept of *configuration (that corresponds to a connector operating mode)*.

Definition 61. A set of all selected features in a feature model $FM = (F, \phi)$ that satisfy the constraints in ϕ is referred to as ***configuration***:

$$\begin{aligned} &CC \overset{\text{def}}{=} SF = \{f_1, f_2, \dots, f_m\} \\ &SF \subseteq F \\ &\forall f \in SF: f.selected = true \\ &SF \vdash \phi \end{aligned} \tag{6.22}$$

For example, the current configuration of the sequencer connector $S1$ represented in Figure 6.19 is as follows:

$$\begin{aligned} &CC_{S1} = \{Data_{In}, Data_{Out}, MiddleLayer, \\ &Simple\ Buffer, Output, Output1, Output2\} \end{aligned} \tag{6.23}$$

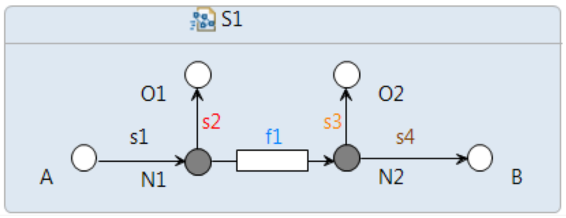


Figure 6.19. Initial configuration (mode) of sequencer connector.

The rules for selecting features from a feature mode that satisfy the constraints of the model can be listed as follows:

1. If a feature is selected, its parent must also be selected.
2. If a feature is selected, all of its mandatory children in an "AND" group must be selected.
3. If a feature is selected, at least one of its children in an "OR" group must be selected.
4. If a feature is selected, exactly one of its children in an "Alternative" group must be selected.

Definition 62. Let us consider $FM =< F, \phi >$ be a feature model, $[\![FM]\!]$ denotes the set of ***valid configurations*** of the feature model FM.

$$\forall CC_i \in [\![FM]\!]: CC_i \subseteq F \wedge CC_i \vdash \phi \tag{6.24}$$

Since features in the feature model for component connectors represent a coarse-grained coordination protocol, there is a need to determine which connector elements are represented by each feature. In the context of this work, we assume that component connectors are defined as **Definition 34**. Figure 6.19 shows a sequencer connector using graphical notations, which are equivalent to its formal representation. We map the features to connectors by the following operator.

Definition 63. The ***feature to connector map*** operator takes a feature $f \in F$ and returns a connector partition $p \in P_{C_{id}}$ (cf. **Definition 50**) of connector $C_{id} = \langle I, O, R \rangle$ as follows:

$$\begin{gathered} F2C: F \to P \\ \forall f_1, f_2: F2C(f_1) \cap F2C(f_2) = \emptyset \end{gathered} \tag{6.25}$$

In the context of the sequencer connector $S1$ in Figure 6.19,

$$\begin{gathered} F2C(Data\_In) =< \{A\}, \{(A, s1, Sync, N1)\} > \\ F2C(Simple\ Buffer) =< \{N1, N2\}, \{(N1, f1, FIFO, N2)\} > \end{gathered} \tag{6.26}$$

Assume a need to restrict the second output port such that it is accessed after it receives an acknowledgement from the component that is connected to the first output. The *pro-active waiting* feature lets the first component acknowledge its termination and the coordination protocol to memorize it. By choosing this feature as part of feature set in the current configuration, it yields the *proactive-waiting sequencer* connector mode $S8$ as illustrated in Figure 6.20. In the context of $S8$,

$$\begin{gathered} F2C(Proactive\ waiting) = < \{N3, i1\}, \{(N1, f1, FIFO, N3), \\ (N3, sd1, SyncDrain, i1), (N3, f2, FIFO, N2)\} > \end{gathered} \tag{6.27}$$

Note that we can apply, for example, more of such basic constructs (see **Definition 36**) to grow this connector up for creating more connector modes.

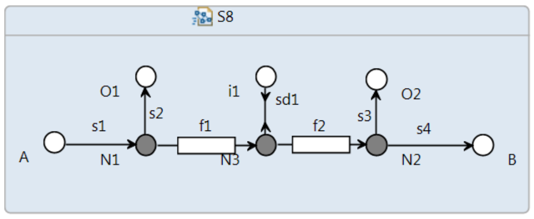


Figure 6.20. The "pro-active waiting" sequencer connector mode.

Consider again the sequencer connector, but with new capability. Imagine that the components connected to the ports $O1, O2$ may fail for a long period of time, which leads to a deadlock in the connector. The possible solution is to choose weak versions of output instead of strong counterparts. This avoids the problem by not enforcing the components to respond when they are not working properly. Figure 6.21 illustrates the new mode of the connector, which is called weak sequencer. Note that the connector mode illustrated in Figure 6.21 is a structurally equivalent connector (cf. **Definition 54**) to the connector shown in Figure 6.19.

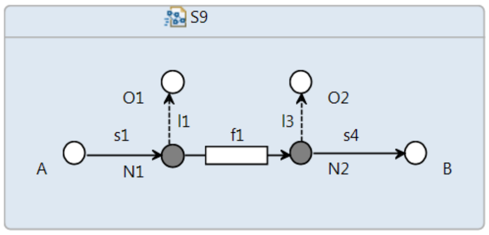


Figure 6.21. The weak sequencer connector mode.

Now consider the situation in which the result of the second component is complementary with respect to the first one. Therefore, whenever it fails, the system should proceed normally through port B and disregard port $O2$. This requirement is met by selecting the weak version of the second output resulting in a new mode called quasi-weak sequencer shown in Figure 6.22.

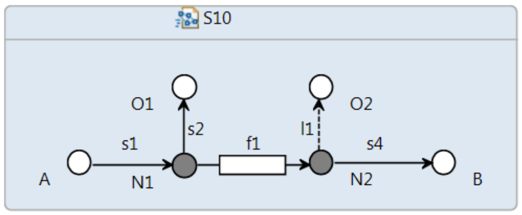


Figure 6.22. The quasi-weak sequencer connector mode.

Now suppose that a new requirement forces a dependence between components in the sequence. To be more specific, consider the second component connected to the port $O2$ is executed with the result of the first component and whenever the second component is not ready to consume the result, it should be memorized. The *pro-active dependent* feature meets the envisage requirement. By selecting this feature, a new mode called pro-active dependent sequencer results as shown in Figure 6.23.

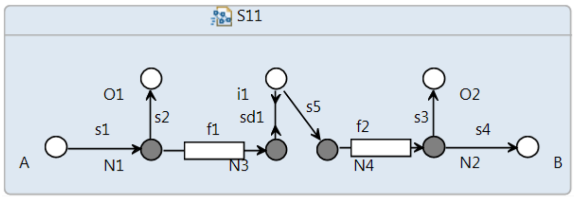


Figure 6.23. The “pro-active dependent” sequencer connector mode.

6.4.3. Mode-based adaptation of component connectors through feature models

In this section, we use the feature-based representation (see Section 6.4.2) of component connectors to facilitate the mode-based adaptations of component connectors. Note that mode-based adaptation is the mechanism we adopted to enable the self-adaptation process for component connectors that is proposed in this thesis.

Figure 6.24 represents the process of product derivation, where artifacts are composed according to a valid configuration, which is compliant with the feature model. The process of product derivation is accomplished at design-time to produce a variant of a software solution among different valid variants by resolving the variability points in the model. However, the process of variant derivation based on DSPLs is slightly different.

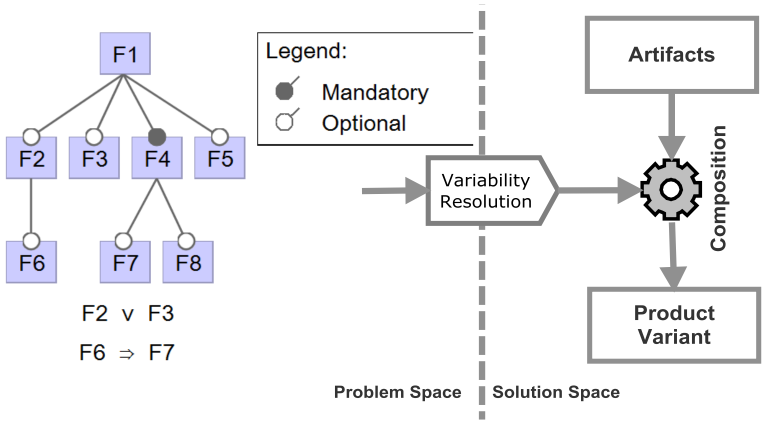


Figure 6.24. Product derivation process in SPL.

The process for dynamic adaptation based on DSPL comprises two steps. As depicted in Figure 6.25, at design-time, an initial feasible configuration of the system is derived. At runtime when the situation of the operating environment is changed and the current configuration cannot satisfy a desired non-functional requirement, a new configuration needs to replace the current one. In this thesis, we follow such adaptation process. This process comprises adaptation reasoning (see Chapter 5) to find a suitable configuration that can be enacted (the subject of this chapter) through variability resolution at runtime.

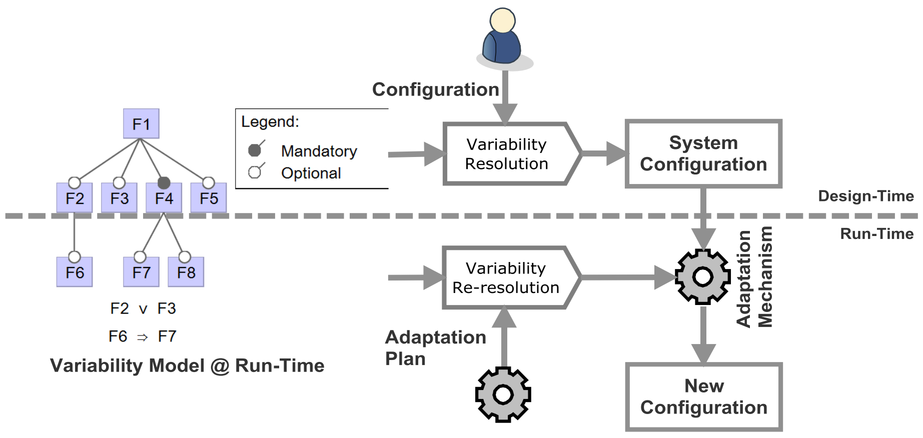


Figure 6.25. Dynamic adaptation of software system with variability model at runtime.

In some circumstances, for example when the initial requirements have been changed, the feature model itself needs to be adapted. The change in the variability model can be quite common for software systems whose target configurations are dynamically discovered at runtime. The change in the variability model results in a new feature model that may be required to be resolved at runtime and as a result causes some changes in the current configuration. In this thesis, this type of change is not considered as a part of the adaptation process since we assume that the modes of the component connectors are entirely discovered at design-time.

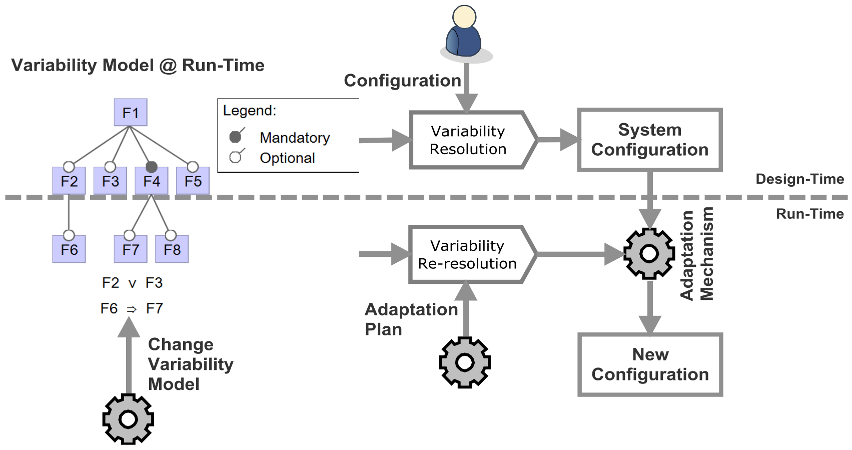


Figure 6.26. Dynamic adaptation of software system with dynamic variability model at runtime.

Definition 64. The ***Feature Configuration Model (FCM)*** defines a feature model, $FCM = (F_{cm}, \phi_{cm})$, where all variabilities of the original feature model $FM = (F, \phi)$ are resolved and the subsequent product derivation results in only one variant corresponding to the configuration, CC (see **Definition 61**), of an specific mode of the connector:

$$\begin{aligned} F_{cm} &= CC \subseteq F \\ \phi_{cm} &\vdash \phi \end{aligned} \tag{6.28}$$

An adaptation execution for a connector can be simply seen as set of variant substitutions for given variation points. Let us consider that the current configuration of the running connector corresponds to mode CM_s and the target configuration that is derived based on the adaptation reasoning corresponds to CM_t. Each of these connector modes have their corresponding resolved variability models denoted by $FCM_s = (F_s, \phi_s), FCM_t = (F_t, \phi_t)$, see **Definition 64**. Because of such a definition, the key problem of adaptation execution (see Figure 6.1) boils down to the transition from current mode to the target mode through their corresponding feature configuration models. The added and removed features can be identified by calculating the following equations:

$$\begin{aligned} F_{add} &= F_t - F_s \\ F_{rem} &= F_s - F_t \end{aligned} \tag{6.29}$$

Then by adopting the feature to the connector map operator, the added and removed connector partitions $p_{add} \in P_{C_s}, p_{rem} \in P_{C_s}$ (cf. **Definition 50**) of connector $C_s = \langle I, O, R_s \rangle$ can be located and respectively be added and removed from C_{id}. This results in a new connector mode $C_t = \langle I, O, R_t \rangle$ with the following relations:

$$\begin{aligned} p_{add} &= F2C(F_{add}) =< N_{add}, R_{add} > \\ p_{rem} &= F2C(F_{rem}) =< N_{rem}, R_{rem} > \\ R_t &= R_s + R_{add} - R_{rem} \end{aligned} \tag{6.30}$$

At the execution level, we need to determine the order in which these changes have to be applied, and maintain the consistency of the connector by checking the connectivity (to evaluate whether the condition in **Definition 49** is valid) of the connector throughout this transition. Checking this connectivity is straightforward because it should always be possible to find a path (**Definition 40**) from the input nodes I to output nodes O by traversing the elements in R in the current configuration of the temporal connector.

6.5. Limitations and Threats to Validity

The approach for adaptation execution that we proposed in this chapter is demonstrated through the Reo language. However, this approach is general enough as long as we can map the language structural constructs to the graph theory constructs as we demonstrated for Reo in Section 6.3 via various definitions. The operationalization of adaptation process that is discussed in Section 6.4 is based on tree-based feature models that can be mapped to the structural constructs via **Definition 63**.

However, there are different connector models and languages in the literature (see a review of existing connector languages in (Bruni et al., 2013; Kell, 2007)) and these are based on different theories and technical machineries (Bliudze & Sifakis, 2007). For instance, a connector may impose a handshaking constraint between a sender component and a receiver component (Milner's Calculus of Communicating Systems), or it may demand for an agreement on the action to be executed next by the connected components (Hoare's Communicating Sequential Processes). As a result, a threat to external validity of this approach is the choice of specific connector language (i.e., Reo) used for demonstration purposes and communicating the results of this thesis. This threat, however, is mitigated as far as possible by formally defining general-purpose structures using graph theory and widely adopted feature models.

6.6. Conclusions

In this chapter, we have presented the execution phase of the MAPE-K loop. We have adopted two solid and well recognized theories to enable such adaptation effectuation for component connectors, i.e., graph theory and variability modeling. We adopted graph theory to formally define the structure of connectors and reason about structural changes at an appropriate level of abstraction. We also showed how to reason about mode changes based on feature models corresponding to the modes of connectors and use the formal structural constructs to derive adaptation actions. We have also presented the adaptation itself as the change of a connector from one current operating mode to a new target mode. The variability model changes of our connectors are mapped into reconfiguration actions in order to adapt the connectors at runtime. We have based the reconfiguration on Reo that, due to its reconfiguration properties, enables products to switch channels and structural configurations.

This chapter concludes the technical contribution of this dissertation, which started in Chapter 4 with a learning mechanism to calibrate analytical models corresponding to the component connectors embracing the existence of uncertainties. We have explained the adaptation reasoning to derive the right mode for the connector given the current situation at runtime in Chapter 5. Although this section introduces a novel structural reasoning for component connectors, and the combination of feature-based and structural reasoning based on graph theory is new, this chapter acts as a supplementary solution component. The next part of this dissertation discusses the advantages and limitations of the proposed solution framework, and provides comprehensive details on the experimental evaluation and tool support developed for this research work.

Chapter 7

7. Implementation and Evaluation

"The difference between theory and practice is that in theory, there is no difference between theory and practice." Anonymous

Contents

7.1. Introduction

In this chapter, we show how the three key parts of the solution framework (i.e., RCU framework) are integrated to enable self-adaptation of component connectors through a real-world case study. To conduct this research, we followed the guidelines of the action research methodology (Chapter 1) that provides a rigorous set of steps focused on planning (Chapter 2, Chapter 3) and conducting the research (Chapter 4, Chapter 5, Chapter 6) along with the evaluation of the research results (this chapter). In the self-adaptation process, we have devised mechanisms to calibrate the runtime models as presented in Chapter 4. After a requirement violation is detected in an updated analytical model based on monitoring data, a decision needs to be made for what operational mode is appropriate to fix the violation. Then the decision needs to be enacted on the running connector and the operating mode of the connector is changed accordingly. We present the mechanisms for such decision-making and change execution in Chapter 5 and Chapter 6 respectively. The focus of this chapter, on the other hand, is on an experimental evaluation of these above-mentioned solution components (cf. Figure 7.1).

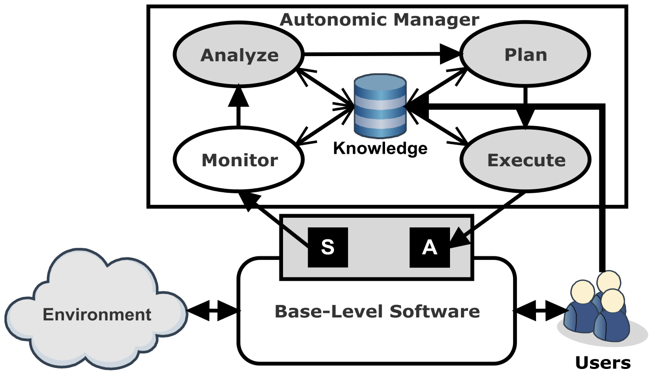


Figure 7.1. Scope of Chapter 7.

The main contributions of this chapter are to show the validity of the research claims we have made in the Introduction (Chapter 1):

- How the solution components are integrated with each other to realize the feedback control loop.
- To evaluate the “computational complexity”, “stability”, and “robustness” of the adaptation reasoning controller that enables the self-adaptation of component connectors.
- To show the “applicability” and “usability” of self-adaptive connectors in a real-world context.

The outcome of this chapter is a number of empirical and experimental results as well as proof of concepts that results that not only demonstrate the validity of the research claims but also address **RQ3** (cf. Chapter 1) that calls for evidences of real-world applicability of our solution framework, i.e., RCU.

The remainder of this chapter is organized as follows. This chapter begins with a high-level overview of the proposed solution framework in Section 7.2. Details about the criteria for evaluating the solution framework are presented in Section 7.3. The key section of this chapter (i.e., Section 7.4) includes details of the case study through an empirical research. Results regarding the experimental evaluation of the solution framework are also presented in Section 7.4. A discussion of limitations and threats to validity is presented in Section 7.5. Finally, we conclude the chapter by reviewing the research claims and their support in Section 7.6.

7.2. An Overview of the Proposed Solution Framework

An overview of our approach to enabling reliable self-adaptation of component connectors is shown in Figure 7.2. As illustrated, the approach covers both design-time and runtime. During design-time, the aim is to design appropriate architectural modes of the connector and verify them against expected requirements. At runtime, while the connector starts operating, the framework monitors quality data. The non-functional requirements are continuously verified with respect to runtime data that may reflect changes in the environment's behavior. In the case of detected violations, a mode change is decided and enacted accordingly. In the following, we briefly discuss each phase in turn and describe the relevant activities.

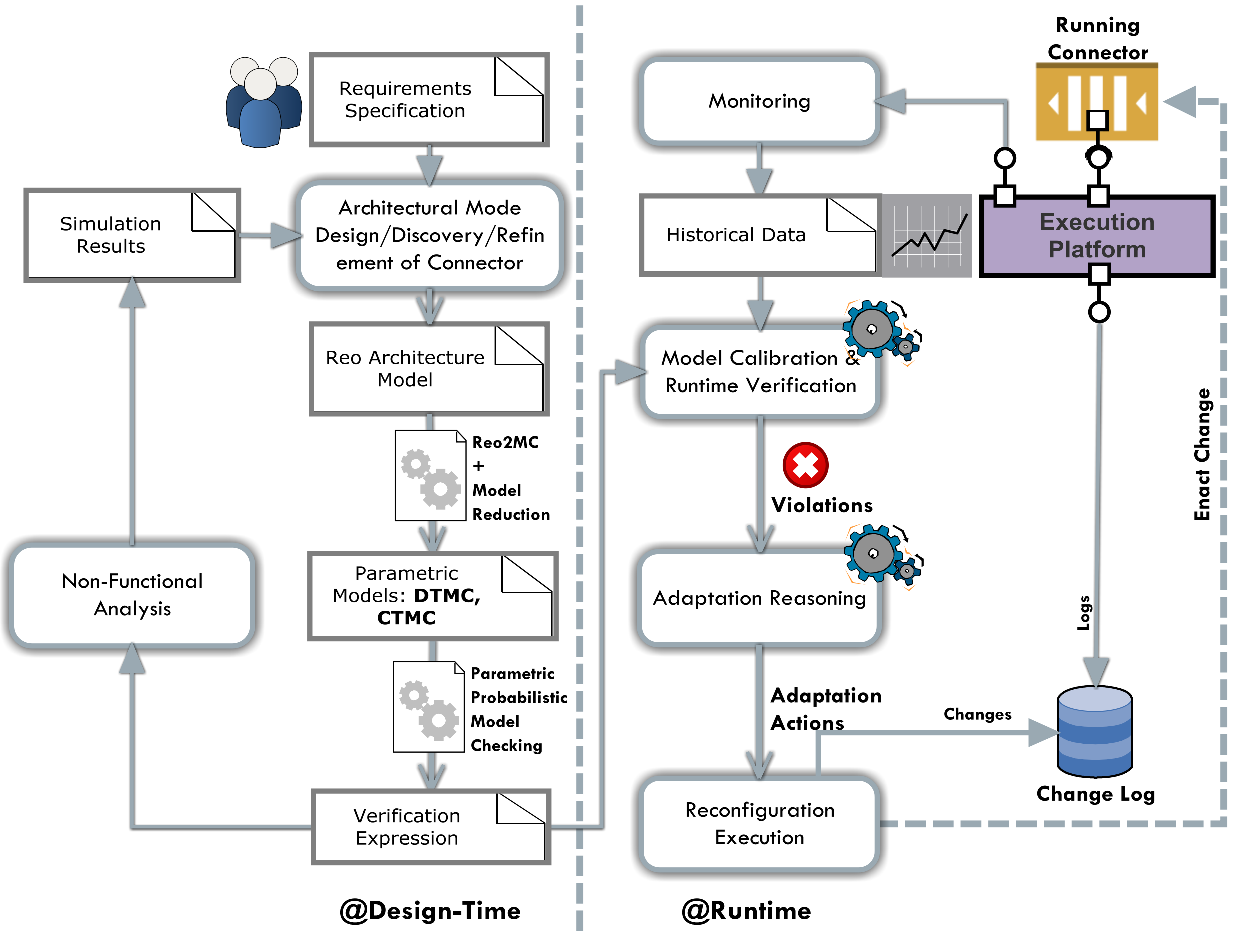


Figure 7.2. The proposed framework.

(i) Design-Time. The approach begins at design-time when the operating modes of the connector are determined. The key point of the design is to specify both the architecture of the connector in each mode as well as the known properties of its enclosed elements. The architectural design is then transformed to parametric versions of probabilistic models comprising Discrete-Time Markov Chains (DTMC) for reliability purposes and Continuous-Time Markov Chains (CTMC) for performance purposes. Then, the probabilistic counterpart of the architectural design is verified against expected non-functional requirements by using parametric model checking. Different configurations resulting from different applications of reconfiguration patterns are model checked in the different environment conditions for which they are conceived. The goal is to show whether or not the different configurations can satisfy non-functional requirements. The designer may look at the analysis results and may modify the architecture design accordingly. We chose parametric model checking because some channel properties are not known at design-time so they need to be specified by variable parameter.

Model-to-Model Transformation and Parametric Verification. The objective here is to enable a runtime efficient verification that evaluates non-functional requirements of the system while it is executed. One possibility would be to use traditional model checking to achieve this goal. In this case, at design-time, we would model check the architectural design in the different environment conditions in which they are intended to work. At runtime, the model can also be analyzed by the model checker in the current environment conditions. A failure of requirement satisfaction would then drive the application of a reconfiguration pattern. This approach, unfortunately, is unlikely to work in practice, especially because of the imposed time required by the verification step, which may lead to unacceptably late reactions at runtime. To make runtime analysis feasible, we apply a *parametric verification* approach instead of the classical one. In this case, parametric verification is performed at design-time and a formula is generated, which is later evaluated quite efficiently at runtime when updated real data are available. The only imposed overhead at runtime is the substitution of variables with real value. This is a fairly scalable approach that has been shown that its runtime overhead is practical for even large-scale models. Note that we borrowed this quantitative verification technique (Calinescu et al., 2012) from the PhD work of Antonio Filieri reported in (A Filieri et al., 2013; Antonio Filieri, Ghezzi, & Tamburrelli, 2011; Antonio Filieri, 2013). An overview of such parametric verification is shown in Figure 7.3.

To evaluate requirements, the architectural design is transformed into parametric Markov models. The transformations from Reo architecture models into Markov models are performed by Reo2MC tool chain (Moon, 2011). Regarding Markov models, parametric DTMCs are used to verify reliability properties, while parametric DTMCs with rewards are used to verify cost properties (e.g. channel utilizations). Note that such properties (i.e., quantifiable non-functional requirements) are expressed as formulae written in the PCTL and CSL temporal logic and their extension is based on the concept of rewards as we described in Chapter 2.

The parametric Markov models and the property formulae are fed into PARAM model checker ("PARAM Model Checker," 2013). The resulting formulae are used for two purposes. Firstly, they are used for design-time verification of different coordination configurations. In this case, we have to make assumptions about quality data for the parameters. The values represent the environment conditions we predict, and for which we want to prove that appropriate modes are discovered at design-time that can satisfy the requirements. Secondly, when no mode is able to satisfy the requirements, the designer should change the set of connector modes. Furthermore, these formulae are used for runtime analysis and planning to perform continuous verification and reconfiguration.

(ii) Runtime. When the procedure moves to runtime, the quality data collected through monitoring must be transformed into values that can be used in the parametric model checker. This transformation in general depends on the abstraction that model parameters realize on measurable data in the environment. The transformation from monitoring data to model parameters is described in Chapter 4. The updated parameters are used to substitute the formulae in order to verify the current satisfaction. In case the verification detects any violations, an adaptation is decided to replace the current configuration of the connector. The details of such adaptation reasoning are described in Chapter 5 of this thesis. Having decided about the appropriate operating mode, the change to the configuration of the running connector needs to be executed. The details of such change enactment are described in Chapter 6.

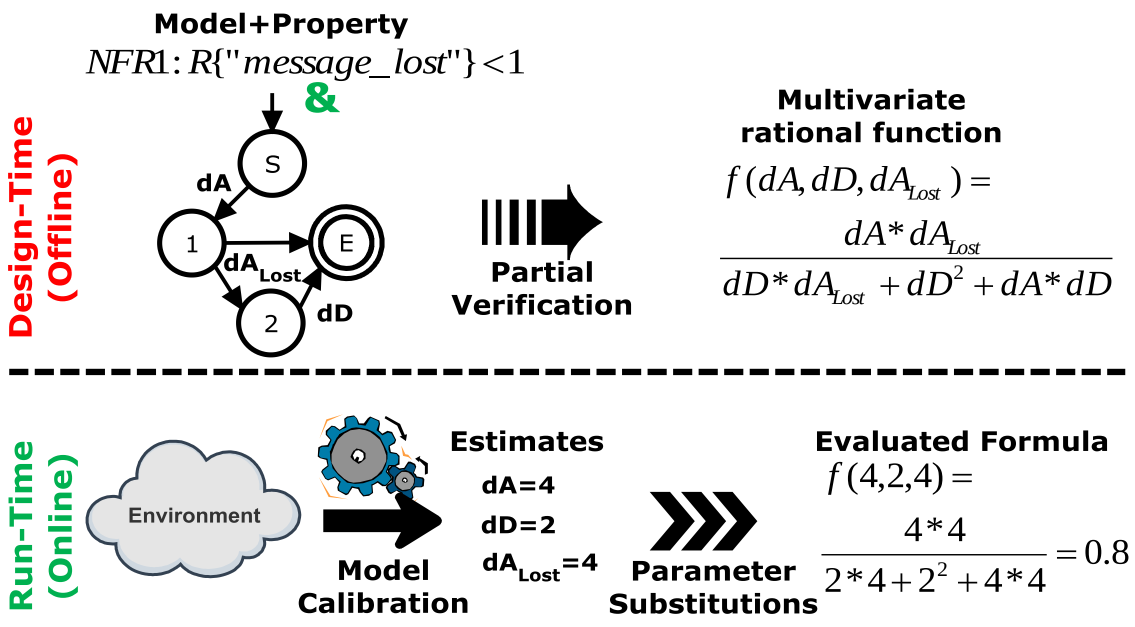

Model+Property
$NFR1: R\{"message\_lost"\} < 1$
&
Design-Time (Offline)
S
dA
1
E
dA_Lost
2
dD
Partial Verification
Multivariate rational function
$f(dA, dD, dA_{Lost}) = \frac{dA * dA_{Lost}}{dD * dA_{Lost} + dD^2 + dA * dD}$
Run-Time (Online)
Environment
Model Calibration
Estimates
dA=4
dD=2
dA_Lost=4
Parameter Substitutions
Evaluated Formula
$f(4,2,4) = \frac{4*4}{2*4+2^2+4*4} = 0.8$

Figure 7.3. Parametric requirement verification process.

7.3. Evaluation Criteria

Since a critical solution component of our RCU framework responsible for reasoning on adaptation is a fuzzy controller, we needed to borrow some criteria that has been typically used in control engineering for evaluating the properties of controllers. From the perspective of control engineering, a controller should be able to provide the following properties (Hellerstein et al., 2004):

- *Stability*. A control system is stable if there exists a converge point to which the system approaches. As time tends to infinity, the distance to the equilibrium point tends to zero. In other words, when a controlled system becomes unstable, the output of the system will not converge.
- Absence of *overshooting*. An overshoot occurs when the system exceeds the setpoint prior to convergence.
- Low *settling time*. Settling time refers to the time required for the controlled system to reach the setpoint.
- *Robustness*. A robust control system converges to the setpoint despite errors or variations in the initial model. This property defines how well the system will react to disturbances and inaccurate feedback measurements, as well.

These properties can be interpreted from the perspective of software engineering. The adoption of control theory has recently become popular in software engineering community. For example, we observe that noticeable studies have been publishing in several venues, e.g., the ACM Transactions on Autonomous and Adaptive Systems (TAAS), the International Conference on Autonomic Computing and Communications (ICAC), the Symposium on Software Engineering for Adaptive and Self-Managing

Systems (SEAMS), the IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SASO), the Schloss Dagstuhl seminar on Software Engineering for Self-Adaptive Systems, and more recently, the GI Dagstuhl Seminar "Control Theory meets Software Engineering". It is very interesting that the most recent ACM/IEEE International Conference on Software Engineering (ICSE) in 2014, there are two papers that focus on the adoption of control theory for solving software engineering problems, i.e., (Antonio Filieri et al., 2014) and (D'Ippolito et al., 2014). As a result, such control theoretic evaluation criteria can be interpreted from the perspective of software engineering. A controller in this new perspective should be able to provide the following properties (Antonio Filieri et al., 2014):

- *Stability*. Stability refers to the property that the self-adaptive system maintains the objective despite unpredictable deviations from expected behaviors; e.g., changing workloads or hardware failures. In addition, the system should not react to transient external changes. For example, a controller for adjusting system resources should not react to transient load changes. Instead, such a controller should be able to distinguish between a condition of stabilized load changing that effects performance and a short-lived load changes that will not have a lasting effect on the system.
- *Robustness to inaccurate measurements*. Controlling a running system usually relies on monitoring and/or other measurement mechanisms. Each of these might be affected by noise, or might require a certain time to converge to a convenient accuracy. A controller should provide a reasonable behavior even in presence of such measurement errors. Besides reducing the sensitivity to measurement errors, robustness allows for the use of less invasive monitoring instruments, sometimes required for high accuracy but expensive in terms of performance overhead.

Note that here we do not consider the overshooting as it is not relevant in fuzzy controllers. The settling time is also considered in the stability analysis of the designed controller.

In this chapter, we use the above-mentioned controller properties (i.e., stability and robustness) in conjunction with the research claims of this thesis (i.e., runtime efficiency, scalability and applicability, see Chapter 1) as evaluation criteria to assess the validity of the solution framework proposed in this thesis. We use a real-world connector as a case study and we use an experimental evaluation approach to demonstrate the validity of the solution framework in the next section.

7.4. Case Study, Implementation and Experimental Evaluation

In this section, we present a real-world component connector, we call it here ElasticQueue, which is used in many cloud-enabled software applications. We use this component connector as a case study through which we demonstrate the validity of the **research claim 4** of this thesis (i.e., real-world applicability, refer to Introduction chapter, cf. RQ3).

The approach presented in this thesis develops a set of techniques and methods to control the uncertainties in the self-adaptation control loop of component connectors. However, it is not evident that these techniques and methods are actually useful in real-world settings. In order to evaluate the applicability of our approach in a real-world context, we present a case study and a number of experimental evaluations to provide evidence of the applicability of our approach in real-world scenarios.

7.4.1. ElasticQueue as a concrete case of self-adaptive component connectors

In this section, we provide a number of usage scenarios of ElasticQueue in the context of cloud architectures. A software application running in the cloud is typically expected to handle a large number of requests from different geographic regions. If the application is designed to process each request synchronously, it would then result in a high response time and a bad user experience. In order to resolve this issue, a common design pattern is to pass requests through an intermediate messaging system to another service (a *consumer* service) that handles them asynchronously. This strategy helps to ensure that the business logic in the application is not blocked while the requests are being processed.

Cloud storage such as Queues makes it possible to architect decoupled applications. Queues are powerful because they allow for client applications to submit messages at a high rate of speed, one that may exceed the ability of the backend server to process. As the queue size begins to grow, more resources can be added to increase scale and therefore process messages in a timely fashion. Therefore, queues have become a core building block of cloud architectures. This design pattern is common in most cloud-based applications and is not limited to the arriving workloads from outside the application, it may also be used internally for smoothing requests in different parts of the application concerning different purposes. In this section, we review different usage scenarios of such a pattern by giving concrete examples of applications of this pattern in the architectural design of cloud applications. Note that we have identified these usage scenarios by systematic investigation of existing literature and our own experience in the development of cloud-based applications.

Usage scenario 1. *Competing consumers* (Homer, Sharp, Brader, Narumoto, & Swanson, 2014)**.**

The number of requests could vary significantly over time for many expected or unexpected reasons. A sudden burst in aggregated requests from multiple tenants may cause unpredictable workloads. At peak hours, a system may need to process many hundreds of requests per second, while at other times this number could be very small. Additionally, the type of the process performed to handle these requests may be highly variable. Using a single instance of the processing component (cf. Figure 7.4) may cause an overload in the messaging system by the arrival of messages to the application. To handle this fluctuating workload, the system can run multiple instances of the processing component. The workload needs to be load-balanced across consumers to prevent an instance from becoming a bottleneck. In this scenario, the elastic queue stores the messages and consumers can pick up the messages from a single point for processing.

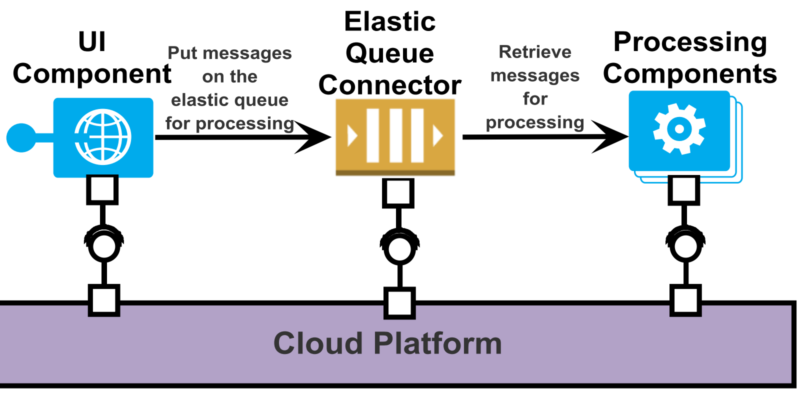

UI Component
Put messages on the elastic queue for processing
Elastic Queue Connector
Retrieve messages for processing
Processing Components
Cloud Platform

Figure 7.4. An instance of competing consumers in a typical cloud architecture.

Usage Scenario 2. *Prioritized requests* (Fehling, Leymann, Retter, Schupeck, & Arbitter, 2014; Homer et al., 2014)**.**

Applications may delegate specific tasks to other services; for example, to perform background processing or to integrate with other applications or services. In cloud applications, a message queue is typically used to delegate tasks to background processing. In many cases, the order in which requests are received by a service is not important. However, in some cases it may be necessary to prioritize specific requests (cf. Figure 7.5). These requests should be processed earlier than others of a lower priority that may have been sent previously by the application. The elastic queue, in this scenario, plays the role of temporal storage for each priority line.

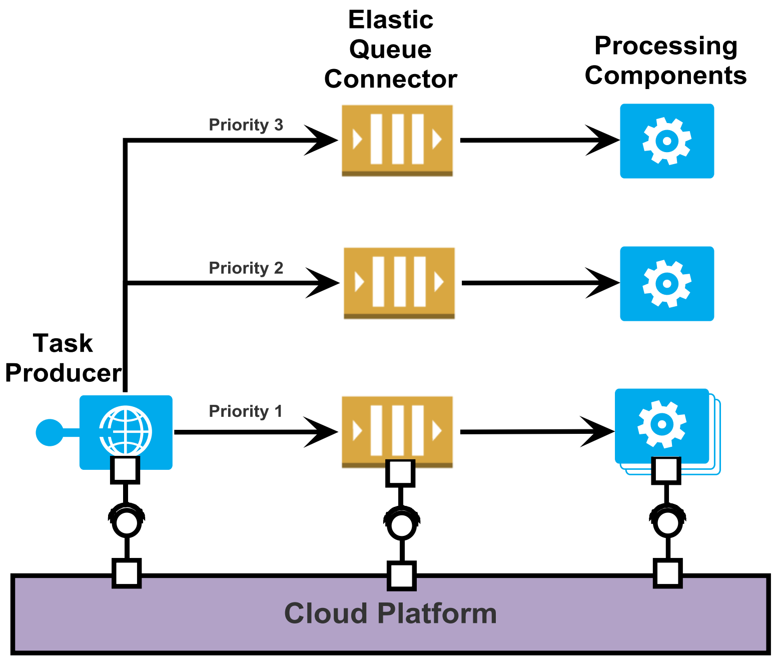


Figure 7.5. The adoption of elastic queue in prioritized requests.

Usage scenario 3. *Pipes and filters* (Fehling et al., 2014; Homer et al., 2014; Medvidovic & Taylor, 2009)**.**

An application may be required to perform a variety of tasks of varying complexity. The processing tasks performed by each module, or the deployment requirements for each task, could change as business requirements are amended. Some tasks might be compute-intensive and could benefit from running on powerful hardware, while others might not require such expensive resources. Furthermore, additional processing might be required in the future, or the order in which the tasks performed by the processing could change. A sequence of message queues can be used to provide the infrastructure required to implement a pipeline. An initial message queue receives unprocessed messages. As illustrated in Figure 7.6, a component, acting as a filter, listens for a message on this queue, performs its work, and then posts the transformed message to the next queue in the sequence. Another filter task can listen for messages on this queue, process them, and post the results to another queue, and so on until the fully transformed data appears in the final message in the queue. In this scenario, the elastic queue plays the role of pipes in this architectural style.

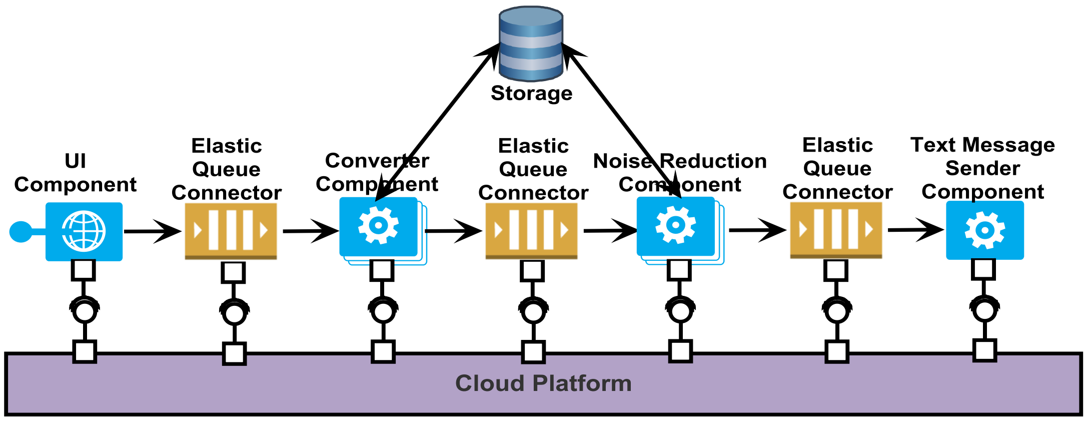
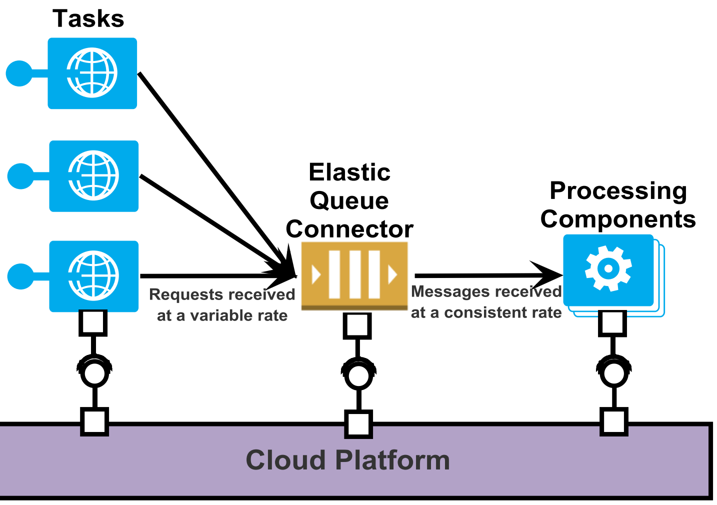


Figure 7.6. An example of pipes-and-filters architecture in the cloud by exploiting elastic queues.

Usage scenario 4. *Load leveling* (Wilder, 2012)**.**

Many solutions in the cloud involve running tasks that invoke services. In this environment, if a service is subjected to intermittent heavy loads, it can cause performance or reliability issues. A service could be a component that is part of the same solution as the tasks that utilize it, or it could be a third-party service providing access to frequently used resources such as a cache or a storage service. If the same service is utilized by a number of tasks running concurrently, it can be difficult to predict the volume of requests to which the service might be subjected at any given point in time (cf. Figure 7.7). It is possible that a service experiences peaks in demand that cause overload and is unable to respond to requests in a timely manner. Overloading a service with a large number of concurrent requests may also result in the service failing if it is unable to handle the contention that these requests could cause. In this scenario, the elastic queue is responsible for leveling the requests (cf. Figure 7.7).



Tasks
Elastic Queue Connector
Processing Components
Requests received at a variable rate
Messages received at a consistent rate
Cloud Platform

Figure 7.7. The adoption of elastic queue for load leveling.

Usage scenario 5. *Request scheduling* (Homer et al., 2014)***.***

An application performs tasks that comprise a number of steps, some of which may invoke remote services or access remote resources. The individual steps may be independent of each other, but they are orchestrated by the application logic that implements the task (cf. Figure 7.8). Whenever possible, the application should ensure that the task runs to completion and resolves any failures that might occur when accessing remote services or resources. These failures could occur for a variety of reasons. For example, the network might be down, communications could be interrupted, a remote service may be unresponsive or in an unstable state, or a remote resource might be temporarily inaccessible—for example, due to resource constraints. If the application detects a more permanent fault from which it cannot easily recover, it must be able to restore the system to a consistent state and ensure integrity of the entire end-to-end operation. In this scenario, the elastic queue facilitates the scheduling of tasks by temporarily storing them in a reliable and manageable storage.

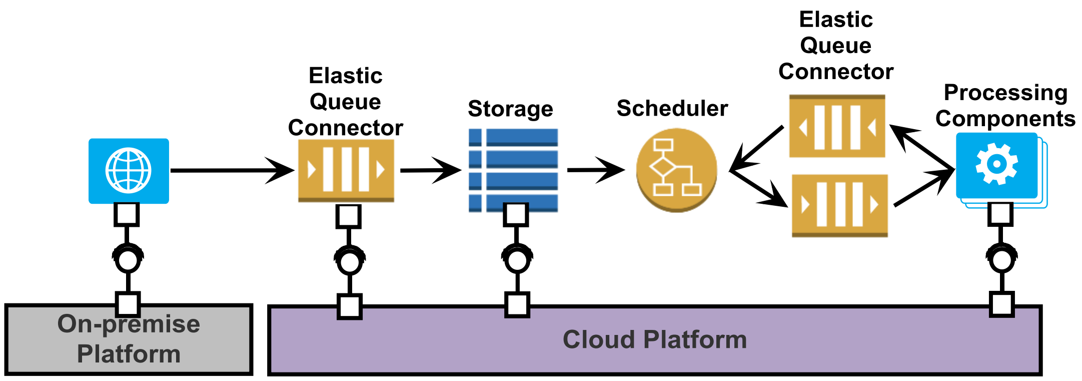


Figure 7.8. The adoption of elastic queue for request scheduling.

Usage scenario 6. Multi-cloud integration (Jamshidi & Pahl, 2014)**.**

As illustrated in Figure 7.9, the integration dimension is very important when building cloud-native applications that are distributed among different cloud environments of a hybrid cloud, or have to be integrated with other applications (of one or several customers) hosted in different environments, on-premises and in the cloud. The elastic queue, in this scenario, may reside on one (or both) cloud platform(s) to enable the communications between different application layers.

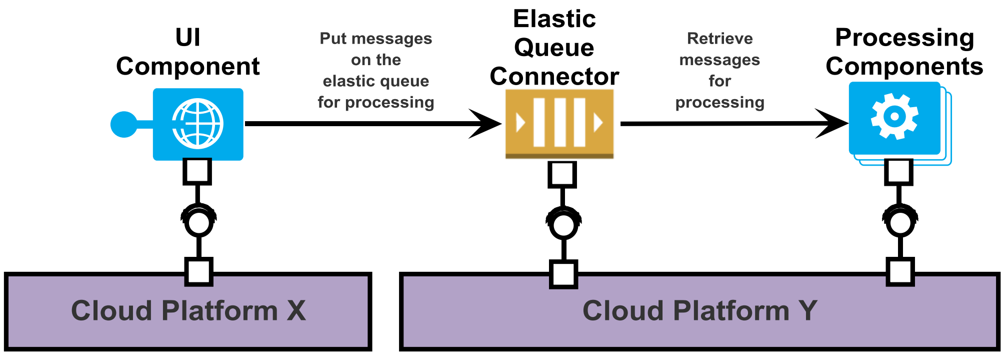


Figure 7.9. An instance of integration in multi/hybrid cloud by heterogeneous components with elastic queue.

Usage scenario 7. Hybrid integration (Jamshidi & Pahl, 2014)**.**

Figure 7.10 shows a processing functionality that experiences varying workload. This component is hosted in an elastic cloud while the rest of an application resides in a static environment (such as an on-premise data center). The elastic queue, in this scenario, plays an important role for integrating the different parts of the hybrid deployment.

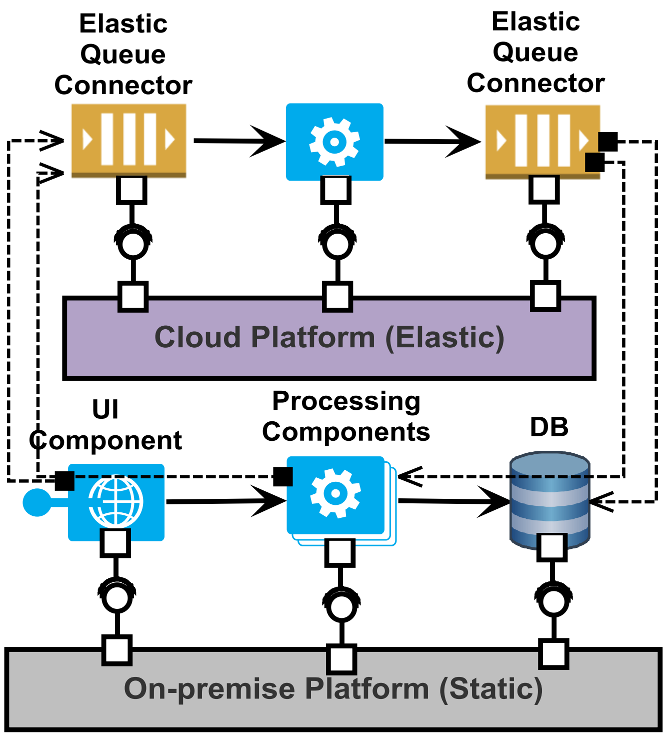


Figure 7.10. An instance of integration in hybrid cloud by connecting static layers with elastic queues.

In this section, we reviewed different real-world adoptions of different variations of the ElasticQueue connector. The main objective of presenting such a comprehensive set of usage scenarios was to demonstrate that such software connectors have been adopted in real domains (cf. research claim 4 in Chapter 1). This was the main motivation in choosing ElasticQueue as a case study to evaluate our solution framework. The other motivation behind this choice was that the cloud environment contains several sources of uncertainty, see (Jamshidi et al., 2014).

7.4.2. Architectural modes of ElasticQueue component connector

In the experiments that we performed as a part of this case study, we only considered 5 operating modes for ElasticQueue. The key objective of the experiments is to evaluate the research claims (see Chapter 1) of this thesis. Table 7.1 lists the modes and their corresponding components and Figure 7.11 represents their corresponding architectural designs.

Table 7.1. Linguistic labels to describe ElasticQueue operating mode.

ElasticQueue Mode	Interface Component	Processing Components
Normal	1	1
Effort	1	2
Medium Effort	1	3
High Effort	1	4
Maximum Effort	1	5

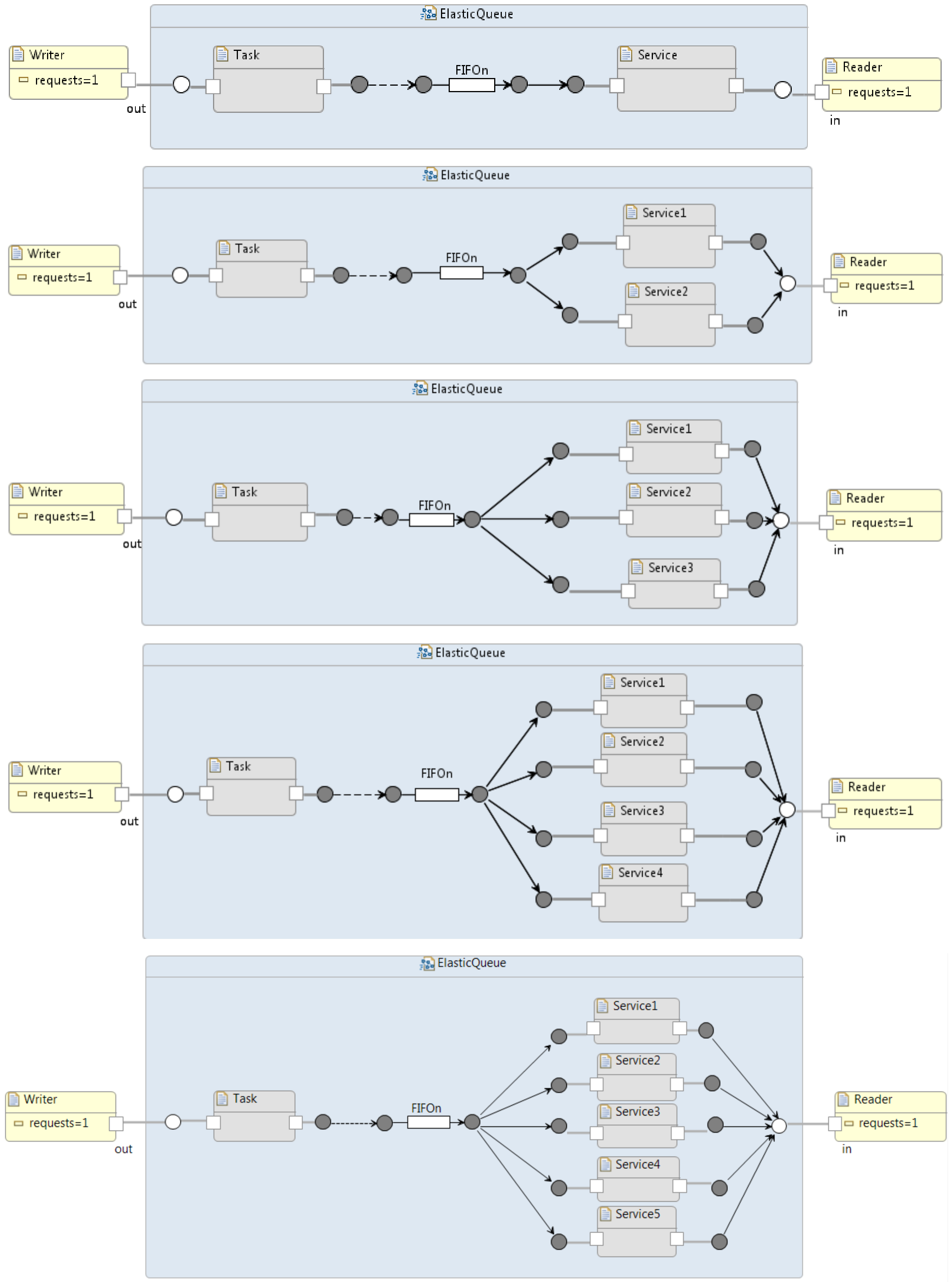


Figure 7.11. Five architectural modes of ElasticQueue (cf. Table 7.1).

7.4.3. Tool Support: Design Components of the ElasticQueue

In this section, we present different parts of the tool that we implemented to demonstrate the validity of the research claims that we mentioned in Section 7.1. The tool that we implemented is divided into several parts covering the different phases of the feedback control loop to enable the self-adaptation of ElasticQueue.

Figure 7.12 illustrates these parts and how they correspond to different phases of the feedback control loop architecture. There are six main parts: (1) **ElasticQueue**,is the connector that we want to adapt based on the environmental situation (i.e., request load) and system performance (response time); (2) **Load generator**, synthetically generates workload to simulate the usage pattern of a cloud-based application; (3) **Monitoring** measures the metrics, which are required to decide about adaptation of the connector; (4) **Smoothing/prediction** covers the process of model calibration for requirement verification and predicting the future workload based on historical data; (5) **Scaling engine** takes the smoothed monitoring data as input and produces appropriate adaptation actions according to the policies in the knowledge base; and finally (6) **Change actuator** enacts the change to the running connector on the fly.

In order to have a better understanding of the tools, in this section we describe in detail the set of tools designed and implemented around ElasticQueue that we call RobusT2Scale (Jamshidi et al., 2014). It is important to mention that RobusT2Scale, in essence, is a concrete realization of the RobusT2 framework (that we have described in Chapter 5) for the specific type of component connector, i.e., ElasticQueue. The main purpose of this realization is to demonstrate that the proposed framework (in Chapter 5) can be adopted for adapting real-world connectors.

In the following, we describe the architectural design and overview of each module and in the next section (i.e., Section 7.4.4), we describe the implementation and technological details of the tool chain as shown in Figure 7.12. Note that for describing each module of this realization of the RobusT2 framework, we use specific scenario to demonstrate the details of the architectural design of that module.

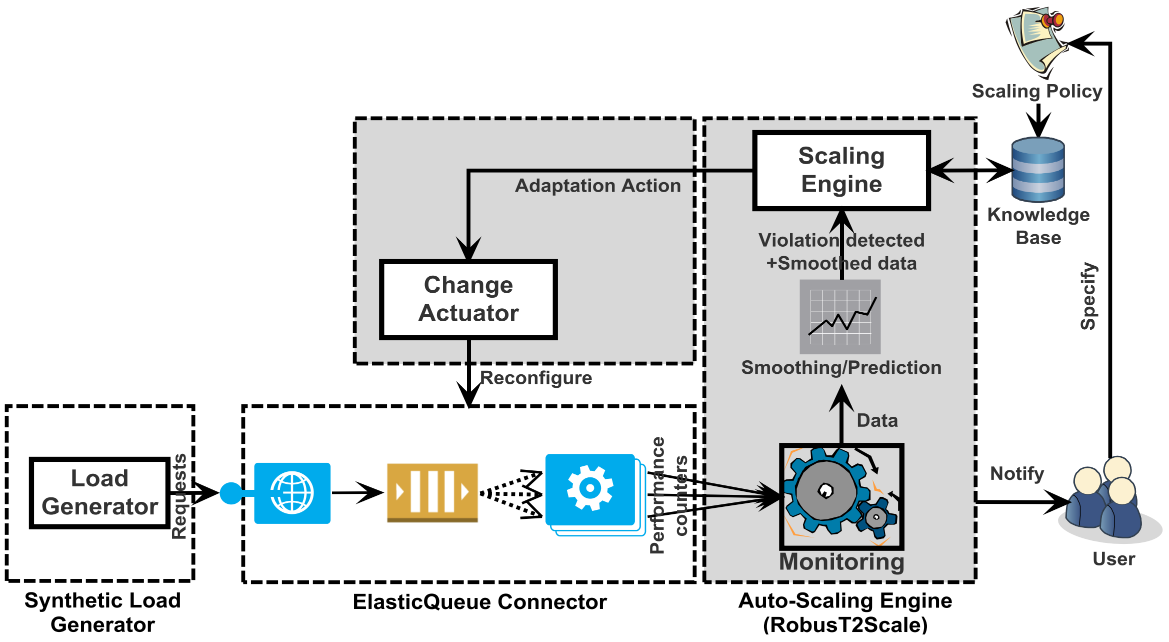


Figure 7.12. Tool chain architecture.

ElasticQueue is a cloud service containing a web role and a (number of) worker role(s) designed with a layered architectural style. In this sample, it represents a service that needs dynamic scaling to handle the load being placed on it. The web role exposes a web service that adds task items to a queue. The worker role(s) picks items off the queue and processes them. The RobusT2Scale framework is responsible for reconfiguring the ElasticQueue connector at runtime.

Load Generator is a client-side (on-premise) application (cf. Figure 7.17) that calls the service hosted by the ElasticQueue web role. The Load Generator is a syntactic workload generator employed to simulate various levels of load on the ElasticQueue connector (cf. Figure 7.13).

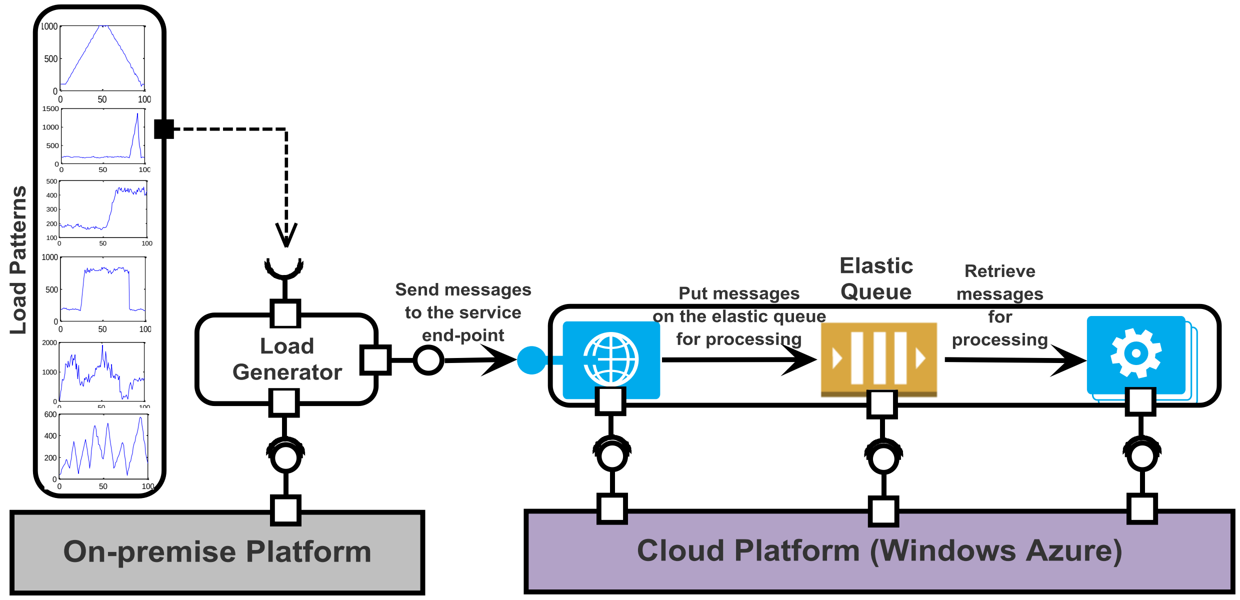

Load Patterns
Load Generator
Send messages to the service end-point
Put messages on the elastic queue for processing
Elastic Queue
Retrieve messages for processing
On-premise Platform
Cloud Platform (Windows Azure)

Figure 7.13. ElasticQueue architecture – load injection scenario.

Monitoring is a web application that serves two purposes. As illustrated in Figure 7.14, it 1) serves up a client to the user and then 2) exposes the metrics gathered by the Scaling Engine to this client via a service. We have developed this module in a way that it can be hosted either on-premise or as a cloud hosted application. The dashboard (cf. Figure 7.16) displays metric data, pending reconfiguration actions and the configuration of the ElasticQueue in an easily understandable way.

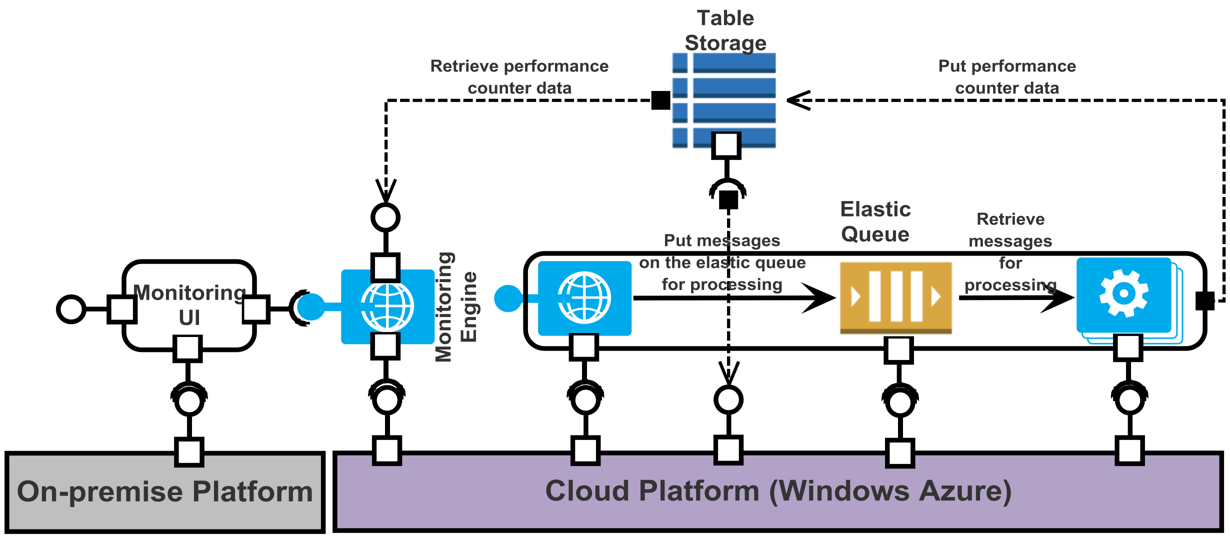


Table Storage
Retrieve performance counter data
Put performance counter data
Elastic Queue
Retrieve messages for processing
Put messages on the elastic queue for processing
Monitoring UI
Monitoring Engine
On-premise Platform
Cloud Platform (Windows Azure)

Figure 7.14. ElasticQueue architecture - monitoring scenario.

Smoothing/prediction enables requirement verification at runtime and predicts short-term future usage of the connector based on historical data collection. This module gathers historical data metrics, such as the number of queued-up work items and the number of requests per second, from the cloud storage. Using these metrics, it determines whether the application needs to be adapted. Note that the enabler of this part is the model calibration mechanism as a part of RobustMC framework proposed in this thesis. The details of the RobustMC framework are given in Chapter 4.

Scaling Engine is the part of the design that is responsible for enforcing the scaling policies. We built this module in a way that can be hostable both on-premise and in the cloud. For this experiment, it has been built as a console application (cf. Figure 7.18) running on-premise. The module is responsible for reasoning about adaptation based on fuzzy reasoning. Note that the adaptation reasoning in this module is based on the RobusT2 framework developed as a part of this thesis. The details of the RobusT2 framework and the fuzzy reasoning is given in Chapter 5. If a need to reconfigure the ElasticQueue is determined, it calls the Cloud Service Management API to start off this action. While we host this logic client-side on-premise. It would also be possible to locate this logic in a cloud platform (here Windows Azure worker role) - the scaling engine stores all its data in a cloud storage (cf. Figure 7.15).

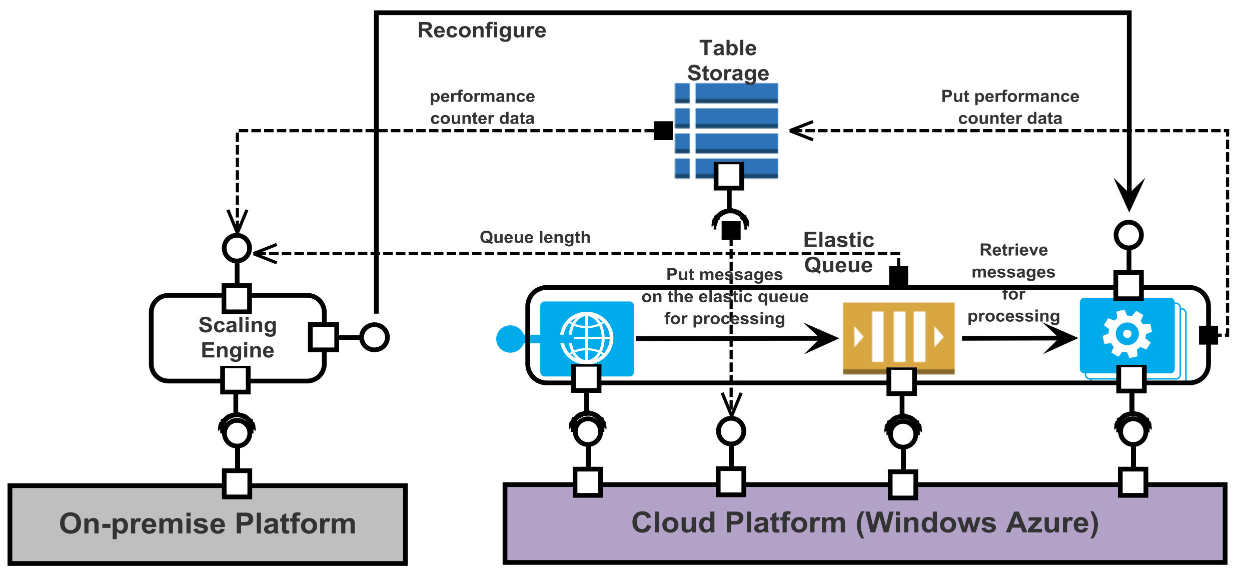

Reconfigure
Table Storage
performance counter data
Put performance counter data
Queue length
Elastic Queue
Put messages on the elastic queue for processing
Retrieve messages for processing
Scaling Engine
On-premise Platform
Cloud Platform (Windows Azure)

Figure 7.15. ElasticQueue architecture - scaling scenario.

Change actuator enacts the change to the running connector on the fly. The key benefit of this evaluation is that we adopt a real-world public cloud platform to demonstrate the validity of our approach rather than an artificial simulation environment. For enacting the change, after determining the required reconfiguration commands, this module calls the Cloud Service Management API (in this case is Azure Service Management REST API[3]) to start off the reconfiguration actions. The details of the adaptation mechanism is described in Chapter 6.

Until this point, we have described the high-level design of the tool chain in order to enable self-adaptation of the ElasticQueue connector. In the next section, we describe the implementation details and the technologies that we adopted to realize self-adaptive connectors.

7.4.4. Implementation technologies of the ElasticQueue

In this section, we describe the implementation view of the framework that we have realized (by extending and adapting an existing Azure monitoring and enactor modules in MSDN code library) to enable reliable self-adaptation of ElasticQueue as our case study in this chapter. The implementation consists of 3 .NET solutions, each with a certain area of responsibility as follows:

1. The solution named ElasticQueue contains the projects that represent the functionalities of the ElasticQueue. This project contains an Azure service with one Azure web role and one worker role. The Web Role exposes a web-service with a single method that places a message on a queue. The worker role then takes one message off the queue and processes the message accordingly.
2. The ScalingEngine solution contains two projects:
    a. A console application called ScalingEngineClient (see Figure 7.18) and a Windows Presentation Foundation (WPF) application called LoadClient (see Figure 7.17). The ScalingEngineClient is responsible for most of the work in this experiment- it houses the RobusT2 framework. It is responsible for continuously monitoring the queue length, the

[3] http://msdn.microsoft.com/en-us/library/azure/ee460799.aspx

requests per second performance counter as well as the current instance count. It takes these metrics and saves them to 2 tables in Azure table storage. It also feeds them to the RobusT2 framework to determine, based on the fuzzy reasoning, to which mode the ElasticQueue needs to be reconfigured. If any adaptations are needed, it uses the Service Management API to initiate the change on the cloud platform. The Service Management API provides programmatic access to much of the functionality available through the Management Portal. The Service Management API is a REST API. All API operations are performed over security certificates.

   b. The LoadClient is a syntactic load generator that is responsible for simulating load patterns. It allows the user to determine the amount of load to simulate and then starts calling the ElasticQueue end-point with the desired amount of times per second. It also lets the user to track the current queue length through its UI as illustrated in Figure 7.17.

3. The Monitoring solution includes an ASP.NET web-application as well as a Windows Azure web role, making it possible to host it both on-premise and in Azure. The solution also contains a Silverlight application that is then hosted within the web role. The Monitoring service makes it possible to watch the current state of the ElasticQueue in close to real-time. The monitoring client uses a simple to understand UI (see Figure 7.16), allowing an end user to monitor the current queue length, instance count as well as any adaptation actions, currently happening as well as previously executed.

As discussed above the primary logic for adapting the ElasticQueue connector is contained in the ScalingEngine. A number of useful visualization components are also contained in the Monitoring application. This section will discuss the detailed implementation of both of these components of the experiment.

The ScalingEngine, as mentioned before, is responsible for tracking the current load on the application and then reconfiguring the ElasticQueue accordingly. To enable this it utilizes a couple of configurable objects. First, it uses a list of MetricProvider objects. The sole responsibility of a MetricProvider is to collect metric data from the Azure-based application. As soon as new data is obtained, it is evaluated to determine if there is a need to reconfigure the ElasticQueue. This is done by the use of another list of custom objects called ScalingLogicProviders, which can be thought of as the encapsulation of a scaling rule. The metric data collected by the MetricProvider is passed to each of the defined ScalingLogicProviders, one at a time. These then take that data collected by the MetricProvider and use some logic to determine if there is a need to initiate an adaptation action. If this is the case, the requested adaptation is passed to the last list of objects. They are called TimeLogicProviders and have two responsibilities. First, they are responsible for verifying that any scaling change initiated by the ScalingLogicProviders will keep the instance count within the configured values for the current time. If they accept the requested scaling change, the scaling engine makes a call to the Azure Service Management API to initiate the change. The second responsibility for the TimeLogicProviders is to initiate any scaling change needed to stay within the configured max and min values for the current time. For example shutting down worker instances of the ElasticQueue once the weekend has arrived (see Figure 7.19). Calls made to the Azure Service Management API are highly privileged operations. These calls will affect both the performance of an application as well as the costs. As such, this service needs to be secured appropriately.

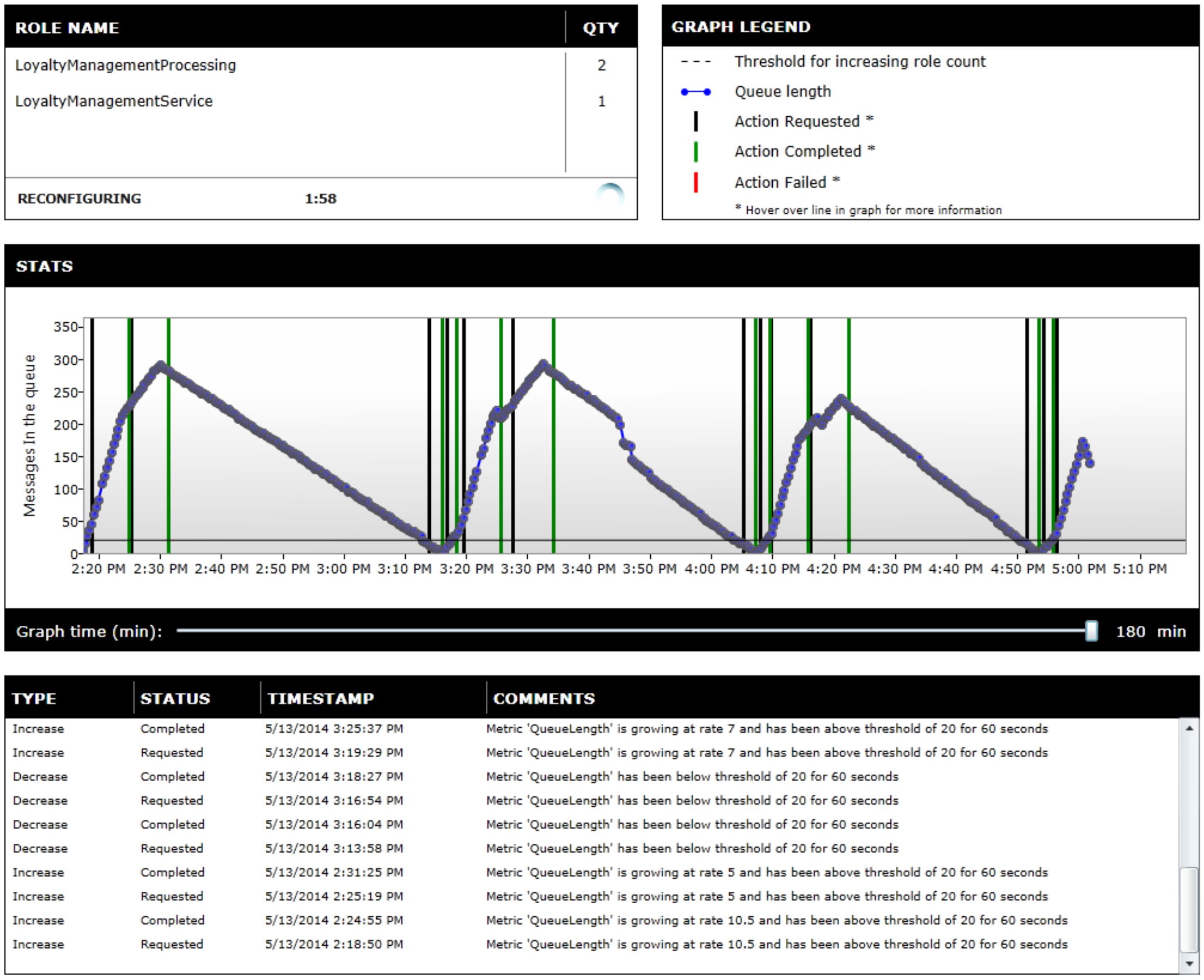
The other implementation issue was to avoid oscillations due to constantly adding or removing resources. For avoiding such situations, in each control interval, the ScalingEngine checks whether there is a pending scaling action that has not been enacted to the ElasticQueue yet. If it finds that there is a pending action, it basically ignores (see Figure 7.20) the decisions made in this control interval and proceeds to the next round.

The Monitoring part is a web application. This module has two responsibilities: 1) it is responsible for serving up the monitoring data to the client, as well as 2) hosting a web service. The monitoring module to get the necessary information from Azure Storage uses the web service. Other clients if needed also could use the web service. The monitoring takes the data retrieved from the web service and displays it in a visual dashboard using lists and graphs. The monitoring module is built using the Model-View-ViewModel pattern (Microsoft, 2014).

7.4.4.1. Architectural reconfiguration challenges in the cloud

In order to reconfigure ElasticQueue in the cloud, there are some known technical challenges that we have also faced during the experimental evaluation of this work:

- Taking action to scale on cloud platforms is exposed through the use of the Management APIs. Through a call to this API, it is possible to change the instance count in the service configuration and in doing so to change the number of running instances.
- When adding and removing instances, it is important to remember that the Management API is an asynchronous API. This means that once a change has been requested, an application will need to poll the service to determine if and when that change has taken effect.
- Applications need to ensure that rules are aware of the time delay in adding more capacity and that they do not result in significant excess capacity being added as a result of subsequent evaluations of rules while new instances are being started.
- The load placed on most applications is quite stochastic; however, at a higher level most applications will display some broad trends in load.



ROLE NAME	QTY
LoyaltyManagementProcessing	2
LoyaltyManagementService	1

RECONFIGURING 1:58

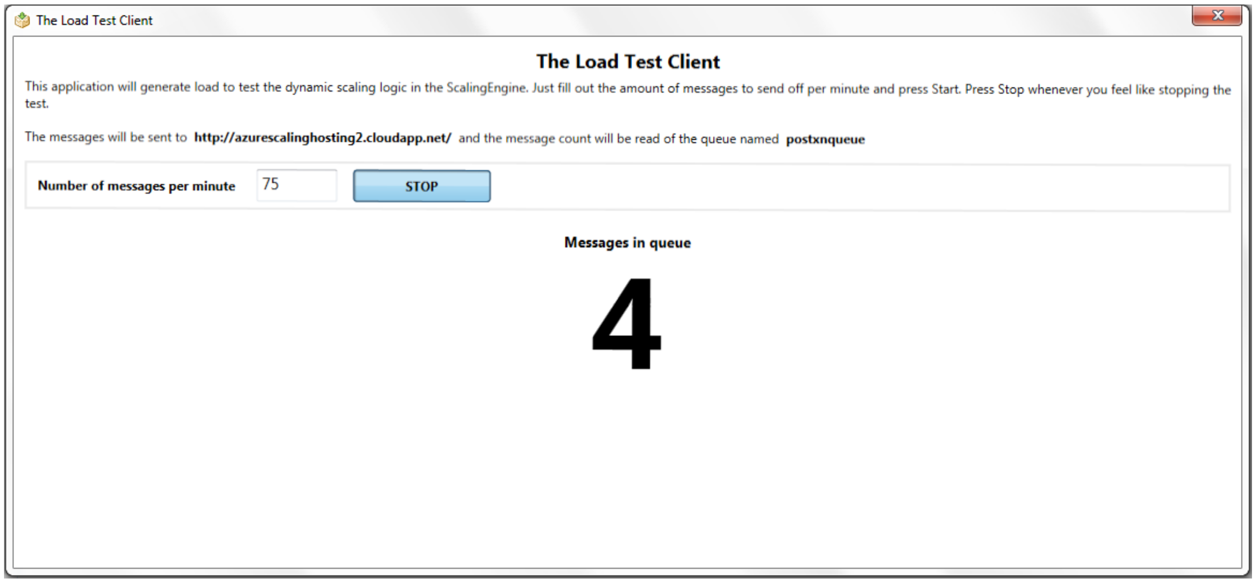
GRAPH LEGEND	
- - -	Threshold for increasing role count
●─●	Queue length
▮	Action Requested *
🟩	Action Completed *
🟥	Action Failed *

* Hover over line in graph for more information

STATS

TYPE	STATUS	TIMESTAMP	COMMENTS
Increase	Completed	5/13/2014 3:25:37 PM	Metric 'QueueLength' is growing at rate 7 and has been above threshold of 20 for 60 seconds
Increase	Requested	5/13/2014 3:19:29 PM	Metric 'QueueLength' is growing at rate 7 and has been above threshold of 20 for 60 seconds
Decrease	Completed	5/13/2014 3:18:27 PM	Metric 'QueueLength' has been below threshold of 20 for 60 seconds
Decrease	Requested	5/13/2014 3:16:54 PM	Metric 'QueueLength' has been below threshold of 20 for 60 seconds
Decrease	Completed	5/13/2014 3:16:04 PM	Metric 'QueueLength' has been below threshold of 20 for 60 seconds
Decrease	Requested	5/13/2014 3:13:58 PM	Metric 'QueueLength' has been below threshold of 20 for 60 seconds
Increase	Completed	5/13/2014 2:31:25 PM	Metric 'QueueLength' is growing at rate 5 and has been above threshold of 20 for 60 seconds
Increase	Requested	5/13/2014 2:25:19 PM	Metric 'QueueLength' is growing at rate 5 and has been above threshold of 20 for 60 seconds
Increase	Completed	5/13/2014 2:24:55 PM	Metric 'QueueLength' is growing at rate 10.5 and has been above threshold of 20 for 60 seconds
Increase	Requested	5/13/2014 2:18:50 PM	Metric 'QueueLength' is growing at rate 10.5 and has been above threshold of 20 for 60 seconds

Figure 7.16. Monitoring dashboard UI of RobusT2Scale.


The Load Test Client

The Load Test Client

This application will generate load to test the dynamic scaling logic in the ScalingEngine. Just fill out the amount of messages to send off per minute and press Start. Press Stop whenever you feel like stopping the test.

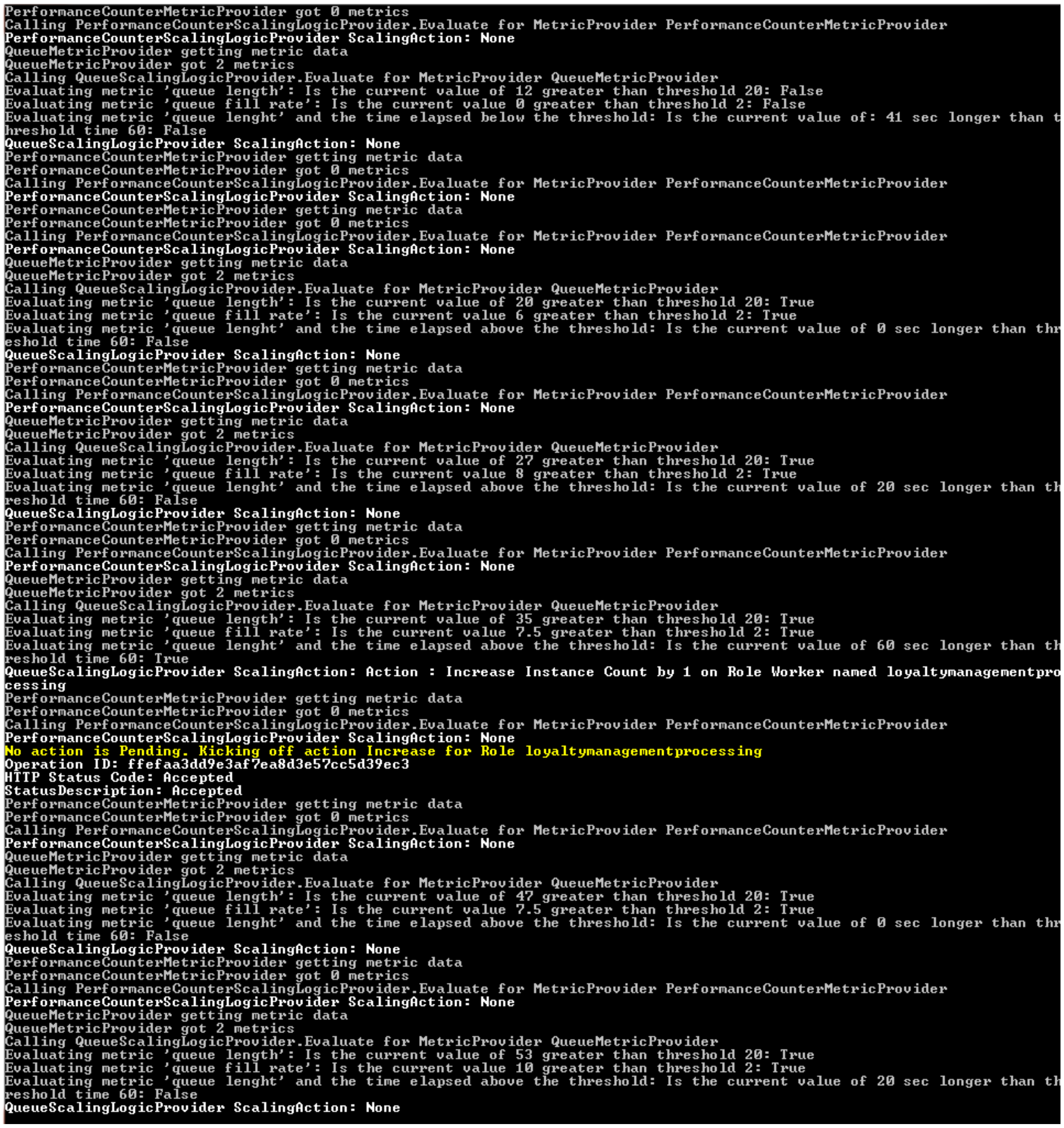
The messages will be sent to **http://azurescalinghosting2.cloudapp.net/** and the message count will be read of the queue named **postxnqueue**

Number of messages per minute 75

Messages in queue

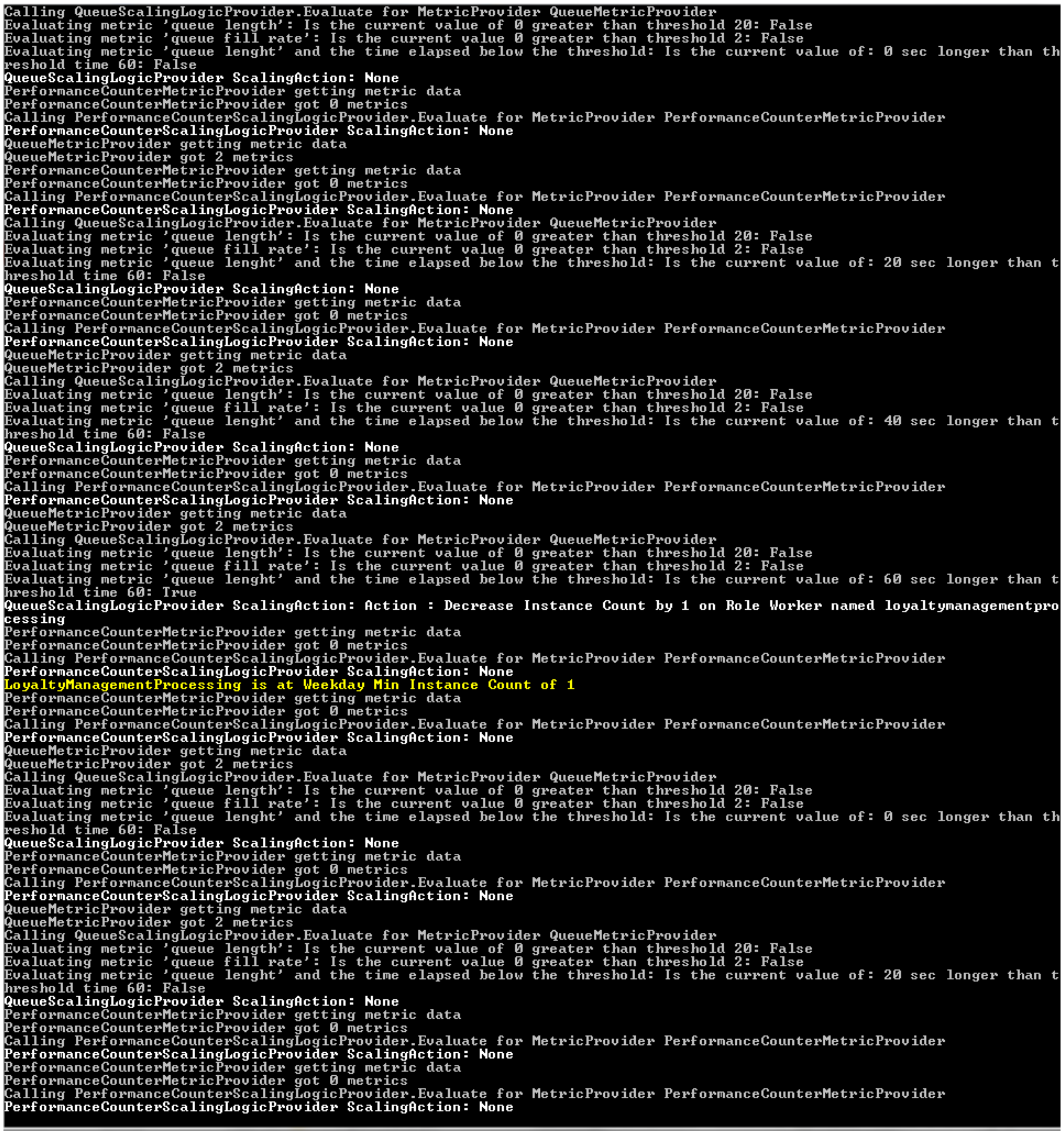
4

Figure 7.17. Load generator UI.



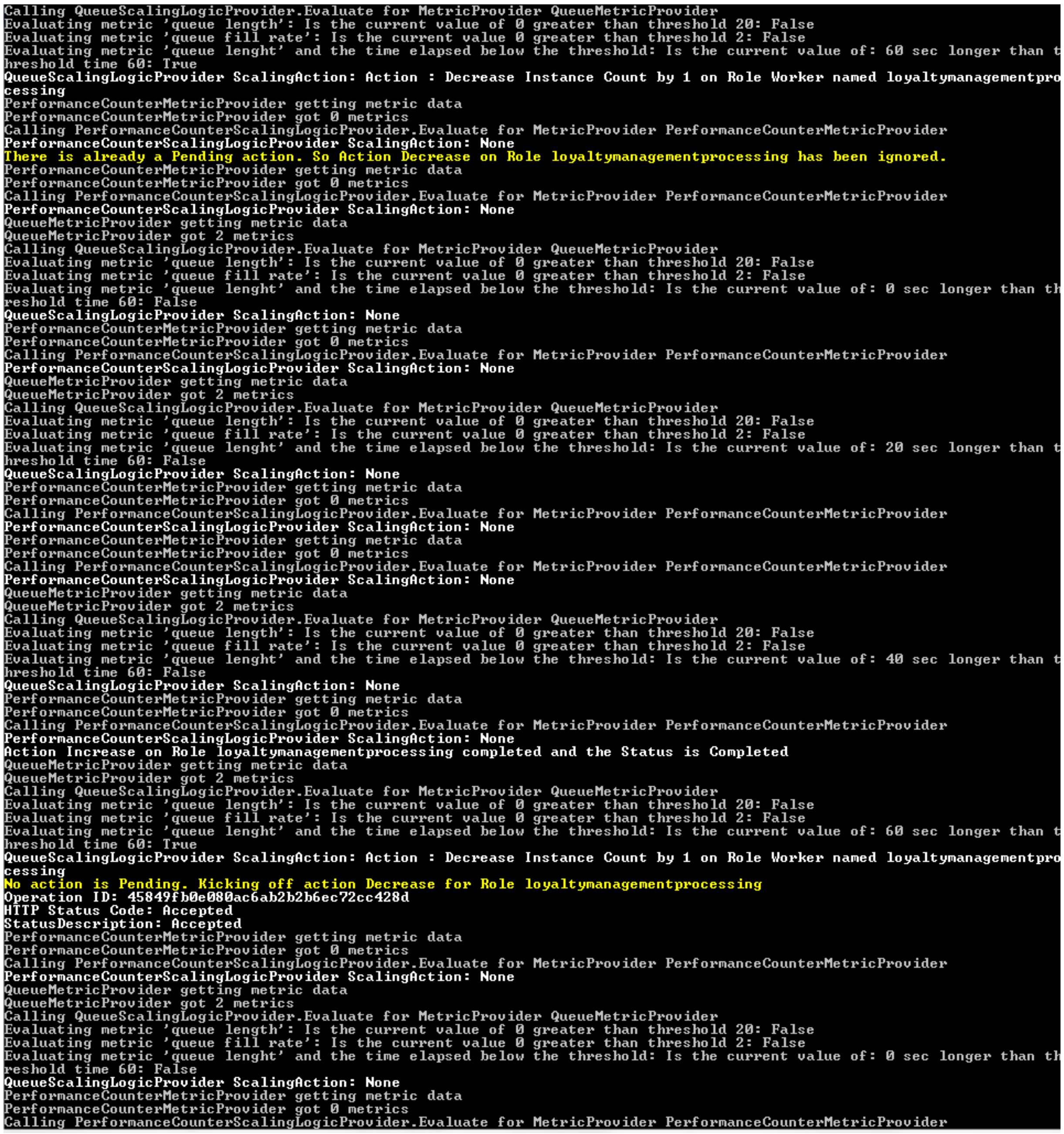
```
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 12 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 41 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 20 greater than threshold 20: True
Evaluating metric 'queue fill rate': Is the current value 6 greater than threshold 2: True
Evaluating metric 'queue lenght' and the time elapsed above the threshold: Is the current value of 0 sec longer than thr
eshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 27 greater than threshold 20: True
Evaluating metric 'queue fill rate': Is the current value 8 greater than threshold 2: True
Evaluating metric 'queue lenght' and the time elapsed above the threshold: Is the current value of 20 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 35 greater than threshold 20: True
Evaluating metric 'queue fill rate': Is the current value 7.5 greater than threshold 2: True
Evaluating metric 'queue lenght' and the time elapsed above the threshold: Is the current value of 60 sec longer than th
reshold time 60: True
QueueScalingLogicProvider ScalingAction: Action : Increase Instance Count by 1 on Role Worker named loyaltymanagementpro
cessing
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
No action is Pending. Kicking off action Increase for Role loyaltymanagementprocessing
Operation ID: ffefaa3dd9e3af7ea8d3e57cc5d39ec3
HTTP Status Code: Accepted
StatusDescription: Accepted
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 47 greater than threshold 20: True
Evaluating metric 'queue fill rate': Is the current value 7.5 greater than threshold 2: True
Evaluating metric 'queue lenght' and the time elapsed above the threshold: Is the current value of 0 sec longer than thr
eshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 53 greater than threshold 20: True
Evaluating metric 'queue fill rate': Is the current value 10 greater than threshold 2: True
Evaluating metric 'queue lenght' and the time elapsed above the threshold: Is the current value of 20 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
```

Figure 7.18. Scaling engine UI (a reconfiguration is initiated).



```
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 0 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 20 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 40 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 60 sec longer than t
hreshold time 60: True
QueueScalingLogicProvider ScalingAction: Action : Decrease Instance Count by 1 on Role Worker named loyaltymanagementpro
cessing
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
LoyaltyManagementProcessing is at Weekday Min Instance Count of 1
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 0 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 20 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
```

Figure 7.19. Scaling engine UI (sensitivity to environmental data).



```
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 60 sec longer than t
hreshold time 60: True
QueueScalingLogicProvider ScalingAction: Action : Decrease Instance Count by 1 on Role Worker named loyaltymanagementpro
cessing
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
There is already a Pending action. So Action Decrease on Role loyaltymanagementprocessing has been ignored.
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 0 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 20 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 40 sec longer than t
hreshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
Action Increase on Role loyaltymanagementprocessing completed and the Status is Completed
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 60 sec longer than t
hreshold time 60: True
QueueScalingLogicProvider ScalingAction: Action : Decrease Instance Count by 1 on Role Worker named loyaltymanagementpro
cessing
No action is Pending. Kicking off action Decrease for Role loyaltymanagementprocessing
Operation ID: 45849fb0e080ac6ab2b2b6ec72cc428d
HTTP Status Code: Accepted
StatusDescription: Accepted
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
PerformanceCounterScalingLogicProvider ScalingAction: None
QueueMetricProvider getting metric data
QueueMetricProvider got 2 metrics
Calling QueueScalingLogicProvider.Evaluate for MetricProvider QueueMetricProvider
Evaluating metric 'queue length': Is the current value of 0 greater than threshold 20: False
Evaluating metric 'queue fill rate': Is the current value 0 greater than threshold 2: False
Evaluating metric 'queue lenght' and the time elapsed below the threshold: Is the current value of: 0 sec longer than th
reshold time 60: False
QueueScalingLogicProvider ScalingAction: None
PerformanceCounterMetricProvider getting metric data
PerformanceCounterMetricProvider got 0 metrics
Calling PerformanceCounterScalingLogicProvider.Evaluate for MetricProvider PerformanceCounterMetricProvider
```

Figure 7.20. Scaling engine UI (a reconfiguration is ignored due to a pending action).

One of the key objectives of this evaluation is to show the validity of research claim 4, i.e., the real-world applicability of the solution framework of this thesis. Therefore, we decided to evaluate the solution framework in a real and practical environment rather than through a simulation. The key objective of this thesis is to develop mechanisms to enable self-adaptation of component connectors. These mechanisms should be robust against measurement noise, evaluated in a real practical environment that can provide assurance that this evaluation is trustworthy. It is also remarkable that elastic systems on cloud platforms contain different sources of uncertainties and this provides an appropriate real-world environment in which we can evaluate our solution framework.

7.4.5. Controller design for ElasticQueue: an empirical perspective

In Chapter 5, we present a methodology for designing fuzzy logic controllers appropriate for adapting connectors. In this section, however, we take an empirical perspective, and we describe the data collection procedure for adaptation policy elicitation. We also elaborate on how to design the controllers based on the collected data. The data collection is described in Section 7.4.5.1 and the controller design comprising fuzzy membership function derivation and adaptation rule elicitation is presented in Section 7.4.5.2.

7.4.5.1. Adaptation policy elicitation through survey

An interval type-2 fuzzy logic controller is used to perform the adaptation management of the ElasticQueue. The controller is used to determine an operating mode for the ElasticQueue at runtime. The design of the fuzzy logic system (adaptation rules and membership functions) is done using the knowledge elicitation technique discussed in Chapter 5. This technique, based on surveys, allows extracting adaptation policies from experts in the form of IF-THEN rules.

In order to design the configuration adaptation controller, a data collection was conducted in a survey among 10 experts in cloud computing. The survey was mainly performed among the participants of the Third National Conference on Cloud Computing and Commerce (NC4), April 2014 in Dublin, Ireland. The participants of this survey are affiliated with: University College Cork (UCC), Athlone Institute of Technology (AIT), University of Limerick (UL), and Dublin City University (DCU). The survey was designed at the Irish Centre for Cloud Computing and Commerce (IC4), Ireland. The experts that we asked for this experiment were PhD students and university professors in software engineering and cloud computing. These experts had basic knowledge about fuzzy logic and type-2 fuzzy logic, however, for knowledge elicitation such knowledge is not necessary as we explained in Section 5.5.5.2. Note that since they all had experience in web-based application development, they had a good understanding of workload and response time. For a more detailed description of the survey, we refer to Appendix A.

7.4.5.2. Survey processing

A fuzzy logic controller is completely defined by its fuzzy membership functions and rules. We present the details of how to transform the collected data to type-2 fuzzy membership function in Section 7.4.5.2.1. The rule elicitation is described in Section 7.4.5.2.2 and the output surface of the fuzzy controller is presented in Section 7.4.5.2.3.

The processed results of the survey are used to define the fuzzy sets (two inputs, five MFs per input) and the 25 rules (one MF per rule). Table 7.2 presents the processed results for Questions 1 and 2 of the survey (mean values and standard deviation of the answers, see the survey template in Appendix A). This information is used to define the parameters required to create the membership functions for the inputs of the fuzzy system. Table 7.3 presents the processed results for Question 3 of the survey (summarizes the rules defined by the experts). This information is used to define the 25 fuzzy rules.

Table 7.2. Processed survey results: Workload and Response time.

Linguistic		Means		Standard Deviations	
		Start (a)	End (b)	Start (σ_a)	End (σ_b)
Workload	Very low	0	27	0	8.23
	Low	22	41.5	7.15	7.09
	Medium	36.5	64	5.80	3.94
	High	61	82.5	4.59	6.77
	Very high	78	100	6.32	0
Response time	Instantaneous	0	7.2	0	5.20
	Fast	6.1	20	4.07	5.27
	Medium	18.2	41.5	5.59	8.51
	Slow	38.5	63.5	7.09	9.44
	Very slow	60	100	7.82	0

Table 7.3. Processed survey results: Adaptation rules.

Rule (l)	Antecedents		Consequent					c^l_{avg}
	Workload	Response time	Normal (-2)	Effort (-1)	Medium Effort (0)	High Effort (+1)	Maximum Effort (+2)	
1	Very low	Instantaneous	7	2	1	0	0	-1.6
2	Very low	Fast	5	4	1	0	0	-1.4
3	Very low	Medium	0	2	6	2	0	0
4	Very low	Slow	0	0	4	6	0	0.6
5	Very low	Very slow	0	0	0	6	4	1.4
6	Low	Instantaneous	5	3	2	0	0	-1.3
7	Low	Fast	2	7	1	0	0	-1.1
8	Low	Medium	0	1	5	3	1	0.4
9	Low	Slow	0	0	1	8	1	1
10	Low	Very slow	0	0	0	4	6	1.6
11	Medium	Instantaneous	6	4	0	0	0	-1.6
12	Medium	Fast	2	5	3	0	0	-0.9
13	Medium	Medium	0	0	5	4	1	0.6
14	Medium	Slow	0	0	1	7	2	1.1
15	Medium	Very slow	0	0	1	3	6	1.5
16	High	Instantaneous	8	2	0	0	0	-1.8
17	High	Fast	4	6	0	0	0	-1.4
18	High	Medium	0	1	5	3	1	0.4
19	High	Slow	0	0	1	7	2	1.1
20	High	Very slow	0	0	0	6	4	1.4
21	Very high	Instantaneous	9	1	0	0	0	-1.9
22	Very high	Fast	3	6	1	0	0	-1.2
23	Very high	Medium	0	1	4	4	1	0.5
24	Very high	Slow	0	0	1	8	1	1
25	Very high	Very slow	0	0	0	4	6	1.6

7.4.5.2.1. Fuzzy sets design (membership functions)

The MFs has been selected to be triangular for the labels "Low" ("Fast"), "Medium" and "High" ("Slow") and trapezoidal for the labels "Very low" ("Instantaneous") and "Very high" ("Very slow").

Table 7.4. Locations of the main points of IT2 MFs.

Triangular	Trapezoidal
$l_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$	$ll_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$
$m_{UMF} = ((a+b)/2\,, 1)$	$ul_{UMF} = (a - \alpha\sigma_a, 1)$
$r_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$	$ur_{UMF} = (b + \alpha\sigma_b, 1)$
$l_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$	$lr_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$
$m_{LMF} = ((a+b)/2\,, 1)$	$ll_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$
$r_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$	$ul_{LMF} = (a + \alpha\sigma_a, 1)$
	$ur_{LMF} = (b - \alpha\sigma_b, 1)$
	$lr_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$

As illustrated in Figure 5.24 and Figure 5.25, we used trapezoidal MFs to represent "Very low" ("Instantaneous") and "Very high" ("Very slow"), and triangular MFs to represent "Low" ("Fast"), "Medium" and "High" ("Slow"). Let a and b with standard deviations σ_a and σ_b, respectively, be the mean values of the interval end-points of the linguistic labels (cf. Table 7.2). For "Low", "Medium" and "High" labels, the triangular T1 MF is then constructed by connecting: $l = (a - \sigma_a, 0), m = ((a + b)/2,1), r = (b + \sigma_b, 0)$. Accordingly, for "Very low" and "Very high" labels, the associated trapezoidal MFs can be constructed by connecting: $(a - \sigma_a, 0), (a, 1), (b, 1), (b + \sigma_b, 0)$, see dashed lines in Figure 5.24 and Figure 5.25. As it is indicated by the standard deviations in Table 7.2, there are uncertainties associated with the ends and the locations of the MFs. For instance, one may imagine a triangular T1 MF in $l' = (a - 0.3 * \sigma_a, 0), m = ((a + b)/2,1), r' = (b + 0.4 * \sigma_b, 0)$.

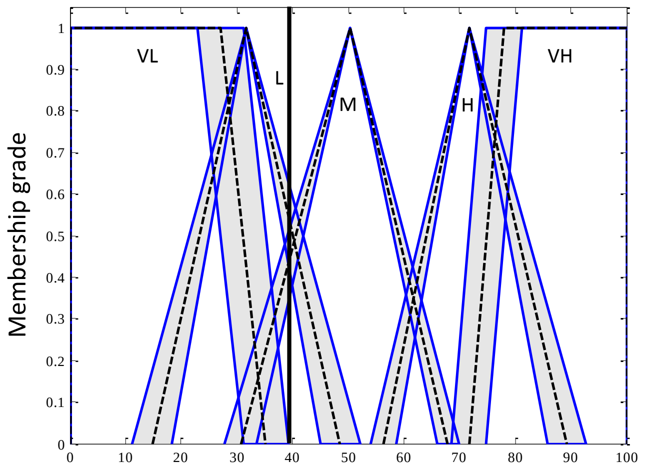


Figure 7.21. IT2 MFs of the workload labels.

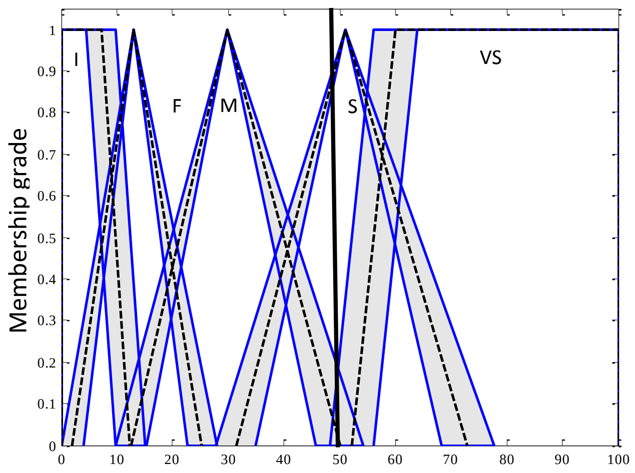


Figure 7.22. IT2 MFs of the response time labels.

7.4.5.2.2. Fuzzy rules design

The survey asked the experts to associate labels to each rule as summarized in Table 7.3. Each of these labels is now associated with a value. As different experts defined different rules, an 'average' rule is retained. Each rule is then associated with the average of the values defined by the experts. The details of the creation of the MFs for the rule base is explained in Chapter 5.

7.4.5.2.3. Fuzzy logic control surface

The fuzzy logic system is completely defined by its membership functions and fuzzy rules. An uncertainty value of $\alpha = 0.5$ is considered. Figure 7.23 shows the fuzzy logic surface representing the fuzzy logic controller designed from the survey.

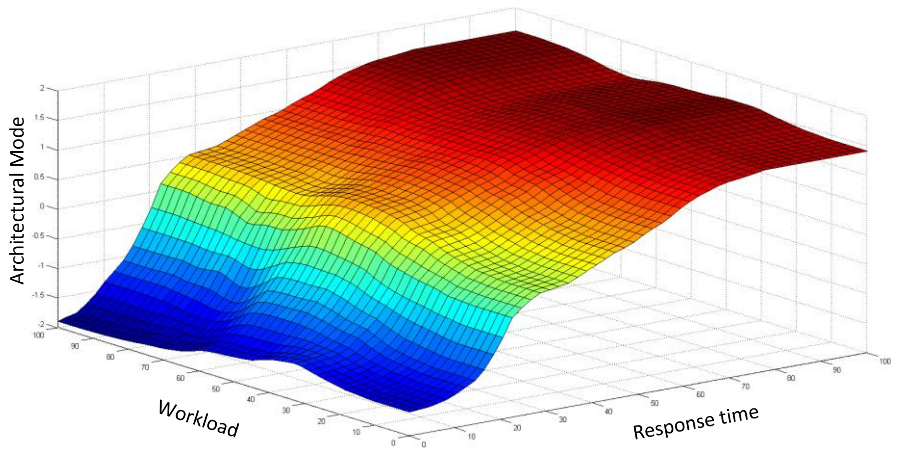


Figure 7.23. Output of the IT2 FLS for elasticity reasoning.

7.4.6. Experimental evaluations

This section presents the validation of the controller developed in Section 7.4.5 for the ElasticQueue adaptation reasoning. As we discussed earlier in the implementation section, the experimental conditions are quite different from those in simulation and particularly regarding the control of the elastic queue in the cloud. In this context, the ElasticQueue is controlled by cloud platforms. The ElasticQueue and its adaptation infrastructure is implemented using .NET technologies and Microsoft Azure PaaS services, see Section 7.4.4. The ElasticQueue makes sure that the tasks submitted to the system are processed reliably within the specified SLA. The controller (i.e., the heart of decision making) is implemented in Matlab and then integrated with the .NET technologies using MATLAB Builder NE for Microsoft .NET Framework. MATLAB Builder™ NE allows creating .NET and COM components from MATLAB® programs that include MATLAB code. This then enables integrating these components into larger .NET, COM, and Web applications and deploying them to computers that do not have MATLAB installed using the MATLAB Compiler Runtime (MCR) that is provided with the MATLAB Compiler™ ("MATLAB Builder NE," 2014).

7.4.6.1. Experimental setting

The architecture of our experimental setup is depicted in Figure 7.24. The client side is JMeter, which generates workload based on our predefined patterns. In our case, the server side is the System Under Test (SUT), which is the ElasticQueue connector, controlled by RobusT2Scale. Here we defined test cases in which the number of users and their usage vary according to time-dependent patterns. A workload generated in this manner hits the SUT and triggers its controller. The controller ensures that the connector remains elastic. Here we followed the guidelines of cloud testing, e.g. (Gambi, Hummer, Truong, & Dustdar, 2013).

Typically, the variances in the generated workload should be large enough to warrant a scaling action. In this work, we injected different patterns of workloads, most of which are drawn from real-world workloads (e.g. ("Anonymized access logs," 2001), similar patterns are also used in (Anshul Gandhi, Harchol-Balter, Raghunathan, & Kozuch, 2012)), to explore the platform's elasticity behavior for a range of demand patterns. In our measurements, we use a set of six different workloads – see Figure 7.28. Across time, some workloads show recurring cycles of growth and decrease, such as an hourly news cycle. Others have a single burst, such as during a special event. Further, we scale the duration of the traces to 1 hour. We evaluate RobusT2Scale against the full set of workloads (see Table 7.9). We ran the SUT on Microsoft Azure VMs. VMs were located in the same availability zone in Ireland; see the deployment details in Table 7.5.

Table 7.5. Deployment details of our experimental setting.

Experimental Deployment Units	**Clients**	**Elastic Application**			**Elasticity Controller**
	JMeter	**UI**	**BL (Scalable)**	**DS**	**RobusT2Scale**
Specification	Desktop, Intel Core i7 CPU, 2.8GHz, 12 GB	**1** Small (A1) Azure VM	**2-6** Small (A1) Azure VMs	**1** Small (A1) Azure VM	**1** Small (A1) Azure VM

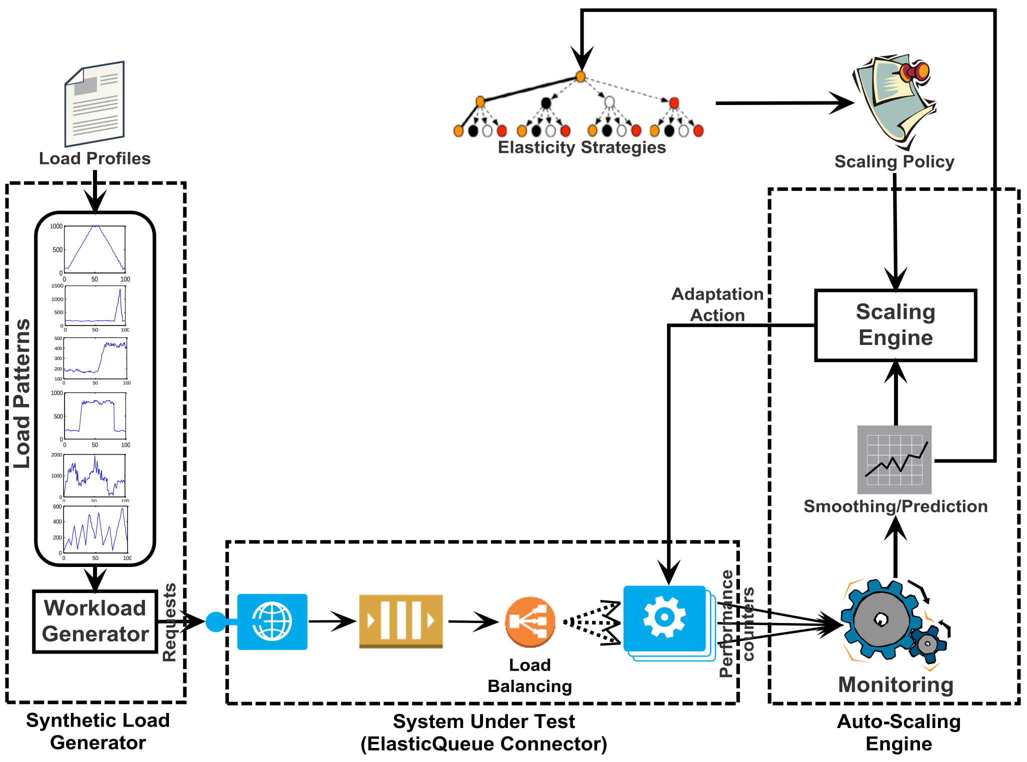


Figure 7.24. Experimental setting for ElasticQueue.

7.4.6.2. Results

In this section, we present a number of experimental studies on RobusT2Scale to answer the following research questions:

- **Q1 (Scalability)**. *What is the runtime overhead of our approach?*
- **Q2 (Accuracy)**. *What is the accuracy of the employed estimation techniques and does the error of estimation vary across different workloads*?
- **Q3 (Effectiveness)**. *Is it effective for guaranteeing SLAs and minimizing cost*?
- **Q4 (Robustness)**. *Is it robust against uncertainties and noises*?
- **Q5 (Stability)**. *Does the controller guarantee stability property?*

Runtime Performance (Q1)

A lengthy adaptation reasoning process hinders usefulness and as a result the adoption of self-adaptive software. In order to assess the runtime overhead of the proposed fuzzy adaptation reasoning process, we have conducted a set of experiments using simulation with different settings. In a practical setting, the size of the rule base and the number of antecedents in each rule in any particular target system is expected to be in the range of $[10,100]$ and $[1,3]$ according to (S.-W. Cheng & Garlan, 2012; Fleurey & Solberg, 2009). However, for evaluating the scalability of our approach, we vary the number of *adaptation rules*, number of rule *antecedents* and number of *linguistics* in our experimental evaluations by orders of magnitude in the range $[9,1000]$, $[2,6]$, $[3,10]$ respectively. We performed the experimental evaluations on a machine with Intel Core i7 CPU, 2.80 GHz, 12 GB memory, 64-bit Windows 7 Professional and MATLAB R2013a.

Scalability with respect to linguistics-rules

To examine the scalability of our approach with respect to the number of linguistics (i.e., MFs) and rules, we provided the following experimental setup. Each input domain consisted of M MFs, where $M = \{3,4,\dots,10\}$. We kept the left-most and right-most MFs in Figure 5.24 (and Figure 5.25) and add another MF identical with the middle MF in Figure 5.24 (and Figure 5.25) and place the center of it in a random number uniformly generated in $[0,5]$. The interval consequent of each rule was generated as two uniformly distributed random numbers in $[0,5]$. Each input was discretized into 100 points and therefore computing a control surface needs $100 \times 100 = 10{,}000$ iterations of the reasoning process. For example, in scenario 3, the MFs ($M = 5$) are shown in Figure 5.24 (and Figure 5.25) and the corresponding control surface is shown in Figure 7.23. To compare the performance of the IT2 controller with the T1 counterpart, we also recorded the computation time for baseline T1 FLSs, whose MFs were centrally embedded in the corresponding IT2 FSs, as an example, see the dashed line in Figure 5.24 (and Figure 5.25).

To make the results statistically meaningful, we performed 10,000 trial runs in nine different scenarios classified by pair of linguistics-rules and measured the duration of time, beginning once inputs are ready to feed the system and ending when the execution receives the best architecture mode to enact. The results are shown in Figure 7.25 separated based on the scenarios, while Table 7.6 summarizes the means of the data in each scenario. The data in Table 7.6 confirms “10× increase in the number of adaptation rules approximately results in 10× increase in runtime” in both experiments (i.e., with and without MATLAB optimization). The data indicates that the reasoner performs well even in large rule bases with significantly higher rules than nominal usage (i.e., 1000 rules). It took approximately 1-190ms to decide for a suitable architectural change, which is acceptable (S.-W. Cheng & Garlan, 2012).

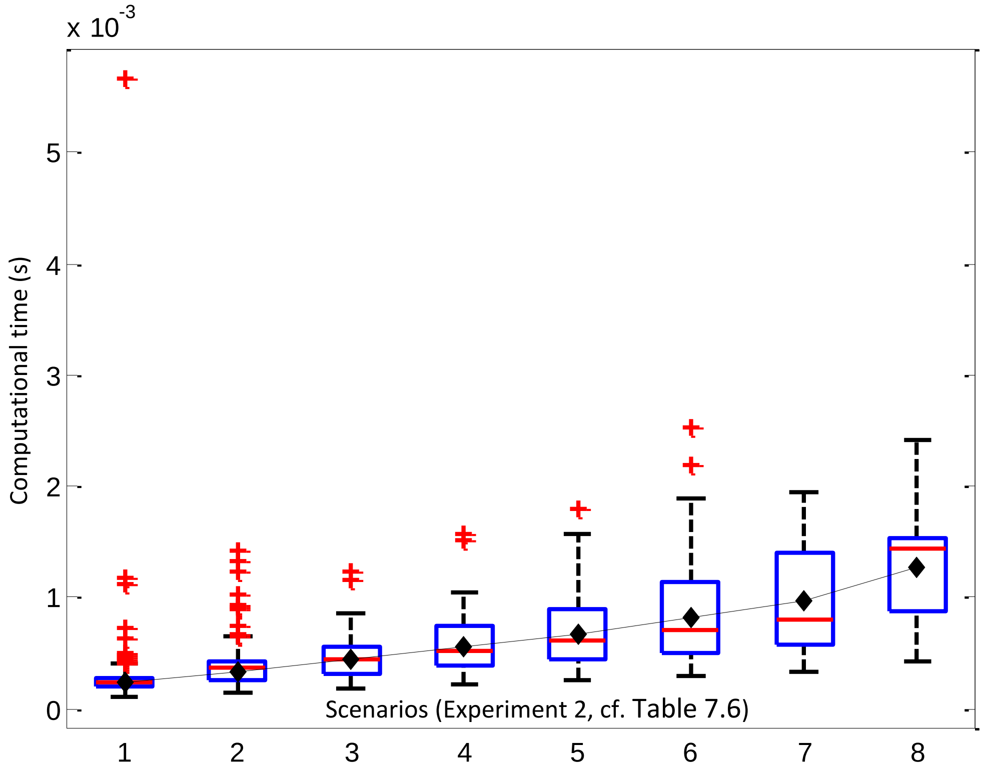


Figure 7.25. Runtime performance w.r.t. # of linguistics/rules.

Table 7.6. Summary of runtime performance evaluations.

Scenario	Setting (# linguistics, # rules)	Performance (s)		
		IT2	Optimized IT2	T1
1	3, 9	0.0011	0.0002	0.0010
2	4, 16	0.0019	0.0003	0.0011
3	5, 25	0.0030	0.0004	0.0012
4	6, 36	0.0044	0.0006	0.0013
5	7, 49	0.0061	0.0007	0.0014
6	8, 64	0.0080	0.0008	0.0016
7	9, 81	0.0101	0.0010	0.0017
8	10, 100	0.0126	0.0013	0.0019
9	10, 1000	0.1901	0.0100	0.0111

Scalability with respect to the number of antecedents/rules

For evaluating the scalability with respect to the number of antecedents/rules, we fixed the number of MFs for each input domain but increased the number of input domains $A = \{2,3,..,6\}$. We also considered the highest amount of rules for each generated FLSs by generating possible combinations of MFs of each input. Similarly, we performed 10,000 trial runs in each of the five different scenarios. The results are shown in Figure 7.26. The means of the runtime performance in each scenario are summarized in Table 5.13. The data in Table 7.7 confirms "3× increase in the number of rules approximately brings about 4× increase in runtime". This demonstrates that the reasoner performs well in large rule bases that have twice the number of antecedents per rule than nominal usage (i.e., 6 antecedents). The memory footprint for the largest rule base is about $1/3\,MB$, which is suitable for resource constrained systems such as embedded systems (see Table 5.13).

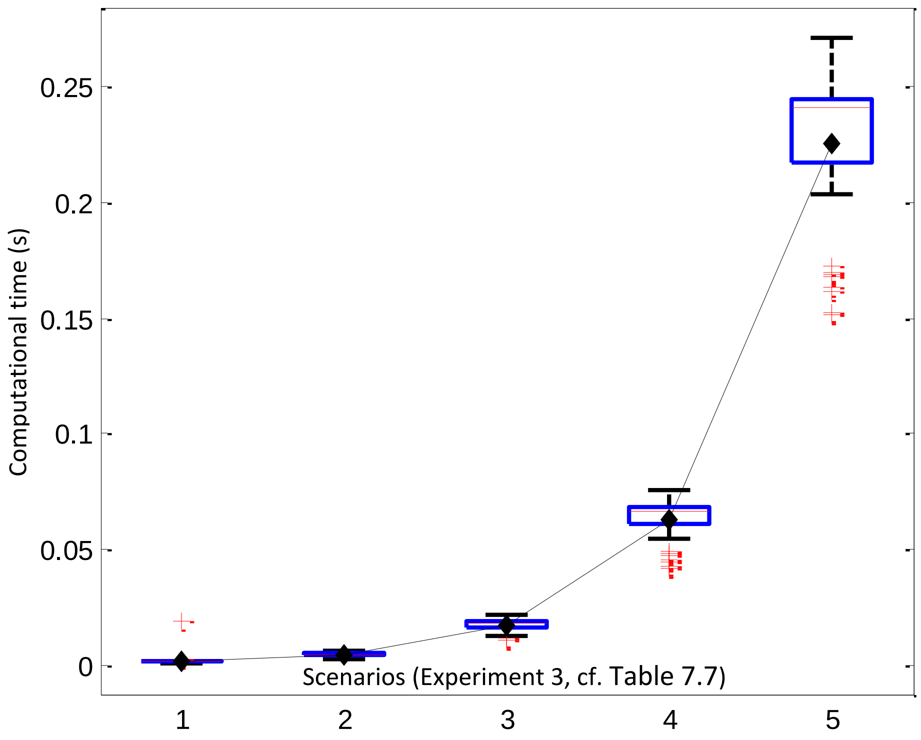


Figure 7.26. Runtime performance w.r.t. # of antecedents/rules.

Table 7.7. Summary of runtime performance evaluations.

Scenario	Setting (# antecedents, # rules)	Performance (s) without Matlab optimization	Memory footprint (Bytes)
1	2, 9	0.0011	1440
2	3, 27	0.0044	6480
3	4, 81	0.0169	25920
4	5, 243	0.0631	97200
5	6, 729	0.2264	349920

Computational Complexity (Q1)

In Section 7.4.5, we designed a fuzzy-based adaptation controller using knowledge elicitation through a survey. In the reasoning process, two steps (i.e., computing firing degrees and type-reduction) are computationally expensive. The computational cost of the firing degrees depends on the size of the rule base and the worst case is of the order of magnitude $O(\#rules)$. The computational cost of the type-reduction is proportional to the centroid and, therefore, depends on the number of discretization of the input variables. Table 7.8 presents the results of the centroid of the "medium" IT2 MF in Figure 7.8 calculated using the KM algorithm with different values of discretization, N. The number of iterations to find the value of the centroid with respect to a naïve calculation before the invention of the KM algorithm and its enhanced version. As it is evident in this table, the computational cost of the type-reduction step (i.e., EKM) is of the order of magnitude $O(\sim Log(N))$.

Table 7.8. Computational complexity of centroid calculation.

N	KM iterations	Enhanced KM (EKM) iterations	2^Niterations
4	4	1	16
16	6	1	65536
100	6	2	$1.2677e+30$
256	7	2	$1.1579e+77$
1024	8	3	$> 8.9885e+307$

Since the fuzzy reasoning is used for software adaptation in a closed loop, an efficient way of reasoning in frequently changing environments was desirable. In Chapter 5, we described a solution that performs a (design-time) computationally expensive derivation of rule-consequent centroids that can be used at runtime, when rule-firing intervals become known. The approach fits the situation in which time consumption during controller design is not critical, but runtime reasoning is subject to tight time constraints. Since in our approach, at design-time, calculate the centroid of each rule consequents, we conclude that the computational costs of the activities at design-time is $O(\sim(K_{dt} + \#rules \times Log(N)))$, where K_{dt} is constant. However, the cost of calculating the centroid at runtime has to be paid only once. This makes the overall computational complexity of the fuzzy reasoning process at runtime $O(\sim(K_{rt} + \#rules + Log(N)))$, where K_{rt} is constant (note the use of plus instead of multiply here).

Workload Estimation Accuracy (Q2)

In order to evaluate the accuracy of the adopted estimation technique, we simulated different workloads and measured the error of estimation by root relative squared error (RRSE). Figure 7.27 shows sample data and different estimations from changing the parameters of the model. It is evident that the estimations with different parameters result in different levels of prediction accuracy. For this sample, the estimation with $\beta = 0.27, \gamma = 0.94$ is more accurate than the other two estimations.

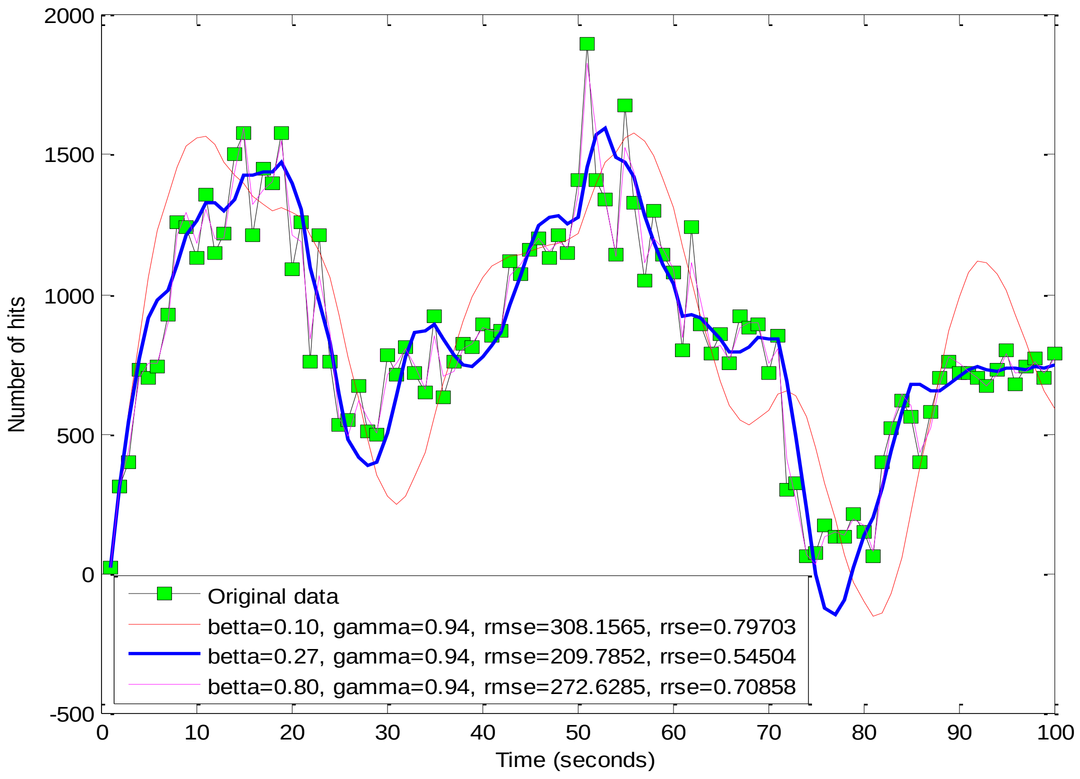


Figure 7.27. Predicted vs. actual workload.

We also evaluated the accuracy of the prediction techniques for different workload patterns. As it is depicted in Figure 7.28, for different patterns (i.e., big spike, etc.), the estimator shows different estimation errors. For three patterns, i.e., ‘slowly varying’, ‘dual phase’, ‘steep tri phase’, the relative error and variations are quite low. The ‘large variation’ shows the large mean of error and ‘big spike’ and ‘quickly varying’ demonstrate the largest variations.

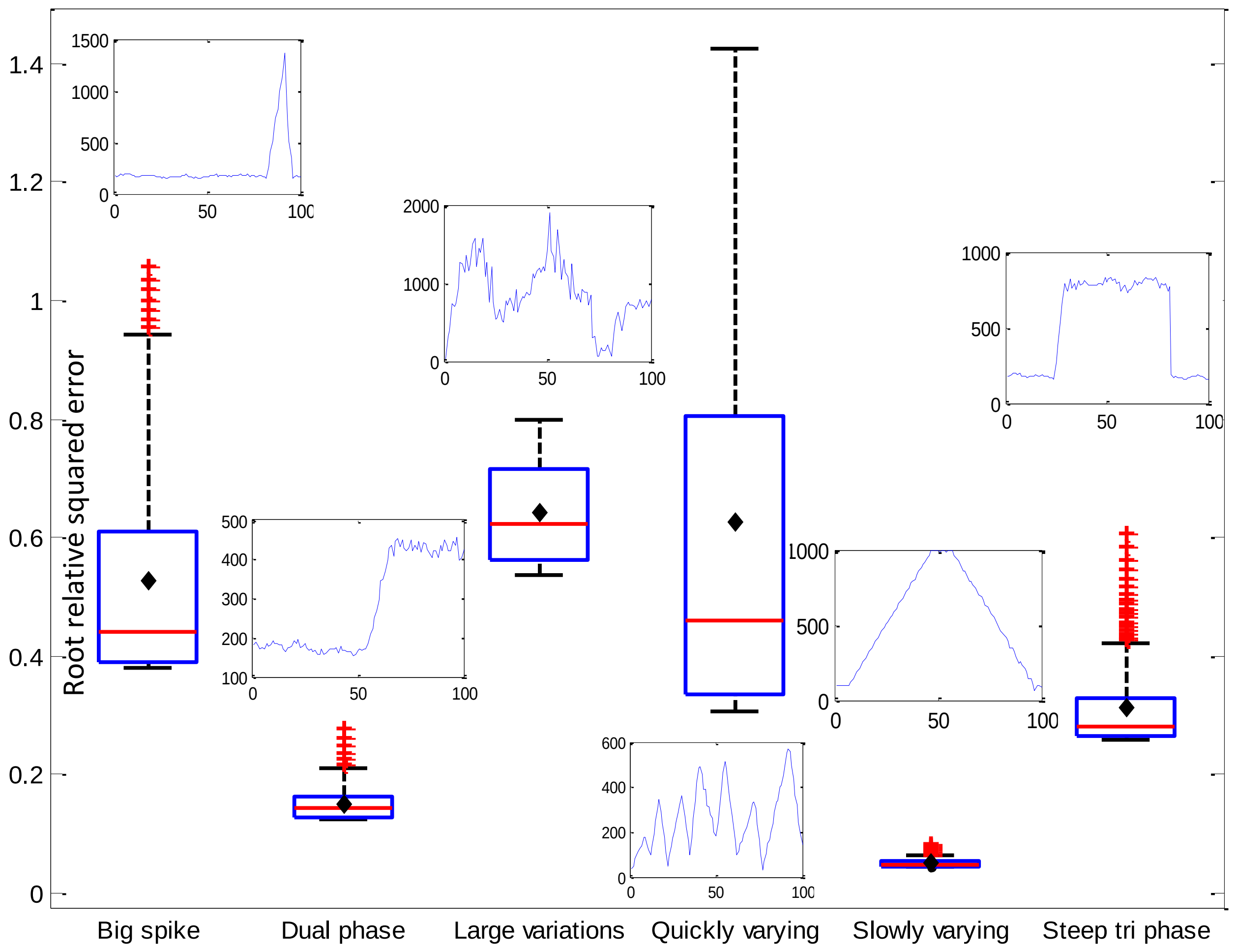


Figure 7.28. Estimation errors w.r.t. workload patterns.

Effectiveness of RobusT2Scale (Q3)

With the rise of multi-tenant software such as cloud-native applications, it is more common to measure higher percentiles of response time rather than just the average response time. The motivation behind looking at higher percentiles is to confirm that most of the system users can access the functionalities of that system with low response times and only a small fraction, if any, of them face slow access. As a result, we decided to use the 95th percentile of response times, $\boldsymbol{rt_{95}}$ (**Definition 65**) to evaluate the effectiveness of RobusT2Scale. This choice is motivated by recent studies (Computing, Gandhi, Dube, & Karve, 2014; A Gandhi, 2013) which point out that 95th percentile response times is an appropriate metric for measuring effectiveness.

Definition 65. For a given workload, we define $\boldsymbol{rt_{95}}$ as the $95th$ percentile of response times (in milliseconds) for requests that complete during the course of the workload.

As a benchmark for measuring the effectiveness of RobusT2Scale, we consider (1) the 95th percentile of response time (rt_{95}), which represent our SLA and (2) the weighted average number of node instances acquired over time ($\overline{vm}$), which determines the cost of ownership. These criteria cover the three main aspects of elasticity comprising scalability, cost and time efficiency. The goal is to meet the response time

SLA, here we assume $rt_{95} = 600ms$, while keeping $\overline{vm}$ as low as possible. The drop in $\overline{vm}$ represents the potential capacity to be released back to the cloud to save on costs.
To evaluate the effectiveness of RobusT2Scale, we compared our approach with two provisioning policies: *over-provisioning* (here we correspond this with a connector in Maximum Effort mode throughout its lifetime, see Table 7.1) and *under-provisioning* (here we correspond this with Normal mode throughout its lifetime, see Table 7.1). A summary of the results is shown in Table 7.9. In comparison with over-provisioning policy, RobusT2Scale has acquired less nodes, saving as much as a factor of two in cost. In comparison with under-provisioning policy, RobusT2Scale is significantly better in terms of rt_{95}, giving a cloud-based application a better chance to guarantee the SLAs.

Table 7.9. Comparison of the effectiveness of RobusT2Scale.

SUT	Criteria	Big spike	Dual phase	Large variations	Quickly varying	Slowly varying	Steep tri phase
ElasticQueue with RobusT2Scale	$rt_{95\%}$	973ms	537ms	509ms	451ms	423ms	498ms
	$\overline{vm}$	3.2	3.8	5.1	5.3	3.7	3.9
Over-provisioning	$rt_{95\%}$	354ms	411ms	395ms	446ms	371ms	491ms
	$\overline{vm}$	6	6	6	6	6	6
Under-provisioning	$rt_{95\%}$	1465ms	1832ms	1789ms	1594ms	1898ms	2194ms
	$\overline{vm}$	2	2	2	2	2	2

As seen in Table 7.9, the SUT with RobusT2Scale has not violated the response time SLA in any patterns of workloads except for the "big spike". The SUT with overprovisioning has satisfied the SLA for all the patterns, however, by imposing a cost of up to a double amount (for 'big spike', but for the other patterns the difference is less) of what has been imposed by RobusT2Scale. The SLA is never met for the SUT with the under-provisioning.

Robustness of RobusT2Scale (Q4)

In Chapter 5, we showed that the utilized estimation approach, i.e. double exponential smoothing, contains unavoidable errors. In this thesis, we have claimed that RobusT2Scale is resilient against input noises, one of which is the estimation error. In this section, we provide some experimental evidence to support this claim.

Earlier we observed that the worst estimation error happens for 'large variation' and 'quickly varying' patterns and is less than 10% of the actual workload. As a result, we injected a white noise to the input measurement data (i.e., x_1) with an amplitude of 10%. We ran RMSE measurements for each levels of blurring, and for each measurement, we used 10,000 data items as input. Figure 7.29 shows RMSE values for the four different blurring values. We observed two interesting points. First, the error of control output produced by the elasticity controller is less than 0.1 for the blurring levels. Second, the error of control output is decreased when we designed the controller with a higher blurring. A higher blurring leads to a bigger FOU, which is a representative for the supporting levels of uncertainty. Therefore, designers should make a choice in terms of the level of uncertainty that the controller can support. Note in some circumstances an overly wide FOU results in performance degradations (JM Mendel, 2001). These observations provide enough evidence that RobusT2Scale is robust against input noise. This achievement is one of the important benefits of using IT2 FLS rather than T1 FLS for elasticity reasoning in cloud-based software, where uncertainty in terms of noise and events are prevalent (Gambi et al., 2013).

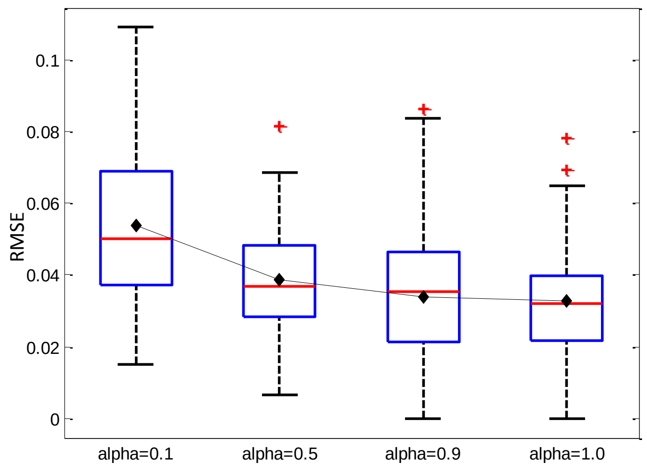


Figure 7.29. RMSEs of the controller with different blurrings.

Stability Analysis of the Controller (Q5)

While FLSs are typically used in open loop decision making, in our approach, we exploit them here in a closed loop adaptation process. There are some properties of closed loop control systems that should be evaluated when comparing controllers for computing systems (Hellerstein et al., 2004). We analyzed the robustness of the controller in previous section, while in this section we examine its *stability*. When a controlled system becomes unstable, the output of the system will not converge, which often either incurs a higher cost or results in a bad user experience. More concretely, by stability, here we mean that for any bounded input over any amount of time, the output will also be bounded. This is called bounded-input-bounded-output (BIBO) (Hellerstein et al., 2004). A system is defined to be BIBO stable if there exists a constant k such that for all bounded input conditions, the output absolute value never exceeds k. In other words, as long as a stable signal is input, a stable output is guaranteed.

In control theory, it is common to use a theoretical analysis to prove that the controlled system is stable. However, such analysis is beyond the scope of this work and we only use informal analysis based on the control surfaces, as shown in Figure 7.23. The inputs, i.e., workload and system performance, are bounded in $(x_1, x_2) \subseteq [0,100]$, and as shown in Figure 7.23, the controller output is also bounded $Y^{IT2}(x_1, x_2) \subseteq [-2, +2]$. Since for any bounded input over any amount of time, the output is also bounded, the designed controller satisfies the BIBO stability property. Moreover, because of the smooth control surface, small changes in the inputs correspond to small output changes.

Effects of Conflicting Policies (Q4, Q5)

The main objective of this work is to find an approach to capture the uncertainty on expressing conflicting adaptation policies. In this section, we discuss the effects of contrasting advices by domain experts on the adaptation decisions and control surface to clarify such claim.

In the data collection for constructing the rule base, see Table 7.3, different domain experts had different advices for each combination of input parameters. In Chapter 5, we provided a methodology to combine these contrasting advices into a coherent adaptation rule. Essentially this formula provides a compromise between the contrasting opinions and provides a mechanism to find an appropriate tradeoff by weighting the centroids of advices based on the number of domain experts that voted for that specific advice. For instance, for the first and last rule, we can see a relatively high dispersion in the opinions, but for the third rule, a relatively high agreement between experts can be observed.

Some other parts of the controller also affects the control output. For instance, the level of the uncertainty that is embedded in the MFs through blurring parameter α will be directly translated to the output interval of the controller, i.e., $[y_l(x_1, x_2), y_r(x_1, x_2)]$. Figure 7.30 shows the difference between the upper layer and lower layer of the control surface for all possible inputs, i.e. $y_r - y_l$. As it is evident, for some combinations of the inputs the boundary is larger than the other points. This is due to the uncertainty that is embedded in the MFs. A larger J_x, results in a bigger output interval for that point.

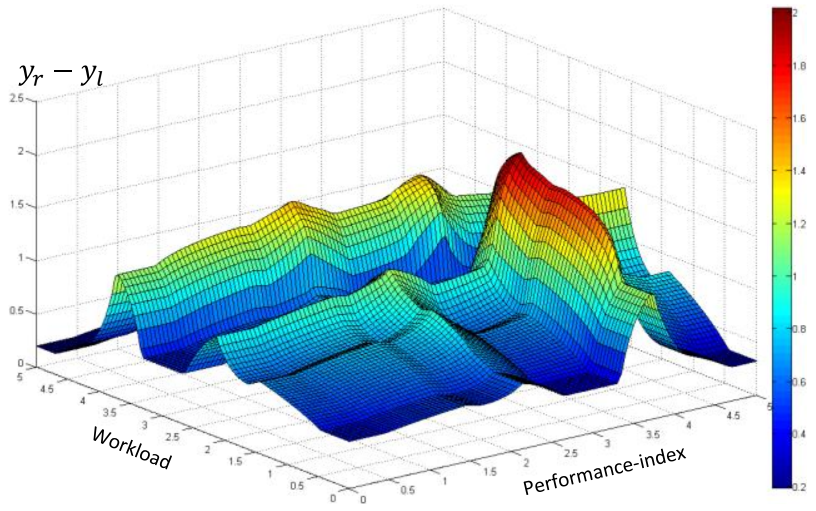


Figure 7.30. Distance between decision boundaries (IT2FLS).

7.5. Limitations and Threats to Validity

In this chapter, the RobusT2 framework was evaluated though a cloud-based case study. This evaluation is based on a typical real-world cloud-based connector without any previous knowledge of workload requests (only real-time information of the loads is used). The principal contribution is the use of interval type-2 fuzzy logic controllers and human expertise in operating mode of the connector:

- As far as we know, it is the first application of type-2 fuzzy logic in cloud computing and the first experimental validation of an adaptation management system on a configuration adaptation of a cloud-based application using 1) type-2 fuzzy logic and 2) using experience of multiple experts.
- It is an application for a cloud application, but it could be used in any elastic or adaptive application (the parameters of the adaptation reasoning depend only on the characteristics of the environment within which the connector is operating and the quality factors of the connector itself)
- In this self-adaptive connector, the priority source is the response time because it is the most performing source in SLAs.

In the remainder of this section, we discuss limitations and threats to validity of this work.

7.5.1. Limitations

Runtime performance. A performance evaluation shows low runtime and memory overhead of the decision-making part of the self-adaptive loop. In order to compare our results with some benchmark, we extracted a number of adaptation reasoning performances reported in the literature in Table 7.10. By comparing it with the results in Section 7.4.6.2, our approach performs better by roughly an order of magnitude. However, we cannot claim that our approach outperforms these approaches in all circumstances because of the differences between the experimental settings. In terms of scalability, according to Table 7.10, Rainbow (S.-W. Cheng & Garlan, 2012) also confirms "10× increase in the number of adaptation rules approximately yields 10× increase in runtime".

Table 7.10. A benchmark on performance for reasoning.

Rules	Adaptation Reasoning Performance (s)		
	Rainbow/Stitch (S.-W. Cheng & Garlan, 2012)	StarMX (Asadollahi, Salehie, & Tahvildari, 2009)	MADAM (Geihs et al., 2009)
10	0.017	-	-
100	0.167	-	0.087
1000	1.454	2.8183	-
10000	13.730	-	-

Control of noise. We evaluated the robustness of adaptation reasoning against sensory noises. We showed that the robustness of control could improve or deteriorate depending on the blurring values as the only design parameter for transitioning from T1 to IT2 FLS. Therefore, we can find the optimal blurring value that makes the IT2 FLS design more resilient against dynamic noises. We also observed a better noise control by the IT2 FLS compared with its T1 counterpart. The study of robustness against colored noises, e.g. pink noise, can be an interesting future work.

Limited stability analysis. For analyzing the stability of the controlled system, we only provided informal comments based on the observation of the control surfaces. A theoretical analysis to establish the BIBO stability of the controller is left as a future work.

Limitations of design-time mode discovery. RobusT2 assumes that prior to deployment, system architects are able to identify appropriate architectural modes that could resolve the issues that may arise at runtime. This means that RobusT2 scope of adaptation is limited to situations that can be addressed with a set of preconceived system modes. RobusT2 may not be able to resolve an issue that could be resolved through runtime adaptation, simply because the domain experts have not included the appropriate modes.

7.5.2. Threats to validity

Threat to internal validity. Regarding the internal threats, there is only one issue. RobusT2 is dependent on the rules that are specified by a number of domain experts through an elicitation methodology. This means that RobusT2 decision making is implicitly dependent on the selection of these domain experts and their advices on adaptation policies. RobusT2 may not be able to adapt the system appropriately if the expert advices are overall incorrect and sub-optimal.

Threat to external validity. An external threat is related to the methodology for data collection and fuzzy rule elicitation, as the accuracy of adaptation decisions is heavily depends on the ability to specifying fuzzy membership functions and fuzzy rules. In Chapter 5, we presented a methodology for data collection and concrete examples to show feasibility of adaptation policy elicitation through survey. In fact, in Section 7.4.5, we showed how the data collected via a group of users impacts the design of fuzzy controller and how it reasons about adaptation. This is mitigated as far as possible by designing a precise data collection protocol and well-defined template for data collection, see Appendix A.

Another external threat is the choice of analytical models (i.e., DTMC and CTMC) used for requirement verification at runtime. As we described in Chapter 4, we focused on quantifiable non-functional requirements, which are specified through constraints (via temporal logic PCTL and CSL, see Chapter 2) on the analytical models. As we briefly discussed in Section 7.2, the verification of such requirements triggers adaptations. The temporal logic that we employed (recall Chapter 2) is already able to formally specify a fair number of non-functional requirements. However, the analytical models used for verifying requirements may not be applicable to all quality properties of interests for component connectors.

7.6. Conclusions

In this chapter, we presented the principal findings and the results of the research evaluation. In summary, we evaluate the RobusT2 framework's support for adaptation management of component connectors. We utilized the ElasticQueue as a concrete and real-world case of component connectors that require self-adaptation in the cloud. We evaluated the efficiency, scalability, robustness and applicability of the proposed solution. More specifically, in the RobusT2 framework we evaluated the self-adaptation process to address the challenges in RQ1, RQ2 and RQ3 with the specific claims we made in Chapter 1:

- **Research claim 1** (runtime efficiency). The activities that need to be integrated in the self-adaptation loop are required to be time efficient. Therefore, as a part of ensuring the practicality of the approach, we provided evidence of runtime efficiency of the adaptation process.
- **Research claim 2** (scalability). It is not sufficient for the approach to be time efficient with small models, it also needs to impose an acceptable overhead on large-scale systems, which correspond to complex models. We ensured the scalability of the approach by investigating the computational complexity of the approach.
- **Research claim 3** (robust against dynamic uncertainty). It is desirable that the approach is resilient against different amplitudes of noises, which resemble the reality of uncertain environments that component connectors are operating in. We injected different levels of noise to the input parameters of the approach and evaluated the robustness of the approach under dynamic uncertainty.
- **Research claim 4** (applicability). The approach presented in this thesis developed a set of techniques and methods to control the uncertainties in the self-adaptation loop of component connectors. We applied the solution proposed in this thesis to a real-world case study and evaluated different aspects of the solution in real-world experimental settings.

Some additional ideas to improve the adaptation management of the component connectors have been identified but have not been evaluated or presented in this thesis:

- To apply the adaptation management framework developed in this thesis on multi-cloud scenarios.
- The ElasticQueue test bench operation is very constrained in terms of mode variations. We have not considered on the fly adaptation of the modes themselves in this work. More specifically, the adaptation strategies can be changed throughout time.

When the adaptation management survey for the ElasticQueue was presented to the experts, not all the information about the ElasticQueue nor the adaptation management platform was presented and not all the implementation constraints were clearly identified (before experimental validation). It could be interesting to conduct a new survey presenting more information about the implementation of the RobusT2 framework and the results.

Chapter 8

8. Conclusions

"It is more fun to arrive at a conclusion than to justify it." Malcolm Forbes (1919-1990)

Contents

8.1. Chapter Overview

In this chapter, we present the main conclusions of our research by highlighting the significance of this thesis, its limitations and we discuss short-term and long-term future research directions. This chapter is divided into four parts. First, in Section 8.2, we provide a summary of the research through revisiting the research questions and the hypothesis of this thesis. In Section 8.3, we discuss the core contributions of this research. In Section 8.4, we point out limitations and short-term directions for future research. Finally, we suggest the long-term future work opened up by the research, drawing the thesis to a close.

8.2. Research Summary: A Reminiscence

This section starts by returning to the research questions defined at the beginning of the thesis in Chapter 1. The answers to the questions, which have emerged throughout the research and presented in the core contribution chapters are presented and discussed. The section then revisits the research hypothesis by examining the results of the case study and experimental evaluations presented in Chapter 7 to see whether they support the original hypothesis.

8.2.1. Research Questions Revisited

In this thesis, we enable the reliable and dependable self-adaptation of component connectors in unreliable environments with imperfect monitoring facilities by providing: (a) techniques for robust model calibration, (b) a mechanism for robust adaptation reasoning, and (c) tool support that allows an end-to-end application of the developed techniques.

In this section, we revisit the research questions presented in Chapter 1 and we discuss the answers to each that have appeared throughout the thesis.

The first research question is:

***Research Question 1 (RQ1)*. How to estimate the parameters (i.e., calibrate) of the analytical models at runtime that we employ for non-functional requirement verification of component connectors in the presence of noisy monitoring data?**

Chapter 4 proposes mechanisms for model calibration in the presence of uncertainty. In Chapter 4, we presented the analytical models through which we model the component connector behavior. We also proposed mechanisms to calibrate the unknown parameters of the models at runtime. The key contribution here is that the mechanisms are capable of carefully determining the parameters even in the presence of uncertainty. The proposed method is comprehensively evaluated with a thorough discussion of the results.

Therefore, the ability of the model calibration mechanisms to *handle* and *robustly control* the *uncertainties* in the monitoring data provides an explicit answer for this research question, i.e., **RQ1**. Since the model calibration can estimate unknown parameters of the analytical models at runtime, we can ensure that the non-functional requirement verification that ultimately triggers the adaptation actions provides a reliable mechanism for enabling self-adaptation of connectors.

The second research question is:

***Research Question 2 (RQ2)*. How to reason about adaptation and derive appropriate configurations for component connectors at runtime in the presence of noisy measurements and imprecise objectives?**

Chapter 5 describes in detail the design, implementation and experimental validation of the adaptation reasoning that we have devised for component connectors. In this chapter, we proposed the RobusT2 framework to realize the adaptation reasoning using a type-2 fuzzy logic system. This chapter presented the application of type-2 fuzzy logic control developed in this research for adaptation reasoning. The developed RobusT2 framework has the following features: 1) It combines the input end-to-end response performance and number of requests to the connector that it controls in the decision of operating mode changes of the connector; 2) it combines the opinions from different experts, so that an acceptable decision boundary can be obtained; 3) it provides an interval decision, so that a flexible decision can be made based on a design tradeoff between the internal situation of the connectors and their environmental conditions. This chapter also presented experimental evaluations of the framework.

Once the decision for mode change has been made, a change needs to be enacted to the running connector. **Chapter 6** presents a mechanism to enact the transitions from the current connector configuration to the target configuration. Considering the high heterogeneity of models involved in connectors, this chapter introduced an approach to derive a reconfiguration plan using reasoning based on graph theory and feature models. We described a mechanism for transforming these feature models corresponding to the connector modes to an executable reconfiguration plan. Note that this chapter is not a core contribution chapter, but rather acts as an operationalization part of the framework.

Therefore, the proposed methodology for designing fuzzy logic controllers and the integration of the designed type-2 fuzzy logic controllers in the feedback control loop to decide about mode change at runtime provides an explicit answer to this research question, i.e., **RQ2**. Since the type-2 fuzzy logic controllers are capable of combining different opinions of multiple users to produce reliable output in the presence of noisy inputs, we can ensure that the integration of the controllers as the decision makers for the self-adaptive connectors enable reliable adaptation.

The third and final research question is:

***Research Question 3 (RQ3)*. How well can our approach for model calibration and adaptation reasoning in the feedback control loop ensure the reliability of the self-adaptation of component connectors in a real-world unreliable environment?**

Chapter 7 reports an end-to-end evaluation of individual research components and provides an overall validation of the proposed framework. In this chapter, we showed how the three key parts of the RCU framework are integrated to enable self-adaptation of component connectors through a real-world case study. To conduct this research, we followed the guidelines of the action research methodology (Chapter 1) that provides a rigorous set of steps focused on planning (Chapter 2, Chapter 3) and conducting the research (Chapter 4, Chapter 5, Chapter 6) along with the evaluation of the research results (Chapter 7). Therefore, in this chapter, we focused on an experimental evaluation of the adaptation management of component connectors in the RCU framework. In general, we demonstrated the validity of the research claims (i.e., runtime efficiency, scalability, robustness and applicability) of this thesis through experimental evaluations. The experimental results in this chapter provided a positive answer to **RQ3**.

8.2.2. Research Hypothesis Revisited

In this section, we revisit the research hypothesis presented in Chapter 1, which is repeated here:

The application of parameter estimation for calibrating models for non-functional requirement verification, in the presence of imprecise monitoring data and fuzzy logic in adaptation reasoning, and the integration of the two in self-adaptation process enables component connectors to become robust against uncertainty in the surrounding environment.

The *first part* of the hypothesis concerns reliable model calibration for the sake of non-functional requirement verification considering that the monitoring data are inherently uncertain. This thesis has demonstrated how adopting stochastic techniques (i.e., Bayesian and Markov Chain Monte Carlo) provides a more accurate parameter estimation, with thorough experimental evaluations presented in Chapter 4. As discussed in Chapter 4, some classes of monitoring data contain unstable measurement noise. Therefore, it is of course impossible to claim that all uncertainties in monitoring data can be handled. However, this thesis demonstrates that if we assume that the monitoring data contains missing data and stable measurement noise, the technique that we proposed in Chapter 4 can provide a reliable estimation of unknown parameters of the analytical models.

The *second part* of the hypothesis concerns the use of fuzzy logic to reason about mode changes of component connectors considering that different users specify the adaptation policies and they may not have a unified opinion about the policies. Chapter 5 of this thesis demonstrates a methodology to elicit user opinions at design-time and transforms them into type-2 fuzzy membership functions. A fuzzy logic controller is designed to enable decision-making about such adaptations at runtime. Thus, this clause of the hypothesis is supported by the solution framework presented in this thesis.

The *final part* of the hypothesis concerns the integration of the two mechanisms and the application of the developed solution to real-world connectors. Chapter 7 of this thesis demonstrates the applicability of the developed solution framework for enabling self-adaptation of real-world software connectors in an inherently unreliable environment with many different sources of uncertainties. The experimental evaluations that we have reported in Chapter 7 reveal that the developed solution enables a dependable and robust adaptation of the connectors in unreliable environments. Thus, this clause of the hypothesis is also supported by the research presented in this thesis.

Chapter 1 describes the research methodology that we have followed to conduct the research presented in this thesis. The heterogeneous nature of the software engineering discipline impedes the widespread adoption of a single research methodology (K Welsh, 2010). Because of the analytical and synthetic nature of this research, we followed the principles of the design-science paradigm. The evaluation of the artifacts (i.e., solution framework and its comprising analytical techniques and mechanisms) are mostly performed through *controlled experiments*. Controlled experiments provide a better understanding of the problem, and feedback to improve the mechanisms has been obtained so far throughout research. Experiments also explain the contributions of the mechanism when compared to existing practices. In Chapters 4 and 5, we make use of experimental evaluations to more objectively validate the claimed benefits of this thesis within the more detailed case study as presented in Chapter 7. The outcomes of the case study with the experimental results lead to an overall conclusion that the hypothesis is substantially supported.

8.3. Research Contributions

The principal contribution of this thesis is an approach for enabling the self-adaptation of component connectors considering uncertainties including measurement noises and users' conflicting opinions about adaptation policies. This main contribution incorporates several parts:

- A set of stochastic techniques to facilitate model calibration as a part of runtime verification task in the feedback control loop of self-adaptive component connectors. The proposed stochastic approach is able to update the unknown parameters of the models at runtime even in the presence of incomplete and noisy observations.
- A general methodology based on fuzzy logic for deciding the adaptations that adjust the configuration of component connectors to the appropriate operating mode. The methodology enables a systematic development of a fuzzy logic controller that can determine the right operating mode for connectors. This methodology provides a means of defining adaptation policies in a way such that different opinions of the users about the policies can be incorporated. The derived fuzzy controllers can decide about the operating mode based on a tradeoff of the user-specified policies.
- The evaluation of the adaptation reasoning by applying the solution to enable self-adaptation of some real-world connectors.

As we have reviewed in Chapter 3, some research has recently started to address the challenges posed by uncertainty in self-adaptive software. In the state-of-the-art chapter in this thesis, we systematically pointed out areas that have been covered by existing work and areas that are left open. The approach presented in this thesis is the first systematic method for incorporating *uncertainty* regarding *multiple user opinions* about adaptation policies.

In addition to this primary contribution, this work makes a couple of secondary contributions that are also significant:

- Systematic identification of different sources of uncertainty present in the feedback control loop of elastic systems and characterization of them using a well-known taxonomy. We discussed challenges to manage the impact of uncertainty on elastic software systems. We focused on elastic systems because connectors play a central role in elastic systems and our main concern was to demonstrate that there is a need for controlling uncertainty in a real domain where connectors have been adopted as an essential entity.
- Findings on the experimental evaluations of the approach based on the development of several prototype tools on practical platforms. These prototypes have demonstrated how the approach presented in this thesis can be applied to practical environments that leverage component connectors.
- In addition to these general results on validity, they have allowed us to explore how various aspects of our approach can be automated, such as the architectural mode change in component connectors. At the same time, this implementation has revealed areas where significant research challenges still remain, such as incorporating uncertainty related to the change enactment in connectors. More specifically, the implementation on practical environments revealed that for the same change the time that it takes to enact the change (i.e., change execution latency) on the connectors takes different times depending on some situational parameters affecting the platform such as the usage pattern, network connection, platform availability and many other reasons.

8.4. Limitations

Although some significant results of our approach have been demonstrated, it is important to pinpoint the limitations of this thesis. In this section, we point out a number of notable limitations, which remain for immediate future work. Such short-term future research would enhance the support of the hypothesis of this thesis.

The empirical work justifying the real-world applicability of this approach consists of only 1 case study in elastic cloud-based application and a couple of experimental evaluations.

We have argued that the case study presented in this thesis (see the evaluation chapter) provides strong evidence of the applicability of our approach in a practical domain. We have given particularly careful attention to experimentally evaluate the "scalability", "accuracy", "effectiveness", "robustness" and "stability" of the solution to strengthen this claim. However, experimental evaluations have their limitations, and in attempting to generalize from the results of the experimental evaluations, we can go only up to certain point. Future empirical evaluations of the approach to other types of connectors that we have not considered in this research or other types of architecturally significant software would be of great help in evaluating the scope of our results.

More specifically, regarding the experimental evaluations that we have performed and reported in Chapter 7, we have the following limitations:

1. *Limited workload patterns*. We considered six different workload patterns. However, in production environments, workload may change in many unpredictable ways.
2. *Evaluations with different connector type*. Our experimental evaluation is limited to ElasticQueue. For this type of connector, although popular in the cloud domain, there are many different varieties of connectors in practice in different domains.
3. *Evaluations with different platforms.* Although the solution we introduced in this thesis is independent of a specific platform, we only evaluated the RCU framework on Microsoft Azure.

The approach requires design-time discovery of appropriate operating modes, but it may not be effective in cases where modes that can resolve issues regarding specific situations are not defined.

The solution that we proposed in this thesis assumes that prior to deployment, system architects are able to identify appropriate architectural modes that could resolve the issues that may arise at runtime. This means that the connectors' scope of adaptation is limited to situations that can be addressed with a set of preconceived operating modes. The solution may not be able to resolve an issue that could be resolved through runtime adaptation, simply because the domain experts have not included the appropriate mode(s).

The approach effectiveness is dependent on the users' specification of the adaptation policies.

The solution is also dependent on the rules that are specified by a number of domain experts through an elicitation methodology. This means that the decisions that are made by our solution framework are implicitly dependent on the selection of these domain experts and their advices on adaptation policies. Our solution framework may not be able to adapt the connectors appropriately if the expert advices are incorrect and sub-optimal.

The approach requires tool support in order to be adopted for practical use, but such tool support may not be available on some platforms.

The approach for enabling the self-adaptation of component connectors we have presented in this thesis is inherently dependent on two particular facilities in order to be practical: I. Monitoring, II. Enactment. To evaluate the approach and to show that the approach is actually implementable, we developed prototype tools that support key elements of our solution. While demonstrating the validity of the approach in principle is one goal of this thesis work, actually applying the approach to different types of connectors on different platforms requires monitoring facilities that retrieve runtime data and an enactment module to execute changes. This is not a limitation of our approach in principle, but it is beyond the scope of this thesis. Further tool development would be necessary to make the approach adoptable by practitioners on different platforms.

8.5. Future Work

Throughout the development of the work for this thesis, we have intermittently noted some areas in which further investigation would be necessary for clarifying wider open issues or advancing the state of knowledge regarding certain aspects. In this section, we enumerate some major future areas of work where we believe a number of significant research challenges remain and as a result, several perspectives for long-term work can be anticipated:

Integration with other uncertainty control approaches. There are different sources of uncertainty in the context of self-adaptive software. However, the approach proposed in this thesis can only handle uncertainties regarding measurement noise and conflicting user opinions regarding adaptation policies. We believe that the integration of this approach with the existing approaches for controlling uncertainty regarding other sources has potential to be the basis of further investigations. We believe that an end-to-end solution for controlling the uncertainties makes self-adaptive systems more resilient against noise and makes them more dependable.

Dynamic update of adaptation rules. Runtime knowledge evolution and sharing is a topic that has attracted less attention so far and is considered as an open challenge in self-adaptive software (Abbas et al., 2011). In this thesis, we have not discussed dynamic updates to the adaptation mechanism. The inference engine chooses from a set of rules each time an adaptation cycle is performed. Therefore, it would be feasible to add new rules to the rule base at runtime. By adaptation cycle, we mean the time from receiving input measurements until the calculation of the output and sending it for execution. This allows dynamic incorporation and removal of adaptation rules and indicates another avenue of future work. A promising approach is fuzzy rule learning (L. Wang & Mendel, 1992). Over time, the adaptation outcomes can be captured in a repository. Then, by applying runtime efficient fuzzy rule learning, for example the WM method (L. Wang & Mendel, 1992), new rules can be learned and potentially improve the effectiveness of the adaptation mechanism. For instance, this facility can be used to avoid mode switches that have not historically resulted in better system quality. The rule learning approaches can also be applied at design-time to assist users in rule specifications.

Change of user opinion over time. Uncertainty not only comes from the multiplicity of stakeholders, but also from changes in their preferences over time. Users may change their opinions about adaptation policies over time due to several reasons. For example, they may change their opinions based on the effectiveness of the controller that has been designed based on their initial opinions. This change introduce another type of uncertainty which is related to the user's lack of knowledge. The development of mechanisms to incorporate such uncertainty is considered a future direction of this research.

Dynamic switch between adaptation strategies. In this work, we consider that only one set of adaptation rules will be determined by the users. However, one generalization of the work is to determine different sets of adaptation policies at design-time and switch between them at runtime. As illustrated in the upper right part of Figure 8.1, the strategies derive the adaptation policies that themselves determine the adaptation actions based on the reasoning process. The decision for when to switch between strategies and switch to what strategy needs its own reasoning mechanism.

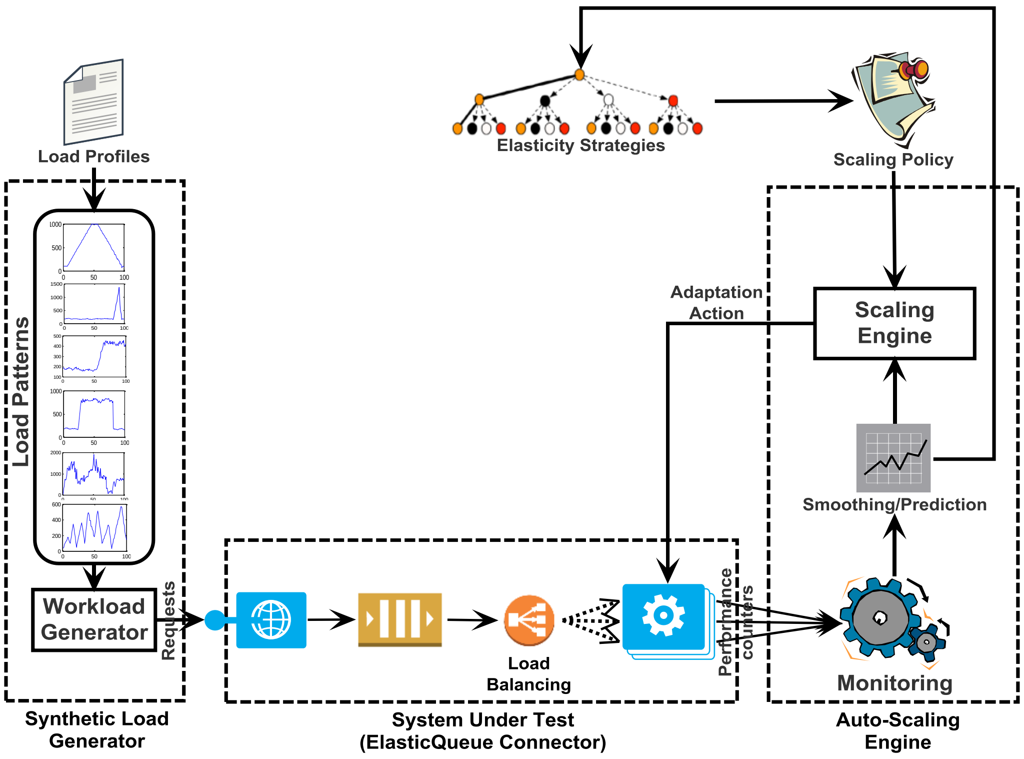


Figure 8.1. Dynamic switch between adaptation strategies.

Extending the requirements engineering framework to explicitly accommodate uncertainty. In this thesis, we have provided a solution for dynamically changing architectural modes of software connectors based on environmental and internal situations of the connectors, considering the traditional interpretation of Zave and Jackson's (Zave & Jackson, 1997) requirements engineering framework to enable dynamic adaptation as described in details in (Calinescu et al., 2012). However, one potential direction of this work can be an extension of this traditional framework to explicitly account for uncertainty. In order to make it clearer, we are going to briefly discuss the principles of an extended architectural requirements engineering framework. Zave and Jackson (Zave & Jackson, 1997) have conceptualized requirements engineering in the sense of this framework as:

$$(\mathbb{S}, \mathbb{D}) \vdash \mathbb{R} \tag{8.1}$$

where $\mathbb{D}$ is a set of domain assumptions, $\mathbb{S}$ is the specification that satisfies the set of requirements $\mathbb{R}$.

A requirement in this framework is a prescriptive statement about the desired phenomena in the environment and it obviously should not refer to phenomena in the specification. In the traditional interpretation of this framework (Chopra, 2012), the satisfaction of the statement (8.1) has binary nature – i.e., it is *satisfied* or *falsified*. We call a requirement *satisfiable,* if at runtime we can verify if the requirement is satisfied for particular instances. On the other hand, we call a requirement *falsifiable*, if at runtime we can verify that the requirement is violated. There are some special requirements that are neither satisfiable nor falsifiable. These requirements are called *vague*. There are also some requirements that are either non-satisfiable or non-falsifiable.

In order to have a legitimate requirement, one should be able to provide evidence that shows that requirements are both satisfiable and falsifiable upon specific environmental states. Chopra (Chopra, 2012) argues that imprecise requirements are vague. However, we now provide evidence that imprecise or adaptive (Luciano Baresi et al., 2010) requirements that are relaxed (Whittle et al., 2009) or become reflective (Nelly Bencomo, Whittle, Sawyer, Finkelstein, & Letier, 2010) are not vague. As illustrated in Figure 8.2, α–cuts (JM Mendel & John, 2002) can be used to precisely determine the intervals that satisfy or violate the requirement, e.g., ${}^{\alpha=0.18}R = \{x|\mu_R(x) \geq 0.18\}$ determines the three zones as indicated in Figure 8.2 – note that $\alpha = 0$ is the default choice. Therefore, Chopra's claim about the vagueness of imprecise requirements is not accurate as, opposed to his argument, the violation zones can be identified precisely without any vagueness associated with them.

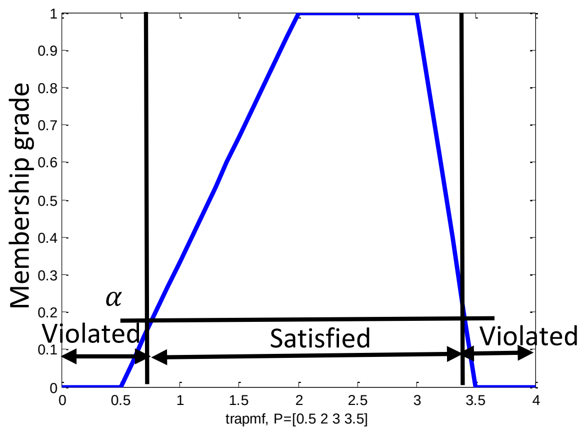


Figure 8.2. Satisfied and violated intervals in an imprecise requirement.

Although T1 FSs provide flexibility in requirements specification, such FSs have limited capabilities for handling uncertainty with this type of fuzzy sets (JM Mendel, 2007). Here, by handling uncertainty, we specifically mean the capability for *specifying* and *minimizing the effects* of such uncertainty. Of course, there are different sources of uncertainty for self-adaptive architectures (Esfahani & Malek, 2013) and they require different approaches to handle them. As indicated in Figure 8.3, IT2 FSs are able to model the uncertainty in the membership function by blurring the fixed membership functions.

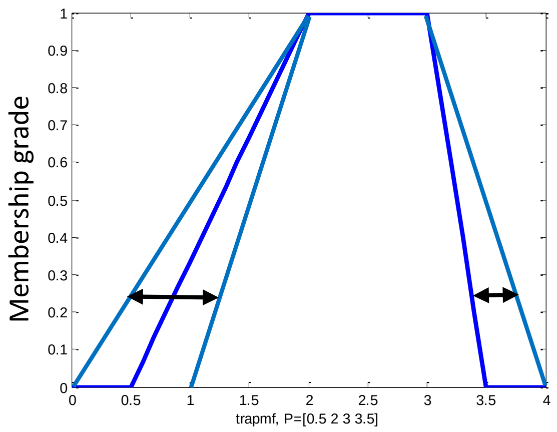


Figure 8.3. Blurring of T1-FS to build an IT2-FS.

We argue that Zave and Jackson's framework must be extended to explicitly handle uncertainty. We believe that uncertainties should be considered as first-class citizens. As indicated in equation (8.2) below, $\mathbb{S}_I$ is the system specification by considering the internal uncertainty associated with it. $\mathbb{D}_E$ shows the domain assumptions with environmental uncertainty and $\widetilde{\mathbb{R}}$ is the set of imprecise requirements with uncertain meanings in its specification. In contrast to traditional requirements in the Zave and Jackson framework, the requirements in this setting are adaptive and flexible by nature. While traditional prescriptive requirements are either satisfied or violated, imprecise requirements would be verified at a satisfaction degree (Luciano Baresi et al., 2010; Whittle et al., 2009) in the presence of uncertainty. The specification and management of these requirements provides a way to trade these requirements off against each other at runtime. The notation $\vDash$ (note the difference with $\vdash$ in Equation (8.1)) indicates that satisfaction verification can be performed at runtime as opposed to the traditional view, which is solely an offline activity. For instance, we use IT2 FS and the reasoning based on this type of theory:

$$(\mathbb{S}_I, \mathbb{D}_E) \vDash \widetilde{\mathbb{R}} \tag{8.2}$$

Such a framework, if developed in the future, can accommodate different sources of uncertainty in self-adaptive software that stem from unreliable entities (i.e., environment, human, mathematical techniques, or separation of concerns) and consequently leads to a more dependable solution that potentially has a better chance for widespread adoption. We consider the development of such an extended requirements engineering framework as a potential and fruitful future direction of this thesis.

Extending to other application domains. The final front for extending this work is application of RCU to other application domains. We have already done this for the problem of dynamically adjusting queues in cloud computing. However, multiple other application domains can be extended to benefit from the contributions of this thesis. One notable example is the problem of network applications in ubiquitous environments (Inverardi et al., 2010). In the domain of networked ubiquitous computing, heterogeneous devices need to detect services discoverable in the ubiquitous networked environments and adapt their own communication protocols to interoperate with them, since networked applications are realized on different middleware (Inverardi, Spalazzese, & Tivoli, 2011). Because the ubiquitous environments contains many sources of uncertainty, the approaches dynamically adjust protocols such as the proposal in (Di Marco, Inverardi, & Spalazzese, 2013) that requires explicit consideration of the impact of uncertainty on the device interoperability decisions. As a result, the RCU framework can naturally be an appropriate fit to this problem.

Bibliography

Abbas, N., Andersson, J., & Weyns, D. (2011). Knowledge evolution in autonomic software product lines. In *Proceedings of the 15th International Software Product Line Conference on - SPLC '11*. Accessed: 06/25/2014. http://dl.acm.org/citation.cfm?id=2019177 (p. 1). New York, New York, USA: ACM Press.

Ahmad, A., Jamshidi, P., & Pahl, C. (2014). Classification and comparison of architecture evolution reuse knowledge-a systematic review. *Journal of Software: Evolution and Process*. Accessed: 07/03/2014. http://onlinelibrary.wiley.com/doi/10.1002/smr.1643/full.

Amin, A., Colman, A., & Grunske, L. (2012). An Approach to Forecasting QoS Attributes of Web Services Based on ARIMA and GARCH Models. In *2012 IEEE 19th International Conference on Web Services*. Accessed: 06/30/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6257792 (pp. 74–81). IEEE.

Andersson, J., de Lemos, R., Malek, S., & Weyns, D. (2009). Reflecting on self-adaptive software systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5069072 (pp. 38–47). IEEE.

Anonymized access logs. (2001). Accessed: 09/23/2014. ftp://ftp.ircache.net/Traces/.

Apel, S., & Kästner, C. (2009). An Overview of Feature-Oriented Software Development. *Journal of Object Technology*. Accessed: 02/12/2014. http://www.jot.fm/issues/issue_2009_07/column5/index.

Arbab, F. (2004). Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*. Accessed: 06/28/2014. http://journals.cambridge.org/production/action/cjoGetFulltext?fulltextid=223762, *14*(3), 329–366.

Ardagna, D., Ghezzi, C., & Mirandola, R. (2008). Rethinking the use of models in software architecture. In *Quality of Software Architectures. Models and Architectures*. Accessed: 02/11/2014. http://link.springer.com/chapter/10.1007/978-3-540-87879-7_1.

Arora, S., Sampath, P., & Ramesh, S. (2012). Resolving uncertainty in automotive feature interactions. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6345807 (pp. 21–30). IEEE.

Asadollahi, R., Salehie, M., & Tahvildari, L. (2009). StarMX: A framework for developing self-managing Java-based systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. Accessed: 05/26/2014. http://www.uwspace.uwaterloo.ca/handle/10012/4728 (pp. 58–67). IEEE.

Aughenbaugh, J. (2006). *Managing uncertainty in engineering design using imprecise probabilities and principles of information economics*. Accessed: 12/16/2013. http://westinghouse.marc.gatech.edu/Members/jaughenbaugh/papers_presentations/aughenbaugh_jason_m_200608_phd.pdf. Georgia Institute of Technology.

Aughenbaugh, J. M., & Paredis, C. J. J. (2006). The Value of Using Imprecise Probabilities in Engineering Design. *Journal of Mechanical Design*. Accessed: 06/13/2014. http://link.aip.org/link/?JMDEDB/128/969/1, *128*(4), 969.

Autili, M., Cortellessa, V., Di Ruscio, D., Inverardi, P., Pelliccione, P., & Tivoli, M. (2011). EAGLE: engineering software in the ubiquitous globe by leveraging uncErtainty. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=2025199 (p. 488). New York, New York, USA: ACM Press.

Autili, M., Cortellessa, V., & Ruscio, D. Di. (2012). Integration architecture synthesis for taming uncertainty in the digital space. In *Large-Scale Complex IT Systems. Development, Operation and Management*. Accessed: 06/29/2014. http://link.springer.com/chapter/10.1007/978-3-642-34059-8_6.

Aziz, A., Sanwal, K., Singhal, V., & Brayton, R. (1996). Verifying continuous time Markov chains. *Computer Aided Verification*. Accessed: 02/03/2014. http://link.springer.com/chapter/10.1007/3-540-61474-5_75.

Baier, C., & Katoen, J. (2008). *Principles of model checking*. Accessed: 02/03/2014. http://mitpress.mit.edu/books/principles-model-checking. Cambridge, Massachusetts: The MIT Press.

Baresi, L. (2006). Toward Open-World Software: Issue and Challenges. *Computer*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1707632, *39*(10), 36–43.

Baresi, L., & Ghezzi, C. (2010). The disappearing boundary between development-time and run-time. In *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*. Accessed: 06/30/2014. http://dl.acm.org/citation.cfm?id=1882367 (p. 17). New York, New York, USA: ACM Press.

Baresi, L., Pasquale, L., & Spoletini, P. (2010). Fuzzy Goals for Requirements-Driven Adaptation. In *2010 18th IEEE International Requirements Engineering Conference*. Accessed: 06/18/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5636887 (pp. 125–134). IEEE.

Batista, T., Joolia, A., & Coulson, G. (2005). Managing dynamic reconfiguration in component-based systems. *Software Architecture*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/11494713_1.

Becker, S., Koziolek, H., & Reussner, R. (2007). Model-Based performance prediction with the palladio component model. In *Proceedings of the 6th international workshop on Software and performance*

- *WOSP '07*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1217006 (p. 54). New York, New York, USA: ACM Press.

Bencomo, N., & Belaggoun, A. (2013). Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks. *Requirements Engineering: Foundation for Software Quality*. Accessed: 01/27/2014. http://link.springer.com/chapter/10.1007/978-3-642-37422-7_16.

Bencomo, N., Grace, P., Flores, C., Hughes, D., & Blair, G. (2008). Genie: supporting the model driven development of reflective, component-based adaptive systems. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1368207 (p. 811). New York, New York, USA: ACM Press.

Bencomo, N., & Ramirez, A. (2012). RELAXing Claims: Coping With Uncertainty While Evaluating Assumptions at Run Time. *Model Driven Engineering Languages and Systems*. Accessed: 02/12/2014. http://hal.inria.fr/hal-00718997/.

Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., & Letier, E. (2010). Requirements reflection. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=1810329 (Vol. 2, p. 199). New York, New York, USA: ACM Press.

Bennani, M., & Menasce, D. A. (2005). Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *Second International Conference on Autonomic Computing (ICAC'05)*. Accessed: 05/26/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1498067 (pp. 229–240). IEEE.

Bertrand, D., Déplanche, A.-M., Faucou, S., & Roux, O. H. (2008). A Study of the AADL Mode Change Protocol. In *13th IEEE International Conference on Engineering of Complex Computer Systems (iceccs 2008)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4492905 (pp. 288–293). IEEE.

Bertsekas, D., & Tsitsiklis, J. (2002). *Introduction to probability*. Belmont, Mass: Athena Scientific.

Bianculli, D., Filieri, A., Ghezzi, C., & Mandrioli, D. (2014). Incremental Syntactic-Semantic Reliability Analysis of Evolving Structured Workflows. *Proceedings of the 6th International Symposium On Leveraging Applications of Formal Methods Verification and Validation (ISoLA 2014)*. Accessed: 07/03/2014. http://people.svv.lu/bianculli/pubs/bfgm-isola2014.pdf.

Billingsley, P. (1961). Statistical Methods in Markov Chains. *The Annals of Mathematical Statistics*. Accessed: 07/03/2014. http://www.jstor.org/stable/10.2307/2237603, *32*(1), 12–40.

Bladt, M., & Sorensen, M. (2005). Statistical inference for discretely observed Markov jump processes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. Accessed: 07/03/2014. http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9868.2005.00508.x/full, *67*(3), 395–410.

Blair, G., Bencomo, N., & France, R. B. (2009). Models@ run.time. *Computer*. Accessed: 06/08/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5280648, *42*(10), 22–27.

Bliudze, S., & Sifakis, J. (2007). The algebra of connectors. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software - EMSOFT '07*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1289935 (p. 11). New York, New York, USA: ACM Press.

Bolstad, W. (2011). *Understanding computational Bayesian statistics*. Wiley.

Borde, E., Haik, G., & Pautet, L. (2009). Mode-based reconfiguration of critical software component architectures. In *2009 Design, Automation & Test in Europe Conference & Exhibition*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5090838 (pp. 1160–1165). IEEE.

Briand, L., & van der Hoek, A. (2014). Companion Proceedings of the 36th International Conference on Software Engineering. http://dl.acm.org/citation.cfm?id=2591062. ACM.

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., & Stefani, J.-B. (2006). The FRACTAL component model and its support in Java. *Software: Practice and Experience*. Accessed: 06/30/2014. http://onlinelibrary.wiley.com/doi/10.1002/spe.767/abstract, *36*(11-12), 1257–1284.

Bruni, R., Melgratti, H., & Montanari, U. (2013). A Survey on Basic Connectors and Buffers. *Formal Methods for Components and Objects*. Accessed: 02/05/2014. http://link.springer.com/chapter/10.1007/978-3-642-35887-6_3.

Bures, T., Hnetynka, P., & Plasil, F. (2006). SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1691359 (pp. 40–48). IEEE.

Calinescu, R., Ghezzi, C., Kwiatkowska, M., & Mirandola, R. (2012). Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*. Accessed: 06/23/2014. http://dl.acm.org/citation.cfm?id=2330686, *55*(9), 69.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., & Tamburrelli, G. (2011). Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*. Accessed: 06/11/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5611553, *37*(3), 387–409.

Calinescu, R., Johnson, K., & Rafiq, Y. (2011). Using observation ageing to improve markovian model learning in QoS engineering. In *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering - ICPE '11*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1958823 (p. 505). New York, New York, USA: ACM Press.

Calinescu, R., & Kwiatkowska, M. (2009). Using quantitative analysis to implement autonomic IT systems. In *2009 IEEE 31st International Conference on Software Engineering*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=1555026 (pp. 100–110). IEEE.

Cámara, J., Moreno, G. A., & Garlan, D. (2014). Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proceedings of the 9th International Symposium on Software*

Engineering for Adaptive and Self-Managing Systems - SEAMS 2014. Accessed: 06/16/2014. http://works.bepress.com/gabriel_moreno/23/ (pp. 155–164). New York, New York, USA: ACM Press.

Cavallo, B., Di Penta, M., & Canfora, G. (2010). An empirical comparison of methods to support QoS-aware service selection. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems - PESOS '10*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1808899 (p. 64). New York, New York, USA: ACM Press.

Cazzola, W., Savigni, A., Sosio, A., & Tisato, F. (1998). Architectural reflection: Bridging the gap between a system and its architectural specification. *REF'98*. Accessed: 02/12/2014. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.388.2958.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2009). Using Feature Models for Developing Self-Configuring Smart Homes. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4976601 (pp. 179–188). IEEE.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2010). Designing and prototyping dynamic software product lines: techniques and guidelines. In *Software Product Lines: Going Beyond*. Accessed: 06/18/2014. http://link.springer.com/chapter/10.1007/978-3-642-15579-6_23.

Cetina, C., Haugen, Ø., & Zhang, X. (2009). Strategies for variability transformation at run-time. *Proceedings of the 13th International Software Product Line Conference*. Accessed: 02/12/2014. http://dl.acm.org/citation.cfm?id=1753245.

Chan, A. (2008). Dynamic QoS Adaptation for Mobile Middleware. *IEEE Transactions on Software Engineering*. Accessed: 06/21/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4547430, *34*(6), 738–752.

Chauvel, F., Barais, O., Borne, I., & Jezequel, J.-M. (2008). Composition of Qualitative Adaptation Policies. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1642998 (pp. 455–458). IEEE.

Chauvel, F., Song, H., & Chen, X. (2010). Using qos-contracts to drive architecture-centric self-adaptation. *Research into Practice – Reality and Gaps*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-13821-8_9.

Chen, B., Peng, X., Yu, Y., & Zhao, W. (2014). Uncertainty handling in goal-driven self-optimization – Limiting the negative effect on adaptation. *Journal of Systems and Software*. Accessed: 06/05/2014. http://www.sciencedirect.com/science/article/pii/S0164121214000065, *90*, 114–127.

Cheng, B., Sawyer, P., Bencomo, N., & Whittle, J. (2009). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. *Model Driven Engineering Languages and Systems*. Accessed: 01/26/2014. http://link.springer.com/chapter/10.1007/978-3-642-04425-0_36.

Cheng, S., & Garlan, D. (2007). Handling uncertainty in autonomic systems. *ASE*. Accessed: 12/16/2013. http://acme.able.cs.cmu.edu/pubs/uploads/pdf/IWLU07-HandlingUncertainties-pub.pdf.

Cheng, S.-W., & Garlan, D. (2012). Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*. Accessed: 06/11/2014. http://www.sciencedirect.com/science/article/pii/S0164121212000714, *85*(12), 2860–2875.

Cheng, S.-W., Garlan, D., & Schmerl, B. (2006). Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems - SEAMS '06*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1137679 (p. 2). New York, New York, USA: ACM Press.

Cheung, L., Roshandel, R., Medvidovic, N., & Golubchik, L. (2008). Early prediction of software component reliability. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*. Accessed: 06/03/2014. http://dl.acm.org/citation.cfm?id=1368104 (p. 111). New York, New York, USA: ACM Press.

Cheung, R. (1980). A User-Oriented Software Reliability Model. *IEEE Transactions on Software Engineering*. Accessed: 06/21/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1702709, *SE-6*(2), 118–125.

Chopra, A. K. (2012). The Meaning of Requirements and Adaptation. *arXiv Preprint arXiv:1209.1551*. Accessed: 07/03/2014. http://arxiv.org/abs/1209.1551, 12. Software Engineering.

Ciancone, A., Filieri, A., & Drago, M. (2011). KlaperSuite: an integrated model-driven environment for reliability and performance analysis of component-based systems. *Objects, Models, Components, Patterns*. Accessed: 02/02/2014. http://link.springer.com/chapter/10.1007/978-3-642-21952-8_9.

Computing, A., Gandhi, A., Dube, P., & Karve, A. (2014). Adaptive, Model-driven Autoscaling for Cloud Applications. *Usenix.org*. Accessed: 06/30/2014. https://www.usenix.org/system/files/conference/icac14/icac14-paper-gandhi.pdf.

Cooray, D., Malek, S., Roshandel, R., & Kilgore, D. (2010). RESISTing reliability degradation through proactive reconfiguration. In *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1859011 (p. 83). New York, New York, USA: ACM Press.

Cortellessa, V., Marco, A. Di, & Inverardi, P. (2007). Integrating performance and reliability analysis in a non-functional MDA framework. *Fundamental Approaches to Software Engineering*. Accessed: 02/11/2014. http://link.springer.com/chapter/10.1007/978-3-540-71289-3_6.

Coulson, G., Blair, G. S., Clarke, M., & Parlavantzas, N. (2002). The design of a configurable and reconfigurable middleware platform. *Distributed Computing*. Accessed: 07/03/2014. http://link.springer.com/article/10.1007/s004460100064, *15*(2), 109–126.

Crnkovic, I., Sentilles, S., Vulgarakis, A., & Chaudron, M. R. V. (2011). A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*. Accessed: 05/26/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5587419, *37*(5), 593–615.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2004). Staged configuration using feature models. *Software Product Lines*. Accessed: 03/14/2014. http://link.springer.com/chapter/10.1007/978-3-540-28630-1_17.

D'Ippolito, N., Braberman, V., Kramer, J., Magee, J., Sykes, D., & Uchitel, S. (2014). Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. Accessed: 06/16/2014. http://www.doc.ic.ac.uk/~das05/icse2014appendix.pdf (pp. 688–699). New York, New York, USA: ACM Press.

David, P., & Ledoux, T. (2006). An aspect-oriented approach for developing self-adaptive fractal components. *Software Composition*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/11821946_6.

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*. Accessed: 05/27/2014. http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2575.2004.00162.x/full, *14*(1), 65–86.

Di Marco, A., Inverardi, P., & Spalazzese, R. (2013). Synthesizing self-adaptive connectors meeting functional and performance concerns. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6595500 (pp. 133–142). IEEE.

Diaconis, P., & Ylvisaker, D. (1979). Conjugate priors for exponential families. *The Annals of Statistics*. Accessed: 03/28/2014. http://projecteuclid.org/euclid.aos/1176344611.

Dobson, S., Zambonelli, F., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., … Schmidt, N. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*. Accessed: 06/11/2014. http://dl.acm.org/citation.cfm?id=1186782, *1*(2), 223–259.

Dubois, D., Foulloy, L., Mauris, G., & Prade, H. (2004). Probability-possibility transformations, triangular fuzzy sets, and probabilistic inequalities. *Reliable Computing*. Accessed: 02/18/2014. http://link.springer.com/article/10.1023/B:REOM.0000032115.22510.b5.

Eder, K., Villegas, N., Trollmann, F., Pelliccione, P., Muller, H. A., Schneider, D., … Perini, A. (2013). Assurance Using Models at Runtime for Self-Adaptive Software Systems. *Springer, Berlin, LNCS*. Accessed: 02/11/2014. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.309.3457&rep=rep1&type=pdf.

Eliassen, F., Gjørven, E., Eide, V. S. W., & Michaelsen, J. A. (2006). Evolving self-adaptive services using planning-based reflective middleware. In *Proceedings of the 5th workshop on Adaptive and reflective middleware (ARM '06) - ARM '06*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1175856 (p. 1). New York, New York, USA: ACM Press.

Elkhodary, A., Esfahani, N., & Malek, S. (2010). FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1882296 (p. 7). New York, New York, USA: ACM Press.

Engels, G., & Bencomo, N. (2014). Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. http://dl.acm.org/citation.cfm?id=2593929. ACM.

Epifani, I., Ghezzi, C., Mirandola, R., & Tamburrelli, G. (2009). Model evolution by run-time parameter adaptation. In *2009 IEEE 31st International Conference on Software Engineering*. Accessed: 06/02/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5070513 (pp. 111–121). IEEE.

Esfahani, N., Elkhodary, A., & Malek, S. (2013). A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems. *IEEE Transactions on Software Engineering*. Accessed: 06/11/2014. http://dl.acm.org/citation.cfm?id=2521631, *39*(11), 1467–1493.

Esfahani, N., Kouroshfar, E., & Malek, S. (2011). Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*. Accessed: 06/16/2014. http://cs.gmu.edu/~smalek/papers/esecfse2011.pdf (p. 234). New York, New York, USA: ACM Press.

Esfahani, N., & Malek, S. (2013). Uncertainty in self-adaptive software systems. *Software Engineering for Self-Adaptive Systems II*. Accessed: 12/15/2013. http://link.springer.com/chapter/10.1007/978-3-642-35813-5_9.

Esfahani, N., Malek, S., & Razavi, K. (2013). GuideArch: Guiding the exploration of architectural solution space under uncertainty. In *2013 35th International Conference on Software Engineering (ICSE)*. Accessed: 06/03/2014. http://dl.acm.org/citation.cfm?id=2486795 (pp. 43–52). IEEE.

Famelis, M., Salay, R., & Chechik, M. (2012). Partial models: Towards modeling and reasoning with uncertainty. In *2012 34th International Conference on Software Engineering (ICSE)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6227159 (pp. 573–583). IEEE.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). *Cloud Computing Patterns. Springer*. Accessed: 07/03/2014. http://link.springer.com/content/pdf/10.1007/978-3-7091-1568-8.pdf. Vienna: Springer Vienna.

Filieri, A. (2013). *Model based verification and adaptation of software systems@ runtime*. http://www.politesi.polimi.it/handle/10589/74321. Politecnico di Milano.

Filieri, A., Ghezzi, C., Grassi, V., & Mirandola, R. (2010). Reliability analysis of component-based systems with multiple failure modes. *Component-Based Software Engineering*. Accessed: 02/03/2014. http://link.springer.com/chapter/10.1007/978-3-642-13238-4_1.

Filieri, A., Ghezzi, C., Leva, A., & Maggio, M. (2012). Reliability-driven dynamic binding via feedback control. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Accessed: 05/26/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6224390 (pp. 43–52). IEEE.

Filieri, A., Ghezzi, C., & Tamburrelli, G. (2011). Run-time efficient probabilistic model checking. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=1985840 (p. 341). New York, New York, USA: ACM Press.

Filieri, A., Ghezzi, C., & Tamburrelli, G. (2012). A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*. Accessed: 01/27/2014. http://link.springer.com/article/10.1007/s00165-011-0207-2.

Filieri, A., Hoffmann, H., & Maggio, M. (2014). Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?doid=2568225.2568272 (pp. 299–310). New York, New York, USA: ACM Press.

Filieri, A., Tamburrelli, G., & Ghezzi, C. (2013). Supporting Self-adaptation via Quantitative Verification and Sensitivity Analysis at Run Time. *IEEE Transactions on Software Engineering*. Accessed: 05/29/2014. http://www.iste.uni-stuttgart.de/fileadmin/user_upload/iste/zss/publications/supplementaryMaterial/2013-TSE-WM-PaperReviewCopy.pdf.

Fleurey, F., & Solberg, A. (2009). A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. *Model Driven Engineering Languages and Systems*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-04425-0_47.

Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., & Gjorven, E. (2006). Using architecture models for runtime adaptability. *IEEE Software*. Accessed: 06/30/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1605180, *23*(2), 62–70.

Fredericks, E., DeVries, B., & Cheng, B. (2014). AutoRELAX: automatically RELAXing a goal model to address uncertainty. *Empirical Software Engineering*. Accessed: 05/30/2014. http://link.springer.com/article/10.1007/s10664-014-9305-0.

Fredericks, E. M., DeVries, B., & Cheng, B. H. C. (2014). Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. Accessed: 06/10/2014. http://dl.acm.org/citation.cfm?doid=2593929.2593937 (pp. 17–26). New York, New York, USA: ACM Press.

Fredericks, E. M., Ramirez, A. J., & Cheng, B. H. C. (2013). Towards run-time testing of dynamic adaptive systems. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-*

Managing Systems (SEAMS). Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6595504 (pp. 169–174). IEEE.

Gallotti, S., & Ghezzi, C. (2008). Quality prediction of service compositions through probabilistic model checking. *Quality of Software Architectures. Models and Architectures*. Accessed: 02/02/2014. http://link.springer.com/chapter/10.1007/978-3-540-87879-7_8.

Gambi, A., Hummer, W., Truong, H.-L., & Dustdar, S. (2013). Testing Elastic Computing Systems. *IEEE Internet Computing*. Accessed: 07/02/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6682972, *17*(6), 76–82.

Gambi, A., Toffetti, G., & Pezzè, M. (2010). Protecting SLAs with surrogate models. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems - PESOS '10*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1808900 (p. 71). New York, New York, USA: ACM Press.

Gandhi, A. (2013). *Dynamic Server Provisioning for Data Center Power Management*. Accessed: 06/30/2014. http://reports-archive.adm.cs.cmu.edu/anon/anon/home/ftp/usr0/ftp/2013/CMU-CS-13-110.pdf. Carnegie Mellon University.

Gandhi, A., Harchol-Balter, M., Raghunathan, R., & Kozuch, M. A. (2012). AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Transactions on Computer Systems*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=2382556, *30*(4), 1–26.

Garlan, D. (2010). Software engineering in an uncertain world. In *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1882389 (p. 125). New York, New York, USA: ACM Press.

Garlan, D., Cheng, S., Huang, A., Schmerl, B., & Steenkiste, P. (2004). Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*. Accessed: 06/23/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1350726, *37*(10), 46–54.

Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., … Stav, E. (2009). A comprehensive solution for application-level adaptation. *Software: Practice and Experience*. Accessed: 07/03/2014. http://onlinelibrary.wiley.com/doi/10.1002/spe.900/abstract, *39*(4), 385–422.

Gelernter, D., & Carriero, N. (1992). Coordination languages and their significance. *Communications of the ACM*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=376083, *35*(2), 97–107.

Georgas, J. C., Hoek, A. van der, & Taylor, R. N. (2009). Using Architectural Models to Manage and Visualize Runtime Adaptation. *Computer*. Accessed: 07/03/2014. http://cat.inist.fr/?aModele=afficheN&cpsidt=22088650, *42*(10), 52–60.

Georgas, J. C., & Taylor, R. N. (2008). Policy-based self-adaptive architectures: Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems - SEAMS '08*. Accessed:

06/16/2014. http://dl.acm.org/citation.cfm?id=1370038 (p. 105). New York, New York, USA: ACM Press.

Georgiadis, I., Magee, J., & Kramer, J. (2002). Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems - WOSS '02*. Accessed: 06/11/2014. http://dl.acm.org/citation.cfm?id=582135 (p. 33). New York, New York, USA: ACM Press.

Ghezzi, C., Pinto, L. S., Spoletini, P., & Tamburrelli, G. (2013). Managing non-functional uncertainty via model-driven adaptivity. In *2013 35th International Conference on Software Engineering (ICSE)*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=2486794 (pp. 33–42). IEEE.

Ghezzi, C., & Sharifloo, A. (2013). Dealing with Non-Functional Requirements for Adaptive Systems via Dynamic Software Product-Lines. *Software Engineering for Self-Adaptive Systems II*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-35813-5_8.

Ghezzi, C., & Sharifloo, A. M. (2011). Verifying Non-functional Properties of Software Product Lines: Towards an Efficient Approach Using Parametric Model Checking. In *2011 15th International Software Product Line Conference*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6030058 (pp. 170–174). IEEE.

Ghezzi, C., & Tamburrelli, G. (2009). Predicting performance properties for open systems with KAMI. *Architectures for Adaptive Software Systems*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-02351-4_5.

Glinz, M. (2005). Rethinking the notion of non-functional requirements. *Proc. Third World Congress for Software Quality*. Accessed: 02/21/2014. http://www.ptidej.net/course/log3410/fall11/Lectures/Article_5.pdf.

Gmach, D., Krompass, S., Scholz, A., Wimmer, M., & Kemper, A. (2008). Adaptive quality of service management for enterprise services. *ACM Transactions on the Web*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1326569, *2*(1), 1–46.

Gokhale, S. (2007). Architecture-Based Software Reliability Analysis: Overview and Limitations. *IEEE Transactions on Dependable and Secure Computing*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4099190, *4*(1), 32–40.

Goldsby, H. J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Hughes, D. (2008). Goal-Based Modeling of Dynamically Adaptive System Requirements. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4492385 (pp. 36–45). IEEE.

Graph operations. (2014). Accessed: 09/23/2014. http://en.wikipedia.org/wiki/Graph_operations.

Grunske, L. (2008). Specification patterns for probabilistic quality properties. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*. Accessed: 07/03/2014.

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4814114 (p. 31). New York, New York, USA: ACM Press.

Hagras, H. (2007). Type-2 FLCs: A New Generation of Fuzzy Controllers. *IEEE Computational Intelligence Magazine*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4195040, *2*(1), 30–43.

Hallsteinsen, S., Hinchey, M., & Schmid, K. (2008). Dynamic Software Product Lines. *Computer*. Accessed: 05/26/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4488260, *41*(4), 93–95.

Hallsteinsen, S., Stav, E., Solberg, A., & Floch, J. (2006). Using Product Line Techniques to Build Adaptive Systems. In *10th International Software Product Line Conference (SPLC'06)*. Accessed: 06/26/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1691586 (pp. 141–150). IEEE.

Hansson, H., & Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*. Accessed: 07/03/2014. http://link.springer.com/article/10.1007/BF01211866, *6*(5), 512–535.

Heaven, W., Sykes, D., Magee, J., & Kramer, J. (2009). A case study in goal-driven architectural adaptation. *Software Engineering for Self-Adaptive Systems*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-02161-9_6.

Hellerstein, J., Diao, Y., Parekh, S., & Tilbury, D. (2004). *Feedback control of computing systems*. Hoboken, NJ: John Wiley & Sons.

Hevner, A., March, S., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*. Accessed: 12/15/2013. http://dl.acm.org/citation.cfm?id=2017217.

Hielscher, J., & Kazhamiakin, R. (2008). A framework for proactive self-adaptation of service-based applications based on online testing. *Towards a Service-Based Internet*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-540-89897-9_11.

Hirsch, D., Kramer, J., Magee, J., & Uchitel, S. (2006). Modes for software architectures. *Software Architecture*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/11966104_9.

Hoff, P. (2009). *A first course in Bayesian statistical methods*. Springer.

Homer, A., Sharp, J., Brader, L., Narumoto, M., & Swanson, T. (2014). *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Microsoft.

Immonen, A., & Niemelä, E. (2007). Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software & Systems Modeling*. Accessed: 07/03/2014. http://link.springer.com/article/10.1007/s10270-006-0040-x, *7*(1), 49–65.

Inamura, Y. (2006). Estimating continuous time transition matrices from discretely observed data. Accessed: 04/03/2014. https://www.boj.or.jp/en/research/wps_rev/wps_2006/wp06e07.htm/. Japan.

Inverardi, P., Issarny, V., & Spalazzese, R. (2010). A theory of mediators for eternal connectors. *Leveraging Applications of Formal Methods, Verification, and Validation*. Accessed: 06/29/2014. http://link.springer.com/chapter/10.1007/978-3-642-16561-0_25.

Inverardi, P., Spalazzese, R., & Tivoli, M. (2011). Application-layer connector synthesis. *Formal Methods for Eternal Networked Software Systems*. Accessed: 06/29/2014. http://link.springer.com/chapter/10.1007/978-3-642-21455-4_5.

Jafry, Y., & Schuermann, T. (2004). Measurement, estimation and comparison of credit migration matrices. *Journal of Banking & Finance*. Accessed: 02/12/2014. http://www.sciencedirect.com/science/article/pii/S0378426604001037.

Jamshidi, P., Ahmad, A., & Pahl, C. (2014). Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?doid=2593929.2593940 (pp. 95–104). New York, New York, USA: ACM Press.

Jamshidi, P., Ghafari, M., Ahmad, A., & Pahl, C. (2013). A Framework for Classifying and Comparing Architecture-centric Software Evolution Research. In *2013 17th European Conference on Software Maintenance and Reengineering*. Accessed: 06/28/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6498478 (pp. 305–314). IEEE.

Jamshidi, P., & Pahl, C. (2014). Cloud Migration Patterns - Supplementary Materials. Accessed: 09/23/2014. http://www.computing.dcu.ie/~pjamshidi/Materials/CMP.html.

Jean-Baptiste, L., Maria-Teresa, S., Jean-Marie, G., & Antoine, B. (2013). Modeling dynamic adaptations using augmented feature models. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*. Accessed: 05/29/2014. http://dl.acm.org/citation.cfm?id=2480690 (p. 1734). New York, New York, USA: ACM Press.

JMeter. (2014). Accessed: 09/23/2014. http://jakarta.apache.org/jmeter/.

Kalekar, P. (2004). Time series forecasting using Holt-Winters exponential smoothing. *Kanwal Rekhi School of Information Technology*. Accessed: 03/28/2014. http://www.it.iitb.ac.in/~praj/acads/seminar/04329008_ExponentialSmoothing.pdf.

Karnik, N., & Mendel, J. (1998). Introduction to type-2 fuzzy logic systems. In *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228)*. Accessed: 06/13/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=686240 (Vol. 2, pp. 915–920). IEEE.

Karnik, N., & Mendel, J. (1999). Type-2 fuzzy logic systems. *IEEE Transactions on Fuzzy Systems*. Accessed: 06/13/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=811231, *7*(6), 643–658.

Karnik, N. N., & Mendel, J. M. (2001). Centroid of a type-2 fuzzy set. *Information Sciences*. Accessed: 07/03/2014. http://www.sciencedirect.com/science/article/pii/S002002550100069X, *132*(1-4), 195–220.

Katz, E., & Katz, S. (2008). Incremental analysis of interference among aspects. In *Proceedings of the 7th workshop on Foundations of aspect-oriented languages - FOAL '08*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1394500 (pp. 29–38). New York, New York, USA: ACM Press.

Kell, S. (2007). Rethinking software connectors. In *International workshop on Synthesis and analysis of component connectors in conjunction with the 6th ESEC/FSE joint meeting - SYANCO '07*. Accessed: 07/01/2014. http://dl.acm.org/citation.cfm?id=1294918 (pp. 1–12). New York, New York, USA: ACM Press.

Kephart, J., & Chess, D. (2003). The vision of autonomic computing. *Computer*. Accessed: 06/01/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1160055, *36*(1), 41–50.

Kephart, J., & Das, R. (2007). Achieving Self-Management via Utility Functions. *IEEE Internet Computing*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4061119, *11*(1), 40–48.

Kiczales, G. (1996). Aspect-oriented programming. *ACM Computing Surveys*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=242420, *28*(4es), 154–es.

Kitano, H. (2004). Biological robustness. *Nature Reviews. Genetics*. Accessed: 05/28/2014. http://www.nature.com/nrg/journal/v5/n11/abs/nrg1471.html, *5*(11), 826–37.

Klein, G. (2007). Flexecution as a Paradigm for Replanning, Part 1. *IEEE Intelligent Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4338498, *22*(5), 79–83.

Klir, G., & Yuan, B. (1995). *Fuzzy sets and fuzzy logic*. Upper Saddle River, New Jersey: Prentice Hall.

Kotonya, G. (2010). Combining Service-Orientation with Product Line Engineering. *IEEE Software*. Accessed: 05/28/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5406497, *27*(3), 35–41.

Kramer, J., & Magee, J. (2007). Self-Managed Systems: an Architectural Challenge. In *Future of Software Engineering (FOSE '07)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4221625 (pp. 259–268). IEEE.

Kwiatkowska, M. (2007). Quantitative verification. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1287688 (p. 449). New York, New York, USA: ACM Press.

Kwiatkowska, M., Norman, G., & Parker, D. (2007). Stochastic model checking. *Formal Methods for Performance Evaluation*. Accessed: 02/04/2014. http://link.springer.com/chapter/10.1007/978-3-540-72522-0_6.

Kwiatkowska, M., Norman, G., & Parker, D. (2010). Advances and challenges of probabilistic model checking. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5707120 (pp. 1691–1698). IEEE.

Lapouchnian, A., Yu, Y., Liaskos, S., & Mylopoulos, J. (2006). Requirements-driven design of autonomic application software. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1188976 (p. 7). New York, New York, USA: ACM Press.

Lau, K., Elizondo, P., & Wang, Z. (2005). Exogenous connectors for software components. *Component-Based Software Engineering*. Accessed: 06/18/2014. http://link.springer.com/chapter/10.1007/11424529_7.

Lee, J., & Kang, K. (2006). A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In *10th International Software Product Line Conference (SPLC'06)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1691585 (pp. 131–140). IEEE.

Leitner, P., Michlmayr, A., Rosenberg, F., & Dustdar, S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *2010 IEEE International Conference on Web Services*. Accessed: 06/21/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5552763 (pp. 369–376). IEEE.

Lemos, R. De, Giese, H., & Müller, H. (2013). Software engineering for self-adaptive systems: A second research roadmap. *Software Engineering for for Self-Adaptive Systems II*. Accessed: 12/15/2013. http://link.springer.com/chapter/10.1007/978-3-642-35813-5_1.

Letier, E., Stefan, D., & Barr, E. T. (2014). Uncertainty, risk, and information value in software requirements and architecture. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. Accessed: 06/10/2014. http://www0.cs.ucl.ac.uk/staff/e.letier/publications/2014-ICSE-Uncertainty.pdf (pp. 883–894). New York, New York, USA: ACM Press.

Letier, E., & van Lamsweerde, A. (2004). Reasoning about partial goal satisfaction for requirements and design engineering. *ACM SIGSOFT Software Engineering Notes*. Accessed: 06/01/2014. http://dl.acm.org/citation.cfm?id=1029905, *29*(6), 53.

Liang, Q., & Mendel, J. M. (2000). Designing interval type-2 fuzzy logic systems using an SVD-QR method: Rule reduction. *International Journal of Intelligent Systems*. Accessed: 07/03/2014. http://onlinelibrary.wiley.com/doi/10.1002/1098-111X(200010)15:10%3C939::AID-INT3%3E3.0.CO;2-G/abstract, *15*(10), 939–957.

Linda, O., & Manic, M. (2011). Uncertainty-Robust Design of Interval Type-2 Fuzzy Logic Controller for Delta Parallel Robot. *IEEE Transactions on Industrial Informatics*. Accessed: 06/02/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6009194, *7*(4), 661–670.

Littlewood, B. (1975). A Reliability Model for Systems with Markov Structure. *Applied Statistics*. Accessed: 07/03/2014. http://www.jstor.org/stable/10.2307/2346564, *24*(2), 172.

Liu, X. (Frank), Azmoodeh, M., & Georgalas, N. (2007). Specification of Non-functional Requirements for Contract Specification in the NGOSS Framework for Quality Management and Product Evaluation. In *Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4273474 (pp. 7–7). IEEE.

Liu, X., & Yen, J. (1996). An analytic framework for specifying and analyzing imprecise requirements. In *Proceedings of IEEE 18th International Conference on Software Engineering*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=227738 (pp. 60–69). IEEE Comput. Soc. Press.

Load Runner. (2014). Accessed: 09/23/2014. http://www8.hp.com/us/en/software-solutions/loadrunner-load-testing/.

Marzolla, M., & Mirandola, R. (2010). Performance aware reconfiguration of software systems. *Computer Performance Engineering*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-642-15784-4_4.

MATLAB Builder NE. (2014). Accessed: 09/23/2014. http://www.mathworks.co.uk/products/netbuilder/index.html.

Maximum clique. (2014). Accessed: 09/23/2014. http://en.wikipedia.org/wiki/Maximum_clique#Definitions.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P., & Cheng, B. H. C. (2004). Composing adaptive software. *Computer*. Accessed: 06/18/2014. http://di.ufpe.br/~redis/middleware/mckinley-composition04.pdf, *37*(7), 56–64.

Medvidovic, N., & Taylor, R. (2009). *Software architecture: foundations, theory, and practice*. Wiley.

Mendel, J. (2000). Interval type-2 fuzzy logic systems: theory and design. *IEEE Transactions on Fuzzy Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=873577, *8*(5), 535–550.

Mendel, J. (2001). *Uncertain rule-based fuzzy logic system: introduction and new directions*. Prentice Hall.

Mendel, J. (2007). Type-2 fuzzy sets and systems: an overview. *Computational Intelligence Magazine, IEEE*. Accessed: 02/05/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4197699.

Mendel, J. (2008). Encoding Words Into Interval Type-2 Fuzzy Sets Using an Interval Approach. *IEEE Transactions on Fuzzy Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4607249, *16*(6), 1503–1521.

Mendel, J. (2009). Enhanced Karnik--Mendel Algorithms. *IEEE Transactions on Fuzzy Systems*. Accessed: 06/16/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4505357, *17*(4), 923–934.

Mendel, J., Hagras, H., & John, R. (2010). Standard background material about interval type-2 fuzzy logic systems that can be used by all authors. *IEEE Computational Intelligence Society*. Accessed: 02/05/2014. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.125.4195&rep=rep1&type=pdf.

Mendel, J., & John, R. (2002). Type-2 fuzzy sets made simple. *IEEE Transactions on Fuzzy Systems*. Accessed: 06/12/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=995115, *10*(2), 117–127.

Mendel, J., Karnik, N., & Liang, Q. (2000). Connection admission control in ATM networks using survey-based type-2 fuzzy logic systems. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=885114, *30*(3), 329–339.

Mendel, J. M., & Coupland, S. (2012). Enhanced Interval Approach for Encoding Words Into Interval Type-2 Fuzzy Sets and Its Convergence Analysis. *IEEE Transactions on Fuzzy Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6086759, *20*(3), 499–513.

Mendel, J. M., & John, R. I. (2002). Footprint of Uncertainty and Its Importance to Type-2 Fuzzy Sets. *Actapress.com*. Accessed: 02/15/2014. http://www.actapress.com/PaperInfo.aspx?PaperID=26102&reason=500.

Mendel, J. M., John, R. I., & Liu, F. (2006). Interval Type-2 Fuzzy Logic Systems Made Simple. *IEEE Transactions on Fuzzy Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4016089, *14*(6), 808–821.

Mendel, J., & Wu, D. (2010). *Perceptual computing: aiding people in making subjective judgments*. John Wiley & Sons.

Metzger, A., Sammodi, O., & Pohl, K. (2013). Accurate proactive adaptation of service-oriented systems. *Assurances for Self-Adaptive Systems*. Accessed: 02/11/2014. http://link.springer.com/chapter/10.1007/978-3-642-36249-1_9.

Microsoft. (2014). Model View ViewModel. http://en.wikipedia.org/wiki/Model_View_ViewModel.

Montero, I., Pena, J., & Ruiz-Cortes, A. (2008). Representing Runtime Variability in Business-Driven Development Systems. In *Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4464029 (pp. 228–231). IEEE.

Moon, Y. (2011). *Stochastic models for quality of service of component connectors*. Accessed: 02/06/2014. https://openaccess.leidenuniv.nl/handle/1887/17975.

Moon, Y., Arbab, F., & Silva, A. (2011). Stochastic Reo: a Case Study. *ENTCS*. Accessed: 02/06/2014. https://www.duo.uio.no/bitstream/handle/123456789/8998/TR409.pdf?sequence=1#page=90.

Morandini, M., Penserini, L., & Perini, A. (2008). Towards goal-oriented development of self-adaptive systems. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems - SEAMS '08*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1370021 (p. 9). New York, New York, USA: ACM Press.

Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., & Solberg, A. (2009). Models@ Run.time to Support Dynamic Adaptation. *Computer*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5280651, *42*(10), 44–51.

Morin, B., Fleurey, F., & Bencomo, N. (2008). An aspect-oriented and model-driven approach for managing dynamic variability. *Model Driven Engineering Languages and Systems*. Accessed: 06/18/2014. http://link.springer.com/chapter/10.1007/978-3-540-87875-9_54.

Müller, H., Pezzè, M., & Shaw, M. (2008). Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems - ULSSIS '08*. Accessed: 07/03/2014. http://www.researchgate.net/publication/228670531_Visibility_of_control_in_adaptive_systems/file/3deec5171b4a937da9.pdf (pp. 23–26). New York, New York, USA: ACM Press.

Narayanan, D., & Satyanarayanan, M. (2003). Predictive Resource Management for Wearable Computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03*. Accessed: 06/12/2014. http://dl.acm.org/citation.cfm?id=1189041 (pp. 113–128). New York, New York, USA: ACM Press.

Oliveira, N., & Barbosa, L. (2013). Reconfiguration mechanisms for service coordination. *Web Services and Formal Methods*. Accessed: 02/05/2014. http://link.springer.com/chapter/10.1007/978-3-642-38230-7_9.

Oliveira, N., & Barbosa, L. S. (2013). On the reconfiguration of software connectors. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=2480712 (p. 1885). New York, New York, USA: ACM Press.

Oreizy, P., Medvidovic, N., & Taylor, R. (1998). Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*. Accessed: 06/28/2014. http://dl.acm.org/citation.cfm?id=302181 (pp. 177–186). IEEE Comput. Soc.

OSGi Alliance. (2014). Accessed: 09/23/2014. http://www.osgi.org/Main/HomePage.

Papadopoulos, G., & Arbab, F. (1998). Coordination models and languages. *Advances in Computers*. Accessed: 12/14/2013. http://www.sciencedirect.com/science/article/pii/S0065245808602089.

PARAM Model Checker. (2013). Accessed: 09/23/2014. http://www.avacs.org/tools/param/.

Park, S. (2009). Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. Accessed: 05/26/2014. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5069076 (pp. 76–85). IEEE.

Parra, C., Blanc, X., Cleve, A., & Duchien, L. (2011). Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*. Accessed: 05/30/2014. http://tel.archives-ouvertes.fr/tel-00583444/, *76*(12), 1247–1260.

Patikirikorala, T., Colman, A., Han, J., & Wang, L. (2012). A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6224389 (pp. 33–42). IEEE.

Pawlak, R., Seinturier, L., Duchien, L., Florin, G., Legond-Aubry, F., & Martelli, L. (2004). JAC: an aspect-based distributed dynamic framework. *Software: Practice and Experience*. Accessed: 07/03/2014. http://onlinelibrary.wiley.com/doi/10.1002/spe.605/abstract, *34*(12), 1119–1148.

Perez-Palacin, D., & Mirandola, R. (2014). Uncertainties in the modeling of self-adaptive systems. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering - ICPE '14*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?doid=2568088.2568095 (pp. 3–14). New York, New York, USA: ACM Press.

Perrouin, G., & Chauvel, F. (2008). Modeling the variability space of self-adaptive applications. *2nd Dynamic Software Product Lines Workshop*. Accessed: 02/12/2014. http://hal.inria.fr/inria-00456531/.

Pfleeger, S. (1995). Experimental design and analysis in software engineering. *Annals of Software Engineering*. Accessed: 02/12/2014. http://link.springer.com/article/10.1007/BF02249052.

Pham, H. (2006). *System software reliability*. London: Springer.

Pinsky, M., & Karlin, S. (2010). *An introduction to stochastic modeling*. Elsevier.

Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Accessed: 02/01/2014. http://dl.acm.org/citation.cfm?id=1869735. Springer.

Poladian, V., Garlan, D., Shaw, M., Satyanarayanan, M., Schmerl, B., & Sousa, J. (2007). Leveraging Resource Prediction for Anticipatory Dynamic Configuration. In *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4274905 (pp. 214–223). IEEE.

Poladian, V., Sousa, J., Garlan, D., & Shaw, M. (2004). Dynamic configuration of resource-aware services. In *Proceedings. 26th International Conference on Software Engineering*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1317482 (pp. 604–613). IEEE Comput. Soc.

Pop, T., Plasil, F., Outly, M., Malohlava, M., & Bures, T. (2012). Property networks allowing oracle-based mode-change propagation in hierarchical components. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering - CBSE '12*. Accessed: 06/24/2014. http://dl.acm.org/citation.cfm?id=2304753 (p. 93). New York, New York, USA: ACM Press.

Ramamoorthy, C. V. (1966). The analytic design of a dynamic look ahead and program segmenting system for multiprogrammed computers. In *Proceedings of the 1966 21st national conference on -*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=810702 (pp. 229–239). New York, New York, USA: ACM Press.

Ramirez, A., & Cheng, B. (2012). Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. *Model Driven Engineering Languages and Systems*. Accessed: 01/27/2014. http://link.springer.com/chapter/10.1007/978-3-642-33666-9_5.

Ramirez, A. J., & Cheng, B. H. C. (2010). Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '10*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1808990 (pp. 49–58). New York, New York, USA: ACM Press.

Ramirez, A. J., Jensen, A. C., & Cheng, B. H. C. (2012). A taxonomy of uncertainty for dynamically adaptive systems. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6224396 (pp. 99–108). IEEE.

Ramirez, A. J., Jensen, A. C., Cheng, B. H. C., & Knoester, D. B. (2011). Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=2190127 (pp. 568–571). IEEE.

Ramirez, A. J., Knoester, D. B., Cheng, B. H. C., & McKinley, P. K. (2009). Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the 6th international conference on Autonomic computing - ICAC '09*. Accessed: 06/11/2014. http://dl.acm.org/citation.cfm?id=1555258 (p. 97). New York, New York, USA: ACM Press.

Ramirez, A. J., Knoester, D. B., Cheng, B. H. C., & McKinley, P. K. (2010). Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems. *Cluster Computing*. Accessed: 07/03/2014. http://link.springer.com/article/10.1007/s10586-010-0122-y, *14*(3), 229–244.

Roshandel, R., Medvidovic, N., & Golubchik, L. (2007). A Bayesian model for predicting reliability of software systems at the architectural level. *Software Architectures, Components, and Applications*. Accessed: 02/02/2014. http://link.springer.com/chapter/10.1007/978-3-540-77619-2_7.

Salay, R., Chechik, M., & Horkoff, J. (2012). Managing requirements uncertainty with partial models. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6345804 (pp. 1–10). IEEE.

Salehie, M., & Tahvildari, L. (2012). Towards a goal-driven approach to action selection in self-adaptive software. *Software: Practice and Experience*. Accessed: 06/10/2014. http://onlinelibrary.wiley.com/doi/10.1002/spe.1066/full, *42*(2), 211–233.

Salfner, F., Lenk, M., & Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys*. Accessed: 06/15/2014. http://dl.acm.org/citation.cfm?id=1670680, *42*(3), 1–42.

Sato, N., & Trivedi, K. (2007). Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks. *Service-Oriented Computing–ICSOC 2007*. Accessed: 02/11/2014. http://link.springer.com/chapter/10.1007/978-3-540-74974-5_9.

Sepúlveda, R., Castillo, O., Melin, P., Rodríguez-Díaz, A., & Montiel, O. (2007). Experimental study of intelligent controllers under uncertainty using type-1 and type-2 fuzzy logic. *Information Sciences*. Accessed: 07/03/2014. http://www.sciencedirect.com/science/article/pii/S002002550600332X, *177*(10), 2023–2048.

Serugendo, G., Fitzgerald, J., Romanovsky, A., & Guelfi, N. (2007). A generic framework for the engineering of self-adaptive and self-organising systems. In *Software Engineering for Self-Adaptive Systems*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007%2F978-3-642-02161-9_3.

Shen, L., Peng, X., Liu, J., & Zhao, W. (2011). Towards feature-oriented variability reconfiguration in dynamic software product lines. *Top Productivity through Software Reuse*. Accessed: 06/18/2014. http://link.springer.com/chapter/10.1007/978-3-642-21347-2_5.

Smith, C., & Williams, L. (2002). *Performance solutions: a practical guide to creating responsive, scalable software*. Accessed: 02/11/2014. http://ftp.cmg.org/proceedings/2001/1299.pdf. Addison-Wesley Professional.

Solano Martínez, J. (2012). *Energy management of a hybrid electric vehicle: an approach based on type-2 fuzzy logic*. http://www.banrepcultural.org/blaavirtual/tesis/colfuturo/energy-management-of-a-hybrid-electric-vehicle. University of Franche-Comté.

Solano Martínez, J., John, R. I., Hissel, D., & Péra, M.-C. (2012). A survey-based type-2 fuzzy logic system for energy management in hybrid electrical vehicles. *Information Sciences*. Accessed: 09/16/2014. http://www.sciencedirect.com/science/article/pii/S0020025511006529, *190*, 192–207.

Strelioff, C., Crutchfield, J., & Hübler, A. (2007). Inferring Markov chains: Bayesian estimation, model comparison, entropy rate, and out-of-class modeling. *Physical Review E*. Accessed: 07/03/2014. http://pre.aps.org/abstract/PRE/v76/i1/e011106, *76*(1), 011106.

Surajbali, B., Coulson, G., Greenwood, P., & Grace, P. (2007). Augmenting reflective middleware with an aspect orientation support layer. In *Proceedings of the 6th international workshop on Adaptive and reflective middleware held at the ACM/IFIP/USENIX International Middleware Conference - ARM '07*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1376781 (pp. 1–6). New York, New York, USA: ACM Press.

Sykes, D., Corapi, D., Magee, J., Kramer, J., Russo, A., & Inoue, K. (2013). Learning revised models for planning in adaptive systems. In *2013 35th International Conference on Software Engineering (ICSE)*. Accessed: 06/02/2014. http://dl.acm.org/citation.cfm?id=2486797 (pp. 63–71). IEEE.

Sykes, D., Heaven, W., Magee, J., & Kramer, J. (2007). Plan-directed architectural change for autonomous systems. In *Proceedings of the 2007 conference on Specification and verification of component-based systems 6th Joint Meeting of the European Conference on Software Engineering and the ACM SIGSOFT Symposium on the Foundations of Software Engineering - SAVCBS '07*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1292318 (pp. 15–21). New York, New York, USA: ACM Press.

Sykes, D., Heaven, W., Magee, J., & Kramer, J. (2008). From goals to components: a combined approach to self-management. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems - SEAMS '08*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1370020 (p. 1). New York, New York, USA: ACM Press.

Sykes, D., Heaven, W., Magee, J., & Kramer, J. (2010). Exploiting non-functional preferences in architectural adaptation for self-managed systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1774180 (p. 431). New York, New York, USA: ACM Press.

Tesauro, G. (2007). Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies. *IEEE Internet Computing*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4061117, *11*(1), 22–30.

Tourwé, T., Brichau, J., & Gybels, K. (2003). On the existence of the AOSD-evolution paradox. *SPLAT: Software Engineering Properties of Languages for Aspect Technologies*. Accessed: 06/18/2014. http://scholar.google.com/scholar?q=On+the+existence+of+the+AOSD+evolution+paradox&btnG=&hl=en&as_sdt=0%2C5#0.

Trinidad, P., Cortés, A., Peña, J., & Benavides, D. (2007). Mapping Feature Models onto Component Models to Build Dynamic Software Product Lines. *DSPL*. Accessed: 06/18/2014. http://www.lsi.us.es/~trinidad/docs/trinidad07-dspl.pdf.

Walpole, R., Myers, R., & Myers, S. (2011). *Probability and Statistics for Engineers and Scientists (9th Edition)*. *industrialventilation.net*. Accessed: 02/17/2014. http://www.amazon.com/Probability-Statistics-Engineers-Scientists-Edition/dp/0321629116.

Wang, L., & Mendel, J. (1992). Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*. Accessed: 06/16/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=199466, *22*(6), 1414–1427.

Wang, W.-L., Pan, D., & Chen, M.-H. (2006). Architecture-based software reliability modeling. *Journal of Systems and Software*. Accessed: 06/11/2014. http://www.sciencedirect.com/science/article/pii/S0164121205001421, *79*(1), 132–146.

Welsh, K. (2010). *Design-Time and Run-Time Requirements Modelling for Adaptive Systems*. Accessed: 06/14/2014. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.557291.

Welsh, K., Sawyer, P., & Bencomo, N. (2011a). Run-time resolution of uncertainty. In *2011 IEEE 19th International Requirements Engineering Conference*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6051673 (pp. 355–356). IEEE.

Welsh, K., Sawyer, P., & Bencomo, N. (2011b). Towards requirements aware systems: Run-time resolution of design-time assumptions. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6100125 (pp. 560–563). IEEE.

Weyns, D., & Ahmad, T. (2013). Claims and evidence for architecture-based self-adaptation: a systematic literature review. *Software Architecture*. Accessed: 06/18/2014. http://link.springer.com/chapter/10.1007/978-3-642-39031-9_22.

Weyns, D., Iftikhar, M. U., Malek, S., & Andersson, J. (2012). Claims and supporting evidence for self-adaptive systems: A literature study. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Accessed: 06/10/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6224395 (pp. 89–98). IEEE.

Weyns, D., Malek, S., & Andersson, J. (2010). FORMS: a formal reference model for self-adaptation. In *Proceeding of the 7th international conference on Autonomic computing - ICAC '10*. Accessed: 06/16/2014. http://dl.acm.org/citation.cfm?id=1809078 (p. 205). New York, New York, USA: ACM Press.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Bruel, J.-M. (2009). RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *2009 17th IEEE International Requirements Engineering Conference*. Accessed: 06/30/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5328591 (pp. 79–88). IEEE.

Wilder, B. (2012). *Cloud Architecture Patterns*. Oreilly.

Witten, I., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques* (2 edition.). Morgan Kaufmann.

Wolfinger, R., Reiter, S., Dhungana, D., Grunbacher, P., & Prahofer, H. (2008). Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques. In *Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*. Accessed: 05/30/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4464006 (pp. 21–30). IEEE.

Woodside, C. M., & Litoiu, M. (2008). Performance Model Estimation and Tracking Using Optimal Filters. *IEEE Transactions on Software Engineering*. Accessed: 06/21/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4515874, *34*(3), 391–406.

Wu, D. (2012). On the Fundamental Differences Between Interval Type-2 and Type-1 Fuzzy Logic Controllers. *IEEE Transactions on Fuzzy Systems*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6145645, *20*(5), 832–848.

Wu, D., & Mendel, J. M. (2007). Uncertainty measures for interval type-2 fuzzy sets. *Information Sciences*. Accessed: 07/03/2014. http://www.sciencedirect.com/science/article/pii/S002002550700357X, *177*(23), 5378–5393.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., & Nuseibeh, B. (2012). Speculative requirements: Automatic detection of uncertainty in natural language requirements. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6345795 (pp. 11–20). IEEE.

Yang, Q.-L., Lv, J., Tao, X.-P., Ma, X.-X., Xing, J.-C., & Song, W. (2013). Fuzzy Self-Adaptation of Mission-Critical Software Under Uncertainty. *Journal of Computer Science and Technology*. Accessed: 07/03/2014. http://link.springer.com/article/10.1007/s11390-013-1321-9, *28*(1), 165–187.

Yen, J., & Tiao, W. (1997). A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=566845 (pp. 87–96). IEEE Comput. Soc. Press.

Yin, H., Carlson, J., & Hansson, H. (2012). Towards mode switch handling in component-based multi-mode systems. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering - CBSE '12*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=2304766 (p. 183). New York, New York, USA: ACM Press.

Yu, Y., Lapouchnian, A., & Liaskos, S. (2008). From goals to high-variability software design. *Foundations of Intelligent Systems*. Accessed: 02/12/2014. http://link.springer.com/chapter/10.1007/978-3-540-68123-6_1.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*. Accessed: 05/26/2014. http://www.sciencedirect.com/science/article/pii/S001999586590241X, *8*(3), 338–353.

Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning—I. *Information Sciences*. Accessed: 06/19/2014. http://www.sciencedirect.com/science/article/pii/0020025575900365, *8*(3), 199–249.

Zave, P., & Jackson, M. (1997). Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=237434, *6*(1), 1–30.

Zhao, D. (2011). Supervised adaptive dynamic programming based adaptive cruise control. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. Accessed: 07/03/2014. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5967371 (pp. 318–323). IEEE.

Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., & Shin, K. (2009). What does control theory bring to systems research? *ACM SIGOPS Operating Systems Review*. Accessed: 07/03/2014. http://dl.acm.org/citation.cfm?id=1496922, *43*(1), 62.

Appendix A. The Survey Template

Section I. Contact details

Name ____

Affiliation ____

Email ____

Section II. Expert knowledge

Section II aim to extract expert knowledge to design the adaptation management system for the ElasticQueue shown in Figure 8.4.

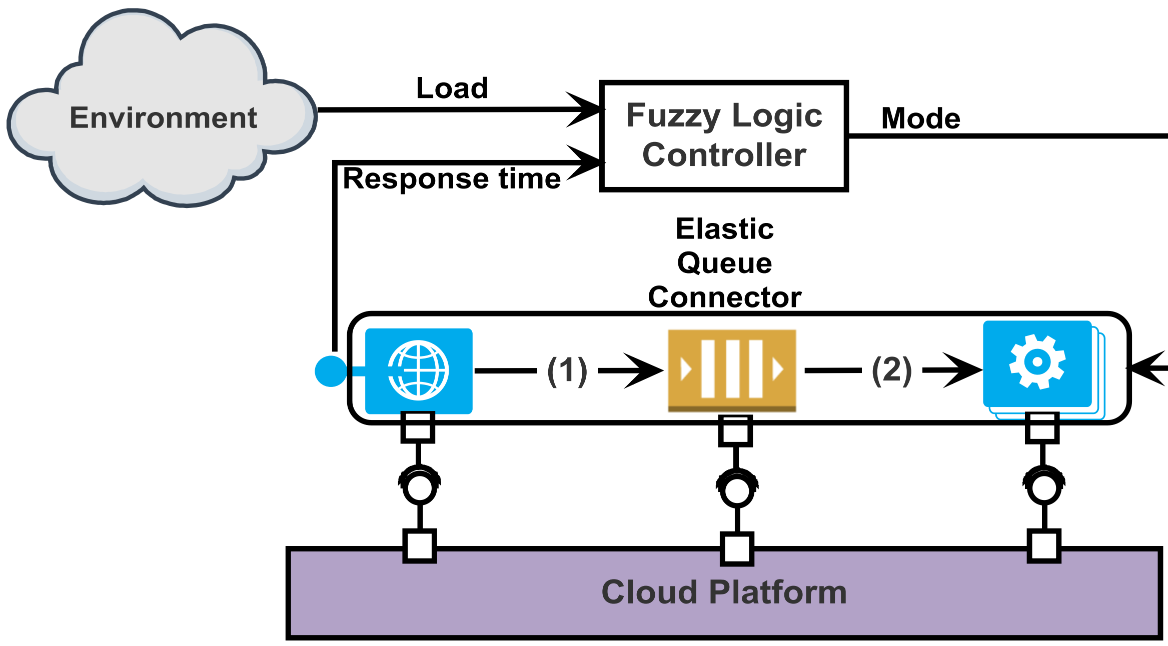


Figure 8.4. ElasticQueue adaptation management with a fuzzy controller.

The objectives of the adaptation management for this ElasticQueue are:

1. Tasks are put into the queue for processing
2. The processing components pick up the tasks from the queue to process them
3. In high loads, the fuzzy controller regulates the number of processing components to meet the response time SLA
4. In low load, the fuzzy controller regulates the number of processing components to decrease the incurred costs

A fuzzy controller is used to determine the operating mode of the ElasticQueue to reach the objectives. Figure 8.4 presents the fuzzy logic controller, and Table 1 explains its inputs and outputs.

Table 8.1. FLS input–output description.

Input/output	Description
Load	Number of requests received at the ElasticQueue end point
Response time	The difference in time that a task is received at the ElasticQueue end point and the time that it has been processed by the processing component
Mode	The operating mode of the ElasticQueue

Section II.1 Linguistic label localization

The purpose of this part of the survey is to locate linguistic labels to determine intervals. This information will be used to construct the fuzzy sets and associated membership functions. The following example demonstrates how to complete this section.

Example 1:

Table 2 summarizes the assigned values to four linguistic labels representing the load to the ElasticQueue with maximum normalized value of 100 (concurrent requests/sec). Obviously these values could be different in the opinion of different people.

Table 8.2. Linguistic labels to describe workload.

Workload	Start ($\boldsymbol{a}$)	End ($\boldsymbol{b}$)
Very low	0	20
Low	15	40
Medium	35	65
High	60	90
Very high	85	100

It can be inferred from Table 8.2 that the person completing this table thinks that:

- A low workload is between 15 and 40 (requests/sec)
- A load of 40 (requests/sec) could be considered low as well as medium

Question 1. Load to the ElasticQueue

Please use your own experience, thoughts and preferences to complete Table 8.3 using values of the Load as defined in Table 8.1.

Table 8.3. Linguistic labels to describe Load.

Workload	Start ($\boldsymbol{a}$)	End ($\boldsymbol{b}$)
Very low	0	
Low		
Medium		
High		
Very high		100

Question 2. Response time of the ElasticQueue

Please use your own experience, thoughts and preferences to complete Table 8.4 using values of the Response time as defined in Table 8.1.

Table 8.4. Linguistic labels to describe Response time.

Response time	Start ($\boldsymbol{a}$)	End ($\boldsymbol{b}$)
Instantaneous	0	
Fast		
Medium		
Slow		
Very slow		100

Section II.2 Rules definition

The FLS presented in the upper part of Figure 8.4 and described in Table 8.1 takes the inputs and processes them to produce outputs using the fuzzy rules and the linguistic labels in Table 7.1. These rules are summarized in Table 8.6.

The following examples demonstrate how to complete this section. However, they are only examples. Please feel free to modify your answers.

Example 2: If Load is “High” and Response time is “Slow” then the mode of the ElasticQueue is ______________?

- The ElasticQueue has less processing components than is needed and the queue is relatively overloaded.
- It is necessary to rapidly increase the computing powers (i.e., the processing components) of the ElasticQueue

If Load is “High” and Response time is “Slow” then the mode of the ElasticQueue is “High-Effort”.

Example 3: If Load is “Medium” and Response time is “Fast” then the mode of the ElasticQueue is ______________?

- The ElasticQueue meet the SLA
- It is necessary to hold the output power in the ElasticQueue

If Load is “Medium” and Response time is “Fast” then the mode of the ElasticQueue is “Effort”.

Question 3. Fuzzy rules definition

Please use your own experience, thoughts and preferences to complete Table 8.6 using the linguistic labels in Table 7.1.

Table 8.5. Linguistic labels to describe ElasticQueue operating mode.

ElasticQueue Mode	Interface Component	Processing Components
Normal	1	1
Effort	1	2
Medium Effort	1	3
High Effort	1	4
Maximum Effort	1	5

The following examples demonstrate how to complete this section. However, they are only examples. Please feel free to modify your answers.

Example 2: If Load is “High” and Response time is “Slow” then the mode of the ElasticQueue is _____________?

- The ElasticQueue has less processing components than is needed and the queue is relatively overloaded.
- It is necessary to rapidly increase the computing powers (i.e., the processing components) of the ElasticQueue

If Load is “High” and Response time is “Slow” then the mode of the ElasticQueue is “High-Effort”.

Example 3: If Load is “Medium” and Response time is “Fast” then the mode of the ElasticQueue is _____________?

- The ElasticQueue meet the SLA
- It is necessary to hold the output power in the ElasticQueue

If Load is “Medium” and Response time is “Fast” then the mode of the ElasticQueue is “Effort”.

Question 3. Fuzzy rules definition

Please use your own experience, thoughts and preferences to complete Table 8.6 using the linguistic labels in Table 7.1.

Table 8.6. Fuzzy adaptation rules.

Rule (l)	**Antecedents**		**Consequent**				
	Workload	**Response time**	**Normal**	**Effort**	**Medium Effort**	**High Effort**	**Maximum Effort**
1	Very low	Instantaneous					
2	Very low	Fast					
3	Very low	Medium					
4	Very low	Slow					
5	Very low	Very slow					
6	Low	Instantaneous					
7	Low	Fast					
8	Low	Medium					
9	Low	Slow					
10	Low	Very slow					
11	Medium	Instantaneous					
12	Medium	Fast					
13	Medium	Medium					
14	Medium	Slow					
15	Medium	Very slow					
16	High	Instantaneous					
17	High	Fast					
18	High	Medium					
19	High	Slow					
20	High	Very slow					
21	Very high	Instantaneous					
22	Very high	Fast					
23	Very high	Medium					
24	Very high	Slow					
25	Very high	Very slow					