Architecture Optimisation of Embedded Systems under Uncertainty in Probabilistic Reliability Evaluation Model Parameters

Indika Meedeniya

Submitted in fulfilment of the requirements of the degree of

Doctor of Philosophy

Faculty of Information and Communication Technologies

Swinburne University of Technology

July 2012

Abstract

Software plays a vital role in most of the embedded systems including safety and mission-critical systems in avionics, automotive, nuclear and medical applications. Along with the functional complexity of software, the quality of software-intensive systems has become a crucial concern. Numerous techniques are being developed to evaluate the quality of a software system from its architecture in terms of quality attributes such as reliability, safety and performance, and to automate the search for alternative designs which provides good trade-offs with respect to those quality attributes of interest. However, the results of these quantitative architecture evaluations depend on design-time estimates for a series of model parameters, which may not be accurate and can change at run-time. Conventional approaches use numerical values (point estimates) as design-time estimates, where the uncertainty in the parameter estimation is not part of the evaluation. As a result, architecture-based quality evaluations at design-time can be inaccurate and thus, sub-optimal design decisions may be taken.

To overcome this problem, this thesis presents a novel design-time architecture evaluation and optimisation approach that incorporates parameter uncertainties. The work specifically focuses on architecture-based reliability evaluation models, where a number of parameters have to be estimated subject to heterogeneous uncertain factors. Instead of using point-estimates for architecture-based reliability evaluation models, this work proposes to incorporate heterogeneous and diverse uncertainty information into the reliability evaluation and architecture optimisation. A framework is devised which can capture uncertainty information associated with parameters and use them for the search for robust and optimal candidate architectures. This approach is able to find good architecture solutions that can tolerate the impact of the uncertainties, and thus provides better decision support. The accuracy and scalability of the presented approach is validated with an industrial case study and a series of experiments with generated examples in different problem sizes and characteristics.

Acknowledgements

First and foremost, my sincere thanks to my initial supervisor, Dr. Lars Grunske who agreed to support my candidature and priceless guidance, especially at the very first stage of this research. I am indebted to Lars for his generous mentoring and many hours spent discussing the topics, reviewing of papers and thesis drafts. The confidence inspired by this regular feedback, interesting discussions and networking made me an enthusiastic PhD candidate.

My warm thanks to Dr. Irene Moser who organised everything to continue my candidature when Lars left Swinburne, and specially for ample help she has given to improve the quality of this thesis. I offer my heartfelt thanks to Prof. Jun Han and Dr. Anthony Tang for their support and guidance throughout the PhD candidature. I am thankful to my fellow research student and friend Aldeida Aleti for her constructive critiques of my work, many interesting discussions and collaborative work. I acknowledge the positive influence to this research from Dr. Barbora Buhnova from her visit to Swinburne and through several collaborative research projects. My thanks also go to fellow PhD students Nargiza, Iman, Ayman, Tharindu, IndikaW, Kaw and Mark for making a friendly and enjoyable research student life.

Many thanks to my mother and two sisters for their continuous encouragement, love and care. Finally, I would like to express my profound gratitude to my wife Thivanka, who always gave her fullest support, care and encouragement to make this research a success.

This project would not have been possible without the scholarships from Cooperative Research Center for Advanced Automotive Technology (AutoCRC, project C4-501: Safe and Reliable Integration and Deployment Architectures for Automotive Software Systems), and Swinburne University of Technology.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

_____

Indika Meedeniya

_____

Date

List of Publications

Publications arising from and included in the thesis

Chapter 3

- Indika Meedeniya, Barbora Bühnová, Aldeida Aleti, and Lars Grunske. Reliability-Driven Deployment Optimization for Embedded Systems. *Journal of Systems and Software (JSS)*, Volume 84(5), Pages 835-846, 2011.

- Indika Meedeniya, Barbora Bühnová, Aldeida Aleti, and Lars Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *International Conference on the Quality of Software Architectures, (QoSA 2010)*, Prague, Czech Republic, June 23 - 25, 2010, volume 6093 of LNCS, Pages 52-67. Springer, 2010.

- Indika Meedeniya, Aldeida Aleti, and Barbora Zimmerová. Redundancy Allocation in Automotive Systems using Multi-objective Optimisation. In *Symposium on Automotive/Avionics Systems Engineering (SAASE 2009)*, San Diego, CA, USA, October 13-16 2009.

Chapter 5

- Indika Meedeniya, Irene Moser, Aldeida Aleti and Lars Grunske. Software Architecture Evaluation under Uncertainty. In *International Conference on the Quality of Software Architectures, (QoSA 2011)*, Boulder, CO, USA, June 20-24 2011, Pages 85-94. ACM. 2011.

  ACM Distinguished Paper Award

Chapter 6

- Indika Meedeniya, Aldeida Aleti and Lars Grunske, Robust Reliability Optimization of Software Architectures with Probabilistic Quality Evaluation Models. *Journal of Systems and Software (JSS)*, Volume 84(10), Pages 2340-2355, 2012.

- Indika Meedeniya, Aldeida Aleti, Iman Avazpour and Ayman Amin. Robust ArcheOpterix: Architecture Optimization of Embedded Systems under Uncertainty. In *ICSE 2012 Workshop on Software Engineering for Embedded Systems (SEES 2012)*, Zürich, Switzerland, June 2-9, 2012. Pages 23-29. IEEE Computer Society 1212.

Chapter 7

- Aldeida Aleti, Stefan Björnander, Lars Grunske, and Indika Meedeniya. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *ICSE 2009 Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2009)*, May 16, 2009, Vancouver, Canada. Pages 61-71. IEEE Computer Society 2009.

Publications arising from but not included in the thesis

- Indika Meedeniya. An incremental methodology for quantitative software architecture evaluation with probabilistic models. In *ACM/IEEE International Conference on Software Engineering (ICSE 2010 Doctoral Symposium)- Volume 2*, Cape Town, South Africa, 1-8 May 2010. Pages 339-340. ACM. 2010.

- Indika Meedeniya and Lars Grunske. An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In *International Symposium on Software Reliability Engineering, (ISSRE 2010)*, San Jose, CA, USA, November 1-4, 2010. Pages 229-238. IEEE Computer Society. 2010.

- Markus Lumpe, Indika Meedeniya and Lars Grunske. PSPWizard: machine-assisted definition of temporal logical properties with specification patterns. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2011)*, Szeged, Hungary, September 5-9 2011. Pages 468-471. ACM. 2011.

- Aldeida Aleti and Indika Meedeniya. Component Deployment Optimisation with Bayesian Learning. In *International ACM Sigsoft Symposium on Component Based Software Engineering, (CBSE 2011)*, Boulder, CO, USA, June 20-24 2011. Pages 11-20. ACM. 2011.

- Aldeida Aleti, Lars Grunske, Indika Meedeniya, and Irene Moser. Let the ants deploy your software - A hybrid ACO/GA based deployment optimisation strategy, In *IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, Auckland, New Zealand, November 16-20, 2009. Pages 505-509. IEEE Computer Society. 2009.

Papers under revision

- Indika Meedeniya, Irene Moser, Aldeida Aleti and Lars Grunske, Evaluating Probabilistic Models under Uncertainty. *Software and Systems Modeling (SoSyM)* – under revision.

- Aldeida Aleti, Barbora Bühnová, Anne Koziolek, Lars Grunske, and Indika Meedeniya, A Systematic Survey on Software Architecture Optimization Methods. *IEEE Transactions on Software Engineering (TSE)* – under revision.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software has conquered most man-made systems in today's world, including safety- and mission-critical systems in avionics, automotive, nuclear plant control and medical devices. Many of the important functions of those systems are implemented or controlled by software which takes the lion's share of the growing functional complexity of today's embedded systems [69, 134, 205, 364]. Consequently, in addition to the functional correctness of the system, the quality of the underlying software has become a concern of paramount importance.

As in many other engineering disciplines, models are increasingly used in software engineering projects to predict quality characteristics of software-intensive systems before they are built [41]. A number of evaluation models have been proposed for specific quality attributes such as performance [36], reliability [182] and safety [191]. The benefit of these evaluation models is especially evident in the architectural design phase, since different design alternatives can be evaluated and software architects can make informed decisions between these alternatives. This decision support, as research and practical evidence show [41, 283], has a significant impact on the quality of the developed system. The ability of constructing quality evaluation models (*e.g.* queuing networks, Markov models or fault trees) based on an annotated architecture specification has provided the capability to automate design space explorations. Finding a suitable architecture design has been formulated as a multi-

objective optimisation problem where the goal is to find architectures with better trade-offs in the quality attributes. Many approaches (*e.g.* [97, 187, 279, 283, 317]) have been developed that apply different optimisation strategies to identify optimal or near-optimal solutions for specific architectural problems.

Gap Identification

In the context of software architecture design, any automated design space exploration is only as good as the accuracy of the results of the model-based quality evaluations, and they rely on the accuracy of parameters that are used in the quality evaluation models. These parameters can be grouped into system-specific and environment-related parameters. The first category includes parameters which are related to the system elements, for instance hardware and software failure rates or throughput metrics. The second category contains parameters that define the operational and usage profile of the systems, and examples for these parameters are call rates of specific services or data sizes that have to be accommodated by the system. These environment parameters also influence the system-specific parameters, such as branching probabilities in the software executions. An accurate determination of some of the parameters is generally hard to achieve, for instance failure rates of software components [85, 181] are hard to determine. Other parameters require information that is only available in later stages of the development project or that depend on the specific usage of the system, and thus are only available during the system run-time. Hence, the values of these parameters used in design-time architecture optimisation are estimates and subject to uncertainty.

In the domain of design-time software architecture optimisation, current approaches commonly use point-estimated parameters for architecture-based quality evaluation models. Model parameters are assumed to be given as distinct numerical values (point estimates) regardless of the characteristics and extent of uncertainty associated with them. The numerical values are then used for quality evaluation and to search for better architecture decisions. However, due to the fact that these

parameters can only be estimated, architecture optimisation approaches may lead to suboptimal and misleading solutions if the estimations are inaccurate.

An emerging direction to overcome the problem of inaccurate design-time parameter estimates is to introduce uncertainty awareness into the system's run-time. Epifani *et al.* [139] and Zheng *et al.* [421] have proposed techniques that monitor the system at run-time and use Bayesian estimators or Kalman filters to refine specific model parameters. Consequently, more realistic quality evaluations are possible, if the models are re-evaluated at run-time. Although this may help trigger restricted optimisations at run-time, it does not solve the general problem since many of the critical design decisions have to be made before the system is built, and they are hard to change at run-time.

Another related branch of research focuses on *sensitivity analysis* which investigates the relationship between the model parameters and architecture-based quality evaluation. Sensitivity graphs are constructed to analyse the response characteristics of a quality attribute of interest with respect to possible variations of one or more parameters. It is a common practice to use parameter sweep techniques to obtain sensitivity graphs. The sensitivity analysis techniques can help analyse the effects of parameter uncertainties on specific quality attributes in detail. However, they are only applicable to a given architecture specification, and do not lend themselves to solving the problem of architecture optimisation under uncertainty.

Complementary to the emerging research work in the domain of software engineering, there exist some uncertainty-aware optimisation approaches in other engineering disciplines such as mechanical engineering [132] and antenna design [127]. Several techniques have been developed to optimise design decisions in order to tolerate certain variations in the system model parameters, which are often called *robust optimisation methods* [46, 55]. A common practice in the robust optimisation is to formulate the desired quality of the system as a mathematical function of uncertain input parameters and design decisions, and then use analytical methods to derive robust design decisions based on the input parameter variables. However,

these concepts are not directly portable to software architecture optimisation due to the significant differences in the problem setting such as mathematical complexity of architecture-based quality evaluation models and heterogeneity of software architecture design decision variables.

Research Problem

This work focuses on the impact of uncertainty in design-time parameter estimates on software architecture optimisation. The work presented in thesis is specifically focused on architecture-based reliability evaluation which is one of the crucial quality attributes in the design of embedded software. Overall, this thesis pursues an architecture optimisation approach that can cater for uncertainty in the input parameters, addressing a set of research issues that arise from the areas of architecture-based reliability evaluation, uncertainty in model parameter estimation and architecture optimisation. In accomplishing this research challenge, this work recognises the impact of estimated parameters on the solutions produced by architecture optimisation as well as the heterogeneous and diverse characteristics of uncertain information associated with parameter estimation. Hence, the initial task is to identify and capture the diverse sources of uncertainty in parameter estimation. Secondly, the prevalent architecture-based quality evaluation models have to be evaluated under uncertain model parameters. The third challenge addressed in this research is the optimisation aspect of the overall problem. In the context of embedded systems design, architecture optimisation has many aspects to consider such as various architecture decisions, required level of tolerance for uncertainty, design constraints, and probabilistic models that are used to quantify the qualities of interest from the candidate architecture solutions. Therefore, to pursue a sustainable solution for architecture optimisation under uncertainty, a framework has to be formulated that can capture those diverse characteristics of the problem. It is further necessary to investigate the adaptation of prevalent optimisation techniques in order to achieve the overall goal of architecture optimisation in the face of uncertainty in the parameter estimates.

Scope of the Research

Software architecture evaluation and optimisation extend over a very large context, where specific sub-research disciplines exist within the main stream of software engineering research. This work acknowledges that addressing the aforementioned research issues in that broader context with a sound validation requires a collective effort across the research communities. Hence, the extent of this research is restricted to a scope suitable for a PhD thesis.

The thesis considers the software architecture design problem with a focus on embedded systems design. Architecture design issues that are applicable to safety- and mission-critical embedded systems are taken into consideration. The scope of design decisions is bounded to cover a set of key choices which have to be made at the architecture design of such systems. This work primarily considers the following design decisions.

- Software/hardware component selection
- Software/hardware redundancy allocation
- Component deployment
- Software/hardware parameters

With regards to the quality attributes, this work specifically focuses on architecture-based reliability evaluation. This research also recognises the multi-objective nature of the architecture optimisation problem. Multiple reliability-related attributes (*e.g.* failure probability of individual services, mean time to failure of a system) are considered, and reliability evaluation models are used throughout the thesis as the probabilistic quality models. Detailed analyses of other probabilistic quality attributes and their evaluation models are considered out of the scope of this work.

This work considers only the uncertainties associated with parameters of the quality evaluation models (*i.e.* type A uncertainties [55]). The *robustness* of the architectures is defined as the ability to tolerate uncertainties in the parameter estimation.

From the architecture optimisation perspective, this work focuses on black-box stochastic approaches rather than exact or approximate algorithms tailored to specific architecture design problem or situation. The rationale behind this decision is that although stochastic algorithms do not guarantee the quality of the results they provide, they have proven successful on large combinatorial optimisation problems applied to the software architecture design context where application of exact algorithms is not practical [61,97,304]. Also, as black-box optimisation algorithms, they are generally applicable to a variety of architecture optimisation problem types.

Research Method and Approach

The primary aim of this research is to provide an extended decision support to architecture optimisation while considering parameter estimation uncertainties, which is an existing gap given current reliance on point-estimate-based methods. The baseline of the research is established with an analysis of existing work relevant to various dimensions of the problem. The specific research gaps are further emphasised by analysing the issues in applying a state-of-the-art architecture optimisation method to an industrial case study. Following a clear identification of the remaining challenges, this thesis takes a bottom-up engineering approach towards a solution to the research problem.

In this work, the overall research problem is decomposed into fine-grained research questions, and solutions are proposed to each of them which eventually lead to an overall solution. The partial solutions are analysed within the scope of the research, and evidence is collected to support the validity of the individual contributions. Experimental evaluation and analytical derivation methods are used as validation strategies in respective contributions. With the support of individual validations, dedicated evidence is collected on the overall solution. The new approach is fully is implemented, and two types of empirical evidence is used to support the overall validity.

(a) The proposed solution is applied to an industrial case study, and new approach's

ability to cater for actual conditions is illustrated. The results of the new method are compared with the solutions obtained from state-of-the-art method illustrating the extended decision support. These experiments establish a *proof of concept* as well as a practical application of the approach.

(b) Empirical evidence is gathered through large numbers of experiments. A series of problem instances are generated from an abstract formulation of the problem model, preserving realistic behaviours in order to cover diverse problem types and sizes as well as different characteristics included in the scope of the thesis. The overall validity is justified with a quantitative analysis of the results collected from the experiments.

Contributions

This thesis addresses the problem of inaccurate estimates for parameters of reliability evaluation models in the context of design-time architecture optimisation by proposing an optimisation approach that incorporates uncertainties. The new approach which I abbreviate as SCOUT (Software arChitecture Optimisation Under uncerTainty), extends the current practice of using point estimates for the model parameters in architecture optimisation [6, 7, 97, 187, 279, 283, 317] by allowing to include uncertainty information available at the early design phase, and combine with optimisation techniques to find better architectures that can tolerate uncertainties.

In summary, the SCOUT approach presented in this thesis contains following novel contributions.

- Complementary to using point estimates, the SCOUT approach incorporates diverse characteristics of uncertain information related to parameter estimation into the parameter specification of the architecture optimisation problem. A generalised probabilistic specification is proposed for the purpose of capturing the extent and characteristics of uncertain parameters. The conventional assumptions on input parameter distributions are relaxed to facilitate heterogeneous parameters in software architecture design.

- An automated and scalable model evaluation technique is introduced which can be used to evaluate complex probabilistic quality models in the presence of uncertain model parameters. A Monte Carlo (MC) simulation technique is adopted to uncertainty analysis to cater for probabilistic quality evaluation models which are hard to solve analytically for variable input parameters. A novel dynamic stopping criterion is introduced that automatically finds the trade-off between time complexity and accuracy of MC simulation. Principles of sequential statistical testing are combined with continuous accuracy monitoring mechanism in MC simulations.
- The SCOUT approach presents a robust optimisation framework which can be used to model and optimise various architecture decisions involved in an embedded systems design. The framework contains a model that integrates architectural elements, quality objectives, design constraints, probabilistic quality evaluation models and optimisation algorithms. Different stochastic algorithms can easily be plugged into the framework for evaluation-expensive, multi-objective optimisation problems. The thesis includes the adaptation of three algorithms, namely Non-dominated Sorting Genetic Algorithm (NSGA-II), Ant Colony Optimisation and Simulated Annealing. The approach also provides the support for optimising architectures with a desired level of tolerance to uncertainty parameter estimation, which can be required in different application domains.

Thesis Outline

The organisation of the thesis is depicted in Figure 1.1.

Chapter 2 provides a foundation for the rest of the thesis, starting with a description of basic concepts in software architecture and embedded systems followed by a second section with a deeper discussion of architecture-based reliability evaluation. The third part of Chapter 2 summarises prevalent techniques and their features in relation to architecture optimisation. Following the background, Chapter 3 presents a description of a case study from the automotive industry, which

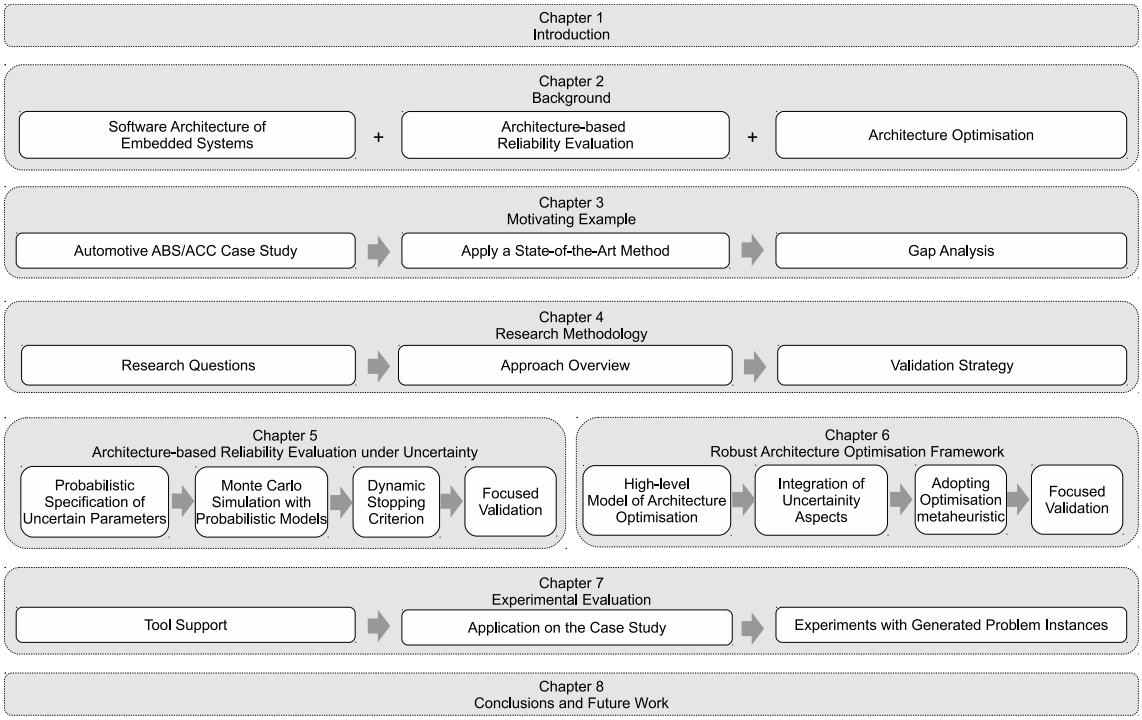


Figure 1.1: Thesis outline

is used as a motivation as well as for the illustration of concepts throughout the thesis. The latter part of the chapter motivates the need for robust optimisation and analyses the gap with respect to the background given in Chapter 2. Chapter 4 articulates the research questions addressed in the thesis followed by a description of the research method and approach taken in this research. The new contributions on the inclusion of uncertain information into input parameter specification as well as the architecture-based quality evaluation considering the uncertain inputs are presented in Chapter 5. Addressing the modelling and optimisation aspects of the research problem, Chapter 6 combines the contributions into the novel framework for robust architecture optimisation. Chapter 5 and Chapter 6 contain more fine-grained related work on specific aspects and evidence of the validity of individual contributions. In Chapter 7, the presentation starts with a brief description of the implementation of the approach and its application to the industrial case study. As a proof of concept, the extended decision support is illustrated on the case study and compared with the results obtained from the conventional approaches. The final

part of Chapter 7 presents a series of experiments on generated problem instances that covers various dimensions of the approach in addressing the identified research problem. Chapter 8 concludes the thesis and discusses on directions for future work.

Chapter 2

Background

Numerous techniques and industrial practices are being developed to enhance the engineering of software-intensive systems. The research problem addressed in this thesis intersects several technical areas where each contains a large body of research on its own. Therefore, this chapter is presented as a source of background information as well as the preliminary literature review establishing a foundation for the contents presented in the remainder of the thesis. The chapter is organised into three distinct blocks. The first part of the chapter provides a brief introduction to the concepts that relate to software architecture, design practices used in embedded systems and architecture-based quality evaluation. Secondly, as this work has specifically scoped down the focus on quality attributes to reliability, a thorough analysis of techniques developed for architecture-based reliability evaluation is presented. This section describes different reliability models and their evaluation mechanisms, which are also used as a basis for the arguments formulated in the following chapters. The third part of the chapter describes aspects and techniques related to the architecture optimisation of software-intensive embedded systems, along with a summary of related approaches.

2.1 Basic Concepts

This section defines the terms and concepts used in relation to the software-architecture in practice, establishing a vocabulary for the rest of the presentation. The research presented in the thesis channels into the field of software-intensive systems, which is defined as follows.

► **Software-intensive systems**. *"any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole" to encompass "individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest"* [218].

The complexity of such systems has grown enormously during the past decades, and it has become more challenging to create, organise and utilise those systems. Hence, concepts, principles and procedures are being developed and applied in practice to manage the increasing complexity of software-intensive systems. The common terms of *architecture* and *architecting* are defined in the ISO standard as follows.

► **Architecture (of a system)**. *"fundamental concepts or properties of a system in its environment embodied in its elements, relationships,and in the principles of its design and evolution"* [219]

► **Architecting**. *"process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system's life cycle"* [219]

Significant efforts have been made to define conceptual entities and their interrelationships systematically in the context of software architecture [41, 218, 219, 240]. Figure 2.1 depicts the conceptual diagram, extracted from the ISO standard 42010 [219].

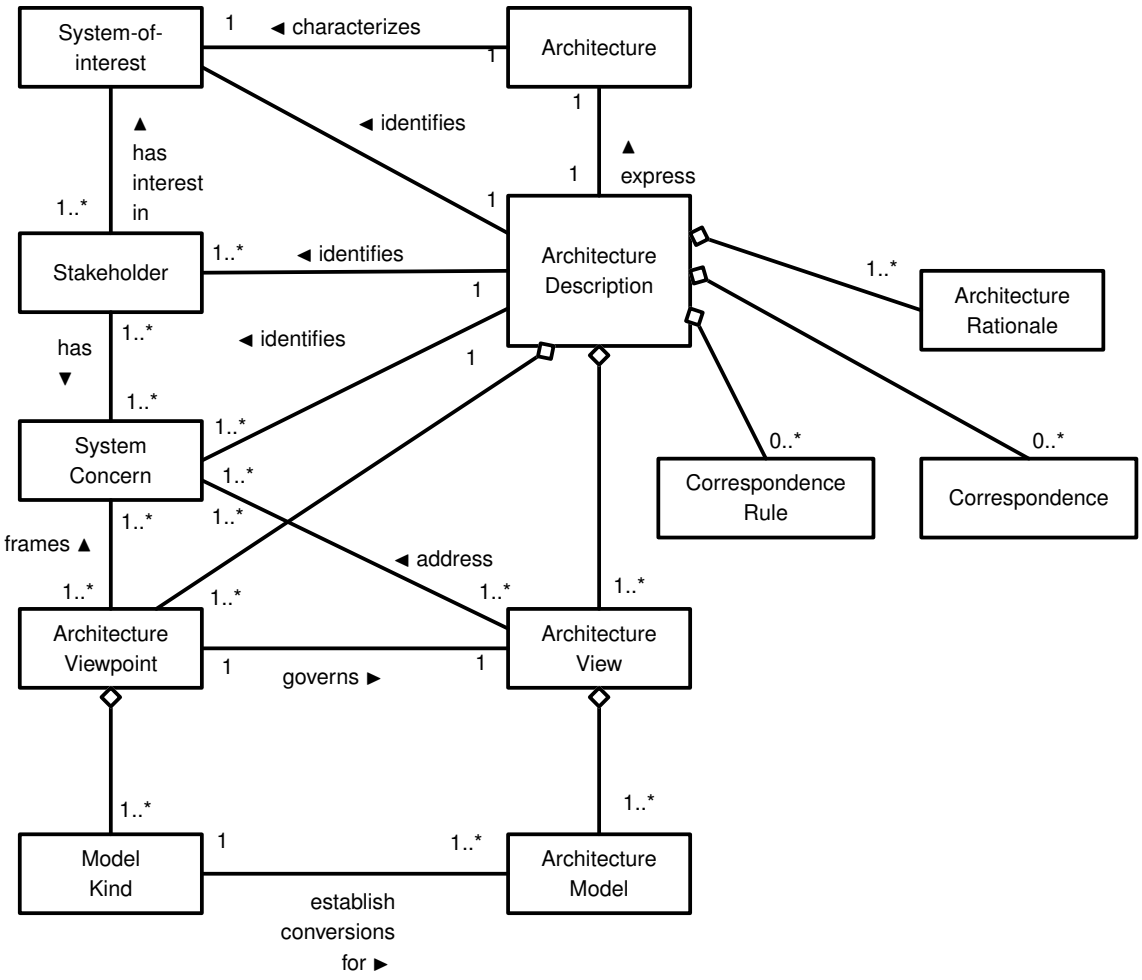


Figure 2.1: ISO/IEC 42010 conceptual model of architecture description

2.1.1 Architecture Views and Viewpoints

Architecture comprises multiple views (perspectives) of a system which enable the exchange of information about a system or system of systems. A description of an architecture can include one ore more *views* of the architecture, and the views are governed by *viewpoint*s. Kruchten [240] presented software architecture as a composite of the four views (logical, process, development and physical) together with the use cases/scenarios as the fifth view. According to the recent ISO 42010 standard [219], the concepts of architecture views and viewpoints are defined in much broader sense, where Kruchten's views constitute a subset.

► **Architecture View**. *"work product expressing the architecture of a system from*

the perspective of specific system concerns" [219]

► **Architecture Viewpoint**. *"work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns"* [219]

In relation to software-intensive systems, these views are subsumed in the hardware and software components, their relationships to each other and the environment, and the principles governing its design and evolution [389]. The architecture of an embedded system encompasses the structure of the system that contains software and hardware components, their properties and relationships. Consequently, the architecture is a vital piece of information at the design stage which helps to decompose a complex system into smaller analysable units and connections among them.

2.1.2 Stakeholders of Architecture

Many parties are interested in, and have an influence on the architecture decisions. In the context of embedded systems, the stakeholders are the end users, the developers, the project managers, the maintainers *etc.* [41], who have different interests regarding the prospective system. For example, end users may be interested in higher performance, reliability or security while project managers are interested in lowering development costs. These interests are formally called *quality attributes*. The role of a *software architect* is to design the system architecture considering those interests.

► **Architect**. *"The person, team, or organisation responsible for systems architecture"* [218].

2.1.3 Software Architecture vs. System Architecture

As presented before, it is apparent that the software architecture subsumes diverse aspects of the system. Different aspects such as the nature of the underlying hard-

ware or the behaviour of the system are manifested in different views of the architecture. Therefore, designs of a software architecture have to consider the entire system, including both hardware and software.

In most practical situations, hardware design is carried out separately [41]. Even in the development of completely new systems, lower level hardware decisions are based on hardware manufacturers and driven by electronic/electrical systems engineers. This does not mean that the software architect does not have any influence on the hardware architecture, but the nature of the decisions that a software architect can make in the hardware design is limited. The architect's freedom of choice in hardware is considered at a higher level, such as the decision on hardware selection and configuration rather than on the gate or transistor level. In this thesis, the focus is on the decisions related to the software architecture (*e.g.* software/hardware component selection, software/hardware redundancy allocation, component deployment) as opposed to the much broader scope covered in the term *system architecture*.

2.1.4 Architecture Description

A formal description of an architecture provides vital benefits at the system's design stage as well as later in the development phase. Architecture descriptions have facilitated the use of computer-aided analysis and development tools including automatic code generation, testing, and model checking. Hence, substantial research efforts from both academia and industry are being dedicated to architecture description, as manifested in ISO/IEC 42010 standard [219]. Several architecture description languages (ADLs) have been developed to represent and capture different aspects of software architectures [284, 286]. Research efforts have been invested in generic ADLs like UML [309], Meta-H [58] as well as special-purpose ADLs like RADL [337], SAE-AADL [346] and EAST-ADL [112] which have been developed for specific application domains (*e.g.* automotive or avionic systems) and modelling requirements.

2.1.5 Architecture of Embedded Systems

Deriving from the software architecture in a broader context, this thesis specifically focuses on the domain of embedded systems. Engineering of software-intensive systems has been experiencing a paradigm shift towards component-based design. At present, a large share of software is being developed by integrating components or adding new functionality to the existing components [214]. Software components are considered self-contained, replaceable parts of a software system which have clearly defined interfaces and functionality. These components can be built in-house, or externally such as Commercial Off-The-Shelf (COTS), Modified Off-The-Shelf (MOTS) and Open Source (OS). With the use of internally and externally developed components, *architecture* can be considered as the first asset that describes the system as a whole.

In embedded systems, different abstraction levels are used in addressing various design purposes [71, 141, 253, 284]. For example, when designing software components, atomic execution blocks can be considered as tasks, and thread level execution platform abstractions can be used [375]. The same architecture can be analysed on a higher level of abstraction, recognising software components and hardware units when designing a deployment [283]. Appropriate levels of abstraction are selected depending on the problem in focus [66, 326].

Architecture-driven engineering practices are being developed and widely applied in the industry including automotive [27, 69, 70, 401], avionics [53, 101, 147], medical applications [205,267] where complex embedded systems are in use. Practitioners use reference architectures and architecture patterns [135,284] due to several advantages in the systems design and development. Standards such as AutoSAR [24], IEEE 1471 [218] and AADL [346] also emerge to enhance the quality of the embedded systems design as well as to establish a common vocabulary and understandability across the industry.

2.1.6 Architecture-based Quality Evaluation

The stakeholders of a software-intensive system have various *concerns* with respect to the prospective system considered in its environment. In a practical systems design context, many of these concerns have to be analysed as early as possible. A failure to identify certain concerns such as reliability, performance or development costs earlier in the development process can result in major disruptions in the development cycle, rework or even complete failures of projects. Therefore, significant efforts are being made to evaluate those concerns of prospective systems well before they are built. As the architecture encompasses holistic aspects of the system which can be examined in different architectural views, it also becomes the blueprint of the system. Hence, in addition to its contribution to managing the complexity of software-intensive systems, the architecture leverages a crucial aspect at the systems design phase; *i.e. architecture-based quality evaluation* [32, 41, 123]. Two major evaluation techniques have been evolved in separate communities with their own compromises.

Qualitative Methods

Qualitative architecture evaluation represents the collection of approaches that take non-numerical means to evaluate the quality attributes from the architecture. In general, these methods involve questionnaires, checklists or observations. The methods are developed by, and often rely on, domain experts who have ample experience and inherently uncertain knowledge on a particular area. Techniques like peer reviews, design discussions, brainstorming sessions and the 'check for rule of thumb' practice are some examples of qualitative assessment methods. Active Design Reviews (ARD) [319], Architecture Trade-off Analysis Method (ATAM) [229], Software Architecture Analysis Method (SAAM) [228] and Active Review for Intermediate Designs (ARID) [88] are some qualitative architecture evaluation methods developed to date. They are widely used in practice, and many of the decisions in today's software-intensive systems are made based on them. However, these techniques are

rather informal and subjective [123]. Hence, qualitative evaluation techniques are rarely used in design tools or standards.

Quantitative Methods

This branch of quality evaluation refers to the methods that derive numerical metrics on the quality of the architecture. This category includes techniques like simulations, mathematical evaluations and experiments on prototypes. In contrast to the subjective questioning techniques conducted in qualitative evaluation, these methods provide quantitative answer to clearly defined concerns in relation to the architecture. Quantitative assessment of architectures has gained notable attraction during the past years [41, 121, 153, 154, 157, 187, 307], mainly due to its objectivity and the support by computer-aided tools. Significant research effort has been directed at the development quantitative evaluation methods for software architecture. Principles of statistics, mathematics and theoretical computer science have been adopted to quantify the properties of interest. However, as quantitative measurements are aimed at addressing specific and pre-defined aspects of the architecture, their applicability is limited in comparison to subjective, but much broader scope in qualitative methods [123].

Quality Attributes

Any software-intensive system is built with a set of functional goals in terms of its input, behaviour and output. According to the ISO 9126 standard, the term *functionality* in the context is defined as *"the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions"* [217]. In addition to the functionality, stakeholders have concerns on other aspects such as reliability, safety or performance of the prospective system, which are called *non-functional or quality attributes*. Hence, the design of software-intensive systems have to satisfy both functional as well as quality goals.

Depending on the stakeholders, the emphasis on different quality attributes varies. The system under consideration is required to meet at least the quality levels given in the requirements. Some of the crucial quality attributes which have to be considered in software-intensive systems design are listed below.

- *Reliability* is the continuity of correct service [26].
- *Availability* is the readiness for correct service [26].
- *Safety* is the absence of catastrophic consequences on the user(s) and the environment [26].
- *Integrity* is the absence of improper system alterations [26].
- *Performance* is a measurement of appropriate utilisation of scares system resources by software functions under certain conditions [153].
- *Energy consumption* is the energy consumed by the system [48].
- *Dependability* of a software system is an integrative concept that encompasses many of the above basic attributes, and can be defined as the ability to avoid service failures that are more frequent and more severe than is acceptable [26].

The ISO/IEC 9126 standard [217] defines a framework for evaluating software product quality. The standard also provides a set of guidelines for the evaluation process. The relative importance of the quality attributes on specific domains has also been investigated. For example, in automotive applications, Åkerholm [5] presented quality attribute ranking based on a survey carried out over a range of industrial practitioners. Many other academic and industrial analyses of quality attributes can also be found in the domains such as automotive [71, 184, 329] and avionics [101, 147].

Quality Attributes and Probability

A specific characteristic of many of these quality attributes is that they are inherently probabilistic in nature [26, 193]. These attributes are typically interpreted in a relative, probabilistic sense rather than strictly deterministic qualities. A key reason behind this notion is that the nature of those quality attributes depends on

unavoidable and imprecise factors of the system and its environment. In relation to the dependability attributes, Avizienis *et al.* state that *"The extent to which a system possesses the attributes of dependability and security should be considered in a relative, probabilistic, sense, and not in an absolute, deterministic sense: Due to the unavoidable presence or occurrence of faults, systems are never totally available, reliable, safe, or secure"* [26]. As a result of the probabilistic nature of the attributes, the modelling and quantifying of these quality attributes inherits the probabilistic notion. A typical process of quantitative architecture evaluation of probabilistic quality attributes is depicted in Figure 2.2. Based on the information extracted from the architecture, a new set of models are constructed for different quality attributes, which are called *quality evaluation models*. The quality evaluation models are also referred *probabilistic models* as they abstract certain probabilistic properties of the prospective system.

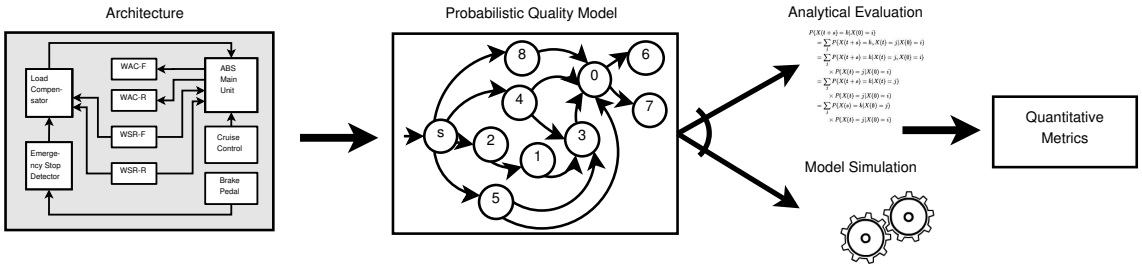


Figure 2.2: Architecture-based evaluation of probabilistic quality attributes

Evaluation of Probabilistic Quality Models

Currently, a large number of evaluation models have been proposed for specific quality attributes such as performance [36,358,408], reliability [172,182,214,239,263], energy consumption [48, 50, 315, 350] and safety [191, 205, 268, 318]. As depicted in Figure 2.2, models are either mathematically analysed or simulated to obtain quantitative metrics of the quality attribute of interest.

Analytical Evaluation

Many of the probabilistic models used in software architecture evaluation are mathematical abstractions of certain aspects of the system which are relevant to an attribute of interest. Some probabilistic properties can be quantified by solving the model analytically. Probabilistic model checking [150, 190, 247] is one example that is used to verify of the probabilistic properties from the system model. In cases where the quality attribute can be given as a algebraic function of architectural parameters, the attribute can be computed by evaluating the function. Some examples of analytical models used in probabilistic quality evaluations are given in Table 2.1.

Quality Attribute	**Quality Model**	**References**
Reliability	Reliability Block Diagrams	[23, 97, 98, 104, 105, 109, 126, 215, 242, 243, 245, 259, 261, 322, 354, 361, 368, 377, 379]
	Markov Chains	[62, 67, 68, 86, 103, 139, 149, 175, 176, 241, 251, 337, 339, 359, 399, 403, 413]
	Fault Trees	[114, 156, 316, 336, 396]
	Dependancy Graphs	[107, 410]
Performance	Aggregation Functions	[158, 409]
	Markov Chains	[176, 357, 359]
	Queuing Networks	[106, 358, 366, 370]
Safety	Block Diagrams	[101, 169, 223, 255]
	Fault Trees	[191, 254, 316, 317, 318]
	Markov Chains	[12, 12, 190]
Energy Consumption	Aggregation Functions	[28, 51, 322, 352, 397]
	Markov Chains	[89, 90, 331, 366]
	State Machines	[48, 49, 185, 264]

Table 2.1: Some examples of probabilistic models used in architecture-based quality evaluation

Model Simulation

Certain types of probabilistic models are hard to solve analytically (*e.g.* Petri Nets or Queuing Networks). Hence, model simulation techniques are used to obtain the

desired attributes from such models. Many simulation-based evaluation schemes can be found in the performance evaluation domain [64, 106]. Model simulation has also been used in architecture-based reliability evaluation in specific conditions [173, 238, 410], whereas some approaches have focused on model simulation for energy consumption estimation [209, 367].

2.2 Architecture-based Reliability Evaluation

In this thesis, the scope in terms of quality attributes and quality models is specifically focused to the ones that are relevant to reliability, which is one of the key quality attributes of interest in the domain of embedded systems. Having a prediction of the reliability of a prospective system at the architecture design stage is crucial not only for safety-and mission-critical systems but also for small, hand-held applications in a competitive market. Research into reliability evaluation has led to a variety of models, each of which focuses on a particular level of abstraction and/or system characteristics. The models' fault coverage, expressive power and underlying mathematical formulation provide a basis for the of research questions and arguments presented in later chapters. This section reviews important classes of widely-used and well-accepted architecture-based reliability evaluation approaches.

2.2.1 Combinatorial Models

Combinatorial reliability modelling is an approach to decomposing complex system into functional entities, which consist of units or subsystems. These models are widely used in the cases where repairs are not feasible, *i.e.* the system fails when all redundancy is exhausted [40].

Reliability Block Diagrams

Reliability Block Diagram (RBD) is a prominent reliability modelling approach that belongs to the category of combinatorial models. An RBD models the structural relationship of how the components and sub-system failures combine to cause system failure. They can also be analysed to predict the availability of a system and to determine the critical components from the point of view of reliability. The system is decomposed into *Reliability Blocks* that have specific failure characteristics. The success path of the system behaviour is identified by the connections between the reliability blocks, or various combinations of working components that form an

operational system. If it is possible to trace at least one path from the start of the RBD to a specific component though operational components, the specific component is considered operational. The components can be organised in series, parallel or other more complex relationships in composing the system. An organisation of a series of blocks of components that are configured in parallel within the blocks are known as a Series-Parallel (SP) system. This is a special, simpler case of RBDs which can be found in the literature [73, 243, 314, 330, 378, 392]. In SP systems, a component with its parallel redundancies is often called a *subsystem*, and the reliability of subsystems can be computed independently from the other parts of the system. Therefore, in SP RBDs, the overall system becomes a series of connected subsystems whose reliability is known. Hence, under SP assumption, the overall reliability of the system can be analytically computed. However, in the context of software architecture-based reliability evaluation, the component compositions do not always form series-parallel structure. Other system concerns compel more complex connections among reliability blocks such as bridge structures. In those cases, the analysis is more complex, hence computationally expensive [17, 23, 29].

Fault Trees

Fault Tree (FT) is another formulation to quantify the reliability of a system's architecture, which is also a widely-used technique in industrial practices [114, 156, 213, 316, 336, 396]. Apart from reliability, fault tree formalism is applied to other dependability attributes such as safety [191] and availability [214]. FT construction is a deductive, top-down approach where the failure events are organised into a tree structure. The tree can also be expanded across different levels of abstractions where top levels corresponds to the system-level failures. In the analysis of complex software system, the effects of lower-level faults and events are systematically propagated by quantifying the reliability of a higher-level abstraction. FT has both graphical and mathematical formulations to decompose the events that lead to a system failure. Conventional logic gate symbols are used in building FTs, and

they can be classified as static or dynamic depending on the types of gates they use [195, 336]. Different variants of FTs such as Component Fault Trees [156, 225], State Event Fault Trees [192, 224] are used in reliability evaluation in different contexts and abstraction levels.

2.2.2 Markov Models

Markov modelling [391] is a powerful technique for analysing complex probabilistic systems considering repair mechanisms and order of events of the system. In this technique, the system behaviour is abstracted into a set of mutually exclusive states and transition between the states [269]. A Markov model is uniquely determined by a set of equations that describe the probabilistic transitions among the states and initialisation probability distributions of the starting states. One important property of a Markov model is that the state transitions are independent of the history, *i.e.* transition from state i to state j only depends on the state i, independently from the history that state i was reached. This also implies that the complete history is summarised in the current state of the process.

It is evident that both combinatorial reliability evaluation models presented earlier are based on tree structures. In contrast, the structural representation of a Markov Chain (MC) is a directed graph – a more powerful formulation which subsumes the capabilities of tree structures. The RBDs and FTs can be transformed to MCs, but the inverse is often not possible. Furthermore, due to the capability of MCs to include the system's internal behaviour abstraction in reliability models, they provide a more comprehensive analysis in the architecture-based reliability evaluation compared to the structure-only abstractions such as RBD and FTs. Three main variants of Markov models that are used in architecture-based quantitative evaluation of reliability, are briefly described in the following sub sections.

Discrete Time Markov Chains (DTMC)

DTMCs are finite state machine formalisms with probabilities attached to transitions which are widely used in modelling discrete-time dynamic systems. They can easily model characteristics such as probabilistic service invocations, execution graphs and failure behaviours of software-intensive systems, and hence they are a prominent technique in software-reliability evaluation. A DTMC can be used to represent all the relevant states of a software execution and the probability to move from each state to another. Among all the states, one state is the *initial state* and one or more among the other states represent successful completion of execution or occurrence of a failure. The formal definition of DTMC can be expressed as a tuple (S, s_0, P, L) where,

- S is a finite set of states
- $s_0 \in S$ is the *initial state*
- $P : S \times S \rightarrow [0, 1]$ is the *transition probability matrix*
- $L : S \rightarrow 2^{AP}$ is the *labelling function*

DTMCs can be used to model a single transition system or the synchronous composition of many systems. The labelling function describes the mapping from the states to the set of atomic propositions AP (typically constraints on variables employed in the model, *e.g.* $(I = 0), (I > 15)$). $P(s, s')$ denotes the probability of making a transition from state s to the state s' . In a DTMC, $\sum_{s' \in S} P(s, s') = 1$ for all states $s \in S$, which implies that even terminating states should have an outgoing transition to itself with a probability 1. When a system is modelled as a DTMC, the execution is represented by a path through the DTMC. A DTMC is called *absorbing* when it contains at least one terminating states [391].

Continuous Time Markov Chains (CTMC)

CTMCs are variant of Markov chains that allow the transitions to take place at continous time. CTMC extension of Markov chains are heavily used in modelling

real-time systems. While a transition in a DTMC refers to a discrete step, transition in CTMC represents a state change taken in real-time. The transitions in a CTMC are labelled with rates instead of probabilities. The formal definition of a CTMC [33] can be expressed as a tuple (S, s_0, R, L) where:

- S is a finite set of states
- $s_0 \in S$ is the *initial state*
- $R : S \times S \to \mathbb{R}_{\geq 0}$ is the *transition rate matrix*
- $L : S \to 2^{AP}$ is the *labelling function*

The delay within a state is represented by the transition rate function which is a form of a negative exponential distribution with a parameter equal to the relevant transition rate. The probability of a transition from state s to s' is enabled within in t time units are given by $1 - e^{-R(s,s')t}$.

When there is more than one transition from a state s, the first enabled transition will be carried out. Therefore, it is said that the transitions are in a race condition which represents real-time behaviour of actual systems. The probability of making a transition from state s to s' can be expressed as $R(s, s')/\sum_{s'' \in S} R(s, s'')$. This leads to the description of embedded DTMC in a CTMC, which has the same S, s_0 and L. The transition matrix P is defined as for the DTMC case described above, with the exception of $P(s, s) = 1$ for the states s which do not have any outgoing transitions.

When a real-time system is modelled as a CTMC, an execution path is defined as a non-empty sequence of $s_0 t_0 s_1 t_1 s_2 t_2$where $R(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{\geq 0}$. The amount of time spent on state s_i is denoted by time t_i. The state at time t in a path w is denoted by $w@t$, that also represented by $w(k)$ where k is the smallest index for which $\sum_{i=0}^{k} t_i > t$. Transient behaviour and steady state behaviour can be described using a CTMC. Transient behaviour represents the system's state of a particular time instance, while the steady state behaviour describes a system's state in long run. Transient state probability $\pi_{s,t}(s')$ is the probability of being in the state s at time t. Steady state probability $\pi_s(s')$ refers to $lim_{t\to\infty}\pi_{s,t}(s')$.

The application of CTMCs are evident in architecture-based reliability evaluation [182, 262] as well as in performance models [357, 358].

Markov Decision Process

A Markov Decision Process (MDP) is an extension of a DTMC which accommodates non-deterministic choices in the models. Non-determinism is useful to model the concurrent behaviour when many probabilistic systems run in parallel as well as when the exact probabilities are not known. As Hoare [208] states, non-determinism is due to deliberate decisions to ignore the factors which influence the selection (of transitions in the MDP). It is also used to simplify the models in analysing specific behaviours by eliminating less-relevant details. The formalisation of a MDP [34] can be given as a tuple $(S, s_0, Steps, L)$ where:

- S is a finite set of states
- $s_0 \in S$ is the *initial state*
- $Steps : S \to 2^{Dist(S)}$ is the *transition function*
- $L : S \to 2^{AP}$ is the *labelling function*

The $Dist(S)$ represents the set of all probability distributions over S, in other words all functions in the form of $f : S \to [0, 1]$ where $\sum_{s \in S} f(s) = 1$. The transition function maps each state $s \in S$ to a non-empty subset of $Dist(S)$. Therefore, the elements in $Steps(s)$ for a given state s indicate the non-deterministic choices available in that state. Since both the non-determinism and probabilistic choice have to be solved in the identification of a path in MDP, the non-determinism is assumed to be handled by adversaries, in most practical cases. *Adversaries* [372] are mechanisms that select non-deterministic transitions based on history of choices made so far as opposed to the history-independent Markov decisions. When an adversary is given, the behaviour of the MDP is probabilistic and can be seen as a DTMC whose states refer to finite paths in the initial MDP. The probabilities of non-deterministic transitions are governed by the adversary. The technique described above can be used in calculation of maximum or minimum probabilities of a specific behaviour by evaluating the model for all possible adversaries.

Markov Model Analysis

Markov chain-based reliability evaluation models can be further distinguished depending on the way they integrate failure behaviour and the system's operational characteristics. They have advantages as well as disadvantages on their own, and good discussions can be found in work of Gokhale *et al.* [172, 174].

Composite Analysis Method

In the composite analysis method, architecture model and failure model are combined into a single reliability model. The technique was introduced originally by Cheung [86], and has been widely used [68, 103, 175, 178, 181, 182, 237, 337, 360, 404] and extended several times. The basic concepts of the composite analysis are explained in the following paragraphs using Cheung's model.

■ *Architecture.* The software system in focus is assumed to be a *terminating application*, which is an application that operates on demand, and a single run of software that corresponds to a terminating execution can be clearly identified. The program flow graph of a terminating application has a single entry and a single exit node. Even though the model is defined for a single entry and single exit application, it can easily be extended to support more complex software systems with multiple initial nodes and multiple final states by introducing *super-initial*, *super-final* states [404]. The transfer of control among modules can be described by an absorbing DTMC with transition probability matrix $P = [p_{ij}]$, where p_{ij} denotes the probability of j^{th} module is called after executing the i^{th} module.

■ *Failure Behaviour.* An assumption of Cheung's model is that the components fail independently and the reliability of the component i is characterised by the probability R_i that the component performs its function correctly, *i.e.* the component produces the correct output and transfers control to the next component without a failure.

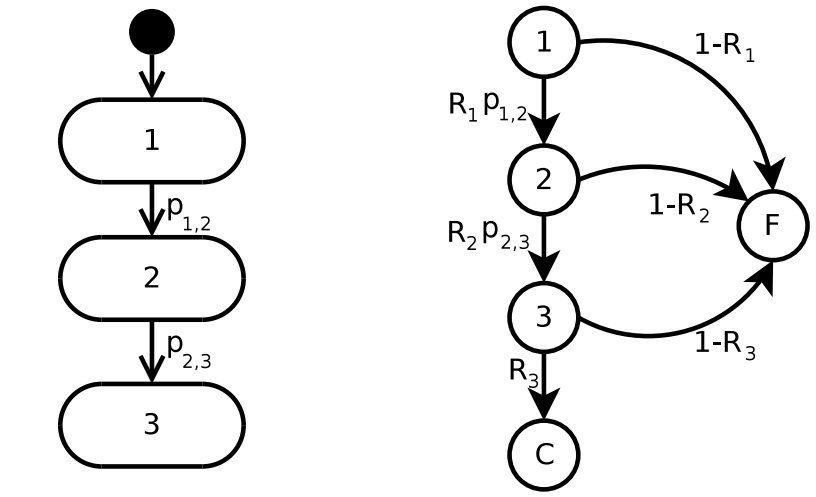
■ *Reliability Evaluation.* The DTMC reliability model is constructed from the architecture of the system such that a node represents the execution of a component and arcs denote the transfer of execution from one component to the other. Two absorbing states C and F are added to the DTMC, representing the correct output and failure, respectively, and the transition probability matrix P is modified accordingly to $\hat{P}$. The original transition probability p_{ij} between the components i and j is modified into $R_i p_{ij}$, which represents the probability that the module i produces the correct result and the control is transferred to component j. From the final (exit) state n, a directed edge to state C is created with transition probability R_n to represent the correct execution. The failures of a component i are considered by creating a directed edge to failure state F with transition probability $(1 - R_i)$. This process integrates the failure behaviour of the components to the functional behaviour described in the original control flow. Thus, a DTMC defined with transition probability matrix $\hat{P}$ is considered as a composite model of the software system [175]. Figure 2.3a illustrates a control flow graph of a software system which can be obtained from its behavioural description or profiling tools. The corresponding DTMC when the two states C and F are added is depicted on the right of the figure. The reliability of the program is the probability of reaching the absorbing state C of the DTMC.

Let Q be the matrix obtained from $\hat{P}$ by deleting rows and columns corresponding to the absorbing states C and F. $Q^k_{(1,n)}$ represents the probability of reaching state n from 1 through k transitions. From initial state 1 to final state n, the number of transitions k may vary from 0 to infinity.
It can be proved that the infinite summation converges as follows [86]:

$$S = I + Q + Q^2 + Q^3 + ... = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1} \tag{2.1}$$

where I is the $n \times n$ identity matrix.

The matrix S is called the *fundamental matrix* of the DTMC, and $S_{(i,j)}$ represents the expected number of visits to the state j starting from state i before it is absorbed.



(a) Control Flow Graph

(b) DTMC when the states C and F are added

Figure 2.3: Control flow graph and corresponding DTMC for composite analysis model.

Cheung [86] introduced an architecture based reliability estimation method in which the reliability of the overall system can be computed from S as;

$$R_s = S_{(1,n)} R_n \tag{2.2}$$

Hierarchical Analysis

Another Markov chain-based reliability evaluation technique has been developed where the architectural model is solved separately, and the failure behaviour of the system is superimposed onto the solution of the architectural model. This hierarchical method has initially been presented by Kubat [241] in relation to DTMCs, and has also been the basis for many recent extensions [172, 175, 288, 289].

The reliability evaluation is carried out in two steps. First, the behavioural model of the system's functionality is solved to obtain architectural statistics, such as mean and variance of the number of visits of components in a single execution. These statistics are then combined with the failure parameters of the components to solve the failure model. To illustrate, a DTMC-based hierarchical model is described

below.

■ *Architecture.* The behavioural description of the system is modelled as an absorbing DTMC [391] where a node represents the execution of a component and arcs denote the transfer of execution from one component to the other. A super-initial node [404] is added to represent the execution start, and arcs are added from that node annotated with relevant execution initialisation probabilities(q_0). Figure 2.4 shows an example DTMC of a software system with multiple entry points.

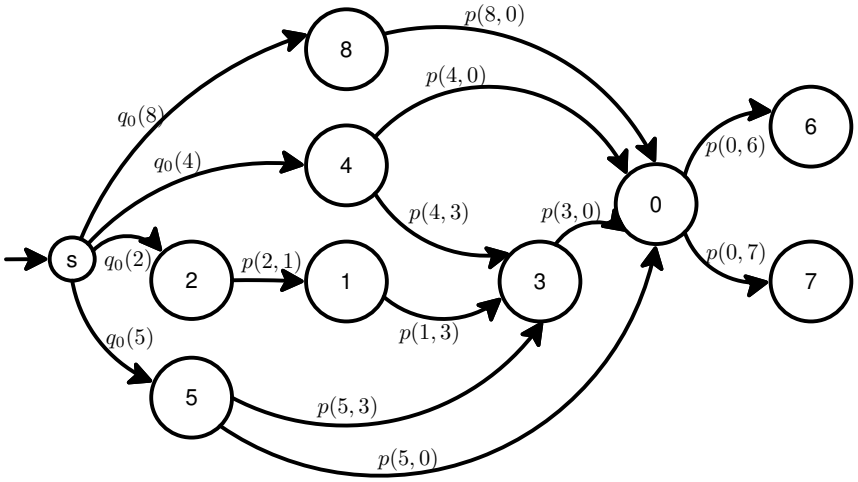


Figure 2.4: Annotated DTMC for service reliability evaluation

■ *Failure Behaviour.* The model assumes that the components fail independently and the reliability of the component i is characterised by the probability R_i that the component performs its function correctly, i.e., the component produces the correct output and transfers control to the next component without a failure.

■ *Reliability Evaluation.* First, the DTMC that abstracts the software system's behaviour is solved to obtain the expected number of visits of each component. The expected number of visits of a DTMC node $v_i : C \to \mathbb{R}_{\geq 0}$, quantifies the expectation of use of a component (or subsystem) during a single system execution. This can be computed by solving the following set of simultaneous equations [175, 241]:

$$v(c_i) = q_0(c_i) + \sum_{j \in \mathcal{I}} (v(c_j) \cdot p(c_j, c_i)) \tag{2.3}$$

where $\mathcal{I}$ denotes the index set of all nodes in the DTMC.

The following expansion of the formula (2.3) can be used to transform the equation in matrix form, where I denotes the identity matrix:

$$\begin{aligned}
v(c_0) &= q_0(c_0) + v(c_0) \cdot p(c_0, c_0) + v(c_1) \cdot p(c_1, c_0) + ... + v(c_n) \cdot p(c_n, c_0) \\
v(c_1) &= q_0(c_1) + v(c_0) \cdot p(c_0, c_1) + v(c_1) \cdot p(c_1, c_1) + ... + v(c_n) \cdot p(c_n, c_1) \\
v(c_2) &= q_0(c_2) + v(c_0) \cdot p(c_0, c_2) + v(c_1) \cdot p(c_1, c_2) + ... + v(c_n) \cdot p(c_n, c_2) \\
&\vdots \\
v(c_n) &= q_0(c_n) + v(c_0) \cdot p(c_0, c_1) + v(c_1) \cdot p(c_1, c_n) + ... + v(c_n) \cdot p(c_n, c_n)
\end{aligned}$$

In matrix form, the transfer probabilities $p(c_i, c_j)$ can be written as $P_{n\times n}$, and the execution initiation probabilities $q_0(c_i)$ as $Q_{n\times 1}$. The matrix of expected number of visits $V_{n\times 1}$ can be expressed as:

$$V = Q + P^T \cdot V \tag{2.4}$$

With the usual matrix operations, the above can be transformed into the solution format:

$$I \times V - P^T \times V = Q \tag{2.5}$$

$$(I - P^T) \times V = Q \tag{2.6}$$

$$V = (I - P^T)^{-1} \times Q \tag{2.7}$$

For absorbing DTMCs, a term that applies to the model used in this illustration, it has been proven that the inverse matrix $(I - P^T)^{-1}$ exists [391]. Therefore the Matrix V can be computed.

After calculating the expected number of visits matrix, the values are combined with the failure information. Assuming the components fail independently and truncating higher order Tailor series terms [172, 241, 289], reliability of the whole system is given as,

$$R \approx \prod_{i\in\mathcal{I}} R_i^{v(c_i)} \tag{2.8}$$

where $\mathcal{I}$ denotes the index set of all nodes in the DTMC.

Model Parameters

A considerable number of parameters are involved in the architecture-based reliability evaluation. Software component reliabilities, hardware failure rates, transition probabilities in the execution models and execution initialisation probabilities are some examples of the parameters used in the models described above. At the design of software-intensive systems, many of these parameters have to be given as inputs, before the system is built. In the prevalent architecture-based evaluation methods, these parameters are given as *point estimates* (distinct numerical values) and the quality metric produced from the probabilistic model evaluation is also *point value* of the quality attribute of interest.

2.2.3 Extended Frameworks

These architecture-based reliability evaluation concepts have been the basis for a large number of novel approaches and frameworks. UML-based frameworks [103, 109], Scenario-based extensions [410], Fault Propagation-based evaluations [107,222, 374], component modelling frameworks [67, 279, 280], graph structure-based methods [113,324], Neural Networks [419], Bayesian Models [344] are some examples. Extensive surveys can be found including Lyu [269], Goseva-Popstojanova *et al.* [182] and Immonen *et al.* [214].

2.3 Architecture Transformation and Optimisation

Architecting software-intensive systems entails making a multitude of decisions regarding the prospective system. Various stakeholders are interested in certain quality attributes of the system, which depend on those decisions made on the architecture. Hence, *architecture optimisation* techniques are being developed to search for better architecture alternatives with respect to numerous quality attributes of interest. The next subsections describe different aspects of architecture optimisation in the context of embedded software-intensive systems.

2.3.1 Architecture Optimisation Aspects

Design Decisions

In relation to embedded systems, a large number of design decisions have to be made in the architecture design phase. As the design decisions are understood in broader terms across many architectural views [219, 240], these decisions can be made by different teams and in a different order of sequence in industrial practice. Changes made to design decisions contained in an architecture constitute a new architecture, often referred to as an architecture alternative [280]. Starting from an initial architectural solution, the process of changing certain design decisions is called *Architecture Transformation* [14, 121, 236], and the actions taken to perform the transformation are called *Architecture Transformation Operators* [39,64,66,187]. Some of the important decisions in the context of embedded systems design are discussed in this section, in order to highlight their different characteristics that have to be considered in an architecture optimisation approach.

■ *Software/Hardware Component Selection.* An important decision in architecting software-intensive is to select software and hardware components. In the hardware domain, *selection* in topology construction refers to employing specific hard-

ware units selected from a set of different available ones with different properties [94, 133, 258, 304, 305, 314, 336, 405]. In the software domain, the system can be built by composing different alternative software components which have similar functionalities with different properties [378, 379]. The architect has to make decisions on which ones to use, considering non-functional attributes such as reliability, performance as well as other factors like cost and delivery time.

■ *Software/Hardware Redundancy Allocation.* A diversity of identical or similar units (hardware or software) are commonly employed in software-intensive systems in order to to gain additional tolerance against faults. As the redundant units have negative consequences like cost, lower performance, excessive weight and higher energy consumption, the problem of finding appropriate redundancy levels in the architecture design has gained significant interest of the industry. This problem is often entitled as *Redundancy Allocation Problem* (RAP) [245, 386, 387]. RAP is divided into two major categories, *i.e.* redundancy allocation of identical components (homogeneous redundancy) which tolerates random faults, and N-Version computing [25, 285] where redundancy of non-identical units (heterogeneous redundancy) are employed with the goal of tolerating both logical and random failures. One of the most studied design configurations of the RAP is a series system of s independent k-out-of-n subsystems. A subsystem is considered as a properly functioning block if at least k_i of its n_i components are operational. If k_i is one for all subsystems, then it is a series-parallel system. A large amount of research effort has focused on the topic of redundancy allocation itself [43, 97, 242, 243, 259, 316], and it has been proven that the RAP is NP-hard [84]. Many research efforts specifically aim to optimise the redundancy allocation decisions.

■ *Deployment.* During the architecture design of embedded software systems, deployment denotes the allocation of software components to the hardware units, and the assignment of inter-component communication activities to network links. With the emerging trends in component-based software engineering paradigm, *Commercial Off-The-Shelf* (COTS) components are widely used, and are composed in build-

ing variety of complex embedded systems. Even for in-house development projects, the processing and communication responsibilities have to be assigned to a hardware topology. The assignment decisions determine the quality of the prospective system in terms of attributes like reliability, performance, safety and energy consumption. Therefore, deployment is considered a crucial task in architecture design, and numerous approaches [230, 283, 298, 320] are dedicated to deployment modelling and optimisation of component-based software systems.

■ *Software/Hardware Parameters.* A number of hardware and software parameters have to be adjusted in the architecture design phase of software systems. This includes settings like change of memory capacity of processor, bus speed, processing speed, scheduling strategy in the hardware domain as well as changes such as queueing priorities, decision rules and processing algorithms in the notion of software. These decisions are motivated by various quality goals (*e.g.* response time) and also have negative effects such as increased cost or energy consumption.

These different decisions are applied to different elements and aspects of the architecture. Some of the decisions are directly linked to a specific software or hardware unit, while certain changes affect the overall assembly of the system. Therefore, an approach for automated robust optimisation for architecture requires consideration of this heterogeneity in the architecture changes.

Goals of Optimisation

As described in previous sections, software architecture design involves satisfying many stakeholders' concerns in terms of different quality attributes. The architectural design decisions discussed above influence one or many of them, which are often in conflict with each other. Hence these quality attributes become objectives of architecture optimisation, where the goal is to find architectures of better quality. Some of the key quality attributes and their dependence on architectural decisions can be identified as follows.

■ *Reliability.* Reliability is one of the primary design goals in architecture design. In

software-intensive systems, reliability refers to *the continuity of correct service* [26]. The traditional definition of reliability is extended with fault tolerance, which assumes that either the system does not fail at all for a given period or it successfully recovers state information after a failure for a system to resume its service as if it was not interrupted [214]. Several metrics have been developed to provide quantitative assessment on reliability, such as mean time to failure (MTTF), mean time to repair (MTTR), probability of continuous correct operation and failure rate. In the domain of software-intensive systems, probabilistic models are employed for reliability quantifications, and they often require information on software failures, hardware failures as well as the usage profile of the system.

■ *Performance.* The term *performance* in software-intensive systems has a broader scope which involves various system aspects like responsiveness, throughput and resource usage. A performance property can be defined as a metric of a system that shows how fast a system executes a certain functionality. The performance of a software-intensive system depends on a number of architectural decisions, *e.g.* deployment or redundancy allocation. For a specific architecture alternative, quantification of the performance metrics is often achieved analytically or using simulations of performance models such as queueing networks, Markov chains and Petri-Nets [36,37]. Similar to the reliability evaluation, performance prediction techniques requires detailed information on hardware, software and usage profile.

■ *Safety.* Safety is defined as the *absence of catastrophic consequences on the user(s) and the environment* [26], which is a critical design goal in many embedded systems including domains like avionics, automotive, medical or nuclear power. The designs of such systems are often required to satisfy strict safety standards, which are specific to the domains. A safety requirement is often specified as a formal hazard description together with a tolerable hazard probability [189]. Quantitative evaluation techniques like fault-trees [156, 318] and FMEA [169, 190] are used in safety evaluation context, where the valuation models are constructed using architectural parameters.

■ *Energy Consumption.* Energy consumed in electronics platforms gains growing attention in software intensive systems design. In the embedded systems domain, it is more evident as most of the systems must support their operation from a limited battery that is hard to recharge (*e.g.* in deep-sea or outer-space missions) or at least uncomfortable to recharge very often (*e.g.* in mobile phones or electric cars). This is further emphasised in systems requiring a minimal battery size (*e.g.* in nano-devices). Not limiting to embedded systems, there are growing trends to minimise energy consumed by software in enterprise systems with the emerging concepts of cloud computing and server farms. Architecture-based probabilistic evaluation models are being developed [153,172,193,307,359] to provide quantitative metrics on the energy consumption of prospective systems.

■ *Cost.* In almost all industries, cost is undoubtedly a key decision factor. Accordingly, software-intensive systems design has to consider different cost aspects like manufacturing, raw material, maintenance, testing, installation as well as opportunity costs. In software engineering, there is a field of research which provides techniques to quantify cost from architecture decisions such as COCOMO [63] and ArchE [30, 121].

The example objectives considered above demonstrate the diversity of design objectives in architecture design of software intensive systems. The different quality attributes use their own models to obtain quantitative metrics while using information on different architectural aspects. These objectives often have to be optimised in parallel, resulting in conflicting goals.

Design Constraints

In the design of software-intensive systems, often there is a set of conditions that a candidate design must satisfy regardless of the other quality aspects. The enforcement of *constraints* is a mechanism in architecture design to ensure the feasibility of the alternatives.

■ *Attribute-Based Constraints.* It is very common to have design constraints which

are related to numerous quality attributes. In certain systems design settings, the quality attributes are formulated as constraints rather than goals of optimisation. Some constraints are inspired by regulations or service contracts. Examples include requirement like 'Failure rate of automotive brake system should be less than 10^{-6} per hour' or 'Response time for a location query in an autonomous robot should be less than $200\mu s$'.

■ *Memory.* This is an example constraint that has to be satisfied in deployment decisions. Memory requirement of all software components that are allocated to a hardware host (or ECU) must not exceed the memory capacity of the host [272].

■ *Collocation.* The Collocation constraint is used to specify restrictions among components. Malek [272] defines a collocation function which can specify a subset of components that either must be, must not be or may not be deployed on the same host. This formulation reflects design concepts such as 'redundant software components should be deployed to different hardware units'.

■ *Localisation.* This is a deployment constraint which specifies a subset of hosts on which a given component may be legally deployed [272]. This is useful as some software components require hardware specific resources such as access to sensors or actuators.

■ *Redundancy Levels.* In redundancy allocation problems, the software or hardware components in architecture design are often subject to constraints. For instance, N-modular configurations require odd numbers of redundancies to employ majority voting. Furthermore, practical problems often have an upper limit for the possible allocation of redundancy levels.

Similar to the quality attributes and design alternatives, it is apperent that the constraints are applicable to different abstraction levels of the architecture and require information on peculiar aspects of the software architectures.

Dimensionality of the Objective Space

Probabilistic quality attributes such as reliability, availability, energy consumption, safety and performance in combination with other attributes like cost and weight have been the objectives for architecture optimisation. One important aspect of the optimisation approaches is the way these objectives are treated.

Single Objective Optimisation (SOO)

In this category, only one objective is considered for optimisation. Therefore, the final result of the optimisation algorithm is one single solution which can be optimal or near-optimal in terms of that quality attribute.

Multiple Objectives Transformed to Single (MTS)

The multiple design objectives existing in the architecture optimisation problems are sometimes transformed into single objective. In most of the cases, weighted sum functions are used to combine several objectives. However, deciding the weights is a subjective task, and the results of the optimisation depend on the initial decision on the weights. From the optimisation point of view, there is only one objective to be optimised, and consequently the final result is a single solution.

Multi-Objective Optimisation (MOO)

Some optimisation approaches consider multiple design goals as orthogonal objectives to be optimised. This strategy is significantly different from the other two approaches as the optimisation algorithm has to consider multiple objectives. This adds complexity to the optimisation process and necessitates more sophisticated approaches. Due to the multiplicity in the objective space, the final achievement of an algorithm is not necessarily a single solution, but can be a set of non-dominated solutions.

▶ *Pareto Optimality.* Non-domination/Pareto Optimality is a concept used in comparing solutions when more than one objectives are the goals of optimisation. A solution x^* is said to be non-dominated if there is no any other x such that $f_i(x^*) \prec f_i(x)$ for all $i = 1, ..., n$ objectives and $f_j(x^*) \triangleleft f_j(x)$ for at least one j, where $f_i(x)$ denotes the objective i of x, $a \prec b$ to represent b is equal to or better than a and $a \triangleleft b$ representing b is better than a, irrespective of maximisation or minimisation. In other words, if there exists no other feasible solution x which would be superior in at least one objective while being no worse in all other objectives of x^*, then x^* is called *Pareto-Optimal set* or *non-dominated* set. The solutions of a MOO algorithm are called *non-dominated set* or *Pareto-Optimal set*.

The MOO approaches preserves the conflicting-notion of the objectives throughout the optimisation and do not restrict the search based on initial weights, as opposed to above MTS approach. Table 2.2 presents a summary of architecture optimisation aspects considered in some current approaches. For each design decision, many approaches exist that focus on optimisation for different quality attributes considering various design constraints.

2.3.2 Optimisation Techniques

Due to ever-increasing system complexity, software engineers have to choose from combinatorially growing numbers of design options, resulting in a design space that is often beyond human capabilities of understanding, making the architecture design a challenging task. The need for automated design space exploration has been recognised in industry and research [329]. However, the architecture-based quantitative evaluation has made automated exploration possible, and a plethora of architecture optimisation approaches have been developed. This section presents a summary of related work in the context of design-time optimisation of embedded software-intensive systems.

Arch. Dece.	Quality Attributes	Constraints	Dimensionality
Deployment	**RELIABILITY** [138, 144, 215, 216, 220, 273, 279, 280, 302, 304, 305, 306, 323, 330, 375], **PERFORMANCE** [2, 10, 38, 54, 60, 81, 82, 136, 138, 140, 144, 163, 165, 202, 204, 207, 210, 215, 216, 220, 226, 227, 272, 279,280,302,304,305,310,326,327,342,357, 373, 384, 388, 402, 414, 420], **COST** [54, 60, 122, 140, 199, 220, 231, 252, 272, 279, 280, 305, 306, 326, 384], **AREA** [81, 207, 226, 384], **AVAILABILITY** [297], **ENERGY** [122, 140, 273, 326, 351, 415], **FLEXIBILITY** [54], **GENERAL** [6, 35, 61, 167, 265, 355], **SAFETY** [306]	**PERFORMANCE** [61,138,216,226,231,252,310, 375], **PHYSICAL** [2, 165, 199, 204, 210, 273, 327, 330, 342, 388, 414, 415], **PRECEDENCE** [2, 165, 210, 327, 330, 342, 414], **STRUCTURAL** [140, 265, 326], **THROUGHPUT** [373], **TIMING** [10, 136, 204, 220, 327, 330, 388, 415], **AREA** [38], **CAPACITY** [304], **COST** [61, 122, 220, 402], **DELAY** [384], **DEPENDABILITY** [304], **GENERAL** [60, 305], **MAPPING** [6, 81, 167, 302, 304], **MEMORY** [6, 81, 302, 384], **MULTIPLE** [215, 297], **NOT PRESENTED** [35,54,82,144,163,202, 207, 227, 272, 279, 280, 306, 323, 351, 355, 357, 420]	**MOO** [6, 54, 61, 81, 122, 140, 144, 167, 215, 220, 226, 265, 272, 279, 280, 302, 304, 305, 326, 351, 355, 375, 384], **SOO** [2, 10, 38, 82, 136, 163, 165, 199, 202, 204, 210, 227, 231, 252, 297, 323, 327, 330, 342, 357, 373, 402, 414, 415, 420], **MTS** [138, 207, 216, 273, 306, 310, 388]
Hardware Redundancy	**RELIABILITY** [57, 91, 92, 93, 94, 95, 96, 97, 99, 100, 113, 130, 131, 211, 243, 244, 257, 258, 261, 288, 305, 336, 347, 376, 377, 378, 405, 419], **COST** [98, 137, 211, 243, 258, 261, 305, 312, 356, 376, 377, 378, 379, 385], **WEIGHT** [243, 376, 377, 378, 379], **AVAILABILITY** [137, 379], **ENERGY** [288], **GENERAL** [167], **PERFORMANCE** [113, 305]	**COST** [57, 91, 92, 95, 96, 97, 131, 244, 257, 258, 336, 347, 405, 419], **WEIGHT** [57, 91, 92, 95, 96, 97, 130, 131, 211, 244, 257, 258, 336, 356], **GENERAL** [93, 94, 99, 100, 305], **MAPPING** [167], **MULTIPLE** [137, 261], **AVAILABILITY** [312, 385], **NOT PRESENTED** [113, 243, 258, 376, 377, 378, 379], **REDUNDANCY LEVEL** [288], **RELIABILITY** [98, 356], **SIZE** [336], **VOLUME** [130, 131, 211]	**MOO** [93, 94, 113, 167, 211, 243, 258, 261, 288, 305, 336, 376, 377, 378, 379, 405], **SOO** [57, 91, 92, 95, 96, 97, 98, 99, 100, 130, 131, 244, 257, 258, 312, 347, 356, 385, 419],**MTS** [137]
Software Redundancy	**AVAILABILITY** [137,379], **COST** [98, 137, 211, 243, 258, 305, 356, 376, 377, 378, 379, 385], **ENERGY** [288], **PERFORMANCE** [3, 166, 305], **RELIABILITY** [3, 13, 57, 91, 92, 93, 94, 95, 96, 97, 99, 100, 130, 131, 166, 211, 243, 244, 257, 258, 288, 305, 347, 376, 377, 378, 405, 412, 419], **WEIGHT** [243, 376, 377, 378, 379]	**AVAILABILITY** [385], **COST** [3, 13, 57, 91, 92, 95,96,97,131,244,257,258,347,405,412,419], **GENERAL** [93, 94, 99, 100, 305], **MULTIPLE** [137], **NOT PRESENTED** [243, 258, 376, 377, 378, 379], **PERFORMANCE** [166], **POWER** [3], **REDUNDANCY LEVEL** [288], **RELIABILITY** [98, 356], **VOLUME** [13, 130, 131, 211, 412], **WEIGHT** [3, 13, 57, 91, 92, 95, 96, 97, 130, 131, 211, 244, 257, 258, 356, 412]	**MOO** [3, 93, 94, 166, 211, 243, 258, 288, 305, 376, 377, 378, 379, 405], **MTS** [137], **SOO** [13, 57, 91, 92, 95, 96, 97, 98, 99, 100, 130, 131, 244, 257, 258, 347, 356, 385, 412, 419]
Component Selection	**PERFORMANCE** [19, 20, 21, 22, 74, 75, 78,79,80,81,128,133,152,198,201,221,233, 271, 279, 280, 292, 293, 294, 295, 305, 416], **RELIABILITY** [13, 29, 74, 91, 92, 93, 94, 95, 97, 99, 100, 170, 171, 211, 221, 233, 243, 244, 257, 258, 261, 271, 279, 280, 295, 305, 306, 313, 314, 347, 376, 377, 378, 399, 405, 416], **AVAILABILITY** [19, 21, 22, 74, 78, 80, 128, 133, 137, 198, 221, 233, 271, 274, 294, 379, 416], **COST** [17, 19, 20, 21, 22, 29, 74, 75, 77, 78, 79, 80, 98, 102, 104, 108, 128, 133, 137, 198, 211, 221, 233, 243, 252, 258, 261, 271, 274, 279, 280, 292, 305, 306, 312, 328, 353, 356, 369, 376, 377, 378, 379, 385, 393, 394, 395, 416], **ENERGY** [351], **WEIGHT** [243,376,377,378,379] **GENERAL** [203, 282, 369], **N/A** [72, 341, 381, 417, 418], **REPUTATION** [22, 198, 221, 416], **SAFETY** [15,16,306,321,338], **SECURITY** [294], **AREA** [81]	**AVAILABILITY** [312, 385], **COST** [13, 16, 77, 91, 92, 95, 97, 170, 171, 244, 257, 258, 292, 313, 314, 347, 399, 405], **DELIVERY TIME** [104, 108, 328], **DESIGN** [15], **FUNCTIONAL CORRECTNESS** [79], **GENERAL** [29, 74, 93, 94, 99, 100, 203, 305, 418], **LOCAL** [19, 21], **MAPPING** [81], **MEMORY** [81], **MULTIPLE** [137, 261], **NOT PRESENTED** [20, 72, 75, 133, 152, 198, 201, 221, 243, 258, 274, 279, 280, 282, 294, 295, 306, 321, 338, 341, 351, 369, 376, 377, 378, 379, 381, 416, 417], **PERFORMANCE** [252], **QOS VALUES** [19,21,78,79,80,128,233,271,293], **REDUNDANCY LEVEL** [233], **RELIABILITY** [17, 98, 104, 108, 328, 356], **REQUIREMENTS** [22, 102,353,393,394,395], **RESPONSE TIME** [292], **STABILITY** [79], **STRUCTURAL** [22], **SYSTEM DOWNTIME** [16], **TIMING** [19, 21], **VOLUME** [13, 211], **WEIGHT** [13, 91, 92, 95, 97, 211, 244, 257, 258, 313, 356]	**GENERAL** [72, 282, 295], **MOO** [15, 29, 81, 93, 94, 211, 243, 258, 261, 274, 279, 280, 305, 328, 338, 351, 376, 377, 378, 379, 394, 395, 405], **MTS** [19, 20, 21, 22, 74, 75, 78, 79, 80, 128, 133, 137, 198, 203, 221, 233, 271, 292, 293, 294, 306, 341, 353, 369, 381, 393, 416, 417, 418], **SOO** [13, 16, 17, 77, 91, 92, 95, 97, 98, 99, 100, 102, 104, 108, 152, 170, 171, 201, 244, 252, 257, 258, 312, 313, 314, 321, 347, 356, 385, 399]
Hardware Selection	**RELIABILITY** [91, 92, 93, 94, 95, 97, 99, 100,113,211,243,244,257,258,304,305,306, 314, 336, 347, 376, 377, 378, 405], **AVAILABILITY** [137,379], **COST** [98,137,211, 243,258,305,306,312,376,377,378,379,384, 385], **ENERGY** [209], **GENERAL** [61, 265], **PERFORMANCE** [113, 304, 305, 384], **AREA** [384], **SAFETY** [306], **WEIGHT** [243, 376, 377, 378, 379]	**COST** [61, 91, 92, 95, 97, 244, 257, 258, 314, 336, 347, 405], **AVAILABILITY** [312, 385], **CAPACITY** [304], **DELAY** [384], **DEPENDABILITY** [304], **GENERAL** [93, 94, 99, 100, 305], **MAPPING** [304], **MEMORY** [384], **MULTIPLE** [137], **NOT PRESENTED** [113, 243, 258, 306, 376, 377, 378, 379], **PERFORMANCE** [61, 209], **RELIABILITY** [98], **SIZE** [336], **STRUCTURAL** [265], **VOLUME** [211], **WEIGHT** [91, 92, 95, 97, 211, 244, 257, 258, 336]	**MOO** [61, 93, 94, 113, 211, 243, 258, 265, 304, 305, 336, 376, 377, 378, 379, 384, 405], **SOO** [91, 92, 95, 97, 98, 99, 100, 209, 244, 257, 258, 312, 314, 347, 385], **MTS** [137, 306]
Scheduling	**PERFORMANCE** [1, 2, 47, 81, 82, 114, 115,116,125,138,143,165,166,204,210,212, 220,249,301,303,310,327,342,373,384,388, 414, 420], **RELIABILITY** [114, 125, 138, 166,220,330], **COST** [83,114,115,122,220, 231, 252, 384], **ENERGY** [114, 115, 122, 212, 351, 365], **AREA** [81, 384], **GENERAL** [61]	**PERFORMANCE** [47, 61, 83, 114, 115, 116, 138, 166, 231, 252, 310, 365], **PHYSICAL** [1, 2, 165, 204, 210, 327, 330, 342, 388, 414], **PRECEDENCE** [1, 2, 47, 125, 165, 210, 249, 327, 330, 342, 414], **TIMING** [204, 212, 220, 249, 327, 330, 388], **COST** [61, 122, 220], **DELAY** [384], **LATENCY** [301], **MAPPING** [81], **MEMORY** [81,384], **NOT PRESENTED** [82,143,303, 351, 420], **THROUGHPUT** [301, 373]	**SOO** [1, 2, 47, 82, 83, 115, 116, 143, 165, 204, 210, 212, 231, 249, 252, 301, 303, 327, 330, 342, 365, 373, 414, 420], **MOO** [61, 81, 114, 122, 125, 166, 220, 351, 384], **MTS** [138, 310, 388]
Param. Change	**COST** [256, 279, 280, 311], **ENERGY** [28, 49, 209, 331, 331, 332, 365, 366], **PERFORMANCE** [47, 256, 279, 280], **RELIABILITY** [279, 280], **SAFETY** [311]	**DESIGN** [311], **MULTIPLE** [332], **NOT PRESENTED** [256, 279, 280], **PERFORMANCE** [28, 47, 49, 209, 331, 331, 365, 366], **PRECEDENCE** [47]	**MOO** [256, 279, 280], **MTS** [311], **SOO** [28, 47, 49, 209, 331, 331, 332, 365, 366]

Table 2.2: A summary of architecture optimisation aspects

Optimisation Algorithms

From the optimisation point of view, these approaches can be categorised based on optimisation strategy, algorithm and dimensionality in the objective space. Table 2.3 summarises key approaches that have been devised to date.

Algo. Class	Sub Class and References	Constraint Handling
Black-box Stochastic Algorithms	**Evolutionary Algorithms** [6, 29, 54, 56, 60, 61, 73,74,75,76,77,81,97,98,99,100,113,120,122,125, 131, 137, 140, 144, 145, 161, 165, 167, 170, 171, 183, 187, 199, 211, 232, 256, 258, 261, 266, 274, 275, 276, 277, 279, 280, 281, 288, 302, 304, 305, 306, 313, 314, 317, 321, 326, 327, 333, 334, 336, 338, 341, 347, 348, 376, 377, 378, 379, 381, 384, 385, 390, 392, 393, 394, 395, 398, 399, 405, 406, 411, 417, 419]	**Prohibit** [6, 29, 81, 97, 98, 99, 125, 165, 167, 187, 232, 258, 261, 288, 304, 305, 313, 314, 348, 376, 377, 378, 379, 384, 385, 393, 395, 406, 419] **Repair** [60, 140, 326, 394] **Penalty** [56, 60, 61, 74, 76, 100, 120, 122, 131, 137, 170, 171, 199, 211, 258, 275, 281, 327, 336, 341, 347, 392, 399, 405, 406, 411] **N/A** [54,73,75,77,113,144,145,161,183,256,266, 274, 276, 277, 279, 280, 306, 317]
	Simulated Annealing [38, 52, 65, 81, 138, 204, 216, 303, 335, 341, 388, 406, 415, 417]	**Prohibit** [38, 81, 204, 216, 406, 415] **Pennalty** [65, 138, 341, 388, 406] **N/A** [303, 335, 417] **Repair** [52]
	Tabu Search [10, 81, 220, 233, 243, 244, 312, 313, 406]	**Penalty** [10, 243, 244, 312, 406] **Prohibit** [81, 220, 313, 406] **Repair** [233]
	Ant Colony Optimisation [82, 249, 356, 402]	**Prohibit** [249, 402] **Penalty** [356] **N/A** [82]
Heuristics	[1, 4, 10, 15, 16, 18, 28, 47, 49, 51, 52, 83, 87, 95, 114, 115, 136, 142, 143, 163, 166, 196, 198, 201, 202, 203, 207, 209, 212, 215, 226, 231, 252, 265, 271, 273, 292, 295, 297, 310, 324, 330, 331, 353, 357, 369, 373, 375, 380, 407, 414, 416, 418, 420]	**Prohibit** [1, 4, 15, 18, 28, 49, 51, 83, 87, 95, 166, 196, 209, 212, 226, 231, 273, 292, 297, 310, 330, 331, 373, 375, 380, 407, 414, 416] **Repair** [47, 52, 114, 115, 142, 215, 252] **Penalty** [10, 16, 136, 203] **N/A** [143,163,198,201,202,207,265,271,295,324, 353, 357, 369, 418, 420]
Exact Methods	[1, 10, 13, 17, 19, 20, 21, 22, 28, 78, 79, 80, 91, 92, 94, 96, 102, 104, 108, 116, 197, 202, 210, 227, 265, 273, 282, 293, 301, 323, 331, 332, 354, 365, 366, 367, 380, 380, 395, 407, 416]	**Prohibit** [1, 13, 19, 21, 22, 28, 78, 79, 80, 91, 92, 94, 102, 104, 108, 116, 197, 210, 273, 293, 301, 331, 332, 354, 365, 366, 380, 380, 395, 407, 416] **Panelty** [10, 96]

Table 2.3: A summary of architecture optimisation approaches

Exact Algorithms

A set of approaches use *exact algorithms* which guarantees the optimal solution(s). Most of them use brute-force techniques to consider all possible combinations in the input space whereas some use problem-specific constraints and characteristics to traverse through the search space towards the optimal set. However, it has already been proven that most of the architecture design problems in the embedded systems

domain are non-deterministic polynomial-time hard (NP-hard) problems [110],*i.e.* decision problems that are intrinsically harder than those that are solvable in polynomial time by a non-deterministic Turing machine. Consequently, the computational complexity increases exponentially with the problem size, and the applicability of exact algorithms is limited to problems below a certain bound of the problem size.

Problem-Specific Heuristics

Some architecture optimisation approaches use problem-specific heuristics for design space exploration. These approaches belong in the approximative category where the algorithm does not guarantee the discovery of the optimal solution. In contrast to the exact methods which are often inapplicable to larger problems due to the exponential time complexity, approximate optimisation solutions are well-accepted in industrial practices. However, these heuristic approaches are tightly associated with the characteristics of specific problems, and their applicability beyond the conditions in the presentations are hard to generalise. For example, a heuristic can be tailored to solve the deployment architecture optimisation of automotive software, considering the specific nature of constraints. This heuristic may not be applied to solve task-to-module assignment due to differences in constraints, despite the fact that both are assignment problems.

Black-box Stochastic Algorithms

This category (sometimes referred to metaheuristics) refers to stochastic optimisation approaches which make few or no assumptions about the problem being optimised. However they can additionally be *guided* by problem specific knowledge and heuristics. Many of these are nature-inspired algorithms, and have been developed in artificial intelligence disciplines. These algorithms are being applied in different domains and problems. The majority of the architecture design problems have been successfully solved using stochastic approaches, especially evolutionary algorithms.

The review of literature summarised in Table 2.3 confirms that the majority of the architecture optimisation approaches are based on black-box stochastic algorithms. The following examples are the most widely used.

■ *Simulated Annealing (SA).* This is one of the oldest black-box optimisation algorithm applied to combinatorial optimisation problems [340]. SA was originally used as a strategy to escape from local optima (local minima or maxima). The algorithm is inspired by molecular behaviour encountered in the steel hardening process. At high temperatures molecules are free to move around, whereas as the temperature is lowered they are increasingly confined due to the high energy cost of movement. In application to architecture optimisation, a set of architecture solutions are initially generated with a setting of starting temperature. At each iteration of the algorithm, solutions are randomly sampled and modified mimicking the molecular movements. The temperature is lowered throughout the simulation according an annealing schedule where the modification freedom in the architectures is systematically restricted over iterations. Analogous to the finding of a stable structure in annealing process, the goal of the architecture optimisation is manifested as the internal energy of the molecular structure which is minimised during the simulated annealing.

■ *Evolutionary Algorithms (EA).* Evolutionary Algorithms are one of the most applied black-box optimisation algorithms in design space exploration of software architecture context. EAs are population-based algorithms which start with an initial set of solutions and evolves during subsequent iterations. They implement mechanisms that are inspired by biological concepts such as reproduction, mutation, recombination, natural selection and survival of the fittest, to find a final set of solutions considering all of the objectives and constraints. Many variants of EAs have been developed for both single- as well as multi-objective problems.

■ *Ant Colony Optimisation (ACO).* ACO is an optimisation metaheuristic inspired by the foraging behaviour of ants, originally proposed by Dorigo *et al.* [129]. The process mimics the *pheromone* tracks left by ants during their search for food and the return to the nest. It belongs to *constructive* category as the solutions are con-

structed from steps as opposed to changes done in existing solutions in iterative algorithms like EA and SA. Constructive algorithms including ACO, often converge quickly and produce feasible solutions in highly constrained combinatorial optimisation problems [340].

■ *Tabu Search (TS).* Basic idea behind TS is to keep history of the search in order to escape from local optimal as well as to enforce exploratory mechanisms. As an escaping mechanism from local optima, TS imposes restrictions based on a short-term memory of recent solutions. To guide through the search space, it also applies certain strategies mimicking long-term memory processes. TS algorithms have been used in combinatorial optimisation problems including certain architecture optimisation approaches listed in Table 2.3.

A detailed analysis of the optimisation algorithms and their applicability in combinatorial optimisation problems can be found in the survey by Blum *et al.* [340].

Constraint Handling

The optimisation techniques used in architecture optimisation are also distinguished with respect to the way they treat architectural constraints in the design space exploration. The following basic techniques can be found in the architecture optimisation literature.

■ *Prohibit.* One approach to deal with constraints is to prohibit violations. In this technique, the solutions that do not satisfy the design constraints are ignored during the design space exploration. Search is continued until feasible solutions are found.

■ *Penalty.* This a technique used to include constraint-violating architecture solutions into consideration with a negative annotation. Penalty functions are often utilised to compute the level of violation of constraints.

■ *Repair.* In some optimisation approaches, efforts are made to repair infeasible solutions during the architecture optimisation. If a candidate architecture is found to be violating certain constraints, repair attempts are applied. However, this technique

does not grantee to repair a solution, but application of certain repair attempts helps to produce new architecture solutions.

■ *General.* This category covers the optimisation approaches that are abstract and allows the use of different constraint handling mechanisms.

The application of constraint handling techniques has a close connection to the architecture optimisation problem in focus as well as the concrete implementation of the algorithm. They have been studied in the different application contexts as well as in different optimisation algorithms [300, 302]. The constraint handling mechanisms used in the architecture optimisation of embedded systems are listed in the last column of Table 2.3.

2.4 Summary

This chapter presents the background and preliminary literature review on software architecture, architecture-based reliability evaluation and architecture optimisation which is used as the basis for the rest of the thesis. Architecture design of software-intensive systems is a crucial task which influences the life cycle of the system. The stakeholders of a system have various concerns with respect to the prospective system such as reliability, performance or development costs which have to be analysed in early design phases. ISO 42010 international standard presents a conceptual model for the expression, communication and review of architectures of software-intensive systems. The model provides a foundation for architecture-based quality evaluation and optimisation. Several approaches have already been developed to obtain qualitative and quantitative metrics on the quality attributes using information contained in software architecture.

Reliability is one of the most important quality attributes that require consideration in the architecture design of software-intensive systems. With the specific focus of the thesis on reliability motivated from the crucial nature and relevance to the embedded systems domain, the architecture-based reliability evaluation approaches

are described in detail. Several models are used in abstracting reliability aspects from the software architecture with different trade-offs in expressiveness and complexity. Tree structured models such as reliability block diagrams and fault trees can be used to model structural effects of failures to the system's reliability. In addition to the capability to consider structural failure scenarios, Markov chain-based reliability models are being developed that also have the power to include stochastic execution behaviour of the system. However, reliability evaluation using Markov chain-based models involves sophisticated mathematical formalisms. The underlying mathematics is also presented as they will be referenced in the future chapters. Hierarchical and composite reliability analysis methods are discussed with their compromise between different reliability evaluation contexts.

The latter part of the chapter is dedicated to a description and literature review on architecture optimisation. Numerous architecture design choices in the context of embedded software-intensive systems have brought the need for architecture optimisation. A number of approaches have been developed to optimise software architecture decisions considering various quality attributes, optimisation dimensions and design constraints. There are many architecture optimisation approaches in the research literature that aim to solve specific architecture design problems and quality attributes. Stochastic optimisation approaches dominate in use and have successfully been applied in both single and multi-objective optimisation problems with various constraint handling mechanisms.

Overall, this chapter presented the background literature on architecture-based quality evaluation, reliability models and architecture-optimisation. The literature review conducted in this chapter covers much broader scope, where as more specific analysis on related work to research questions is presented in respective chapters. Next chapters use this understanding as the basis, and definitions and models presented in this chapter is referred. Gap analysis and research questions in relation to specific aspects are presented in the following chapters resting on the conception given in this chapter.

Chapter 3

Motivating Example

The previous chapter relates the background of the prevalent architecture-based reliability evaluation and optimisation. Based on the foundation, this chapter motivates the objectives of the thesis elaborating the research gaps with respect to the current practices using an example case study. The case study designates a specific deployment problem from the automotive industry, which aims to map the components of two software-controlled automotive subsystems (services), namely the *Anti-lock Brake System (ABS)* and *Adaptive Cruise Control (ACC)* [158, 188, 288], to a network of electronic control units (ECUs). The same case study is referenced and re-used for the illustration of concepts throughout the thesis.

Following the description of the case study, a state-of-the-art optimisation method is presented where point estimates are used with an optimisation algorithm to identify deployment solutions that have a good overall reliability for both of the subsystems. However, it is shown that even a small change to the point estimates leads to a significant drop in the predicted quality of the solutions. The explicit issues with the use of point estimates in architecture-based quality evaluation and optimisation are further analysed with a set of experiments.

3.1 System Description and Annotations

The illustrative case study presented in this section has been inspired from already published articles [158,188,287,288,289]. The formulation of the automotive software deployment follows the model presented by Malek *et al.* [272, 283], who propose an approach to solving the deployment optimisation problem under very general settings (irrespective of the specific quality attribute). For the illustration, the reliability of the two automotive services are considered as the quality attributes to be optimised.

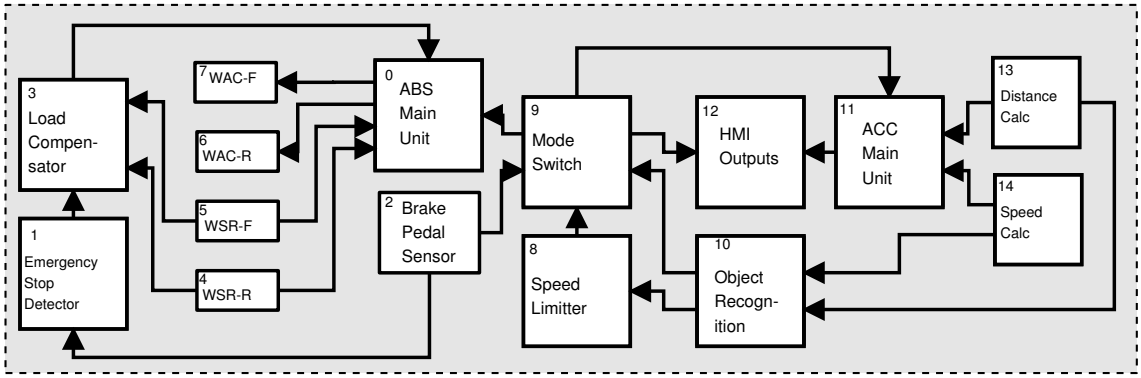
Anti-lock Brake System (ABS)

An ABS is currently used in most modern cars to minimise hazards of skidding and loss of control due to locked wheels during braking. Proper rotation during brake operations allows better manoeuvrability and enhances the performance of braking. The blocks labelled from 0 to 7 in Figure 3.1a denote the software components and their interactions pertaining to the ABS service. The *ABS Main Unit* is the major decision making unit regarding the braking levels for individual wheels, while the *Load Compensator* unit assists with computing adjustment factors from the wheel load sensor inputs. Components 4 to 7 represent the transceiver software components dedicated to each wheel which communicate with sensors and brake actuators. *Brake Pedal Sensor* is the software component that reads from the pedal sensor and sends the data trough the *Emergency Stop Detection* software module.

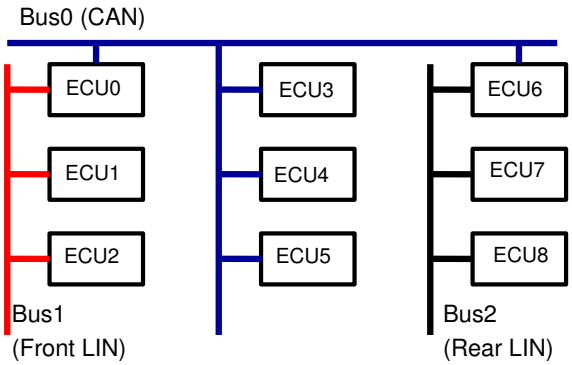
Adaptive Cruise Control (ACC)

The Adaptive Cruise Control (ACC), or intelligent cruise control is one of the driver assistance systems that modern automobiles are commonly equipped with. One important advantage of ACC over conventional cruise control is that a car fitted with ACC slows down automatically when approaching a slower moving vehicle ahead, and then follows the slower vehicle at a set distance. Once the road ahead is clear

again, the ACC accelerates the car to the set cruising speed. In this way, an ACC integrates a vehicle harmoniously into the traffic flow. The main software components used by the ACC system and their interaction diagram are presented as components 8 to 14 of Figure 3.1a. ACC service trigger can occur at software components that communicate with the sensors and user inputs. In the ACC system, the *Speed Limit*, *Object Recognition*, *Mode Switch* and *Brake Pedal Sensor* components contribute to the triggering of the service. The data captured from the sensors triggers the *ACC Main Unit* to communicate with the actuators. *HMI outputs* is a component that gives information to the driver through the vehicle computer display, *e.g.*, information about the vehicle state and latest request. Note that the software components *Brake Pedal* and *ABS Main Unit* are involved in both ABS and ACC services.



(a) Software components and their interactions



(b) Hardware platform

Figure 3.1: Architecture of the ABS/ACC composite system

Hardware Platform

The hardware model of the automotive system which is used to deploy the software components comprises a distributed set of certified Electronic Control Units (ECU) having different capacities of memory, processing power, access to sensors, *etc.* ECUs are assumed to have a fixed and deterministic scheduling, which is a technology used with Time-Triggered Architectures (TTA) in embedded systems in order to maintain the predictability of internal behaviour [206, 234]. With this scheduling strategy, a fixed schedule is determined when the software components are allocated to ECUs. Each component is given a specific time frame within which it must complete the execution. The schedule is done in round-robin, *i.e.* each component gets its own time frame in a circular fashion. The sum of all the execution time slots in the ECU is called the *schedule length*.

The communication among ECUs is achieved by buses as depicted in Figure 3.1b. Automotive systems contain few types of buses which are specifically built for different purposes like control data communication, sensor/actuator interconnection and multi-media data transfer. The figure contains three bus systems consisting of a main CAN (Control Area Network) bus and two LIN (Local Interconnect Network) bus systems for front and rear sensor/actuator connectivity.

In the rest of the thesis, the following annotations and abbreviations are used to denominate the parameters of the elements in the software and hardware architecture. Parameter values for the ABS/ACC composite system are given in Table 3.1 using the conventional specification with numerical values (point-estimates).

Software Components

(a) **Size (sz):** memory size of a component when deployed on to an ECU; expressed typically in KB (kilobytes).

(b) **Workload (wl):** computational requirement of a component for each service; expressed in MI (million instructions).

(c) **Initiation probability (q_0):** the probability that a service starts from the component.

Component Interactions

Specified for a link from component $\mathcal{C}_i$ to $\mathcal{C}_j$.

(a) **Data size (ds):** the amount of data transmitted from software component $\mathcal{C}_i$ to $\mathcal{C}_j$ during a single communication event; expressed in KB (kilobytes).

(b) **Next-step probability (p):** the probability that the execution of component $\mathcal{C}_i$ ends with a call to component $\mathcal{C}_j$.

Electronic Control Units (ECUs)

(a) **Capacity (cp):** memory capacity of the ECU, expressed in KB (kilobytes). The sum of memory requirement of all software component allocated to an ECU should be smaller than the capacity of the ECU.

(b) **Processing speed (ps):** the instruction-processing capacity of the ECU; expressed in MIPS (million instructions per second) [148]. This is used to calculate the execution time, which is a function of the processing speed of the ECU and the computation workload of the service.

(c) **Failure rate (fr):** failure rate (typically written as λ) of the exponential distribution [23, 59] that characterises the probability of a single ECU failure.

Buses

(a) **Data rate (dr):** the data transmission rate of the bus; expressed in KBPS (kilobytes per second). This is used to calculate the time taken for data transmission which is a function of the data rate of the bus and the amount of data transmitted during the communication.

(b) **Failure rate (fr):** failure rate of the exponential distribution characterising the data communication failure of each bus.

Comp. ID	sz (KB)	q_0 ABS	q_0 ACC	wl_{ABS} (MI)	wl_{ACC} (MI)
0	512	0.1	0	1.2	1.2
1	128	0	0	0.6	0
2	128	0.3	0.2	0.4	0.4
3	256	0	0	1	0
4	128	0.3	0	0.4	0
5	128	0.3	0	0.4	0
6	128	0	0	0.4	0
7	128	0	0	0.4	0
8	128	0	0	0	0.8
9	256	0	0	0	1.1
10	512	0	0	0	1.2
11	512	0	0	0	2.1
12	256	0	0	0	0.9
13	256	0	0.4	0	0.9
14	256	0	0.4	0	0.9

(a) Software Components

ECU ID	cp (KB)	ps (MIPS)	fr (h^{-1})
0	512	40	4×10^{-6}
1	512	22	4×10^{-6}
2	512	45	2×10^{-6}
3	512	22	2×10^{-6}
4	1024	22	1×10^{-5}
5	512	22	1×10^{-6}
6	1024	110	8×10^{-7}
7	512	110	2×10^{-7}
8	512	22	9×10^{-7}

(b) ECUs

Trans $c_i \rightarrow c_j$	$p(c_i, c_j)$ ABS	$p(c_i, c_j)$ ACC	ds_{ABS} (KB)	ds_{ACC} (KB)
$0 \rightarrow 6$	0.5	0	2	0
$0 \rightarrow 7$	0.5	0	2	0
$1 \rightarrow 3$	1	0	2	0
$2 \rightarrow 1$	1	0	2	0
$2 \rightarrow 9$	0	1	0	2
$3 \rightarrow 0$	1	0	2	0
$4 \rightarrow 0$	0.7	0	1	0
$4 \rightarrow 3$	0.3	0	2	0
$5 \rightarrow 0$	0.7	0	1	0
$5 \rightarrow 3$	0.3	0	2	0
$8 \rightarrow 9$	0	1	0	1
$9 \rightarrow 0$	0	0.1	0	4
$9 \rightarrow 11$	0	0.4	0	2
$9 \rightarrow 12$	0	0.5	0	1
$10 \rightarrow 8$	0	0.6	0	2
$10 \rightarrow 9$	0	0.4	0	2
$11 \rightarrow 12$	0	1	0	3
$13 \rightarrow 10$	0	0.5	0	1
$13 \rightarrow 11$	0	0.5	0	1
$14 \rightarrow 10$	0	0.5	0	2
$14 \rightarrow 11$	0	0.5	0	2
$15 \rightarrow 17$	0	0	0	0
$16 \rightarrow 17$	0	0	0	0
$16 \rightarrow 18$	0	0	0	0
$17 \rightarrow 18$	0	0	0	0
$17 \rightarrow 19$	0	0	0	0
$18 \rightarrow 17$	0	0	0	0
$18 \rightarrow 19$	0	0	0	0

(c) Component Interactions

BUS ID	dr (KBPS)	fr (h^{-1})
0	128	3×10^{-5}
1	64	1.2×10^{-4}
2	64	4×10^{-5}

(d) Buses

Table 3.1: Parameters of software and hardware elements in the ABS/ACC system

3.2 Reliability Evaluation of Deployment Candidates

3.2.1 Failure Model

In an embedded system, failures may occur on both software and hardware levels. The selection of a failure model is also driven by the goal of the optimisation problem in focus, which is the *optimisation of system deployment* (software components to hardware nodes) with the goal of *maximising the reliability of the two services*. Hence the selected failure model has to capture the dependency between the possible failures of the system and the deployment. The majority of software errors in the system is likely to be manifested the same way on each deployment, independently of the hardware. On the other hand, it is meaningful to abstract away from the hardware-independent software reliability, since this portion of unreliability is in a very similar or even the same way projected to all deployment candidates, and therefore is unlikely to influence the optimisation process [289].

For the purpose of this work, we distinguish two types of failures as follows.

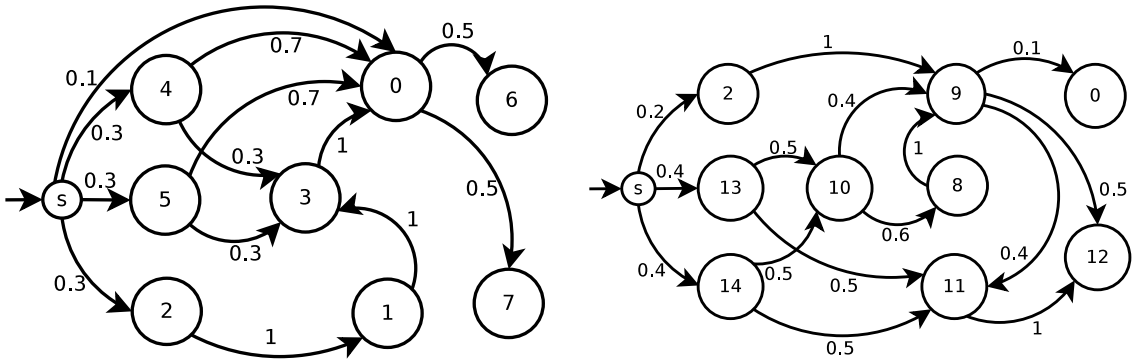
■ *Execution failures.* Failures may occur in the ECUs during an execution of a software component, which affects the reliability of the software modules running on that ECU. Combined with the fixed and deterministic scheduling strategy, any failure that happens in an ECU while a software component is executing or queued leads to a service execution failure.

■ *Communication failures.* Failure of a data communication channel (bus) when a software component communicates with another over the bus can cause a failure in the service that depends on this communication.

Both failure types are considered into the failure model, and the failure behaviours are integrated to the computation of reliability-related quality of a candidate component deployment.

3.2.2 Probabilistic Quality Model

The architecture decision problem in focus is to find the deployments of the software components to the ECUs which maximise the reliability of the two services, *i.e.* ABS and ACC. For the service reliability evaluation at design-time, a widely used Discrete Time Markov Chain (DTMC)-based reliability evaluation model [175, 182, 289] is used. The behaviour of the two automotive systems can be represented by absorbing DTMCs [175]. The DTMCs can be constructed from the behavioural specification of the system such that a node represents the execution of one software component and arcs denote the transfer of execution from one component another. *Super-initial* nodes [404] are added for each service to represent the execution start, and arcs are added from those nodes annotated with relevant execution initialisation probabilities (q_0). Figure 3.2 illustrates the constructed DTMCs for the two automotive services described in Section 3.1 and the node labels correspond to the component ids in Figure 3.1a. The transition probabilities from the super-initial nodes and among other states have been obtained from Table 3.1 as well as execution initialisation probabilities and next step probability values respectively.



(a) ABS (b) ACC

Figure 3.2: DTMCs for service reliability evaluation

Expected number of visits of a component

Let $v_c : C \rightarrow \mathbb{R}_{\geq 0}$, quantify the expected number of visits of a component (or subsystem) during system execution. Note that $v_c(c)$ corresponds to the expected number of visits of state c_i in the underlying DTMC, *i.e.* $E[X(c_i)]$. Hence, it can be written in the following format:

$$v_c(c_i) = q_0(c_i) + \sum_{j \in \mathcal{I}} (v_c(c_j) \cdot p(c_j, c_i)) \tag{3.1}$$

This can be computed by solving the set of simultaneous equations described in Section 2.2.2 of Chapter 2.

Expected number of visits of a communication link

Similar to the expected number of visits of software components, for each link $l_{ij} = (c_i, c_j)$, let $v_l : C \times C \rightarrow \mathbb{R}_{\geq 0}$ denote the expected number of occurrences of the transition (c_i, c_j) in the underlying DTMC. To compute this value, the work of Kubat *et al.* [241], which presented the calculation of expected number of visits of system components can be extended to communication links [289]. In the extension, communication links are understood as first-class elements of the model, and view each probabilistic transition $c_i \xrightarrow{p(c_i,c_j)} c_j$ in the model as a tuple of transitions $c_i \xrightarrow{p(c_i,c_j)} l_{ij} \xrightarrow{1} c_j$, the first adopting the original probability and the second having probability $= 1$. Then the expected number of visits of a communication link can be computed as,

$$v_l(l_{ij}) = 0 + \sum_{x \in \{i\}} (v_c(c_x) \cdot p(c_x, l_{ij})) \tag{3.2}$$

$$= v_c(c_i) \cdot p(c_i, c_j) \tag{3.3}$$

since the execution is never initiated in a link l_{ij} and the only predecessor of link l_{ij} is component c_i.

3.2.3 Model Evaluation

For a single service s_k, failure rates of *execution elements* can be obtained from the ECU parameters, and the time taken for the execution is defined as a function of the software-component workload and processing speed of its ECU. Similar to the models used in previous work [23, 288, 289], service reliabilities consider both ECU and communication link failures. In detail, the reliability of a component c_i that is used in service s_k with the deployment $d : C \rightarrow ECU$ can be computed by Equation 3.4 where $d(c_i)$ denotes the ECU allocation relationship of component c_i.

$$R_i^d(s_k) = e^{-fr(d(c_i))\cdot\frac{wl(c_i,s_k)}{ps(d(c_i))}} \tag{3.4}$$

A similar computation can be employed for the reliability of *communication elements* [289], which are characterised by the failure rates of hardware buses, and the time taken for communication, defined as a function of the buses data rates dr and data sizes ds required for software communication. Therefore, the reliability of the communication between component c_i and c_j for service s_k under deployment d is defined by Equation 3.5.

$$R_{ij}^d(s_k) = e^{-fr(d(c_i),d(c_j))\cdot\frac{ds(c_i,c_j,s_k)}{dr(d(c_i),d(c_j))}} \tag{3.5}$$

The expected number of visits for each node in the DTMC (v_i) can be computed by solving the DTMC [289]. Using the v_i values, the expected visits for the communication elements($v(l_{ij}, s_k)$) are also computed [289]. Based on the relationships obtained in equation(3.4) and (3.5), the reliability of a single service s_k under deployment $d \in D$ is calculated according to Equation 3.6

$$R^d(s_k) \approx \prod_{i\in\mathcal{I}} R_i^d(s_k)^{v_c(c_i,s_k)} \cdot \prod_{i,j\in\mathcal{I}} R_{ij}^d(s_k)^{v_l(l_{ij},s_k)} \tag{3.6}$$

where $v_c(c_i, s_k)$, $v_l(l_{ij}, s_k)$ refer to the expected number of visits for software components and communication links which are computed as described in Section 3.2.2.

3.3 Architecture Optimisation with Non-dominated Sorting Genetic Algorithm

The above described architecture-based reliability evaluation model can be used to quantify reliability of two automotive services for a given candidate deployment. This section presents the architecture optimisation aspect of the problem. The goal of the multi-objective deployment optimisation problem in focus can be restated as *finding the approximate set of component deployment solutions* $A^* \subseteq A$ *that represent a trade-off between the conflicting objectives in* $Q : A \rightarrow \mathbb{R}^2$ (*i.e.* the reliabilities of the ABS and ACC services), *and satisfy the set of design constraints* Ω. In this case study, Ω represents the memory constraints, *i.e.* the memory requirement for all software components allocated to an ECU should not exceed its capacity (*cp*). Different algorithms can be used for solving the optimisation problem. In this illustration, *Non-dominated Sorting Genetic Algorithm (NSGA)-II* [371] is employed, which is one of the most effective multi-objective optimisation algorithms for combinatorial problems including component deployment [117].

For the optimisation process, *NSGA* uses an initial *population* of *chromosomes* consisting of *alleles*. In relation to the deployment model used in this case study, each *allele* in a *chromosome* represents an allocation of a component $c_i \in C$ to an ECU $u_j = d(c_i) \in U$. The initial population of deployment architectures is generated randomly. The *crossover* and *mutation* operators are used to create new solutions by recombining existing ones or changing the allocation of a single component to another ECU. The three genetic operators of the evolution process, *i.e. crossover*, *mutation* and *selection*, can be adapted to the component deployment as follows.

■ *Selection.* The selection operator ranks all solutions, and probabilistically selects the better solutions for the next iteration. The ranking is based on the level of an individual solution's non-domination. In a maximisation problem a solution a^* is non-dominated if there exists no other a such that $Q(a) \geq Q(a^*)$ for all objectives, and $Q(a) > Q(a^*)$ for at least one objective. In other words, if there exists no other

feasible variable a which would be superior in at least one objective while being no worse in all other objectives of a^*, then a^* is said to be non-dominated. The set of all non-dominated solutions is known as the non-dominated set. First, the non-dominated solutions present in the population are identified and assigned rank 0. These solutions will constitute the first non-dominated front in the population, which will be ignored temporarily to process the rest of the population in the same way, finding the solutions which belong to the second non-dominated front. These solutions are assigned with the next rank value, *i.e.* 1. This process is continued until the entire population is classified into separate fronts. A mating pool is then created with solutions selected from the population according to the fitness values that has been assigned to them during the ranking process. The solutions with a lower rank have a higher chance of being selected to be part of the mating pool than the ones with a higher rank. *Binary tournament* can be used in this purpose where one with the lower rank of two randomly selected candidates to be added into the mating pool. The mating pool will then serve for the random selection of the individuals to reproduce using crossover and mutation. After each iteration, a fixed population size is maintained by eliminating weakest solutions (with high non-dominance ranking) of the composite population of parents and offsprings.

■ *Crossover.* Crossover is the creation of new solution candidates $d'_i, d'_j \in D$ from two parents $d_i = [u_{i_1}, u_{i_2}, ..., u_{i_n}]$ and $d_j = [u_{j_1}, u_{j_2}, ..., u_{j_n}]$ coming from existing population by recombining the mapping of components, *i.e.* for a random k: $d'_i = [u_{i_1}, ..., u_{i_{k-1}}, u_{j_k}, ..., u_{j_n}]$ and $d'_j = [u_{j_1}, ..., u_{j_{k-1}}, u_{i_k}, ..., u_{i_n}]$.

■ *Mutation.* Mutation produces a new solution d'_i from existing d_i by switching the mapping of two components, *i.e.* for randomly selected k, l: $d'_i = [u_{i_1}, ..., u_{i_l}, ..., u_{i_k}, ..., u_{i_n}]$ while the original is $d_i = [u_{i_1}, ..., u_{i_k}, ..., u_{i_l}, ..., u_{i_n}]$.

3.4 Analysis of Solutions Obtained Using Point Estimates

The aforementioned architecture evaluation and optimisation approach has been applied to the deployment architecture optimisation problem. The complete process of deployment architecture optimisation presented above is based on design-time parameter estimates, *e.g.* the values presented in Table 3.1. This section presents an analysis of the architecture solutions produced by the optimisation.

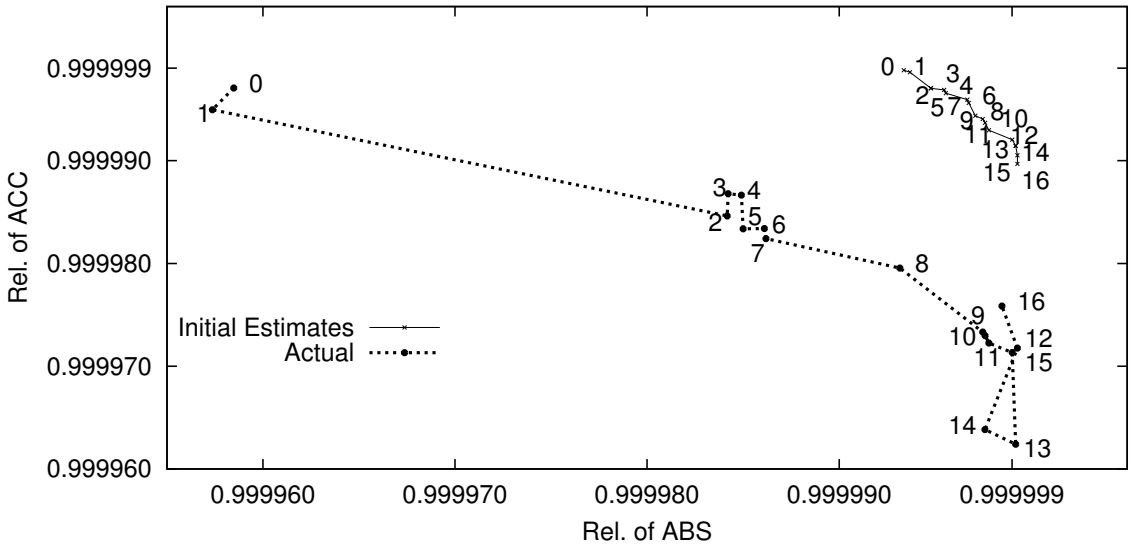


Figure 3.3: Pareto front of point estimates and respective reliabilities when estimates are changed

The solid line in Figure 3.3 represents the set of non-dominated deployment solutions with respect to the trade-offs in the reliabilities of the two services, obtained with point estimated parameters. However, it is emphasised that those estimates are subject to uncertainty. As a practical example, consider three ECUs (0,1,2) located close to the engine, where the operating temperature may vary between $25 - 60^oC$. The failure rates of ECUs are significantly affected by the ambient conditions [59]. As a result of this temperature variation, the failure rate may vary up to 40 times from the original estimate [119]. The dotted line in the figure represents the service reliabilities of the originally obtained solutions if the failure rates are increased only

by 5 times. The new results confirm that optimal architectures obtained from point estimates are misleading. As the estimates are subject to uncertainty, the reliability values are not trustworthy and the optimisation results can be suboptimal. Solutions provided by the conventional approach are not necessarily a set of non-dominated solutions (*e.g.* solutions 1,2,5, 9-11 and 13-15 are dominated by the others). It should also be noted that, this analysis has only used one subset of uncertain parameters (*i.e.* ECU failure rates) as an example. In reality, many parameters including software component reliabilities and usage profile estimates are also uncertain, which would make the point-estimate-based methods even less applicable.

3.5 Gap Analysis

The analysis of the results on the automotive case study has shown that the use of point estimates in architecture optimisation can result in misleading information. Motivated with this illustration, this section further investigates the root causes of the problem at hand. The relevant gaps in the state-of-the-art architecture-based reliability evaluation and optimisation techniques are discussed in detail.

3.5.1 Non-linearity and Mathematical Complexity of Reliability Evaluation Models

A number of models and mathematical formulations have been developed for architecture-based reliability evaluation of software-intensive systems. In a broader sense, architecture-based reliability evaluation can be considered as a mathematical function from architecture annotations and configurations to reliability metrics such as mean-time to failure (MTTF), failure rate or failure probability. The input of the function includes estimated parameters including execution initialisation probabilities, transition probabilities among software components, hardware failures rates and software failure rates. However, as also presented in Section 2.2, the use of simple aggregation functions or linear mathematical formulae (Table 2.1)

ignores certain important information on structural and behavioural aspects of the system (*e.g.* dependences of failures among components). Many researchers including Gokhale *et al.* [172], Goševa-Popstojanova *et al.* [182] and Lyu [270] have pointed out that an accurate prediction of reliability of a software-intensive system considering its operational and failure behaviour require sophisticated mathematical formulations. Markov chains have been successfully adopted and used for comprehensive reliability analysis [172, 182, 289, 391]. With the composition of structural and behavioural aspects into Markovian reliability models, the reliability metric becomes a composite effect of many parameters rather than linear function which can be decomposed into individual relationships. As described in Section 2.2, reliability evaluation models require a series of complex mathematical operations such as application matrix and vector operators. Hence, the mathematical function from the input parameters to the reliability metric is not a linear or simple aggregation function. As a result, it is often hard for an architect to back-propagate an effect on the reliability metric to individual parameters. The following simple experiment further illustrates this aspect.

Experiment ▷ We consider a randomly generated DTMC-based reliability model and observe how reliability changes with respect to different input parameters (transition probabilities and execution initialisation probabilities). In order to observe this, parameters are selected one by one and their values are varied across a possible range (*i.e.* transition probabilities $[0, 1]$, component failure rates $[0.1 \cdot fr, 10 \cdot fr]$) while keeping all the other ones fixed (parameter sweep). The variation of reliability *w.r.t.* the change in parameter are captured as *sensitivity graphs* for individual input parameters.

Figure 3.4 depicts some of the sensitivity graphs corresponding to different input parameters. It can be seen that certain parameters have either positive (*e.g.* 3.4a,3.4f) or negative (*e.g.* 3.4c,3.4d, 3.4e) relationships for some parameters (not necessarily linear). However, the sensitivity graphs of certain parameters (*e.g.* 3.4b) are non-monotonic, *i.e.* the worst reliability is observed at a specific value of a pa-

rameter rather than in its lower/upper margins. Therefore, it is hard for a software systems architect to determine appropriate input parameters for different concerns. For instance, the reliability estimate of ABS system in an automotive system have to be carried out more pessimistically as it is a safety-critical system. When it comes to providing point estimates for the parameters, the architect faces the challenge of propagating the pessimism into the parameter estimation, which is often not possible due to the irregular characteristics. It is to be noted that the sensitivity graphs in Figure 3.4 correspond to a single architecture candidate and one service reliability. Furthermore, the sensitivity analysis is based on considering each parameter separately. The actual effects of parameters to reliability values are even more complicated as the quality metric is a composite effect of many parameters. Hence, analysing these effects for each and every combination of parameters before producing the point estimates is infeasible for practical problems.

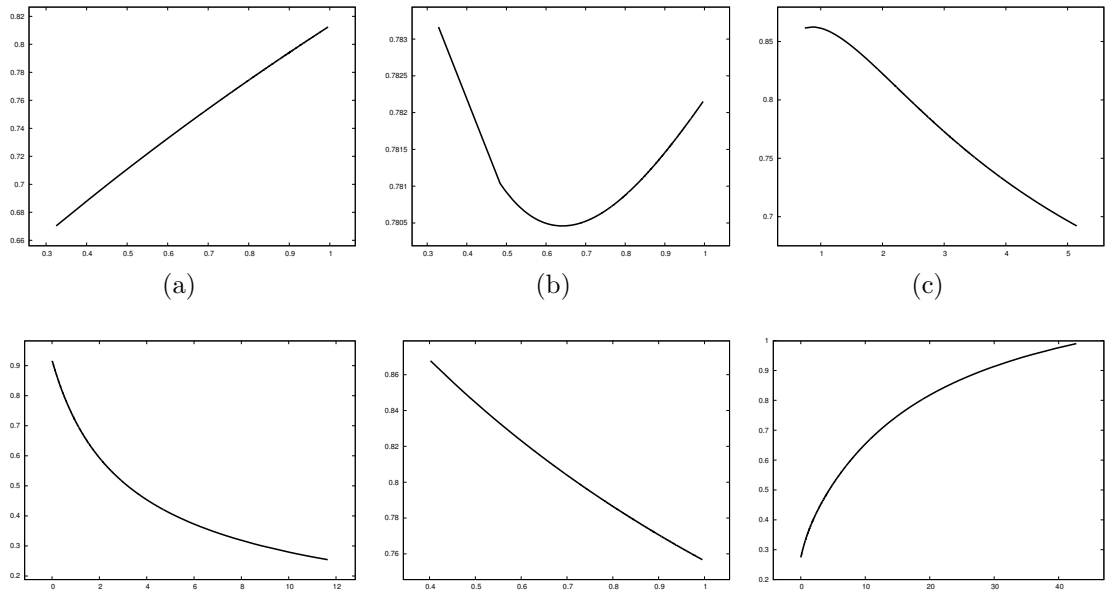


Figure 3.4: Sensitivity of service reliability to different input parameters. In each graph, the x-axis represents the parameter values and the y-axis denotes reliability

◁

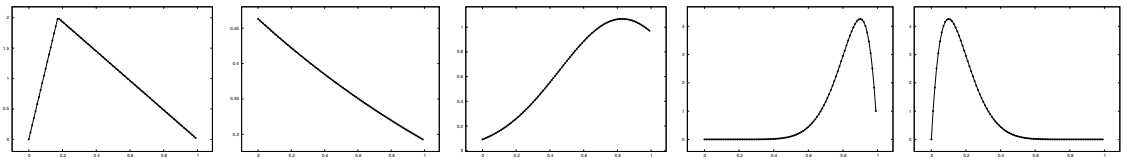
3.5.2 Non-normal Distributions

Parameters of architecture-based reliability models correspond to various elements contained in software architecture. They characterise certain aspects of the prospective system, users or operational environment. Hence, uncertainty associated with the input parameters can have different characteristics, which are often dissimilar and irregular. As a reliability metric (*e.g.* MTTF, failure rate) is computed using these values, the nature of the input parameter distribution affects the distribution of the quantitative metrics produced in architecture-based reliability evaluation. Therefore, a reliability metric can take irregular characteristics with respect to the uncertainty inherited from the input parameters. It is emphasised that the notion of *characteristics* of the reliability metric is different from the common understanding of exponential distribution of failures in the models. More intuitively, the focus here in such model is on the distribution of a failure rate (λ) calculated from the model, under the exponential distribution assumption. This characteristic has not been included in the prevailing architecture-based reliability models and evaluation approaches. The inherent normality assumption in many of the architecture-based reliability evaluation approaches can cause misleading results. For instance, if mean values are provided for all input parameters, the value produced from the reliability model does not necessarily represent the mean value of the metric. The following experiment further elaborates this issue.

Experiment ▷ In order to represent the diversity of characteristics associated with the input parameters in architecture-based reliability models, let some of the parameters in a randomly generated DTMC-based reliability model take variations given in Figure 3.5. Assume that each graph represents a variability of an uncertain parameter and is given as a *probability distribution*. First, let us use the mean values of each parameter as the input for reliability evaluation presented in Section 3.2. When mean values are used, the reachability at the absorbing software component ($S_{1,n}$) given in Section 2.2.2 is 0.37. The probability of reaching the absorbing state without a failure is used as the reliability metric (for further details on the computation

of this, please refer to Section 2.2.2).

Secondly, we sample from each distribution and use the sample values to compute the instantaneous absorption probability. Figure 3.6 depicts the histogram obtained using 10,000 such samples. It can be seen that the histogram does not correspond to a normal distribution (more towards Gamma distribution). The sample mean of the reaching probability is 0.51 which is a 37% deviation from the point estimated value. If the normality assumption is held, these two values should be equal. From this experiment, it can be concluded that the compromises made on point estimates not necessarily correspond to the actual compromises in the reliability of the architecture solutions.



(a) (b) (c) (d) (e)

Figure 3.5: Example characteristics of input parameters in architecture-based reliability evaluation. The x-axis denotes the parameter value and the y-axis depicts probability density

◁

3.5.3 Uncertain Optimisation Goals

A common feature of all architecture optimisation approaches described in Section 2.3 which are capable of catering for NP-hard design problems is that the optimisation goals are used as the criterion to guide the search through the design space. When the objective values are probabilistic properties such as reliability, each architecture alternative is evaluated using the probabilistic models constructed for each quality attribute. However, as consequences of issues described in 3.5.1 and 3.5.2, the use of point estimates for input parameters in architecture-based reliability evaluation does not sufficiently capture the uncertain characteristics of the solutions. Hence, the use of point estimates as optimisation objectives can

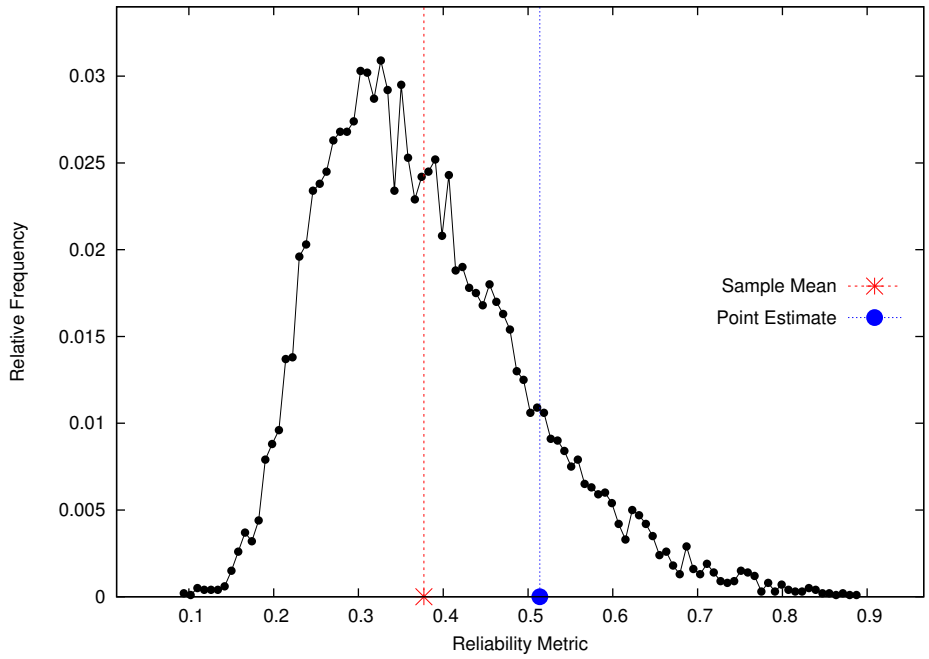


Figure 3.6: Histogram of the reliability metric obtained from 10,000 parameter samples

misguide the optimisation algorithm. Since the optimisation algorithm solely depends on the objective values for design space exploration, the solutions produced from architecture-optimisation can be suboptimal under uncertain input parameters. Even though the solutions can later be assessed for their performance under uncertain conditions, the optimisation process becomes less trustworthy. Better solutions can be missed in the design space exploration as the optimisation algorithm is guided by the point estimates.

Based on these deliberations, the next chapter formulates research questions and describes the SCOUT approach which can overcome these issues.

Chapter 4

Research Methodology

The previous chapters presented the background on software architecture, reliability evaluation and optimisation followed by a spotlight on emerging issues with the use of point-estimated parameters in the current practices of architecture optimisation. Accordingly, the goal of this research is *"to develop a method which can consider heterogeneous uncertainties in design-time parameter estimates and explore the architecture design space to find candidate architectures that are (near-)optimal in terms of a set of reliability goals with a confidence guarantee against the uncertainties"*. Identifying specific gaps in the existing research knowledge, this work recognises two main subgoals that have to be achieved in order to reach the overall goal.

1. *"To develop a method for architecture-based reliability evaluation under uncertainty"*. Prerequisites for this goal can be further refined into three subgoals;
   (a) capturing heterogeneous uncertainty associated with parameters used in architecture-based reliability evaluation,
   (b) evaluating reliability models under uncertainty and
   (c) representing of reliability properties considering the uncertainty.

2. *"To develop a method that can optimise software architectures considering the aspects related to uncertainty and find architecture solutions which are immune to the uncertainty associated with estimated parameters"*. This goal entails the

optimisation aspect of the overall research aim. The goal can be further decomposed into two subareas;

(a) modelling the problem of software architecture optimisation with uncertainty and

(b) integrating of optimisation algorithms.

The hierarchy of research goals is depicted in Figure 4.1. A goal in the upper level of the diagram is decomposed into subgoals listed in its lower-level branches. The structure is organised to represent that achievement of a goal is fulfilled when all the subgoals of that item are achieved.

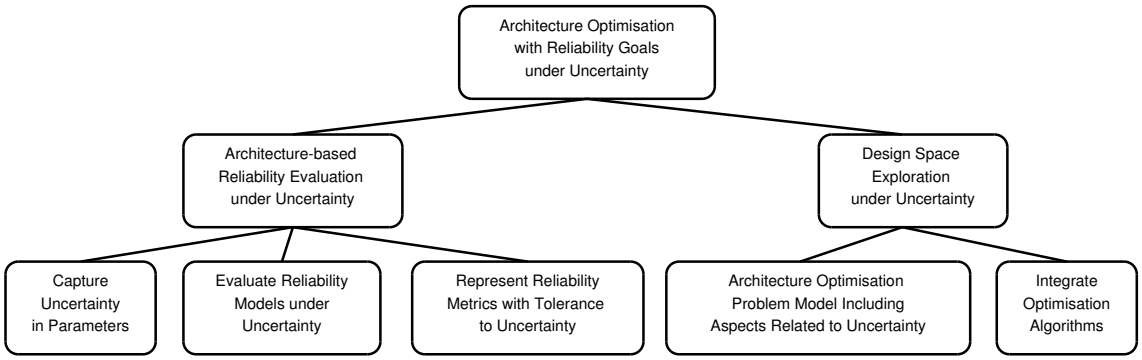


Figure 4.1: Goal structure of the research

The research methodology of this work follows the software engineering research methods characterisation presented by Shaw [362]. The next section formulates refined research questions that are addressed in the thesis, followed by an overview of the SCOUT approach which delivers a solution to fill the identified research gaps. Strategies used to validate the contributions in this thesis are discussed at the end of the chapter.

4.1 Refined Research Questions

Explicit formulation of research questions helps organise the research activities, precisely present the novelty of a solution and validate the achievement of the research goals through the solution [168, 362]. The research goals so far has been presented

on a conceptual level and described in abstract terms. Considering the relevant gaps in the current literature and the scope of this work, the refined research questions that are addressed in the thesis are as follows.

RQ 1. *How to evaluate reliability at the architecture design phase considering heterogeneous uncertainties in the design-time parameter estimates?* Different aspects of this research question are decomposed into the following research subquestions and addressed in this work.

RQ 1.1 *How can heterogeneous information on uncertainty be incorporated into the specification of architectural elements?* The nature of uncertainty in different parameters of software architecture is heterogeneous and diverse. Some parameters like the failure rate of hardware components can have a dependency on the operational environment, and the failure rates have to be derived from uncertainties in the environment. On the other hand, the failure rates of software components depend on factors like the amount of testing, maturity of the design process and the complexity of the algorithms contained in the component. The nature of uncertainty in the usage profile of the prospective system exhibits significantly different characteristics which depend on the uncertain user activities. Difficulty of obtaining these parameters in practice have already been identified [11, 343] and dedicated research efforts emerge on parameter estimation for architecture based quality models [85, 343]. As also explained in gap analysis (Section 3.5), the quantitative metrics obtained from the architecture-based reliability evaluations strongly depend on the parameter values. Hence, it is important to identify and capture this disparate information during the specification of architectural elements. A flexible and powerful method is required to effectively incorporate the notion of uncertainty into architecture description.

RQ 1.2 *How can the reliability of a software architecture be quantified when the input parameters are subject to uncertainty?* As we have discussed in Chapter 2, the architecture-based evaluation of probabilistic properties are performed

by probabilistic models which use the parameters annotated to architectural elements. Considering the non-linearity and mathematical complexity of the reliability evaluation models, this research question can be further decomposed into the following.

a) *How can reliability values be obtained from the probabilistic model evaluation, when the model parameters are uncertain?* In an approach to consider uncertainty in the architecture evaluation, the complex probabilistic evaluation models described in Chapter 2 have to be evaluated in the presence of uncertainty in the input parameter space. Furthermore, the uncertainty which exists in the input parameter space (*e.g.* uncertainty of component reliability, transition probability *etc.*) have to be propagated to the quality metric of interest (*e.g.* uncertainty of Mean-Time to Failure of the system). Therefore, this question requires a method to evaluate the complex probabilistic models in the presence of uncertain model parameters.

b) *In the presence of uncertainty, how can the reliability metrics be estimated with the required level of accuracy?* When the notion of uncertainty is transformed from the architecture specification to the reliability evaluation model, the quality attributes themselves become uncertain. Therefore, the attribute values exhibit a variation across a range rather than resulting in a single numerical value. The reliability metrics have to be *estimated* with respect to the variation. However, the architecture-based reliability evaluation is a crucial element in systems design, and the accuracy of the estimation should be predefined. A new approach should be able to satisfy the accuracy requirements in the standards and recommended practices.

c) *How can the estimation process be automated?* To make an approach applicable to diverse problem settings and sizes of different problem instances, a generic quality estimation process has to be introduced. In

order to achieve the overall goal of the research, the uncertainty analysis process have to be combined with automated architecture optimisation process. Therefore, this question seeks an decision criterion which can automatically control the accuracy-time trade off in the uncertainty analysis.

RQ 1.3 *How can reliability attributes with different degrees of tolerance to uncertainty be accommodated?* In the context of architecture-based prediction of probabilistic quality attributes, conventional quality evaluation models can be used to obtain quantitative metrics that describe the quality of an architecture. However, the uncertainty associated with input parameters propagates through the quality evaluation model and the quality metrics also become variable. For example the architecture-based quantification of reliability of a candidate software architecture will not be a single reliability value, but a variable quantity. This fluctuation of the quality metric represents an important aspect of an architecture which summarises the impact of all uncertainties relevant to the specific quality attribute. Furthermore, the significance of this inconsistency has to be treated differently in different domains and applications. For example, in safety-critical applications, the ambiguity of reliability prediction is often treated pessimistically, whereas in some non-critical web applications, fluctuations can be averaged. Therefore, when referring to a quality attribute of a candidate architecture, it is important to include the possible variation of the quality attribute inherited from input parameters. Hence, there is a need for an approach to represent the attribute in combination with its variability, in addressing RQ 1.

RQ 2. *How can an architecture optimisation method be developed for automated search for better architectures in the presence of uncertainty?* This research question addresses the optimisation aspect of the overall problem, and the following subquestions are formed in relation to this question.

RQ 2.1 *How can the uncertainty of probabilistic evaluation model parameters be*

combined with the diverse aspects which have to be considered in architecture optimisation? Architecture optimisation of software-intensive systems is a complex problem which has many dimensions. In the context of embedded systems, architecture optimisation involves various design decisions such as deployment, software/hardware component selection, software/hardware redundancy allocation and scheduling, while the goals to optimise include many non-functional attributes including reliability. Candidate architectures are often required to satisfy a set of design constraints. From the analysis of related work on architecture optimisation approaches presented in Chapter 2, it is evident that these different aspects are addressed in isolation or in relation to specific problems. However, the overall goal of this research is to formulate an architecture optimisation approach which can incorporate uncertain input parameters. A preliminary research question arises from the need to model these diverse aspects of the software architecture optimisation problem. In order to devise an approach which accommodates uncertainty, the first issue is to formulate a framework which can capture these aspects of the architecture optimisation problem within the scope of the thesis stated in the Introduction. A subsequent research question to address in achieving the overall goal of architecture optimisation under uncertainty is how this uncertain information can be used during the optimisation. As opposed to the methods that use point-estimated input parameters, a new challenge arises in optimisation as the quality attributes (in this case, reliability) of the candidate solutions inherit the notion of uncertainty. The question seeks a modelling support for diverse aspects in software architecture optimisation including the elements induced from parameter uncertainties.

RQ 2.2 *How can commonly used optimisation algorithms be adapted for architecture optimisation with objective functions based on input parameters estimated under uncertainty?* Even for a small architecture optimisation problem in

the domain of embedded systems, the design space is very large due to the combinatorial number of options, and it has been proven that most of the architecture optimisation problems are NP-hard [110]. Consequently, the use of exact optimisation algorithms is limited to very small problems, and becomes counterproductive when solving most of the practical problems of realistic size. On the other hand, problem-specific solvers require knowledge of the particular architecture design problem. Stochastic optimisation algorithms become promising candidates as an optimisation strategy to search for robust optimal solutions due to their adaptability and performance over a range of problems. However, as also discussed in Chapter 2, conventional optimisation algorithms use point values as the quality goals in the candidate solutions. With the introduction of uncertainty into the parameters of quality evaluation models, this assumption is no longer applicable, and optimisation strategy related operators have to be made uncertainty-aware. Given the existing architecture optimisation methods that use stochastic algorithms, a next research subquestion to be addressed is on mapping the conceptual entities of the problem to algorithm elements and providing a feedback strategy to these existing algorithms for the search of (near-)optimal set of candidate architecture solutions that can withstand uncertainty in input parameters.

4.2 Approach Overview

The architecture design of component-based software systems involves a set of architectural elements, such as software components, hardware units and interactions among software components. These architectural elements are annotated with a set of parameters that describe certain characteristics such as the failure rate of a software component or the probability of occurrence of an event. At the architecture design stage, most of the parameters are estimates, and estimations are made subject to uncertainty. This work focuses on the architecture design optimisation of

embedded systems, where a specific design decision has to be made, such as deployment of software components to a hardware platform, allocating redundancy levels to logical or physical units or selecting a set of components from similar alternatives. In this context, each alternative design constitutes a candidate architecture. The goal is to find a (near-)optimal set of candidate architectures with respect to a set of system reliability attributes such as reliabilities of individual services. The reliability properties are evaluated using probabilistic models (*e.g.* Markov chains) which are built using the parameters of architectural elements. However, these model-based reliability evaluations inherit the notion of uncertainty from the estimates of architectural parameters mentioned above. Consequently, the quality objectives have to be optimised in the presence of uncertainty in the input parameter estimates. The overview of the SCOUT approach is outlined in Figure 4.2. The key contributory aspects are briefly presented in following subsections.

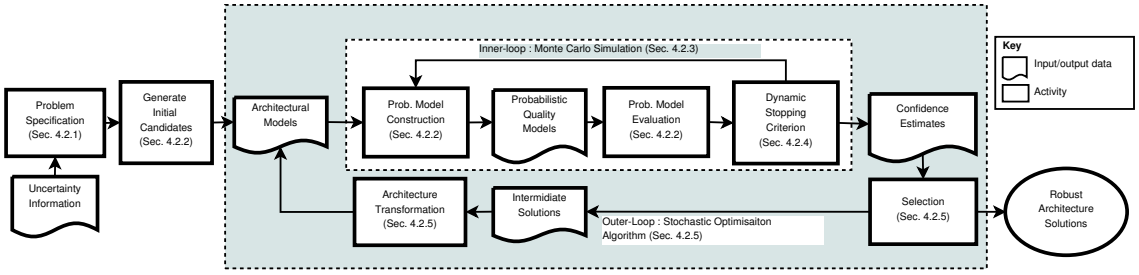


Figure 4.2: Overview of the robust optimisation approach

4.2.1 Parameter Specification with Uncertainty

Firstly, the SCOUT approach proposes to include the uncertainty into the parameter specification. If a parameter that describes some aspect of an architecture is an estimate, the SCOUT approach understands that the parameter specification encapsulates characteristics of uncertainty associated with that parameter. An extension to the specification of architectural element parameters is introduced. As a means to specify parameters with uncertainty, this approach propose to employ probability distributions. The SCOUT approach allows the parameters to be given

as any probability distribution (including discrete distributions) that can represent a parameter with its variability characteristics. The different parameters are allowed to take dissimilar distributions, while some parameters can be precise numerical values. When the provision of probability distributions is impractical, the approach supplies the designers with a method to include a range of variations as a uniform distribution or a discrete set of possible values. The use of probability distributions also entails the support for conversion from other complementary approaches. For instance, interval estimates can be represented as a uniform distribution while the mean-variance estimation methods can be replaced with a normal distribution having the same mean and variance.

4.2.2 Architecture-based Reliability Evaluation under Uncertainty

From a given specification of the elementary units of the architecture, a set of candidate architectures can be built to satisfy the functional requirements and design constraints. This set of candidates are called *initial solutions*, and can be provided by a human architect or automatically generated using problem-specific heuristics or even randomly. The architecture-based quantification of probabilistic properties is carried out thereafter. In order to quantify the desired reliability attributes from the architecture, reliability evaluation models are constructed from the architecture as described in Chapter 2. Parameters of the reliability evaluation model reflects the parameters specified with the architectural elements as described in the previous subsection. However, it should be noted that the provision of the parameters with their uncertainty as heterogeneous distributions hinders the conventional model evaluations as the models expect numerical values for their parameters. This work understands the quality models as mathematically complex formulations, and analytical derivation of quality metrics as a function of input distributions is hard to achieve. A Monte Carlo (MC) simulation-based uncertainty analysis mechanism is proposed that samples from the probability distributions of input parameters. It is

emphasised that probabilistic properties are quantified using conventional reliability evaluation models, but MC is used to re-evaluate the model for the samples taken from input parameter distributions repeatedly. Figure 4.3 illustrates the input of the MC simulation, which is a quality evaluation model with a set of parameters specified as probability distributions. In each MC run, the value of a parameter is sampled from the probability distribution, which is then used as the numerical value to compute the quality metric. In brief, MC-based uncertainty analysis of this approach can be described as follows.

- For the Monte Carlo simulation, the reliability evaluation model which is used to quantify the property of interest from an architecture, acts as a black box. Each MC run samples from the input distributions and instantaneous samples are used as the numerical values in the reliability model evaluation. As the sampling process is independent from the probabilistic quality model, the approach is not limited to specific evaluation models or evaluation techniques.
- MC simulation does not make assumptions about input distributions or their characteristics. Therefore, diverse input distributions can be easily incorporated into the abstract MC model.

4.2.3 Quantification of Uncertain Properties

When all or a subset of the input parameters are uncertain and specified as probability distributions, each MC run can report different quality values for the same architecture due to the use of instantaneous samples from the input parameter distributions. Consequently, the results of the MC simulations is a series of values for the quality attribute of interest. For instance, 100 MC runs of reliability evaluations result in 100 values for the reliability of the candidate architecture such as 0.994, 0.992, .. 0.999. The variability of those values arises from the uncertainty associated with the input parameters. As the goal of the architecture optimisation is to find (near-)optimal architectures that can perform as expected despite the uncertainty of the input parameters, this work proposes to use an appropriate statistical

representative of MC results to indicate the quality attribute of an architecture alternative in terms of both the optimality and robustness, *i.e. robust quality values*. As a means, the SCOUT proposes to use percentiles obtained from MC simulation results. More intuitively, if the 5^{th} percentile reliability of a candidate architecture is regarded as the robust reliability, a numerical value provides a quantitative measure of reliability with a confidence guarantee against the uncertainty, *i.e.* it is 95% confident that the reliability of the candidate solution is greater than that value under all the parameter uncertainties associated with that architecture. In the case of safety-and mission-critical systems like in the automotive or avionics domains, upper/lower bounds of the objectives (conservative approach) for a given confidence interval can be used while for some non-critical applications, the median value (50^{th} percentile) of the quality may be sufficient.

However, an important issue which arises in this step is that the characteristics of the MC results series are unknown. Due to the presence of heterogeneous distributions as input parameters, the distribution of the probabilistic property of interest under uncertainty cannot be determined in advance. Consequently, the robust value of the quality has to be obtained without knowing the distribution. This work adopts non-parametric statistical estimation for this purpose which is further explained in chapter 5.

4.2.4 Dynamic Stopping Criterion

The inequality and variation of individual samples of MC simulation described before is a manifestation of heterogeneous uncertainties associated with the input parameters. With the relaxation of assumptions on input parameter distributions, the distribution of MC samples (*i.e.* samples of a quality attribute) is also not known. An infinite number of MC runs is required to obtain the exact value of the percentile of the quality attribute in this setting. On the other hand, each MC run samples from the input distributions and computes the quality model, which is a computationally expensive process. A careful balance has to be kept between the accuracy

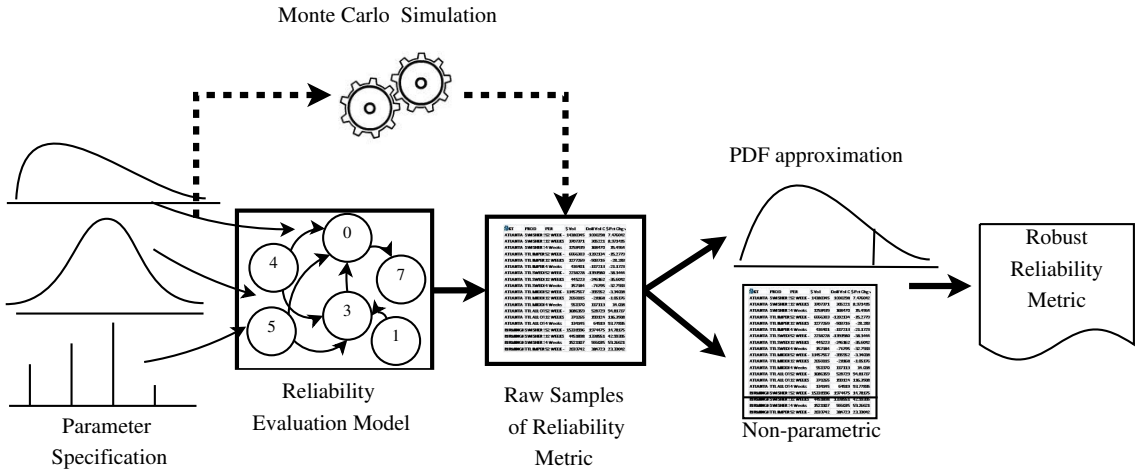


Figure 4.3: MC simulation of probabilistic evaluation models

and time complexity in selecting the MC runs for the robust optimisation.

As a solution to address the above issues, the SCOUT approach introduces a dynamic stopping criterion for the MC simulation based on accuracy monitoring. The stabilisation characteristics of the desired statistical index (*e.g.* percentile) with the increasing samples is exploited. The goal of this element of research is to monitor the accuracy of the MC-based estimation process automatically, and to stop the sampling when sufficient accuracy has been achieved. The assumptions on the monitored distribution are relaxed by applying statistical monitoring on the estimate. Intuitively, the statistical significance test is conducted on the samples of the statistical index (*e.g.* 5^{th} percentile of reliability), rather than the individual reliability samples.

4.2.5 Integrating Stochastic Optimisation Algorithms

The description so far was on obtaining quantitative metrics of reliability properties with the notion of uncertainty, *for a single candidate architecture*. The overall goal require a method to finding architecture alternatives that are (near-)optimal in terms of reliability goals with a confidence guarantee *w.r.t.* the uncertainties in the input parameters. This work proposes to consider the notion of uncertainty

throughout the optimisation process. Both the optimality in terms of the quality and the robustness of the solution with respect to the uncertainty in the parameter estimates are regarded in the search for better architectures, *i.e. design space exploration*.

The embedded systems architecture design involves a number of choices and design decisions such as, software component selection, hardware selection, hardware topology and deployment. The notions of parameter uncertainty and optimisation are applicable for many of these issues, having different quality goals to optimise. Therefore, the SCOUT approach has been developed as a framework that different aspects such as optimisation objectives, architectural changes and robust quality measures are understood in high-level abstraction. The SCOUT defines high-level entities to represent different aspects in the problem architecture optimisation under uncertainty.

As presented in Section 2.3, most of the architecture design problems are NP hard [110], and hence, exact algorithms fail to provide solutions within a feasible time frame. Problem-specific heuristic algorithms exist with limited applicability to domains and optimisation problems. However, stochastic algorithms have proven successful over a range of problems. Therefore, as a design space exploration strategy for candidate architectures which are better in quality while being able to tolerate parameter uncertainties, the SCOUT provides a high-level mapping of software architecture optimisation framework entities to the concepts in stochastic algorithms. Adaptations of three prevalent stochastic optimisation algorithms are presented, namely *Non-dominated Sorting Genetic Algorithm* (NSGA-II), Ant Colony Optimisation and Simulated Annealing which have demonstrated consistent success in architecture optimisation problems (please refer to the summary of architecture optimisation approaches given in Table 2.3). Each algorithm's operators and strategies are tailored to guide the architecture decisions to robust and (near-)optimal solutions. The algorithms' feedback mechanism of measuring the quality of solutions is adjusted to consider the solutions' behaviour in uncertain parameter variations. The

architectures with better quality and tolerance for uncertainty have to be encouraged while weak or uncertainty-sensitive solutions are to be penalised. In order to achieve this, appropriate percentiles of the quality (lower or upper bounds depending on the objective) are proposed to be used as the criterion to compare candidate solutions. The operators that are used to generate a new solution (architecture), are mapped to architecture transformation operators. As a result, a new solution is generated, corresponding quality models are constructed and quality attributes are quantified with the uncertainty as described before. In Figure 4.2, this stochastic optimisation is marked as the *outer loop*, while the MC simulation based uncertainty analysis constitutes an *inner loop*. The formulation of the framework and concrete examples are given in later chapters.

4.3 Validation Strategy

Having described the research questions and the overview of the SCOUT approach, this section presents the validation techniques used with each of the research questions. This section uses characterisation and classification of research questions, research results and validation techniques given by Shaw [362]. In addition to this categorisation, Shaw has explicitly provided generally accepted research strategies in the area of software engineering [362, 363], where this thesis takes the benefit in formulating suitable validation strategies for the proposed solutions.

RQ 1.1 addresses the problem of capturing uncertain information in the parameter specification, which is considered as a software engineering question of type *method or means of development*. The SCOUT approach provides a solution to this research question with a probabilistic parameter specification enabling the inclusion of diverse characteristics on uncertainty associated with model parameters. According to the software engineering solution categorisation presented by Shaw [362], the solution belongs in the *procedure or technique* and *notation* categories. Similar to the most common form of validation to this type of solutions [363], the thesis presents an *evaluation* of the solution. Chapter 5 illustrates the approach's capabil-

ity to capture diverse uncertain parameters pertaining to an automotive embedded system case study as well as with a series of experiments on deployment problem instances. In the evaluation of the overall approach in Chapter 7, a specific evaluation criterion (EC 1) is formulated to evaluate the solution with respect to this research question. The validity of the solution is supported with experiments on a series of generated problem instances covering different architecture optimisation problem types, problem settings and characteristics of uncertain parameters.

RQ 1.2 seeks a *method of analysis* for architecture-based reliability evaluation when parameters are uncertain. Considering the mathematical complexity of reliability models and heterogeneity of the uncertain characteristics in model parameters, the SCOUT approach gives a *procedure or technique* solution with Monte Carlo simulation and a novel dynamic stopping criterion. The new solutions validity is presented through an *evaluation*. Chapter 5 demonstrates a *proof of concept* by applying the MC simulation-based uncertainty analysis method on evaluating reliability of automotive ABS system. Through evaluation of the proposed solution in terms of adequacy, accuracy and scalability is presented by applying the approach on a set of generated problem instances. The proposed solution for this research question is further evaluated in the overall problem of architecture optimisation context in Chapter 7 with Evaluation Criterion 2.

The issue of reliability metric representation under heterogeneous uncertainty in model parameters is channelled with RQ 1.3, which belongs to the *method or means of development* category within the settings of a software engineering research question. In response, this work presents a *procedure or technique* type solution with non-parametric estimation allowing the use of various statistical indices including percentiles. An *evaluation* of the proposed method is conducted to provide evidence on validity of the approach. Chapter 5 presents a *proof of concept* by analysing the variation of reliability metric of a automotive ABS system in the presence of uncertain input parameters. Not limiting to the validity on the specific case, Chapter 7 evaluates the solution with a series of problem instances with respect to specific

research questions. The accuracy and adequacy of the proposed solution in answering the research question is evaluated with evaluation criteria EC 2 and EC 3 respectively.

The modelling requirement to achieve the overall research goal has been formulated in RQ 2.1, which is a *method or means of development* type research question in software engineering. The SCOUT approach devises a framework integrating various aspects involved in software architecture, optimisation algorithms and uncertainty analysis, which belongs to the solution category of *descriptive model* [362]. Firstly, the model is iteratively devised from a commonly accepted formalism given in ISO 42010 [219] . An *analysis* of the modelling aspect of the solution is performed in Chapter 6 with respect to modelling approaches taken in current literature. The model is implemented as a tooling framework, and further *example* type evidence is given by presenting the capability of the model to cater for requirements in a case study.

RQ 2.2 focuses on applying the architecture design space exploration techniques under uncertainty, which is also a question of type *method or means of development*. As a solution, this thesis presents the integration of three stochastic algorithms, namely NSGA-II, Ant colony optimisation and Simulated Annealing, which can cater for NP-hard optimisation problems in architecture optimisation. These adaptations provide *example* type evidence to the validity of RQ 2.2. The proposed solution is fully evaluated and an *evaluation* of the solution to diverse problem settings and characteristics is presented in Chapter 7 with dedicated evaluation criteria EC 4 and EC 5.

Upheld with the individual validations, dedicated evidence is collected on the overall SCOUT approach, which has been in the form of *procedure or technique* type solution. The proposed solution is fully implemented and two types of empirical evidence are used to support the overall validity. On one side, the proposed solution is applied to an industrial case study as an *example* type validity, and the ability to cater for the requirements with extended decision support is illustrated.

Secondly, large number of experiments are conducted as *evaluation* type validation for scalability. A series of problem instances are generated randomly, while preserving realistic behaviours in order to cover diverse of problem types, sizes as well as different characteristics included in the scope of the thesis. The overall validity is justified with quantitative analysis of the results collected from the experiments.

Chapter 5

Architecture-based Reliability Evaluation under Uncertainty

5.1 Introduction

Architecture-based quality evaluation models are an important asset during the design of software-intensive embedded systems. The benefit of these evaluation models is especially evident in the architectural design phase, since different design alternatives can be evaluated and software architects are able to make informed decisions between these alternatives. Chapter 2 presents a number of evaluation models which have been proposed to date for evaluating probabilistic quality attributes. However, a considerable number of parameters involved in the architecture evaluations are based on design-time estimates. These estimations tend to use field data obtained during testing or operational usage, historical data from products with similar functionality, or reasonable guesses by domain experts. In practice, however, parameters can rarely be estimated accurately [11, 151, 181]. This chapter investigates different aspects in relation to the parameter uncertainties in architecture-based quality evaluation having specific focus on reliability, and formulates a framework that constitutes specification, evaluation and quantification of reliability in the presence of uncertainty.

5.1.1 Sources of Uncertainty

In the context of software intensive systems design, the sources of uncertainty can be classified into two major categories.

► **Aleatory Uncertainty**. *"is the inherent variation associated with the physical system or environment under consideration"* [308].

This category refers to the sources that are inherently stochastic in nature. Physical uncertainties such as noise in electrical conductors, humidity, temperature, material parameters, behaviour and instantaneous decisions of operators are examples in the domain of embedded systems. This type of uncertainty can not be avoided [55].

► **Epistemic Uncertainty**. *"is uncertainty of the outcome due to the lack of knowledge or information in any phase or activity of the modelling process"* [308].

This source of uncertainty reflects the lack of knowledge about the exact behaviour of the system. Uncertainties of this type are subjective and depend on factors such as maturity of the design and models, experience of the application domain, and the coverage and extent of testing.

Those two types of sources induce uncertainty in a number of ways in architecture evaluation. An orthogonal categorisation of uncertainty in architecture evaluation can be conducted based on the presence in the design process [55].

- *Type A*. The uncertainties in relation to a changing environment and operating conditions of the prospective system are designated as Type A. Operational temperature of a system, usage profile, frequency of use and occurrence of failure are some of the examples that belong to the Type A category.
- *Type B*. This type captures the notion of uncertainty in the realisation of design parameters. Parameters used in the architecture models may be realised up to certain degree of accuracy. This category is also known as manufacturing tolerances.
- *Type C*. Any *model* used to represent a prospective system's design is an abstraction of real system. Therefore, uncertainties are involved in how accurately the mode describes reality. The type C category captures all sorts of

approximation and modelling uncertainties.

- *Type D.* This category represents the uncertainties with the goals and constraints at the design phase. The actual goals of the system may be realised only after the system has been built, and consequently certain design goals are uncertain at the architecture design phase.

The focus of this work is on the uncertainties of type A and B, which refer to the impact of external and model parameters on the architecture-based quality evaluation. Manifestations of the these two types of uncertainty exist in the parameters of software architecture such as software components, inter-component interactions, hardware components, communication links, behavioural descriptions, operational profile and use cases.

5.1.2 Chapter Overview

This chapter addresses the problem of architecture-based reliability evaluation when the external and probabilistic model parameters are subject to uncertainty as stated in the refined research questions under RQ 1 in Chapter 4. Figure 5.1 illustrates the new elements introduced in this chapter and their relationships which constitute the inner-loop given in Figure 4.2. The leftmost element represents the specification of the software components, hardware, usage profile and quality requirements. This part of the SCOUT approach introduces the ability to incorporate heterogeneous information about the uncertainty of parameters at the specification phase. Model-based quality evaluations are used to determine the quality of the prospective system on the basis of the architecture. Based on the results of multiple Monte Carlo (MC) simulations, estimates for the quality attributes of the architectures are computed. A novel dynamic stopping criterion stops the MC simulations when sufficient samples have been taken. The new approach is illustrated using an example application, followed by a focused validation of the approach with a series of experiments on generated problem instances.

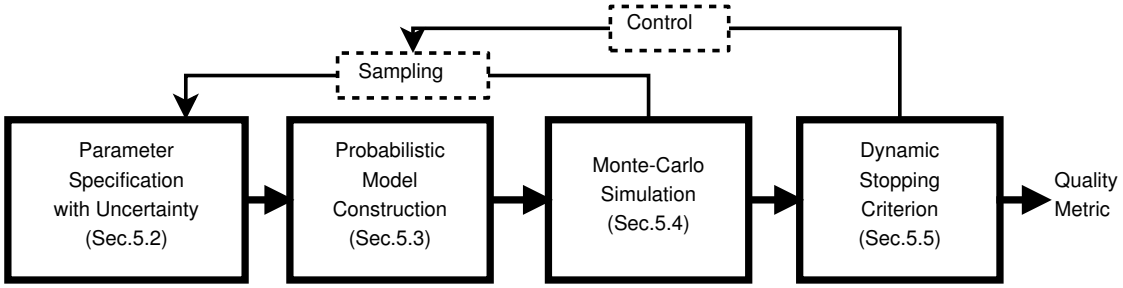


Figure 5.1: Architecture evaluation under uncertainty

5.2 Specification of Uncertain Parameters

Various parameters which are involved in the architecture-based quality evaluation have diverse characteristics in relation to uncertainty. Often there are considerable dissimilarities between parameters whose values cannot be definitively determined. On the other hand, not all the parameters are necessarily uncertain, as some may be precisely determined. Since it has been established that the variability of parameter estimates significantly affects the quality metric of the architecture [42, 177, 349], it is important to capture these diverse characteristics as accurately as possible. Given these considerations, this work understands architecture parameters as a combination of precise and imprecise sets.

5.2.1 Probability Distributions

As a means to capture heterogeneous uncertainties in parameter estimation, this SCOUT approach proposes to use generalised probability distributions. A parameter in an architecture specification is considered as a random variable, whose variability is characterised by its – continuous or discrete – distribution. A generic notation that can cater for any distribution is proposed for the parameter specifications in the architecture descriptions. By *generic distribution* this approach refers to any probabilistic specification of a parameter that satisfies the definitions given below [299].

Definition 1 (Probability Density Function) *For a continuous random variable X, a probability density function is a function such that,*

(1) $f(x) \geq 0$

(2) $\int_{-\infty}^{\infty} f(x) = 1$

(3) $P(a \leq X \leq b) = \int_{a}^{b} f(x)dx =$ *area under* $f(x)$ *from* a *to* b *for any* a *and* b.

Definition 2 (Probability Mass Function) *For a discrete random variable X with possible values* $x_1, x_2, ..., x_n$, *a probability mass function is a function such that,*

(1) $f(x_i) \geq 0$

(2) $\sum_{i=1}^{n} f(x_i) = 1$

(3) $f(x_i) = P(X = x_i)$

Definition 3 (Cumulative Distribution Function) *The cumulative distribution function of a discrete random variable X, denote as* $F(x)$, *is*

$F(x) = P(X \leq x) = \sum_{x_i \leq x} f(x_i)$

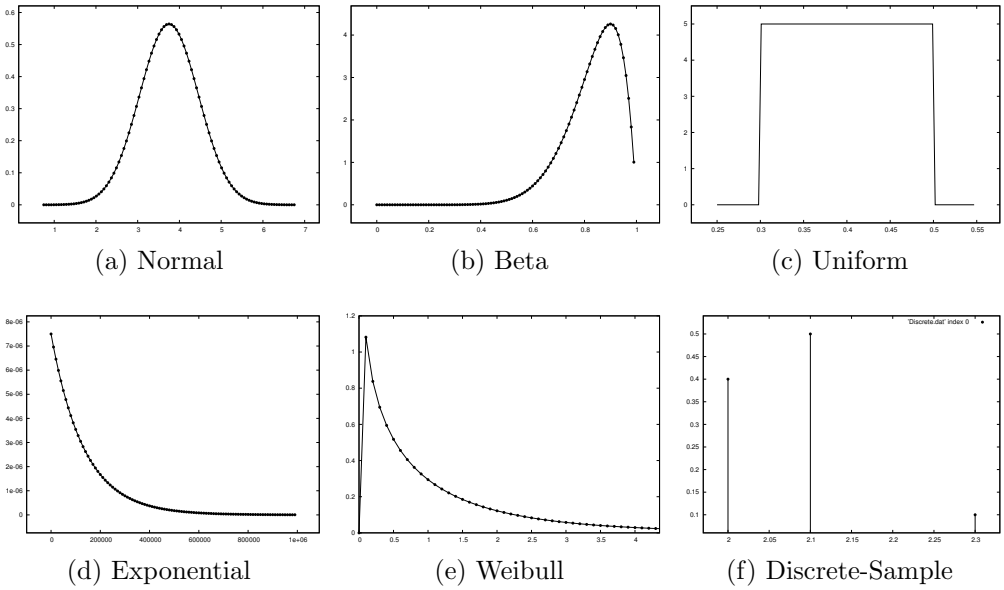
For a continuous random variable X

$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u)du$ *for* $-\infty < x < \infty$.

In order to allow the parameter specifications in the architecture descriptions, SCOUT introduces a generic notation that can cater for any distribution. All or a subset of the parameters in architecture specification can be given as probability distributions. The specification is given as a parameter list, starting with a unique identifier assigned to the distribution. Some examples for Probability Density Function (PDF) specifications are given at Table 5.1, and example PDFs plots illustrated in Figure 5.2

Distribution	**Specification Syntax**	**Range**	**Example**
Normal	NORMAL,μ,σ^2	$(-\infty, \infty)$	$NORMAL, 3.75, 0.05$
Beta	BETA, α, β	[0,1]	$BETA, 10, 2$
Shifted Beta	BETA_SHD,$\underline{x}$, $\overline{x}$, α, β	$(\underline{x}, \overline{x})$	$BETA\_SHD, 3, 5, 2, 10$
Exponential	EXP,λ	$(0, \infty)$	$EXP, 7.5 \times 10^{-6}$
Uniform	UNIFORM,$\underline{x}$, $\overline{x}$	$(\underline{x}, \overline{x})$	$UNIFORM, 3.5, 4.0$
Gamma	GAMMA,λ	$(0, \infty)$	$GAMMA, 1.5$
Weibull	WEIBULL,α	$(0, \infty)$	$WEIBULL, 1.5$
Discrete	DISCRETE, $x_0, p_0, x_1, p_1, ..., x_n, p_n$	(x_0, x_n)	$DISCRETE, 2, 0.4, 2.1, 0.5, 2.3, 0.1$

Table 5.1: Some probability distributions and their PDF/PMF specification



(a) Normal (b) Beta (c) Uniform

(d) Exponential (e) Weibull (f) Discrete-Sample

Figure 5.2: Probability density functions of some distributions

5.2.2 Mapping Heterogeneous Uncertainty into PDFs

The proposed approach allows to combine diverse sources that affect the nominal value of the parameter, and consider their impact on the quality evaluation in addition to the conventional point estimates. Some guidelines to obtain the PDFs at the design stage can be given as follows.

- *Derive from the source variations.* The uncertainty of parameters are often manifestations of different sources. Information from hardware manufactures, third party software vendors or system experts is useful in characterising the uncertainty in specific parameters. In some situations, the distribution of the source variables can be obtained and consequently, the desired parameter's distribution can be approximated from its sources.

  Example ▷ The failure rate (λ) of an ECU is a function of its ambient temperature (T in Kelvin) [59], such as $\lambda = (T + 100) \times 4 \cdot 10^{-6}$. Consider an automotive electronic system where the temperature profile around ECU X varies between 300K and 400K, has a 370K mode and is skewed right. The PDF of λ of ECU X can be derived and specified as $\lambda_X = BETA\_SHD, 400 \times 4 \cdot 10^{-6}, 500 \times 4 \cdot 10^{-6}, 10, 2$ ◁

- *Histogram approximation.* Prior information on some parameters may be available during the architecture design stage. For certain parameters, a considerable amount of raw data may be available as a result of unit testing. In such situations, the PDFs can be approximated from the histograms of the raw data.

  Example ▷ In functional test executions of the system model, the histogram of the test results indicated that the message transfer probability from component C_i to component C_j is normally distributed. The average of the samples is 0.2 with a variance of 0.04. Therefore, the transfer probability can be given as $p_{i,j} = NORMAL, 0.2, 0.04$ ◁

- *Uniform approximation.* It is common to have limited information on the

range of the variation without any specification on variation within the range. Uniform distributions can be used in approximating such situations.

Example ▷ The system has a need to communicate with a new external service X, of which we only know that its worst case response time is 1.0s. The communication link takes at least 5ms for the data transfer. $rt = UNIFORM, 5 \cdot 10^{-3}, 1.0$ ◁

- *Specify distinct information as a discrete-sample distribution.* In cases where the a parameter can only vary within a discrete set, discrete-sample distributions can be used to capture it. This is a very powerful feature in the SCOUT approach as in most of the practical situations, it is relatively easy to obtain discrete estimates.

  Example ▷ Experts have indicated that the request rate (rr) for a service X can be either 200 or 800 per second. In 75% of the cases it is 200. This will be given as $rr = DISCRETE, 200, 0.75, 800, 0.25$ ◁

5.3 Probabilistic Model Construction

The next step of the architecture-based quality evaluation is to construct the quality model from the architectural specifications described before. Different quality attributes can be evaluated using different modelling approaches as discussed in Chapter 2. In the case of probabilistic quality attributes, the evaluation models are also inherently probabilistic. The model parameters are often derived from the parameters of the architectural elements. This process results in one-to-one, one-to-many, many-to-one or many-to-many relationships of architecture parameters to the parameters of probabilistic model. Since the new approach has incorporated probabilistic specifications of parameters of architectural elements, the probabilistic notion is transformed into the evaluation model parameters. Due to the fact that the inputs are probability distributions, the resulting evaluation model parameters

become probability distributions or functions of probability distributions.

5.4 Quality Metric Estimation

The probabilistic model with partially uncertain parameter space has to be evaluated in order to obtain the quantitative metric of the quality of the system architecture at hand. It has been emphasised in Chapter 2 that these models are often hard to represent as linear mathematical functions. When many parameters are uncertain with diverse distributions, the quantitative metric as a distribution cannot be derived analytically. Consequently, the Monte Carlo (MC)-based approach presented here, which draws samples from the probability distributions of input parameters.

Figure 5.3 illustrates the architecture evaluation process using MC simulation. The input of the MC simulation of the probabilistic model (PM) is a set of parameters specified as probability distributions (UPs) as well as deterministic/certain parameters (DPs).

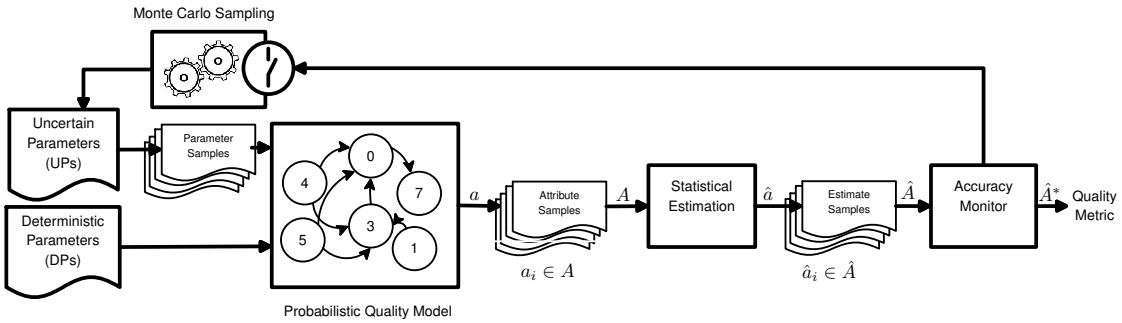


Figure 5.3: Monte Carlo simulation of probabilistic quality evaluation models

5.4.1 Monte Carlo Simulation

The MC simulation takes samples from input probability distributions of the architectural elements within the probabilistic evaluation model. Any single parameter of an architectural element may contribute to more than one parameter in the evaluation model. Every time a sample is taken from an input distribution, all model

parameters dependent on this parameter have to be updated. The steps that are involved in the strategy for a single run of the MC simulation is explained in the following.

1. **Sample:** A sample is taken from probability distributions of each parameter. *Inverse transformation method* [345] can be used to draw a sample from these distributions as follows.
   (a) Obtain the Cumulative Distribution Function (CDF) of the parameter from its PDF.
   (b) Generate a random number x from uniform distribution (0,1).
   (c) Obtain $CDF^{-1}(x)$.
2. **Update:** From the samples obtained from the input distributions, the numerical values for the evaluation model parameters are updated. Since more than one parameter of the probabilistic model may refer to a parameter in the architecture, a subscription mechanism is proposed. Parameters of the evaluation model are subscribed to uncertain parameters in the architecture, similar to the standard *observer* design pattern [162]. When a sample is taken for a specific architectural parameter, all the subscribing model parameters are updated and recomputed.
3. **Resolve dependencies:** The model-specific parameter dependencies are solved at this stage. For example, if the DTMC-based evaluation model is used, the sum of all outgoing transition probabilities of any non-absorbing node should be equal to 1. After obtaining numerical values for the outgoing transition probabilities, a normalisation process will be carried out in order to complies with the model assumptions.
4. **Compute :** Analytically solve/simulate the model and obtain the quantitative metric, as described in Chapter 2.

This single run of MC simulation results in one numerical value of the quality attribute (a). Due to the uncertain parameters (UPs), the values obtained from different runs ($\{a_1, a_2, a_3, .., a_N\} = A$) are most often not identical. From the soft-

ware architect's point of view, it is desirable to have a single measure ($\hat{a}$), which is a *descriptive figure* [164] that can summarise the extent and characteristics of uncertainty in the values. The level of tolerance in the statistical estimation depends on the application domain and the quality attribute. Depending on the characteristics of the system to be designed, the expected value, variance, quartiles, confidence intervals or worst case values can be used in describing certain attributes of the system.

- Expectancy value: For some non-critical cases such as response time in simple web application, the expected value of the quality metric (response time) is considered as the value of the architecture.
- Variance: Certain architecture evaluation goals focus on reducing the variance of the quality. An example is the variance of response time in a fair-trade system. In such cases, the variance has to be estimated.
- Quartile: Some dependable embedded systems design focuses on quartiles of quality, such as choosing the first quartile of the failure rate distribution.
- Value with a given confidence: Confidence value estimates are crucial properties of most of the safety and mission critical systems. Reliability of the ACC system with a 95% confidence under the uncertainty can be mentioned as an example.
- Worst case value: The architects of certain applications are interested in pessimistic estimates. An example is the worst case response time for a automotive brake system.

One important challenge regarding this estimation is that the actual distribution of the quality metric (A) is unknown. The existing uncertainty analysis techniques in software architecture evaluation have a prior assumption regarding the distribution of A. With some exceptions [100,413], most studies assume a normal distribution [94, 181]. Due to the heterogeneity of input parameter uncertainty, and the non-linearity and complexity of model evaluation techniques, the resulting quality distribution after the MC simulation is unpredictable, and the normality assumption is no longer

applicable. As a solution, the SCOUT approach introduces a generalised estimation of ($\hat{a}$), using flexible percentiles while supporting the expected/worst case measures. The following sections describe two types of techniques which can be used to obtain percentiles in unknown distributions.

5.4.2 Parametric Estimation

From the statistical data ($A = \{a_1, a_2, a_3, ..., a_N\}$) in the MC runs, statistical methods can be used to identify parameters of a candidate distribution. Some of the possible approaches are explained below.

Method of Maximum Likelihood

This is a widely used method in obtaining parameters of a specific distribution when only the samples are available. With the notation used in previous subsections, the likelihood function for Monte Carlo samples can be defined as in Equation 5.1

$$L(\theta) = f(a_1;\theta) \cdot f(a_2;\theta) \cdot f(a_3;\theta) \cdot ... \cdot f(a_N;\theta) \tag{5.1}$$

where $f()$ denotes the candidate distribution and θ refers to the parameters of the distribution.

Example ▷ Consider an exponential distribution as an example. The objective is to find the parameter λ from the MC samples A. The likelihood function can be defined as in Equation 5.2.

$$L(\lambda) = \prod_{i=1}^{N} \lambda e^{-\lambda a_i} = \lambda^N e^{-\lambda \sum_{i=1}^{N} a_i} \tag{5.2}$$

Then, λ can be obtained by solving the maximum value of $L(\lambda)$ the Equation 5.2 for the MC samples $A = a_1, a_2, a_3, ..., a_N$. ◁

Method of Moments

This is a technique that is based on equating population moments which are defined in terms of expected values, to corresponding sample moments [299]. In the current context of estimating $\hat{a}$, population refers to candidate distributions (*e.g.* Normal, Beta, Exponential *etc.*) for A, and sample moments refers to the quality value which are the results of individual Monte Carlo runs $(a_1, a_2, a_3, ..., a_N)$.

Example ▷ Consider the normal distribution as an example. To obtain the characterisation parameters μ and σ^2, the first and second moment of a normal distribution can be used.

$$\text{first moment} : E[A] = \mu \tag{5.3}$$

$$\text{second moment} : E[A^2] = \mu^2 + \sigma^2 \tag{5.4}$$

By solving 5.3 and 5.4 with the substitutes $E[A] = \frac{\sum_{i=1}^{N} a_i}{N}$ and $E[A^2] = \frac{\sum_{i=0}^{N} {a_i}^2}{N}$, μ and σ^2 can be found. ◁

Bayesian Estimation

The Bayesian method of estimation can be used to find the parameters of a candidate distribution when some information of the candidate distribution is available. The distribution function with known parameters is called *prior distribution* and the goal is to find the *posterior distribution* with its parameters. In the context of this presentation, the new posterior distribution of A ($f(\theta|A)$) is obtained from the prior distribution ($f(A|\theta)$) and MC samples of A as described by Equation 5.5 .

$$f(\theta|a_1, a_2, a_3, ..., a_N) = \frac{f(a_1, a_2, a_3, ..., a_N|\theta) f(\theta)}{\int_{-\infty}^{\infty} f(a_1, a_2, a_3, ..., a_N, \theta) d\theta} \tag{5.5}$$

where θ refers to parameters of the candidate distribution. For further details on the Bayesian estimation method, please refer to [299].

Example ▷ Assume an exponential distribution with a prior estimate of λ_0. Then, the posterior distribution of exponential parameter λ of A will be given as:

$$f(\lambda) = \frac{\lambda_0^N e^{-\lambda_0 \sum_{i=1}^{N} a_i} \lambda e^{-\lambda\lambda_0}}{\int_{-\infty}^{\infty} \lambda_0^N e^{-\lambda_0 \sum_{i=1}^{N} a_i} \lambda e^{-\lambda\lambda_0} d\lambda} \tag{5.6}$$

◁

The methods discussed above can be used when some prior information about the distribution of the resulting quality metric (A) is available. When the A's distribution is estimated using one of the above methods, statistical indices ($\hat{a}$) can be obtained by integrating the PDFs or referring to statistical tables. However, due to the diverse nature of input parameter distributions and complex quality evaluation models, estimating a prior distribution is hard. No known approach in research or industry currently estimates the output characteristics of distributions of the quality attribute in similar situations. Furthermore, this estimation process is computationally expensive and has to be conducted for each architecture evaluation.

5.4.3 Non-parametric Estimation

An alternative branch of statistics and mathematical foundation which exists for the purpose of estimating *descriptive figures* is referred to *non-parametric* procedures or *distribution-free* statistics. In contrast to the aforementioned three techniques which require some information on the distribution of the quality metric subject uncertainty, non-parametric methods do not depend on any probability distribution or characterisation. Therefore, assumptions on the underlying distribution (A) do not have to be made. General applicability and performance under unrestricted conditions of the population are among the key benefits of non-parametric techniques. Due to the simplicity in computations, non-parametric methods provide fast and efficient applications in variety of statistical problems.

The estimation problem discussed is to obtain various *descriptive figures* (*e.g.* percentiles, mean, variance or confidence bound) of the population of the quality metric (*e.g.* service reliability) from the observed samples of the Monte Carlo simulation (*i.e.* A). The non-parametric methods serve this exact purpose as there exist specific formulations for 'obtaining population indices from sample indices'. Detailed discussions on obtaining various descriptive figures can be found in the work of Gibbons *et al.* [164]. For the purpose of architecture evaluation under uncertainty, the aim is to obtain descriptive figures such as mean, median or percentiles. As mentioned earlier, straightforward formulations exist in the non-parametric stream. Mean and variance can be easily computed from raw samples (*i.e.* for mean $\hat{a} = E(A) = \frac{\sum_{i=1}^{N} a_i}{N}$ or for variance $\hat{a} = Var[A] = \frac{\sum_{i=1}^{N} a_i^2}{N} - \left(\frac{\sum_{i=1}^{N} a_i}{N}\right)^2$). Pessimistic or optimistic figures such as percentiles can also be obtained by simple numerical operations.

Non-parametric methods lend themselves to providing a generic estimation for flexible percentile estimates. Instantaneous objective values for each MC run ($A = a_1, a_2, a_3, ..., a_N$) are sorted into ascending/descending order. Percentile estimates can be obtained from retrieving the correct position in the array. Computational overhead of the percentile estimation can be reduced by inserting into a sorted array each Monte Carlo simulation.

Example ▷ Assume the quantitative predictions reliability of an architecture X for each MC run (A) have been inserted into an array $S = s_1, s_2, s_3, ..., s_N$ sorted in ascending order, and the design objective is to get the 95^{th} percentile of reliability for architecture X.

Let $i_L = int(N * 95/100)$, $i_U = int(N * 95/100) + 1$ be two array indices for S, where $int(x)$ refers to a function that return integer part of x. Then the 95^{th} percentile of reliability ($\hat{a}$) can be given as:

$\hat{a} = s[i_L] + (s[i_U] - s[i_L]) \times \left(\frac{N \times 95}{100} - int\left(\frac{N \times 95}{100}\right)\right)$ ◁

5.5 Dynamic Stopping Criterion

All of the estimation techniques discussed above sample from appropriate distributions and obtain a desired descriptive figure of a quality attribute $\hat{a}$. However, the accuracy of the estimate $\hat{a}$ strongly depends on the sample size, *i.e.* on the number of MC trials carried out. One important characteristic of the problem is that the actual value of the estimate ($\hat{a}$) or the distribution of A is not known. Ideally, an infinite number of MC runs have to be carried out to obtain accurate $\hat{a}$ while small number of MC runs may lead to an inaccurate estimation (Law of Large Numbers). However, conducting large numbers of MC runs is prohibitive for the following reasons.

- MC simulations are computationally expensive. Each MC run involves sampling from distributions and evaluating or simulating the probabilistic quality model. The model evaluation itself is a highly processor intensive activity which often has computational complexity that grows exponentially with the problem size [200, 250, 290].
- The architecture evaluations have to consider more than one objective in most practical cases. Consequently, these MC runs have to be conducted for each of the objectives resulting in multiplication effect for the computation time.
- In the architecture optimisation process which is the focus of this thesis, large numbers of architecture candidates need to be evaluated, and results needs to be given in feasible time that the architecture can take necessary actions.

On the other hand, deriving the number of MC runs which delivers sufficient accuracy from the input variations (UPs in Figure 5.3) is also not possible due to the complex nature of the quality model and heterogeneity of input distributions. As it has been discussed in the previous sections, the quality goals are not limited to the estimation of the mean of a normal distribution hindering the applicability of concepts like sequential probability ratio test [400].

To solve this issue, the SCOUT approach proposes a dynamic stopping criterion based on accuracy monitoring. In this approach, the assumptions on the monitored

distribution (A) are relaxed by transforming the monitoring phase to the estimate $\hat{A}$. A statistical significance test is carried out on the samples of the descriptive figure ($\hat{A}$).

- A minimum of k Monte Carlo runs $(a_1, .., a_k)$ are conducted before estimating the desired index $\hat{A}$. After exceeding k number of runs, one of the methods discussed in Section 5.4.2 can be used to obtain each $\hat{a}$. All of the available attribute samples have to be used in computing each next estimate of the quality $\hat{a}_i$. The rationale behind this is that the statical index of the quality attribute has to be a representative of the population A.
- The variation of the estimate $\hat{A} = \{\hat{a}_1, \hat{a}_2, \hat{a}_3, ..., \hat{a}_k\}$ is monitored for a sliding window of size k. Only the last k samples of the estimate $\hat{A}$ are monitored, as the accuracy of the estimation is a changing property. The objective is to detect if sufficient accuracy has been obtained.
- The following statistical significance test is conducted for the last k estimates [345].

$$w_r = \frac{2z_{(1-\alpha/2)}}{\sqrt{k}} \frac{\sqrt{\overline{\hat{a}^2} - \left(\overline{\hat{a}}\right)^2}}{\overline{\hat{a}}} \tag{5.7}$$

  where : w_r is the relative error,
  $\overline{\hat{a}} = \frac{1}{k} \sum_{i=N-k}^{N} \hat{a}$ is the average of last k estimates,
  $\overline{\hat{a}^2} = \frac{1}{k} \sum_{i=N-k}^{N} \hat{a}^2$ is the mean-square of the last k estimates,
  α desired significance of the test,
  and z refers to the inverse cumulative density value of the standard normal distribution. Also note that the term $\sqrt{\overline{\hat{a}^2} - \left(\overline{\hat{a}}\right)^2}$ refers to the standard deviation of the last k estimates.

  The relative error w_r of the estimate $\hat{A}$ is checked against a tolerance level, *e.g.* 0.05. The complete algorithm is explained in Algorithm 1.

It should be noted that the parameters of the above algorithm, epoch size(k) and significance (α) can be set independently from the architecture evaluation problem. The significance test formula 5.7 computes the relative error, so that the tolerance

Algorithm 1: Monte Carlo simulation with dynamic stopping criterion

$i = 1, j = 1$;
while $w_r >$ *tolerable* w_r **do**
  $a_i :=$ conduct one MC execution();
  if $i \geq k$ **then**
    $\hat{a_j} :=$ non-parametric estimate using $(a_1, a_2, a_3, ..., a_i)$;
    if $j \geq k$ **then**
      $\overline{\hat{a}} = \frac{1}{k} \sum_{p=j-k}^{j} \hat{a}_p$;
      $\overline{\hat{a}^2} = \frac{1}{k} \sum_{p=j-k}^{N} \hat{a}_p^2$;
      $w_r = \frac{2z_{(1-\alpha/2)}}{\sqrt{k}} \frac{\sqrt{\overline{\hat{a}^2} - \left(\overline{\hat{a}}\right)^2}}{\overline{\hat{a}}}$;
    end
    $j++$;
  end
  $i++$;
end
$\hat{a}^* = \hat{a}_j$;

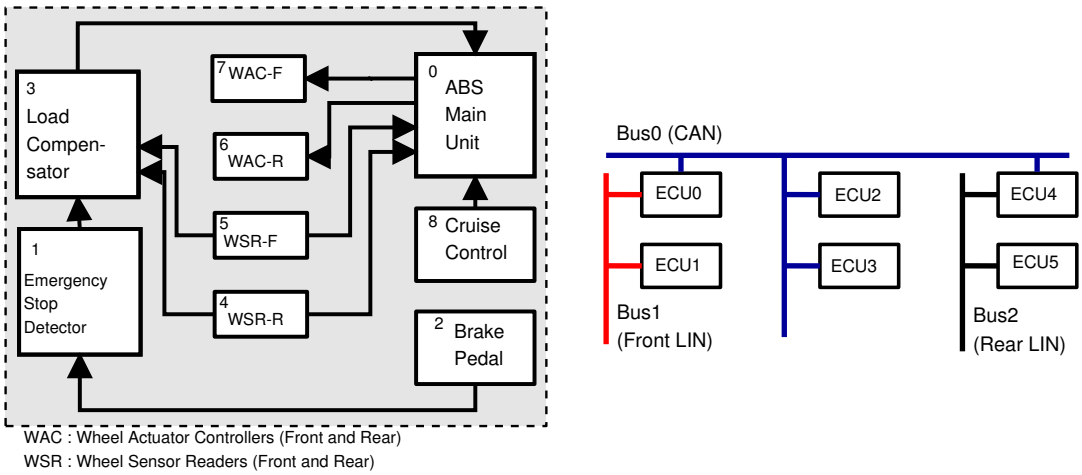
level is independent from the nominal range of the quality attribute. This formulation transforms the stopping criterion into a generic setting that is independent from the size of the problem, quality attribute as well as the numerical range of the estimate.

5.6 Illustration Using an Example Application

5.6.1 System Description

In order to illustrate the concepts described in this chapter, a part of the case study from the automotive industry which has been presented in Chapter 3 is used. The complete case study aims to map the components of software-controlled automotive subsystems (*i.e.* ABS and ACC) to a network of ECUs, optimising the reliability of the two services. This chapter focuses on the evaluation of a single candidate deployment. To maintain the focus of the presentation to new concepts related

to parameter uncertainty, the scope of the case study within this chapter is further narrowed down to one of the services. Figure 5.4(a) depicts the software components and their interactions corresponding to Anti-lock Brake System (left). The ECUs and the bus system that compose the hardware platform for the system is depicted in Figure 5.4(b). It is to be noted that only the subset of ECUs and buses in the whole automotive electronic system that contribute to the ABS service is considered for this illustration. For further details on the ABS system and parameters of software and hardware components, please refer to Chapter 3.



(a) Software components and interactions (b) Hardware platform

Figure 5.4: Software and hardware architectures of the ABS system

Deployment

This example considers one of the feasible deployments of software components to the hardware architecture. The assignment of software components in the considered deployment is given in Figure 5.5. The encircled numbers in the figure refer to the allocated software components with corresponding ids in Figure 5.4(a).

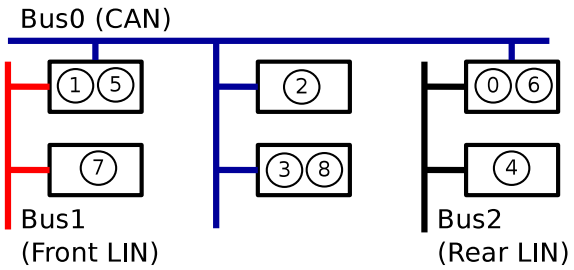


Figure 5.5: Deployment of software components to ECUs

Objectives

The problem in focus is to evaluate the reliability of the ABS function from the architecture. With respect to the deployment, two sources of failure are considered for reliability evaluation.

1. **Execution failures**. Failures may occur in the ECUs during execution of a software component, which affects the reliability of the software modules running on that ECU. For this illustration, a fixed deterministic scheduling of tasks is assumed in the ECU. It is also assumed that the failure that happens in an ECU while a software component is executing or queued leads to a service execution failure.
2. **Communication failures**. Failure of a data communication bus when a software component communicates with another one over the bus directly causes a failure in the service that depends on this communication.

The annotations, model and evaluation of the reliability are progressively presented in upcoming sections.

5.6.2 Specification of Uncertain Parameters

Not every parameter pertaining to the current example is subject to uncertainty. For instance, the processing speed (ps) of an ECU or the computational load (wl) of a software component can realistically be considered fixed and deterministic. However, parameters such as the failure rates of ECUs, failure rates of buses, execution

initiation probabilities and transition probabilities are subject to uncertainty and have to be estimated. Table 5.2 shows an example set of parameters.

Comp. ID	wl (MI)	q_0
0	1.2	0
1	0.6	0
2	0.4	$DISCRETE, 0.03, 0.2, 0.3, 0.4, 1.5, 0.2, 3, 0.2$
3	1	0
4	0.4	$NORMAL, 0.3, 0.075$
5	0.4	$NORMAL, 0.3, 0.075$
6	0.4	0
7	0.4	0
8	0	$DISCRETE, 0.01, 0.2, 0.1, 0.4, 0.5, 0.2, 1, 0.2$

(a) Software Components

Trans $c_i \rightarrow c_j$	$p(c_i, c_j)$	ds (KB)
$0 \rightarrow 6$	$DISCRETE, 0.05, 0.2, 0.5, 0.4, 2.5, 0.2, 5, 0.2$	2
$0 \rightarrow 7$	$DISCRETE, 0.05, 0.2, 0.5, 0.4, 2.5, 0.2, 5, 0.2$	2
$1 \rightarrow 3$	1	2
$2 \rightarrow 1$	1	2
$3 \rightarrow 0$	1	2
$4 \rightarrow 0$	$GAMMA, 0.7$	1
$4 \rightarrow 3$	$GAMMA, 0.3$	2
$5 \rightarrow 0$	$GAMMA, 0.7$	1
$5 \rightarrow 3$	$GAMMA, 0.3$	2
$8 \rightarrow 0$	1	0

(c) Component Interactions

ECU ID	ps (MIPS)	fr (h^{-1})
1	40	$BETA\_SHD, 4 \cdot 10^{-5}, 4 \cdot 10^{-3}, 10, 2$
2	22	$BETA\_SHD, 4 \cdot 10^{-5}, 4 \cdot 10^{-3}, 10, 2$
3	22	$DISCRETE, 2 \cdot 10^{-6}, 0.2, 2 \cdot 10^{-5}, 0.4, 1 \cdot 10^{-4}, 0.2, 2 \cdot 10^{-4}, 0.2$
4	22	$DISCRETE, 1 \cdot 10^{-5}, 0.2, 1 \cdot 10^{-4}, 0.4, 5 \cdot 10^{-4}, 0.2, 1 \cdot 10^{-3}, 0.2$
5	110	$NORMAL, 8 \cdot 10^{-4}, 0.04$
6	110	$NORMAL, 2 \cdot 10^{-4}, 0.01$

(b) ECUs

BUS ID	dr (KBPS)	fr (h^{-1})
0	128	$BETA\_SHD, 3 \cdot 10^{-6}, 3 \cdot 10^{-4}, 10, 2$
1	64	$BETA\_SHD, 1.2 \cdot 10^{-5}, 1.2 \cdot 10^{-3}, 10, 2$
2	64	$BETA\_SHD, 4 \cdot 10^{-6}, 4 \cdot 10^{-4}, 10, 2$

(d) Buses

Table 5.2: Parameter specification of software and hardware elements of the architecture

The probabilistic nature of the parameters in the tables reflects their variability in relation to the automotive ABS system. Different distributions with values which correspond to realistic ranges [23, 59, 178, 289] are used for the illustration. The sources and their influences can be different in each instance. Dissimilar distributions are used even within the same property for different instances. Therefore, the SCOUT approach allows for different PDFs in combination with distinct values, within the same column.

5.6.3 Probabilistic Model Construction

As an example to show how the probabilistic parameters are manifested in the probabilistic quality evaluation model, this section presents the details on the mapping of architecture level annotations to the evaluation model parameters in the considered case study. The mathematical formulation of the model is presented in detail to

explain the propagation of probabilistic specifications through the model evaluation in the model-based reliability evaluation.

In order to obtain a quantitative estimation of the reliability of the automotive architecture in focus, a well-established Discrete Time Markov Chain (DTMC)-based reliability evaluation model [175, 182, 391] is used. An absorbing DTMC [391] is constructed for each subsystem from the software components and hardware specification, such that a node represents the execution of a component and arcs denote the transfer of execution from one component to the other. A super-initial node [404] is added to represent the execution start, and arcs are added from that node annotated with relevant execution initialisation probabilities(q_0). Figure 5.6 illustrates the constructed DTMCs for the running example and node labels corresponding to Figure 5.4(a). Failure rates of *execution elements* can be obtained from

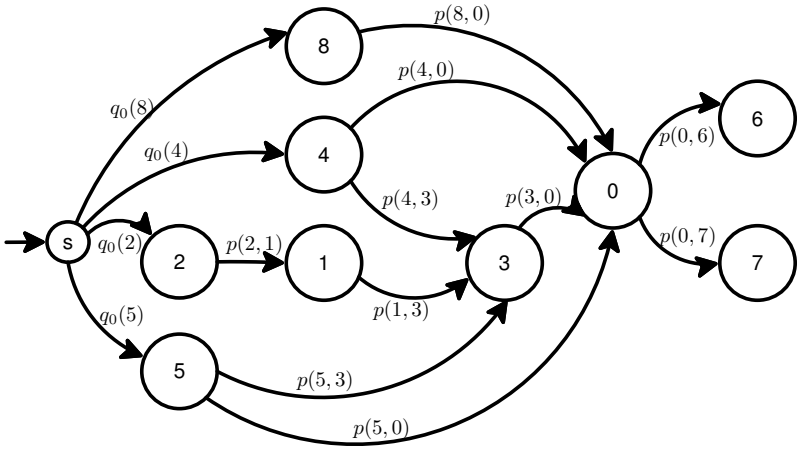


Figure 5.6: Annotated DTMCs for service reliability evaluation

the ECU parameters, and the time taken for the execution defined as a function of the software-component workload and processing speed of its ECU. Similar to the models used in previous work [23, 288], reliability of ABS considers both ECU and communication link failures. In detail, the reliability of a component c_i can be computed as given in Equation 5.8.

$$R_i = e^{-fr(d(c_i))\cdot\frac{wl(c_i)}{ps(d(c_i))}} \tag{5.8}$$

where $d(c_i)$ denotes the ECU allocation relationship of component c_i. A similar computation can be employed for the reliability of *communication elements* [288], which in this system model are characterised by the failure rates of hardware buses, and the time taken for communication, defined as a function of the buses data rates (dr) and data sizes (ds) required for software communication. Therefore, the reliability of the communication between component c_i and c_j is defined by Equation 5.9.

$$R_{ij} = e^{-fr(d(c_i),d(c_j))\cdot\frac{ds(c_i,c_j)}{dr(d(c_i),d(c_j))}} \tag{5.9}$$

Expected number of visits of DTMC nodes $v_i : C \to \mathbb{R}_{\geq 0}$, quantifies expectation of use of a component (or subsystem) per single system execution. This can be computed by solving the following set of simultaneous equations [175, 241]:

$$v(c_i) = q_0(c_i) + \sum_{j \in \mathcal{I}} (v(c_j) \cdot p(c_j, c_i)) \tag{5.10}$$

The formula (5.10) can be expanded as follows:

$$\begin{aligned}
v(c_0) &= q_0(c_0) + v(c_0)\cdot p(c_0,c_0) + v(c_1)\cdot p(c_1,c_0) + v(c_2)\cdot p(c_2,c_0) + ... + v(c_n)\cdot p(c_n,c_0)\\
v(c_1) &= q_0(c_1) + v(c_0)\cdot p(c_0,c_1) + v(c_1)\cdot p(c_1,c_1) + v(c_2)\cdot p(c_2,c_1) + ... + v(c_n)\cdot p(c_n,c_1)\\
v(c_2) &= q_0(c_2) + v(c_0)\cdot p(c_0,c_2) + v(c_1)\cdot p(c_1,c_2) + v(c_2)\cdot p(c_2,c_2) + ... + v(c_n)\cdot p(c_n,c_2)\\
&\vdots\\
v(c_n) &= q_0(c_n) + v(c_0)\cdot p(c_0,c_1) + v(c_1)\cdot p(c_1,c_n) + v(c_2)\cdot p(c_2,c_n) + ... + v(c_n)\cdot p(c_n,c_n)
\end{aligned}$$

In matrix form, the transfer probabilities $p(c_i, c_j)$ can be written as $P_{n\times n}$, and execution initiation probabilities $q_0(c_i)$ as $Q_{n\times 1}$. The matrix of expected number of visits $V_{n\times 1}$ can be expressed as:

$$V = Q + P^T \cdot V$$

With the usual matrix operations, the above can be transformed into the solution format:

$$I \times V - P^T \times V = Q \tag{5.11}$$

$$(I - P^T) \times V = Q \tag{5.12}$$

$$V = (I - P^T)^{-1} \times Q \tag{5.13}$$

For absorbing DTMCs which is also the case for the model used in this illustration, it has been proved that the inverse matrix $(I - P^T)^{-1}$ exists [391].

The expected number of visits of a communication link, $v(l_{ij}) : C \times C \to \mathbb{R}_{\geq 0}$, quantifies for each link $l_{ij} = (c_i, c_j)$ the expected number of occurrences of the transition (c_i, c_j) in the underlying DTMC. To compute this value, the communication links can be understood as first-class elements of the model, and view each probabilistic transition $c_i \xrightarrow{p(c_i,c_j)} c_j$ in the model as a tuple of transitions $c_i \xrightarrow{p(c_i,c_j)} l_{ij} \xrightarrow{1} c_j$, the first adopting the original probability and the second having probability $=1$. Hence, the expected number of visits of a communication link can be computed as:

$$v(l_{ij}) = 0 + \sum_{x \in \{i\}} (v(c_x) \cdot p(c_x, l_{ij})) \tag{5.14}$$

$$= v(c_i) \cdot p(c_i, c_j) \tag{5.15}$$

since the execution is never initiated in l_{ij} and the only predecessor of link l_{ij} is component c_i.

Based on the relationships obtained in equations(5.8) and (5.9), the reliability of the deployment is calculated as defined by Equation 5.16.

$$R \approx \prod_{i \in \mathcal{I}} R_i^{v(c_i)} \cdot \prod_{i,j \in \mathcal{I}} R_{ij}^{v(l_{ij})} \tag{5.16}$$

The mathematical formulation to derive service reliability given in Formula 5.16

illustrates how the architecture level annotations are mapped to the model parameters, and how they are used in computing the predicted reliability of a given deployment architecture. Some of the entries for the parameters in Table 5.2 $p_{i,j}$, fr_i, $fr_{i,j}$, q_{0_i} are probability distributions. These parameters form part of the final reliability calculation in Equation (5.16). The matrix formula (5.13) contains entries of heterogeneous PDFs for p_{ij} and q_{0_i}. Consequently, the model evaluation results of vector V of formula (5.13) become probabilistic, and cannot be solved with numerical matrix operations. Furthermore, R_i and R_{ij} in formulations (5.8) and (5.9) are influenced by the probabilistic specifications of fr_i and $fr_{i,j}$ in Table 5.2. The propagation of parameter uncertainties to the system reliability R can be further observed in the use of probabilistic V in combination with uncertain R_i and R_{ij} in (5.16).

5.6.4 Quality Metric Estimation

The dynamic-stopping MC simulation process described in Section 5.4 can be applied to the example as the follows;

(1). As explained in Section 5.6.3, the reliability model is constructed from the architecture specification. Then the sampling in Monte Carlo simulation takes one sample from each distribution parameters given in Table 5.2. Also note that some parameters are precise values, while some are specified as distributions. However, after taking samples of each distribution, all parameters values appear as numerical figures.

(2). Model-specific parameter dependencies are resolved in this phase. In the DTMC based reliability model, model parameters have to satisfy the following properties.

- The sum of all outgoing transitions (p_{ij}'s) from any non-absorbing node should be equal to 1.
- The sum of execution initialisation probabilities(q_0's) should be equal to 1. This can also be considered as the satisfaction of the previous rule for

the super initial node.

Hence, all outgoing transition probabilities per node are normalised before proceeding to the next step.

(3). These values are substituted into Equation 5.10 and solve the matrix formulae 5.13. Then the expected visits for the links can also be found by solving Equation 5.14.

(4). From sample of failure rates of ECUs and buses, reliabilities of them can be calculated by substituting formulae 5.8 and 5.9. Consequently, the system reliability estimate for the samples obtained in step (1) can be obtained using Equation 5.16. This process represents a single MC run, and the reliability value is labelled as $a_i \in A$.

(5). The steps (1) to (4) are repeated k (*e.g.* 10) times, where k represents the epoch size. Then the accuracy estimation process is started. We assume a goal of the 10^{th} percentile reliability (*i.e.* 90% of the cases are greater than this value), and individual estimates are denoted by $\hat{a_i} \in \hat{A}$. For k samples, the initial estimate of reliability ($\hat{a_1}$) is computed from non-parametric percentile estimation (5^{th} percentile) described in Section 5.4.3.

$$\hat{a_1} = 5^{th}percentile(a_1, a_2, .., a_k)$$
$$\hat{a_2} = 5^{th}percentile(a_1, a_2, .., a_k, a_{k+1})$$
$$\hat{a_3} = 5^{th}percentile(a_1, a_2, .., a_k, a_{k+1}, a_{k+2})$$
$$.$$
$$.$$
$$\hat{a_k} = 5^{th}percentile(a_1, a_2, .., a_k, a_{k+1}, a_{k+2}..., a_{2k})$$

It should be noted that all existing samples are used estimating the values of $\hat{a}$ after each MC run. When k number of estimates ($\hat{a}$ s) are available, the dynamic stopping criterion is applied. This happens for the first time after

$2k$ samples are taken. To check if the stopping criterion has been met, w_r is computed for last k estimates $\hat{a}$:

$$w_r = \frac{3.92}{\sqrt{k}} \frac{Std.Dev(\hat{a}_1, \hat{a}_2, \hat{a}_3, ..., \hat{a}_k)}{Average(\hat{a}_1, \hat{a}_2, \hat{a}_3, ..., \hat{a}_k)}$$

where 3.92 corresponds to the value of $2z_{(1-\alpha/2)}$ in Equation 5.7 with $\alpha = 0.05$.

(6). If w_r is less than a threshold (*e.g.* 0.05), the last $\hat{a}$ is considered as the 10^{th} percentile reliability of the architecture. Otherwise, the process repeats from step 1. When the number of MC runs exceed $2k$, the stopping criterion is checked for each MC run. It is further emphasised that for computing $\hat{a}$'s all the previous samples will be used, but the stopping criterion is only checked for last k of $\hat{a}$'s. When the stop criterion is reached, the final estimate of $\hat{a}$ is taken as the quality metric $\hat{A}^*$.

5.6.5 Experimental Results

The presented MC simulation process has been explored using the partial case study adopted for this chapter. The results of 3000 Monte Carlo runs are presented in Figure 5.7. The instantaneous values of the samples for each MC run, taken as described in step 4 in Section 5.6.4, are scattered as depicted in Figure 5.7a. It can be seen that the values for the reliability vary from 0.85 to 0.999, which is a significant variation with respect to the notion of reliability. Histogram of the reliability obtained from 3000 samples is depicted in Figure 5.7b. It should be emphasised that the conventional assumption on normal or Weibull distribution of the system reliability does not comply with the actual characteristics of the sample distribution obtained from MC simulation.

The goal of the MC run has been set to the 20^{th} percentile reliability, *i.e.* reliability of the system will be greater than the estimate for 80% of the cases (under the uncertainty in the parameter spaces). The minimum samples for the estimation (epoch size, k) has been set to 10, and the values of each estimation is presented

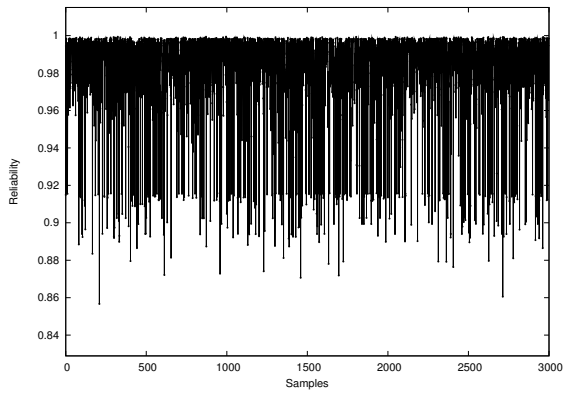
in Figure 5.7c (note that the graph starts at sample size of 10). A considerable variation can be observed at the early estimations, while observable stabilisation is achieved after around 100 MC run. As described in Section 5.5, the stabilisation is identified by a sequential significance test. The variation of the relative error with increasing MC samples is depicted in Figure 5.7d. Note that the x-axis values start at 20, which is $2k$ as initial computation refers to first k estimates. It could be observed that the variation characteristics in Figure 5.7c has been reflected to error values in Figure 5.7d. The proposed dynamic stopping criterion is applied with a setting of an appropriate tolerance value for the relative error, and MC simulation is terminated with significant accuracy of the estimate. The experiments with significance test configuration $\alpha = 0.05$ and $w_r = 0.0005$, the MC simulation achieves the accuracy at 156 runs, with the estimate for 20^{th} percentile reliability of 0.974527.

5.7 Experiments on Generated Problems

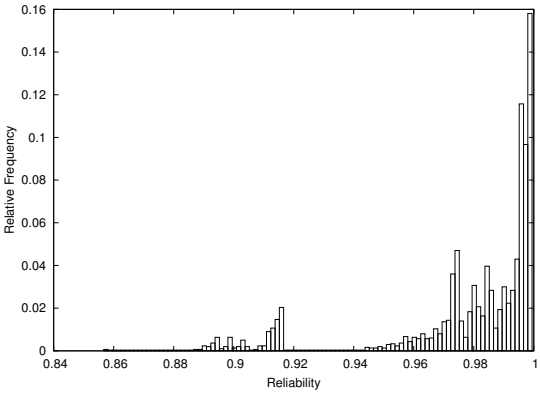
5.7.1 Experiment Setup

A series of experiments has been conducted to investigate the presented architecture evaluation approach under uncertainty. The scalability of the approach is examined by generating instances from a range of problem sizes. The objective of each problem is set to estimation of reliability, and the architecture-based reliability evaluation model presented in Section 5.3 is used for the evaluation.

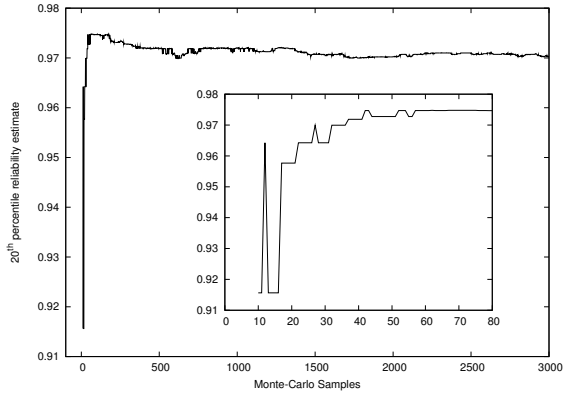
- To represent uncertainties in component reliability figures, each component's reliability is specified as a random distribution in the range $0.99 - 0.9999$. For a single problem, once a component is assigned a PDF, the distribution is kept unchanged throughout the experiments to maintain the comparability of the results.
- Additional distributions are introduced to represent uncertainties associated with other parameters. The number of additional uncertainties are varied from 0 to 10 for each problem size in order to investigate the level or uncertainty in



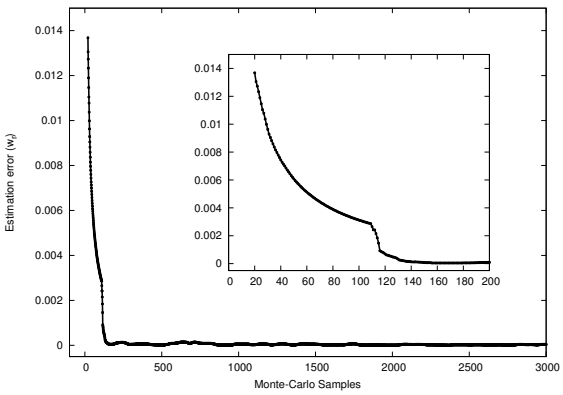
(a) Instantanious samples of reliability in Monte Carlo runs



(b) Histogram of reliability samples



(c) Variation of the accuracy of the estimation with MC runs



(d) Relative error obtained from sequential accuracy monitoring

Figure 5.7: Results of the experiments with ABS system example

different instances. Random parameters in the experiments refers to selecting a distribution from a set of candidate types of Normal, Uniform, Beta, Shifted Beta, Gamma, Exponential, Weibull and Discrete, and their parameters from random sets.

- In order to represent diversity in the relationship between architecture and model, the DTMC is constructed using random combinations of architecture parameters. Architectures are created as a random relationship among a set of components that has parameters selected from random sets. One architecture parameter may have an effect on randomly selected transition probabilities in

the generated DTMC.

- The support for different levels of compromise in the estimation process is captured by conducting each problem instance for median, 25^{th} percentile (75% pessimistic), 5^{th} percentile (95% pessimistic) of the reliability. Dynamic stopping criteria is set to $\alpha = 0.05$, $w_r = 0.005$ and $k = 10$.

The configurations for problem instances are given in Table 5.3.

Case ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DTMC Nodes	10	10	10	10	20	20	20	20	50	50	50	50	100	100	100	100
Additional Uncertainties	0	2	5	10	0	2	5	10	0	2	5	10	0	2	5	10

Table 5.3: Experiment configurations

5.7.2 Results

Table 5.4 summarises the results for the quality estimation goals being the expected value, 25^{th} percentile (75% pessimistic), 5^{th} percentile (95% pessimistic) of the reliability. The value N in the table refers to the Monte Carlo runs that first satisfied the stopping criterion. The estimation of the quality at the stopping condition is listed in the columns with $\hat{a}^*$.

The MC simulations are carried out for a large number of runs (10000), even after the stopping criterion has been met. The final estimation a_f obtained from 10000 runs is compared with the estimation achieved at the stopping condition. The column d_r indicates the relative difference between $\hat{a}^*$ and a_f calculated as:

$$d_r = \left(\frac{\hat{a}^* - a_f}{a_f} \right)^2 \tag{5.17}$$

In all cases, the relative difference d_r is less than the relative error w_r calculated by Equation 5.7. The results show that the approach is applicable to different sizes of the problem as well as diverse levels of uncertainty. The accuracy of the MC simulations comply with the specified tolerance levels. It should be noted that the novel stopping criterion controls the process with the effect of saving large number

Case ID	Median					25^{th} percentile					5^{th} percentile				
	N	$\hat{a}^*$	a_f	d_r	$d_r < w_r$?	N	$\hat{a}^*$	a_f	d_r	$d_r < w_r$?	N	$\hat{a}^*$	a_f	d_r	$d_r < w_r$?
1	19	0.952560	0.954734	0.000005	√	19	0.945523	0.948031	0.000007	√	19	0.935858	0.939333	0.000014	√
2	19	0.965903	0.962922	0.000010	√	19	0.954604	0.957312	0.000008	√	19	0.949370	0.949266	0.000000	√
3	19	0.966705	0.968018	0.000002	√	19	0.962980	0.961934	0.000001	√	19	0.934353	0.952257	0.000353	√
4	19	0.942744	0.932468	0.000121	√	19	0.918319	0.918192	0.000000	√	23	0.903614	0.891964	0.000171	√
5	19	0.897277	0.902449	0.000033	√	19	0.890973	0.892493	0.000003	√	19	0.880296	0.878284	0.000005	√
6	19	0.913447	0.907576	0.000042	√	19	0.900099	0.899102	0.000001	√	26	0.877142	0.887624	0.000139	√
7	19	0.912154	0.916944	0.000027	√	19	0.908060	0.908293	0.000000	√	19	0.888432	0.894926	0.000053	√
8	19	0.966131	0.965325	0.000001	√	19	0.961862	0.960805	0.000001	√	19	0.948812	0.948506	0.000000	√
9	19	0.784776	0.785679	0.000001	√	30	0.767856	0.773189	0.000048	√	19	0.762634	0.756603	0.000064	√
10	28	0.739862	0.736462	0.000021	√	21	0.721135	0.721837	0.000001	√	19	0.698968	0.701438	0.000012	√
11	19	0.783287	0.778369	0.000040	√	19	0.774306	0.762080	0.000257	√	133	0.725934	0.735245	0.000160	√
12	19	0.771074	0.748191	0.000935	√	125	0.727915	0.722666	0.000053	√	257	0.691504	0.681990	0.000195	√
13	19	0.592317	0.594764	0.000017	√	19	0.579529	0.580552	0.000003	√	48	0.562311	0.560448	0.000011	√
14	64	0.593832	0.593420	0.000000	√	34	0.572371	0.579086	0.000134	√	19	0.545807	0.557887	0.000469	√
15	33	0.584036	0.589625	0.000090	√	19	0.576236	0.573726	0.000019	√	21	0.563660	0.552131	0.000436	√
16	269	0.536075	0.530330	0.000117	√	241	0.481548	0.483096	0.000010	√	19	0.438971	0.421210	0.001778	√

Table 5.4: Results of the randomly generated experiments against 16 problem instances and 3 classes of tolerance

of computational resources. For example in 5^{th} percentile estimations, many cases have achieved sufficient accuracy with a small number of MC runs, while cases like 12 are automatically continued for longer to obtain an accurate estimation.

5.7.3 Discussion of the Results

Internal validity

The validity of the reliability evaluation method with uncertainty analysis has been illustrated with respect to the example case study as well as with a series of random experiments. The new framework's capability of handling a diverse range of probability distributions has been validated with the experiments using random distributions correlating to the research questions listed under RQ 1 in Chapter 4. It has been shown that the new approach can successfully evaluate a number of problem instances in different characteristics and sizes, indicating the applicability of the approach over diverse problem settings. The MC simulator has been configured to quantify different percentiles of reliability, in order to cover moderate and more pessimistic tolerance requirements in practice. The novel dynamic stopping criterion has been tested for the 16 random cases and for three different percentile estimations, and the accuracy of the tests has been validated under the specified tolerance levels.

In order to represent diverse conditions in the problem space, the experiments have been generated to be as random as possible. All the parameters, architecture to DTMC relationships have been generated from random sets. The trust on randomness is based on the uniform random number generator implementation in java. In practical situations, the variations of the parameters will not be random, therefore the experiments do not exactly represent a practical example. However, the use of complete random distributions confirms the applicability of the approach to constrained distributions, as any variation in general can inherently be captured in the experiments.

External validity

It is to be noted that all experiments in this chapter explore the model of a single system attribute, reliability. Validity against the spectrum of models and architecture parameters may be carried out with further experiments. However, the framework presented in this chapter is deliberately generic and treats the probabilistic evaluation model as a *black box*, avoiding a dependency on the internal complexity of the model. Therefore, it is suggested that the approach can be applied to any architecture-based evaluation model, even though it has only been validated with regards to a specific reliability model.

5.8 Comparison to Related Work

In the context of software architecture evaluation under uncertainty, a considerable amount of work can be found in the area of sensitivity analysis with respect to the parameters of probabilistic quality models. Most of the approaches to date have concentrated on specific quality attributes. In the area of architecture-based reliability evaluation, Cheung [86] presented a sensitivity analysis method for his original reliability model with a composite Discrete-Time Markov Chain (DTMC) abstraction. The method is purely analytical and consists of a number of 2^{nd} and 3^{rd} order partial derivatives of system reliabilities which are hard to estimate in real cases. Goševa-Popstojanova *et al.* [181] proposed the *method of moments* to calculate the sensitivity of a system's reliability to component reliabilities and transition probabilities analytically. Cortellessa *et al.* [103] discussed the significance of error propagation to other parts of the system. Their sensitivity analysis can help identify the most critical system components. Coit *et al.* [94, 405] have used means and variances of reliability estimates of software components to analytically derive the mean and variance of the reliability of a redundancy allocation. With the assumption of normal distributions for input, Finodella *et al.* [151] derived the distribution of system reliability from a multinomial distribution. Coit *et al.* [100] presented an

analytical approach to obtain the lower bound percentile of the reliability in series-parallel systems whereas similar analytical approach can be seen in evaluation of reliability bounds [56]. All of the above methods have taken an analytical approach to quantify the sensitivity, where the applicability is limited to analytically solvable models. However, these analytical sensitivity analysis methods are hard to generalise. Furthermore, all the discussed approaches assume the parameter distributions are normal and variations can be characterised by the mean and variance alone.

Goševa-Popstojanova *et al.* [179, 180] have shown that analytical methods of uncertainty analysis do not scale well, and proposed a Monte Carlo (MC) simulation-based method. With the help of experimental validation they demonstrated that the MC methods scale better than the *method of moments* approach [177]. Similarly, Marseguerra *et al.* [278] have used mean and variance estimates of component reliabilities to obtain the mean and variance of the system reliability using MC simulation. These approaches have extended the applicability of uncertainty analysis to the analytically solvable models, assuming the applicability of specific reliability models and input distributions. Yin *et al.* [413] have proposed a DTMC-simulation-based approach to derive system reliability from the probability distributions of component reliabilities under the assumption that component and system reliabilities are gamma-distributed. Axelsson [11] has also highlighted the significance of MC based approaches in cost evaluation with uncertainty.

However, the existing MC simulation-based uncertainty analysis approaches assume that there is a specific continuous input distribution and that the resulting sample distribution is normal or Weibull. Practical experience in connection with some studies [156, 261] show that the actual distributions are hard to determine. Mean-and-variance-based quality evaluations are not sufficient for architecture-based decision making in the case of safety-and-mission-critical systems.

In the current approach, assumptions on input distributions have been relaxed by allowing the input parameters to take any probability distribution including the discrete distributions. The SCOUT approach also derives a statistical estimation

method to obtain flexible percentiles without requiring assumption on the distribution of evaluated reliability. A similar concept has been used in safety evaluation domain by Förster *et al.* [156] in obtaining the sensitivity of hazard probabilities to the probability uncertainties of low level failure events. However, the limitations arising from the histogram to PDF approximation in that approach, have been addressed in this chapter with the introduction of statistical estimation phase. It is also acknowledged that orthogonal non-probabilistic approaches are also emerging such as the use of Dempster-Shafer theory of evidence by Limbourg [260] in system reliability evaluation.

All the discussed Monte Carlo simulation approaches in the domain uses a fixed number of MC runs, which needs to be given by the architect or analyst. None of the approaches present a guidance to obtain an appropriate number of MC runs for different problems. No approach has been found that applies any dynamically stopping MC simulation in the domain of software architecture evaluation. The approach presented in the chapter introduces a generic criterion to stop the MC simulations automatically when sufficient accuracy of the estimate has been achieved. The uncertainty analysis framework presented in this chapter encapsulates the powers of existing MC simulation based approaches in a generalised setting for software architecture evaluation, together with the newly introduced concepts of sequential estimation, dynamic stopping criterion.

5.9 Conclusions

This chapter addressed the problem of evaluating probabilistic properties from the software architecture in the presence of uncertainty. The different perspectives of the problem have been investigated with the identified research gaps in relation to the state of the art. The chapter presented an evaluation framework featuring the following:

- A generalised probabilistic specification has been formulated for the purpose of capturing the extent and characteristics at the architecture parameter speci-

fication. The conventional assumptions on input parameter distributions have been relaxed and extended support has been introduced for heterogeneous software architecture parameters.

- Monte Carlo simulation has been adopted for the uncertainty analysis, in order to cater for probabilistic quality evaluation models that are hard to solve analytically as well as for the simulation modes.
- Assumptions on distributions of the quality metric have been relaxed, and distribution-free statistics has been adopted to support different levels tolerance. Conventional limitations on mean, variance of the desired quality analysis have been mitigated with the further support of quartile and percentiles. This extends the applicability of the approach to domains like safety critical embedded systems, where more pessimistic estimates are necessary for certain quality attributes.
- A new dynamic stopping criterion has been introduced which automatically finds the trade-off between time complexity and accuracy of the MC simulation. Principles of sequential statistical testing has been combined with continuous accuracy monitoring mechanism in the MC simulation. The new approach is able to terminate the MC simulation process when specified accuracy level has been reached.

An example of an architecture-based reliability evaluation of automotive ABS system has been used to illustrate the concepts as well as to highlight the gaps in relation to the state of the art analysis techniques. In addition to the experiments on the case study, a series of random experiments has been generated and analysed with respect to the concepts presented in the chapter. The implications of the size of the architecture, different probability distributions and level of compromise have been investigated. The experimental validations are limited to architecture based reliability evaluation, and a DTMC based reliability evaluation model. However, the presented framework is formulated in generic setting and does not have a dependency on the specific model or attribute. Even though extensive experiments over

the spectrum of models and quality attributes is necessary for proper experimental validation on generic applicability, it is hard to argue against applicability of this results to any architecture-based quality evaluation model.

Chapter 6

Robust Architecture Optimisation Framework

6.1 Introduction

The previous chapter discusses how a software architecture can be quantitatively evaluated considering the uncertainty associated with its estimated parameters. The emphasis so far has been on a given candidate architecture. However, the overall goal of this work is to devise a method to find a (near-)optimal set of architecture alternatives in the presence of uncertainty. Having presented the new approach on evaluating the quality of the candidates considering the uncertainty associated with input parameters, this chapter describes the optimisation aspect of the SCOUT approach, *i.e. "how can a (near-)optimal set of candidate architectures be found in the presence of uncertainty in the parameters of the architecture-based quality evaluation models" (RQ 2).* The chapter investigates the pertinent research questions that have been described under RQ 2 in Chapter 4, and formulates a robust optimisation framework which provides a solution to the research questions.

The rest of the chapter is organised as follows. Section 6.2 formulates a high-level model for architecture optimisation which integrates aspects related to parameter uncertainties. The robust architecture optimisation problem is formally defined in

Section 6.3. Using the devised model in the prior sections, Section 6.4 presents the integration of commonly used stochastic optimisation algorithms for the search of robust and (near-) optimal candidate architectures. In Section 6.5, the presented contributions are discussed in the context of existing work on software architecture optimisation modelling and optimisation under uncertainty, with a focus on how the new approach addresses the research questions stated above. The summary and conclusions are made in Section 6.6.

6.2 Modelling

In order to pursue an architecture optimisation approach which can perform under uncertainty, the SCOUT approach first formulates a high-level model for the architecture optimisation and entities related to parameter uncertainty. An abstract view is devised which is designed to capture the conceptual entities and their inter-relationships of the embedded architecture design context. The newly introduced uncertainty-related aspects are integrated into an abstract view of the architecture optimisation.

On the direction towards a high-level model, this work recognises that the ISO/IEC 42010 standard [219] already provides a generic view of the architecture of software-intensive systems. Having described the relevant concepts captured in the ISO 42010 model in Chapter 2, this work specifically focuses on the entities relating to the architecture design process, which are marked with a grey background in Figure 6.1. This conceptual understanding is used as the basis for the development of a high-level model for robust optimisation. The modelling of the SCOUT approach's framework is devised from the common understanding on the related work described in Chapter 2. None of the previously mentioned approaches consider the uncertainty of input parameters into the modelling. The underlying assumption there is that the input parameters are given as point estimates. Using the intuition of the related work, the SCOUT approach derives high-level entities for the concepts related to architecture, architecture-based quality evaluation and optimisation. The model is

further extended by integrating the entities related to the specification and analysis of uncertainty in input parameters.

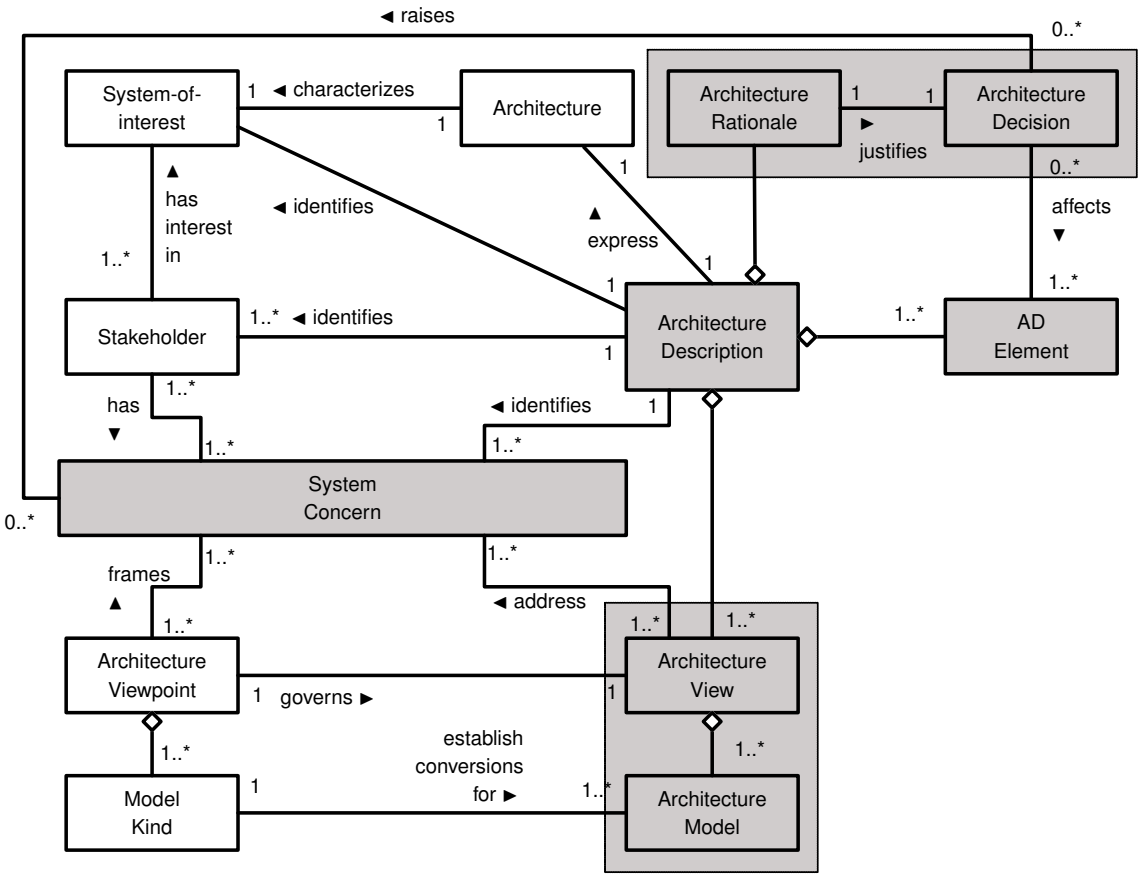


Figure 6.1: ISO/IEC 42010 conceptual model of software-intensive systems architecture

The UML model of the robust architecture optimisation framework the SCOUT approach is depicted in Figure 6.2. The entities with a grey background correspond to the abstract concepts in the ISO model. Note that the relative positions of the elements correspond to each other in the two Figures 6.1 and 6.2. The white elements in Figure 6.2 are the new conceptual entities introduced to capture aspects related to uncertainty and robust architecture optimisation. Individual elements in the model are described in the following paragraphs.

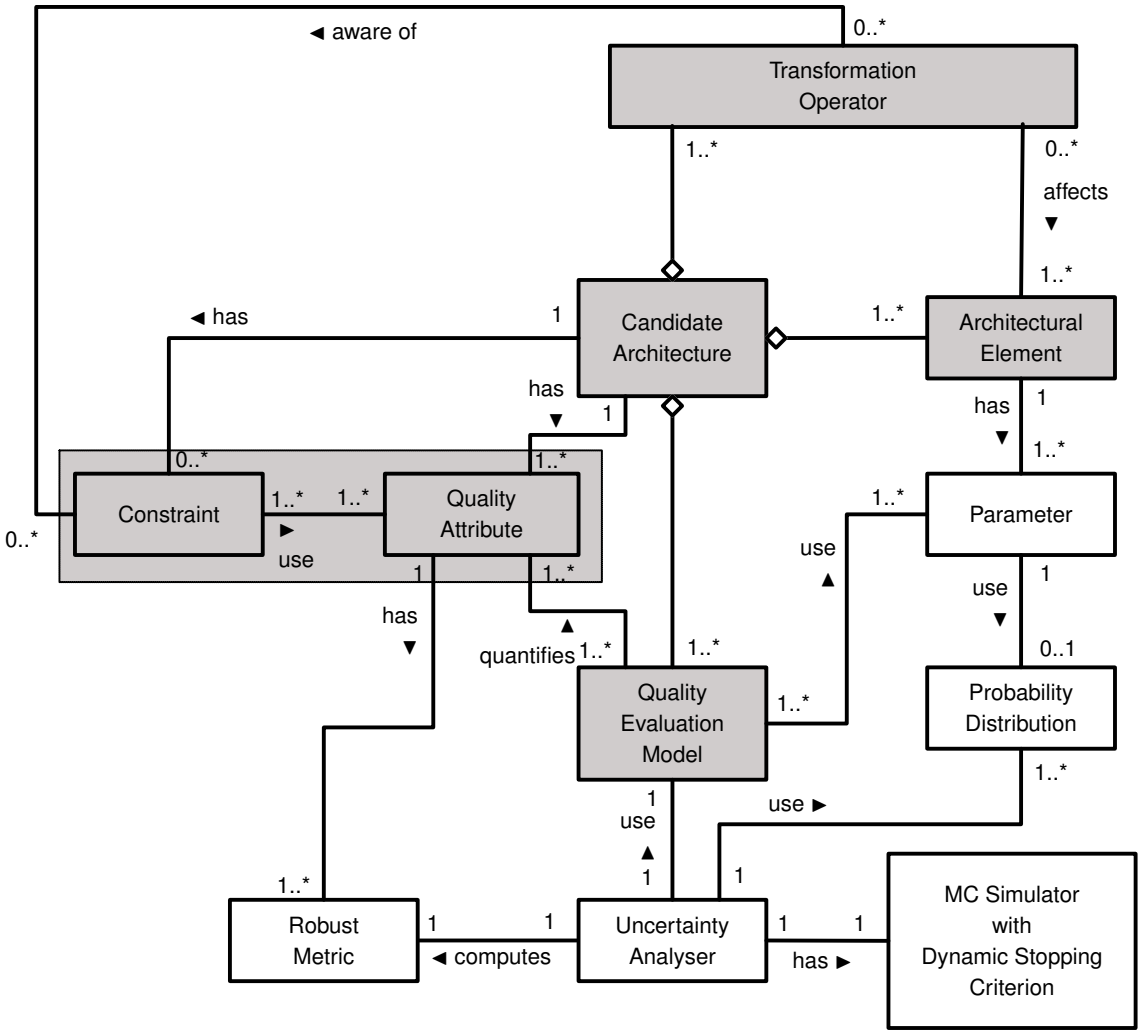


Figure 6.2: High-level model for architecture optimisation under uncertainty. Elements in grey represent the implementation of corresponding elements in the ISO model, approximately in the same layout in Figure 6.2. White elements represents new elements introduced to cater for uncertainty

Candidate Architecture

In this framework, the *Candidate Architecture* is an architecture description to a system in focus. Hence, it reflects the *Architecture Description* conceptual entity in the ISO 42010 model, and refers to the collection of design decisions contained in an alternative design. This model understands that a candidate architecture can be comprised of elements on different levels of abstraction, for example software

components and tasks. Different aspects of an architecture are visible in certain architectural views described in Chapter 2. The appropriate level of abstraction and architecture view have to be selected in modelling a specific problem. For the purpose of representing a candidate architecture in the context of architecture optimisation, it is sufficient to consider only certain aspects of architecture that are part of the optimisation problem. For instance, an architecture can be a deployment, a hardware redundancy allocation or a composite of both.

Architectural Element

An *Architectural Element* in this model is a general abstraction for the items contained in a description of a candidate architecture to the system. These elements can be realised in different architectural views described in Chapter 2. Examples are software components, modules, tasks, interactions, hardware hosts, network links *etc.* The appropriate element types can be selected depending on the architecture design problem at hand. For example, in an embedded software deployment optimisation problem, architectural elements denote software components, inter-component interactions, hardware ECUs and buses which are relevant to the problem in focus.

Parameter

During the presentation of architecture evaluation under uncertainty in the previous chapter, it has been emphasised that some parameters of the architectural elements can be considered as fixed real numbers while some inherit the notion of uncertainty. Complying with this requirement, *Parameter* is a higher level abstraction to represent both certain and uncertain inputs in architecture element specification. In this model, a *parameter* refers to either a fixed number or a probability distribution. The generic specification introduced in Chapter 5 can be used to specify the parameters.

Quality Attribute

A set of *Quality Attributes* is associated with each candidate architecture as design goals to be optimised or constraints to be satisfied, resembling the *system concern* in the ISO model. As it has been discussed in the previous chapters, the evaluation of the quality attributes is often carried out using probabilistic models which are created from the architecture descriptions. Hence, a candidate architecture is associated with a set of models which are used to provide quantitative metrics of the attributes of interest. The responsibility of the tolerance levels and robustness is encapsulated into a separate entity, called *robust metric* in this model. The level of required tolerance of an attribute can be given as a parameter to the quality metric. For example, consider an architecture design problem which requires the optimisation of reliability with higher tolerance against uncertainty. In this case, *reliability* is the quality attribute of interest and the metric associated with the attribute can be set to the 5^{th} percentile of MTTF (mean-time-to-failure). This means that the optimisation goal is a lower bound of MTTF under 95% of the uncertain input parameter variations. Not all quality attributes are necessarily uncertain. The uncertainty is propagated from the parameters of the architectural elements, and therefore only the ones that involve uncertain parameters are required to be assessed against uncertainty. If all parameters that are used in the evaluation of specific attribute are fixed values, the quality attribute is also a distinct value. On the other hand, if at least one of the parameters is uncertain, the quality attribute also becomes an uncertain quantity. The Monte Carlo simulation-based robust objective computation that has been described in Chapter 5 is employed if the objectives are subject to uncertainty. This formulation has made it possible to use quality attributes which inherit uncertainty from input parameters as well as reasonably certain ones, in any mixture in the framework.

Transformation Operator

The *Transformation Operator* entity models the possible actions that an architect can take in order to improve the quality of the architecture, *i.e.* changes to the design decisions which generate new candidate architectures. The robust optimisation formulation understands that the architecture transformations can be performed for individual candidates as well as for combinations of more than one candidate architecture. Certain changes to an architecture (*e.g.* change a deployment of a software component, employ 3-modular voting for a software component) generate a new alternative architecture. It is also common to combine or exchange good design aspects of a group of candidate architectures (*e.g.* swap the deployments of two ECUs in two alternative designs) to generate a new set of alternatives in one step. These common practices of architecture transformation can be implemented as extensions of the generic transformation operators in the model.

Constraints

The robust optimisation framework recognises a set of *Constraints* to be associated with each architecture alternative. The enforcement of constraint satisfaction requires information on individual elements of the architecture (*e.g.* localisation) as well as the architecture as a whole (*e.g.* collocation). Furthermore, a constraint-specific model can be used to check the validity (*e.g.* performance or safety constraints). This model also acknowledges that the constraint satisfaction can be checked at different stages of the architecture optimisation, and sophisticated mechanisms exist in constraint handling. One option would be to ensure constraint satisfaction during the transformation operators while infeasible solutions can also be allowed in the generation. The model supports both the options by realising the relationship with *Transformation Operator* and *Candidate Architecture*.

Quality Evaluation Model

The *Quality Evaluation Model* entity of the framework is a general representation of methods used in architecture-based quality evaluation that reflects a combination *Architecture View* and *Architecture Model* entities in the ISO 42010 model. As already discussed in Chapter 2, a large number of quality models have been developed to quantify different quality attributes from an architecture. These models are certain abstractions of the quality-relevant aspects of a prospective system. From the previous discussion on prevalent architecture-based evaluation and optimisation frameworks, it is evident that some approaches consider the quality attributes are evaluated using mathematical functions while some use specific models. However, both of the methods extract necessary information from the architecture and use it to compute quantitative metrics for the attribute of interest. Therefore, this work understands a high-level relationship among the entities architecture, quality attribute and quality model. The model also adheres to the *generic framework* presented by Grunske *et al.* [189] where quality model in this SCOUT approach's framework is a high level composite of the all 4 elements presented in that work.

Uncertainty Analyser

This element of the framework integrates the uncertainty analysis technique described in the previous chapter. The dynamic stopping Monte Carlo simulation is represented in a separate process element which is used by the *Uncertainty Analyser*. The uncertainty analyser extracts the uncertain parameters of an architecture candidate, and repeatedly generate samples from the probability distributions attached to them. The dynamic stopping criterion leads the MC simulation to produce results with a given level of confidence.

Robust Quality Metric

When the parameters of an architecture are given with the uncertainty associated with them, the quality attributes are no longer point values. Instead of optimising the point values of the quality attributes as in the predominant approaches, this framework allows the optimisation goals to be *robust metrics* which encompass uncertainty. The *Robust Quality Metric* in the framework is the entity introduced to support diverse degrees of tolerance against uncertainty. The required level of tolerance of the quality attribute depends on the quality attribute as well as the domain. This approach allows the architects to define quality goals considering a level of tolerance to the uncertainty. Some examples for the robust values are the following.

- Mean: For some non-critical cases such as response time in simple web application, the mean value of the quality metric (*e.g.* average response time) with respect to the parameter uncertainties is sufficient as the optimisation goal.
- Quartile: Some dependable embedded systems design focuses on quartiles of quality, such as the first quartile of the failure rate distribution. To cater for these requirements, the respective quartile can be given as the robust quality objective.
- Upper/lower bound: The architects can also be interested in pessimistic estimates in some applications, such as in the lower bound of reliability of an automotive ABS system, *w.r.t.* design-time parameter estimates. In such cases, the appropriate bound should be given as the robust quality metric to be optimised.
- Percentile: The confidence value estimate is a crucial requirement in most of the safety and mission critical systems. For example, reliability of the ACC system have to be given with a 95% confidence under the uncertainty.

The *percentile* option in the above listed methods to capture quality attributes with uncertainty is more customisable in comparison to the others. Different levels of tolerance can be represented by changing the value of the percentile. For example

in the context of reliability, moderate tolerance can be achieved by setting the goal to *median* (*i.e.* 50^{th} percentile) while more pessimistic tolerance can be gained by configuring the goal to 5^{th} percentile or first quartile of MTTF. Therefore, different percentiles are used as robust metrics throughout the thesis while it is easily possible to use any other statistical index as well.

6.3 Formal Definition of Robust Architecture Optimisation

Using the underlying model of architecture optimisation-related elements given in the previous sections, the problem of robust architecture optimisation can be formally defined as the following.

- $E = \{e_1, e_2, e_3, ..\}$ represents a finite set of elements composed in an architecture description.
- $D = \{d_1, d_2, d_3, ..., d_m\}$ denotes a finite set of architecture decisions to be made.
- $CA : \mathcal{P}(E) \rightarrow D^m$ to represent the set of all possible candidate architectures that can be generated using the design decisions, where $\mathcal{P}(E)$ denotes the power set of E. Since E and D are both finite sets, CA is also finite. For ease of reference, elements of CA will also be referred as $\{ca_1, ca_2, ca_3, ...\}$ and the same notation is used to refer to elements of any set given using a capital letter.
- The quality attributes $Q : CA \rightarrow \mathbb{R}^k$ represent k different quality goals that have to be considered in the architecture design. The quality attributes also encapsulate the notion of tolerance. In other words, the numerical values of Q denote the robust quality metrics in the framework. The source of uncertainty that comes from E, D is to be considered in the quantification functions $CA \rightarrow \mathbb{R}$.

- Constraints that have to be satisfied in individual architectures are defined as $\Omega : CA \rightarrow \{true, false\}^r$, where r is the number of constraints.

With this notation, the goal of the multi-objective optimisation problem is to

find the approximate set of solutions $CA^* \subseteq CA$ *that represent a trade-off between the conflicting objectives in* Q *which represent robust quality metrics, and satisfy the set of constraints* Ω.

The key role of an optimisation algorithm is the fast and efficient achievement of CA^*. However, as most of the architecture problems are NP-hard, only stochastic algorithms have proven to be scalable for practical-sized problems [84,111,375]. The term *approximate set* is used as the stochastic optimisation does not guarantee the *best* solutions. In a stochastic architecture optimisation algorithm, the process is started with an initial set of architecture solutions CA_0, and the search for better solutions is achieved by a set of transformation operators $T : \mathcal{P}(CA) \rightarrow \mathcal{P}(CA)$. The reason behind the use of power sets in this operator is to include the transformations that generate multiple new architectures using more than one solutions in addition to the transformations on a single solution which generates a single new architecture.

6.4 Integrating Optimisation Algorithms

Having described the high-level model of elements involved in architecture optimisation of software-intensive systems and capturing the effects of parameter uncertainties, this section presents the integration of prominent optimisation algorithms into the framework. From the point of view of an optimisation algorithm, the model presented so far in the chapter is a generic characterisation of the search space, objective space and solution representation. Therefore, different optimisation algorithms can easily be adopted without embedding problem-specific logic into the algorithm. However, the design space exploration in the domain has already been identified as an NP-hard combinatorial optimisation problem, and nature-inspired

optimisation metaheuristics have consistently been successful (summary of architecture optimisation approaches, Table 2.3 given in Chapter 2 gives evidence). This section presents the tailoring of three widespread optimisation algorithms into the robust architecture optimisation framework.

6.4.1 Non-dominated Sorting Genetic Algorithm II

Algorithm Description

The *Genetic Algorithm* (GA) is a guided search technique, which has been inspired by the processes of evolution in nature. In GA, evolution is iteratively carried out on a *population* of candidate solutions. Each individual or population member is referred by a *chromosome* and represented by a string consisting of *allele*s. A chromosome encodes a solution to the problem at hand, and the encoding is referred to *genotype*. An individual solution is also annotated with another set of values representing the goals to be optimised, which is called the *phenotype*. Abstractly speaking, GAs observe the phenotype of the candidates and execute probabilistic transformation rules on the genotype with the use of genetic operators. Three basic genetic operators are used in GAs, namely, *selection*, *crossover* and *mutation*.

Crossover is the process of creating new *offspring* by combining two (or more) parents selected from the population. This is considered as the primary operator in GAs. Chromosomes of the two selected parents are often mixed in different techniques to implement crossovers.

Mutation produces new chromosome by applying random (often small) modification to a selected parent chromosome. This also results a new offspring, mimicking mutation process of biological evolution.

Selection is an important aspect of the all the evolutionary algorithms, where it attempts to ensure *'survival of the fittest'* candidates. When multiple objectives are to be optimised, different selection strategies have been developed such as weighted fitness [155], strength-Pareto [422].

The Non-dominated Sorting Genetic Algorithm (NSGA) is distinct from a simple

genetic algorithm specifically in the implementation of the selection operator. The original proposal of NSGA has been suppressed by the improved and more efficient version, NSGA-II [117].

The NSGA performs a ranking phase before selection, where the population is ranked on the basis of an individual's non-domination. In a maximisation problem a solution p dominates another solution q if $Q(p) \geq Q(q)$ for all objectives, and $Q(p) > Q(q)$ for at least one objective. For minimising objectives, solution p dominates another solution q if $Q(p) \leq Q(q)$ for all objectives, and $Q(p) < Q(q)$ for at least one objective. For both minimisation and maximisation problems, $q \prec p$ denotes the solution p dominates the solution q. If there exists no other feasible solution that dominates a solution p^*, then p^* is said to be non-dominated. The set of all non-dominated solutions is known as the non-dominated set.

The idea behind the non-dominated sorting procedure is a ranking selection method which is used to select good candidates. Since the algorithm is based on non-dominated sorting procedure, the algorithm is known as the *Non-dominated Sorting Genetic Algorithm (NSGA)*.

First, the non-dominated solutions present in the population are identified and assigned a rank of 0. These solutions constitute the first non-dominated front in the population, which is temporarily ignored in processing the rest of the population in the same way, to find the solutions which belong to the second non-dominated front. These solutions are assigned with the next rank value, *i.e.* 1. This process is continued until the entire population is classified into separate fronts. The basic non-dominated sorting process has been significantly improved with the *fast non-dominated sort* in NSGA-II, given in Algorithm 2.

The overall process of the NSGA-II is given in Algorithm 3. A given initial population is first extracted into a set of non-dominated fronts using the `fast-non-dominated-sort` procedure. In cases where a good spread of the solutions in the objective space has to be maintained, sophisticated *crowding-distance assignment* [117] can be applied following the fast non-dominated sorting. NSGS-II

Algorithm 2: Fast non-dominated sorting in NSGA-II

```
1  begin fast-non-dominated-sort(P)
2      foreach p ∈ P do
3          S_p = ∅
4          n_p = 0
5          foreach q ∈ P do
6              if (q ≺ p) then /* if p dominates q                    */
7                  S_p = S_p ∪ q /* Add q to the set of solutions dominated by
                       p                                              */
8              end
9              else if (p ≺ q) then
10                 n_p = n_p + 1 /* Increment the domination counter of p */
11             end
12         end
13         if n_p = 0 then /* p belongs to the first front            */
14             p_rank = 1
15             F_1 = F_1 ∪ p
16         end
17     end
18     i = 1 /* Initialise the front counter                          */
19     while F_i ≠ ∅ do
20         X = ∅ /* Used to store the members of the next front      */
21         foreach p ∈ F_i do
22             foreach q ∈ S_p do
23                 n_q = n_q − 1
24                 if n_q = 0 then /* q belongs to the next front     */
25                     q_rank = i + 1
26                     X = X ∪ q
27                 end
28             end
29         end
30         i = i + 1
31         F_i = X
32     end
33 end
```

presents a *crowded-comparison operator* which guides the NSGA's selection process towards a uniformly spread-out Pareto-optimal front during selection of candidates to the mating pool and selecting a population for the next iteration (Lines 5-10). A mating pool is then created with solutions selected from the population according to the fitness values that has been assigned to them during the ranking process. Solutions with a lower rank have a higher chance of being selected to be part of

the mating pool than the ones with a higher rank. In case the ranks are equal, the comparison uses the crowding distance, where solutions with higher crowding distance are encouraged. *Binary tournament* can be used for this purpose where the better one of two randomly selected candidates (*i.e.* either with a lower rank, or the one with a higher crowding distance if the ranks are equal) will be put into the mating pool. The mating pool will then serve for the random selection of the individuals to reproduce using crossover and mutation (Line 11).

Algorithm 3: NSGA-II main loop

```
1  S* = ∅ /* current non-dominated solutions                          */
2  S = initial-population() /* create or load initial candidates     */
3  while termination criterion not met do
4  |  F = fast-non-dominated-sort(S) /* F = (F_1, F_2, F_3, ...), all non-dominated
   |     fronts of S                                                  */
5  |  P = ∅ and i = 1 while |P| + |F_i| ≤ N do /* until the parent population
   |  is filled                                                       */
6  |  |  P = P ∪ F_i /* include ith non-dominated front in the parent
   |  |     population                                               */
7  |  |  i = i + 1 /* check the next front for inclusion             */
8  |  end
9  |  Sort(F_i, ≺_n) /* sort in descending order using ≺_n           */
10 |  P = P ∪ F_i[1 : (N − |P|)] /* chose the first (N − |P|) elements of F_i */
11 |  P' = make-new-pop(P) /* use selection, crossover and mutation to
   |     create a new population P'                                   */
12 |  S = P ∪ P' /* combine parent and offspring population          */
13 |  S* = update-non-dominated-set(S, S*)
14 end
```

Integration into the Framework

Figure 6.3 presents the integration of NSGA-II concepts into the framework. The conceptual entities in the shaded region denote the high-level model presented in Figure 6.2, and the rest in white background are the newly added entities for NSGA. The population in NSGA terms is mapped to a set of candidate architectures. The solution encoding in NSGA-II is represented by a candidate architecture. The abstract genetic operators are linked to the transformation operators in the architecture

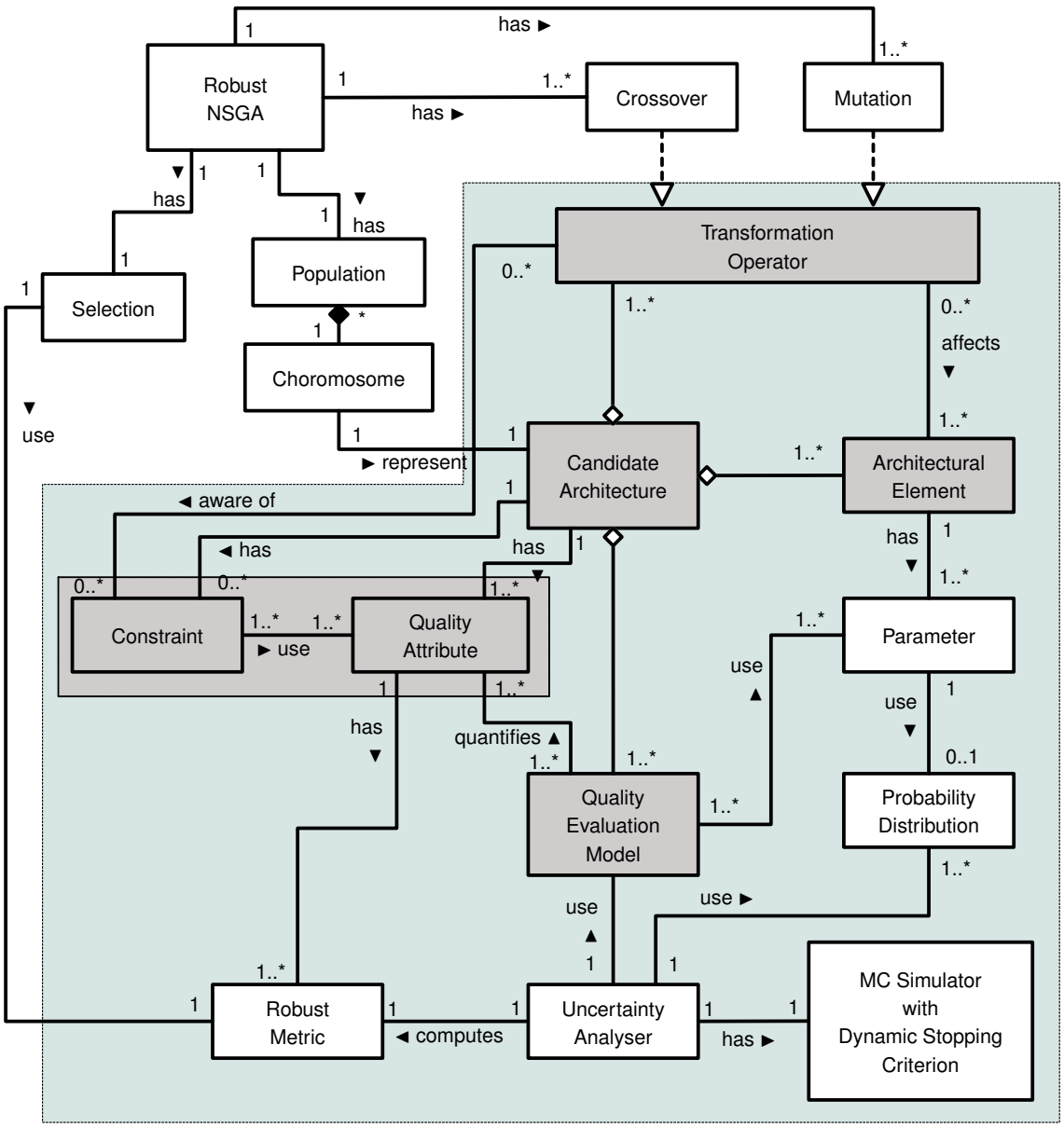


Figure 6.3: Integration of NSGA into the framework

design. The architecture transformation relationship has been defined in many-to-many form, allowing the mapping to any genetic operation in general. For instance, crossover takes two solutions and reproduces two new candidates. This operator can be mapped to swapping deployments of two candidate architectures and producing two new deployment architectures. The SCOUT approach understands the changes

of design decisions as specific to a problem in focus. Different choices in architecture design context can be mapped into the two main operators in NSGA-II.

One important aspect of adapting NSGA-II for robust optimisation is the *selection* operator. As the objective is to find a set of *robust* architecture solutions, NSGA-II's natural selection is exploited to penalise non-robust solutions. This work proposes to consider the *robustness* in non-domination ranking. In the previous chapter, the use of *percentiles* of quality attributes as a flexible measure of tolerance has been introduced. Here, it is proposed to use confidence intervals obtained from Monte Carlo simulation for non-dominated ranking. In maximisation problems, a lower bound will be used while an upper bound is proposed for minimisation problems. The dominance criterion for candidates is defined based on the lower/upper bound. More intuitively, in a maximisation problem a solution p^* is non-dominated if there exists no other p such that $\underline{Q(p)} \geq \underline{Q(p^*)}$ for all objectives, and $\underline{Q(p)} > \underline{Q(p^*)}$ for at least one objective, where $\underline{Q}$ represents the confidence lower bound obtained from the MC runs. The fast-non-dominated sorting given in Algorithm 2 is revised according for robustness-based ranking.

A mating pool is created with the solutions selected from the population according to the rank values which have been assigned to them during the ranking process. Since the dominance criterion is based on the robust quality metrics, it increases the chance of the robust solutions with a higher lower bound to be selected for the next generation, leading the algorithm towards the robust-optimal solutions. The mating pool then serves for the random selection of the individuals in producing new candidate architectures using the crossover and mutation as described before.

6.4.2 Pareto Ant Colony Optimisation

Algorithm Description

Ant Colony Optimisation (ACO) is a stochastic algorithm inspired by the foraging behaviour of real ants which was originally proposed by Dorigo *et al.* [129]. The ACO process mimics the *pheromone* tracks left by the ants during search of food

and the way back to the nest. It belongs to the *constructive* branch of optimisation as the solutions are constructed in steps as opposed to changes made to existing solutions in iterative algorithms. Inheriting the features of constructive algorithms, ACO often converges quickly and produces feasible solutions in highly constrained combinatorial optimisation problems [340].

The central part of an ACO algorithm is the parametrised probabilistic model called *pheromone model* ($\mathcal{T}$). The model consists of a vector of *pheromone trails*, where each trail contains a vector of *pheromone values* (τ). The use of ACO in an optimisation context maps the solutions into the pheromone trails where individual decisions in a solution correspond to pheromone values. As a result, the pheromone model constitutes a parameterised probability distribution over the solution space. In general, ACO approaches attempt to solve an optimisation problem by repetitive execution of two distinctive steps:

- candidate solutions to an optimisation problem at hand are constructed using the pheromone model.
- the generated candidate solutions are used to update the pheromone model in a way that the future sampling of the model guides towards high quality solutions.

Pareto Ant Colony Optimisation (P-ACO) [124] extends these basic ACO concepts to multi-objective optimisation. The main optimisation loop of the P-ACO algorithm is given in Algorithm 4 where individual steps of the algorithm is explained in the following paragraphs.

■ *Initialise Pheromone Model.* In P-ACO, the pheromone model keeps separate maps ($\mathcal{T}^1, \mathcal{T}^2, ..., \mathcal{T}^k$) for individual objectives. A non-zero constant(τ_0) is commonly used as the initial value of all pheromone values.

■ *Construct Solutions.* In ACO, solution construction is conducted by artificial ants. An ant has a limited life span which is often a random number over an interval assigned at ant's creation. In the presence of multiple objectives to optimise in P-ACO, an ant is assigned with random weights ($w^1, w^2, .., w^k$) for each objec-

Algorithm 4: Pareto ant colony optimisation - main loop

```
1  S* = ∅ /* current non-dominated solutions                          */
2  initialize-pheromone-model(T)
3  while termination criterion not met do
4  |  S = ∅ /* solutions produced at the current iteration           */
5  |  for i = 1..n do /* n ants per iteration                        */
6  |  |  s_i = construct-solution() /* the ant constructs a path     */
7  |  |  if s_i is valid then
8  |  |  |  local-search(s)
9  |  |  |  S = S ∪ s_i
10 |  |  |  update-non-dominated-set(s_i, S*)
11 |  |  end
12 |  end
13 |  apply-pheromone-update(T, S)
14 end
```

tive. These weights represent the relative importance of the different objectives to a particular ant. A trail over the pheromone model resembles a path an ant would travel using the pheromone. The construction process starts with an empty set and each step in the trail construct a partial solution. The individual steps of the trial is taken as probabilistic choices made by the ant. The choice is based on the pheromone values in the pheromone model and the ant's weights on different objectives. For an atomic decision step(x), the observable pheromone level for an ant is given by $\tau_x = \sum_{i=1}^{k} w^i \times \tau_x^i$ where k is the number of optimisation objectives. The ant's choices are made according to a *pseudo-random proportional rule* where a random number $q \in [0, 1]$ is checked against a given threshold q_{th}. If $q \leq q_{th}$, the choice with maximum pheromone(τ_x) is taken (greedy selection) and pheromone-proportional choice is made otherwise. In the latter case, the probability that an ant takes the path choice x can be given by $P(x) = \frac{\tau_x}{\sum_{c \in C} \tau_x}$ where C denotes all possible choices available for the specific step.

The construct-solution step of ACO also contains an important diversity preserving technique called *pheromone evaporation*. Once a decision is taken, this mechanism systematically reduce the level of pheromone in that element in order to avoid all the future ants takes the same decision over and over. A *local pheromone*

update rule is introduced to carry out the pheromone evaporation. Once an ant makes a decision on his trail, all the pheromone values corresponding to the decision will be updated as

$$\tau_x^i = (1 - \rho) \cdot \tau_x^i + \rho \cdot \tau_0$$

where ρ is the pheromone evaporation rate. When an ant completes a path using the pheromone model, the steps that were taken in the pheromone trail constitutes candidate solution to the optimisation problem in focus.

■ *Local Search.* The solutions produced by ants are first checked for validity, *i.e.* the satisfaction of certain constraints that are applicable to the problem. A local search procedure may also be applied to the valid solutions in order to improve the quality. The local search techniques involve certain heuristics or random moves which can be specifically tailored to the context of the optimisation problem. Once a solution is finalised, it is checked against the current non-dominated set. If the newly created solution is not dominated by any solution in the Pareto-optimal set (S^*), then the new solution is added to (S^*) and all the existing solutions which are dominated by s are removed from (S^*).

■ *Global Pheromone Update.* The role of directing the automated search towards better solutions in an ACO is largely taken by the *pheromone update* activity. Two pheromone update steps are involved in ACO, where the local pheromone update is carried out during the solution construction. The second one is called *global pheromone update* which is applied on the complete pheromone model. In P-ACO implementation given in Algorithm 4, global pheromone update is conducted after each iteration, *i.e.* sending n ants. The best and second best ants for each objective are selected after each iteration. Individual data structures of the pheromone model corresponding to the each objective k are updated according to the rule:

$$\tau_j^k = (1 - \rho) \cdot \tau_j^k + \rho \cdot \Delta\tau_j$$

where $\Delta\tau_j$ is a quantity indicating the quality of the solution. For example, $\Delta\tau_j$

can be set to 10 for the best ($\Delta\tau_b$) in objective k, 5 for the second best ($\Delta\tau_{sb}$) and 0 for all other solutions. By doing that, the pheromone values corresponding to the best and second best of each objective are updated with higher pheromone values in comparison to the others. This global update rule adds more bias to these decisions in the probabilistic choices made by the future, guiding towards better solutions over iterations.

Integration into the Framework

As a robust architecture optimisation strategy, the P-ACO algorithm can easily be integrated to the high-level model presented earlier in the chapter. In this context, the ant's walk on the pheromone model can be mapped to iterative improvement to the software architecture. Each step taken by an ant resembles individual atomic decisions taken in the architecture design. Hence, the optimisation problem is to find a (near-)optimal set of candidate architectures that are able to tolerate uncertainty in quality evaluation model parameters. The proposed mapping of ACO in to the framework is given in Figure 6.4.

The solution construction and local search in the ACO context are conceptually mapped into the transformation operators in the model. Hence, the solution construction represents the process of making a complete collection of required decisions (*e.g.* deploying all software components in a deployment optimisation problem) and the local search represents iterative changes to these architectural decisions with the application of transformation operators. An important merge of concepts exists in the mapping of global pheromone update to the framework. As the goal is to find the architecture candidates that are better in quality as well as robust *w.r.t.* parameter uncertainties, the bias of architecture decisions should be on robust and better solutions. Therefore, similar to the adaptation of selection operator in NSGA-II, robustness is considered in assessing the quality of the solutions produced by the ants. The same robust quality attribute based non-domination criterion used in NSGA-II is used, and the selection of the best and second best ants of the iteration

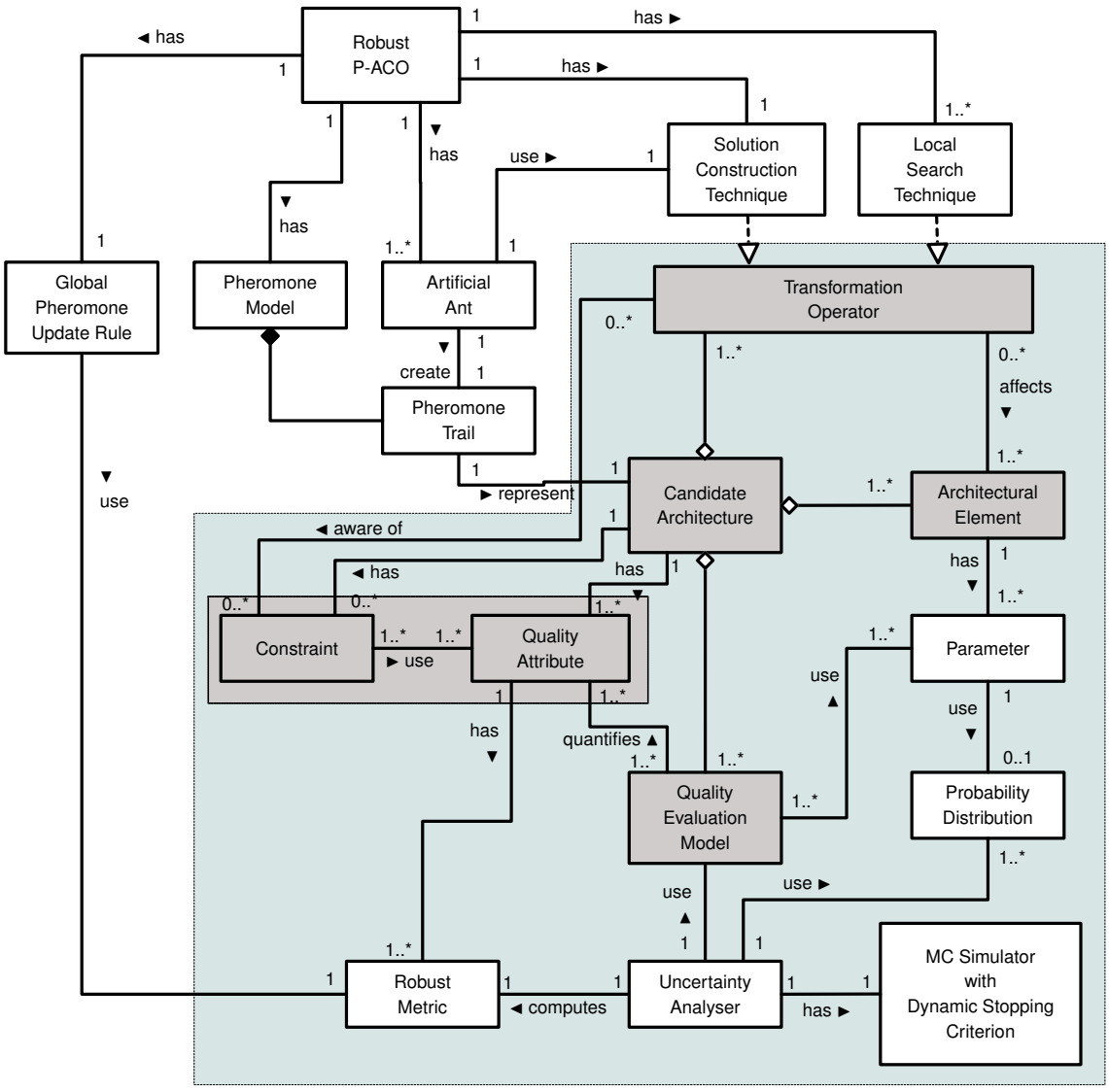


Figure 6.4: Integration of Pareto-ACO into the framework

is carried out using the robust quality attributes. By doing that, the pheromone model is updated with higher values for robust and better candidates in comparison to sensitive or weak solution. Hence, the ants movement can be iteratively guided towards finding robust optimal candidate architectures.

6.4.3 Simulated Annealing

Algorithm Description

Simulated Annealing (SA) optimisation algorithm is inspired by molecular behaviour encountered in the steel hardening process. SA is widely-used as a stochastic solver for combinatorial optimisation problems [340]. The algorithm was originally used as strategy escaping from a local optima (local minima or maxima), and later extensions have pushed its boundary with multi-objective simulated annealing strategies that use a population of potentially efficient solutions rather than single current solution. A non-domination-based multi-objective variant called *Pareto Simulated Annealing* (PSA) is given in Algorithm 5 where specific steps are described in the following paragraphs.

Algorithm 5: Pareto simulated annealing main loop

```
1  S* = ∅ /* current non-dominated solutions                                  */
2  S = burn-in() /* create initial set of solutions                          */
3  T = T_0 /* initial temperature                                            */
4  β = −1 · ^{L*α}√(0.00001 · T_0) /* epoch size L ∈ ℕ+, active period α ∈ [0,1] */
5  while termination criterion not met do
6      for i = 1..L do /* L number of epochs for temperature level          */
7          foreach s ∈ S do
8              s' = neighbourhood-move(s) /* construct a random feasible
                   neighbor solution                                         */
9              update-non-dominated-set(s', S*) /* check and update the
                   current-best solutions                                    */
10             s = accept-criterion-check(s', s, S*, T)/* accept s' using a
                   probablistic acceptance rule                              */
11         end
12     end
13     T = β · T /* compute new temperature with cooling factor β            */
14 end
```

■ *Burn In.* The first step of simulation annealing algorithm is to create a set of initial solutions to a problem at hand. A candidate solution is often represented as a vector of atomic decision variables, similar to the chromosome in GA. The burn-in process initially creates one solution and iteratively applies changes to the solution in

generating a new set of feasible solutions. After the burn in, the algorithm computes the initial temperature (T_0) (*e.g.* T_0 = average normalised quality of all solutions / $ln(2)$) and cooling factor (β) from given parameters. Another parameter involved in the calculation is the active period (α) which indicates the portion of run that should allow for non-hill climbing (exploration) moves.

■ *Neighbourhood Move.* Simulated annealing applies random changes to an existing solution in order to generate new solutions, mimicking the molecular movement in steel annealing. The implementation of neighbourhood moves depends on the solution representation and problem in focus, similar to mutation operator in GAs.

■ *Probabilistic Acceptance Rule.* The contribution in terms of guidance for better solutions over the annealing cycles is provided by the probabilistic decision rule. Algorithm 6 lists the steps of a variant of multi-objective simulated annealing acceptance rule. A newly created (randomly modified) solution is compared with its original form to compute the fitness difference (δ). This comparison is based on the non-domination in the multi-objective domain. The normalised fitness difference is computed using the relative non-domination of the two solutions and a probabilistic decision is taken on whether to keep the original solution or to replace it with the new solution. This probabilistic decision makes a bias towards keeping better (in multi-objective space) solutions in the next annealing cycles. The strength of the decision step (genotypical distance) is also influenced by the current temperature, resembling large molecular movements in higher temperatures.

Integration into the Framework

Similar to the other two optimisation algorithms, Pareto simulated annealing can be naturally integrated into the model introduced for robust architecture optimisation. Figure 6.5 illustrates a conceptual mapping of SA entities into the high-level elements in the model devised earlier in the chapter.

The notion of a *solution* in the SA terms are mapped to the candidate architecture in the framework. Hence, a solution encoding in the SA would represent the

Algorithm 6: Pareto simulated annealing - acceptance rule

```
1  begin accept-criterion-check(s', s, S*, T)
2      n_s = count solutions in S* that dominate s
3      n'_s = count solutions in S* that dominate s'
4      δ = |n'_s − n_s| / sizeof(S*)  /* calculate the fitness difference   */
5      r = random(0, 1)  /* generate a random number between 0 and 1       */
6      if r < e^(−δ/T) then
7          return s'  /* replace s with s' in the solution                 */
8      end
9      else
10         return s  /* keep the previous solution                         */
11     end
12 end
```

collection of architectural decisions enclosed in candidate architecture. The neighbourhood moves and burn in activities can be mapped to architecture transformation operators in the robust architecture optimisation context. Therefore, iterative molecular movements are realised by possible alterations in the architectural decisions which manifests a guided search through the architectural design space. An important mapping has to be carried out in order to make a bias towards robust and Pareto-optimal solutions. In that aspect, the accept-criterion-check rule can be adjusted by embedding the uncertainty into the decision. The aim here is to encourage moves towards architectural decisions that are better and robust while penalising weak or sensitive candidates. Hence, robust quality metrics can be used to make non-domination comparison in accept-criterion-check. Intuitively, in a maximisation problem a solution s^* is non-dominated if there exists no other s such that $\underline{Q(s)} \geq \underline{Q(s^*)}$ for all objectives, and $\underline{Q(s)} > \underline{Q(s^*)}$ for at least one objective, where $\underline{Q}$ represents the confidence lower bound obtained from the MC runs. This rule is applied in selecting the non-dominated solutions as well as in computing fitness difference between s and s'.

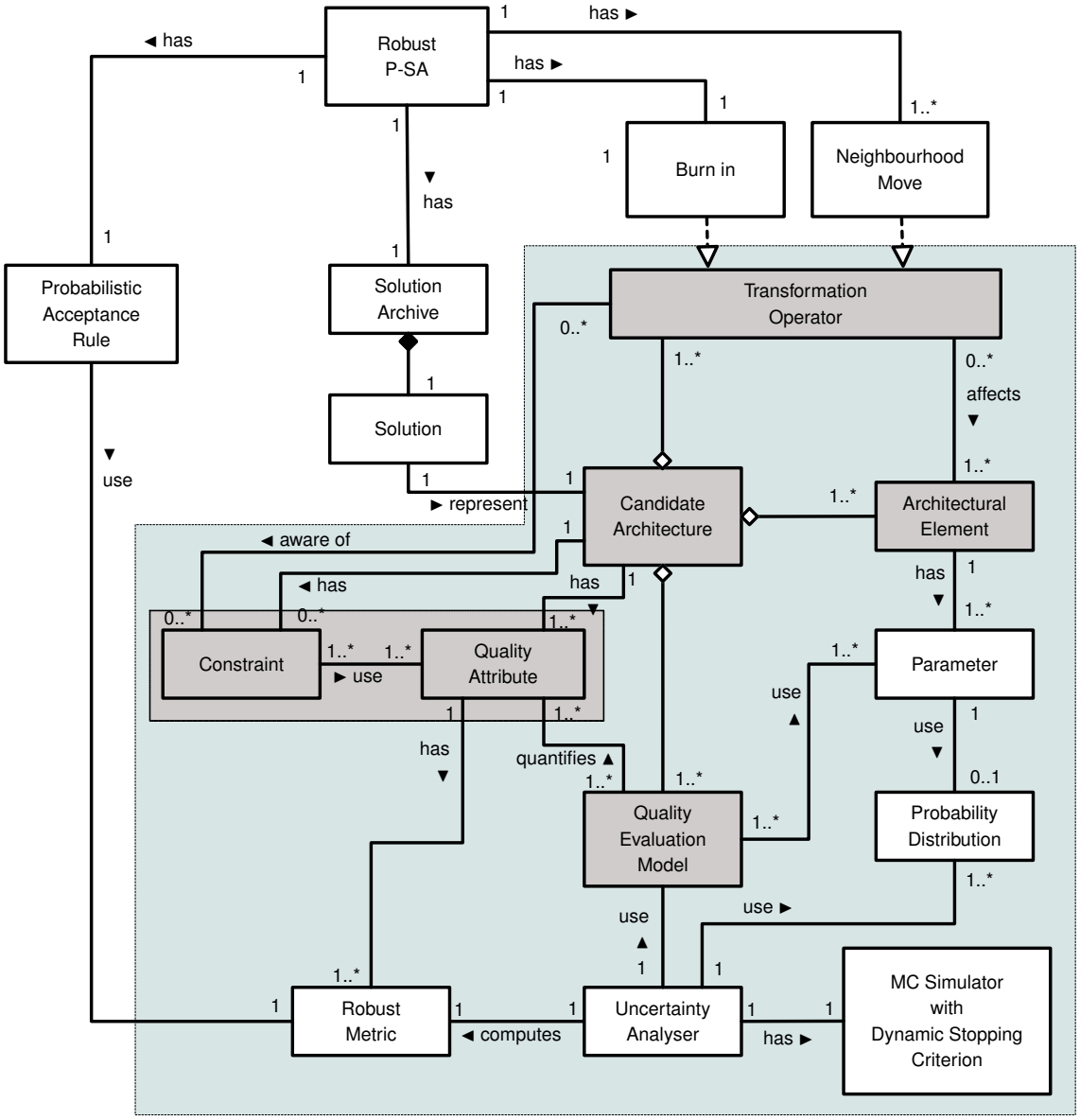


Figure 6.5: Integration of Pareto-simulated annealing into the framework

6.5 Comparison to Related Work

This section presents a discussion of the new framework's contribution in the context of related work to addressing respective refined research questions (under RQ 2) stated in Chapter 4. To the validity of the approach, evidence is provided by bringing the SCOUT model into the context of prominent approaches that address specific aspects in the domain of software engineering. The framework's ability to

cater for the requirements in those modelling approaches is discussed in order to pursue the uncertainty integration into the SCOUT approach as a generic extension to the current methods. The discussion is organised into two areas. First, the high-level model devised for the robust optimisation is examined in the context of existing work. Chapter 2 contains a detailed categorisation of architecture optimisation approaches. Hence, the focus in this section is only on showing that the SCOUT framework integrates the notion of uncertainty without violating or contradicting the current complementary models. The inspiration and influence which this work has obtained from other engineering disciplines that apply robust optimisation techniques for their problems is discussed in the latter part of the section.

6.5.1 High-level Modelling Framework

A number of architecture optimisation techniques have been introduced, which are based on quantitative evaluation of specific probabilistic quality attributes, such as reliability, availability, safety and performance. Apart from these quality attributes, the existing approaches can also be distinguished by the problem they aim to solve. Examples are component deployment (allocation of software components to hardware resources) [283], redundancy allocation [97, 242], topology selection [304], component selection [104] and scheduling [114, 209] problems. However, these architecture optimisation approaches often have a specific focus on an architecture optimisation problem, where as SCOUT's goal is to integrate uncertainty-related aspects into a broader setting described in the research questions. Therefore, the high-level model has been leveraged first for the robust optimisation, which captures the diverse aspects of architecture optimisation problems. Compliance of the presented modelling framework to related meta-approaches is discussed in the following paragraphs, acknowledging that those approaches are based on point-estimated input parameters. The approaches have been selected on the basis that they address the related aspects to the problem in more generic sense, rather than a specific architecture optimisation or evaluation problem.

- Malek *et al.* [272,283] formulated a generic optimisation framework for deployment architecture design of component-based systems. In this approach, software components, interconnections, hardware hosts, networks and services are used as modelling entities. A complete allocation of components to hosts, and interconnections to networks constitutes a deployment architecture. Availability, latency, communication security, energy consumption and memory usage have been used as quality attributes of interest. The approach models quality attributes as mathematical functions defined from annotations to the architectural entities to the real number space. In the viewpoint of the high-level model presented in this chapter, Deployment is a type of *transformation operator*, and the use of mathematical functions to evaluate quality is a special case of *quality evaluation model*. Hence, the model in SCOUT approach complies with and is able to cater for the requirements in Malek *et al.* approach.

- Nicholson [304] introduced an approach for safety-critical embedded systems design which consists of two phases. In the first phase, an optimal architecture topology is designed with the appropriate level of redundancies for hardware elements. Dependability and performance metrics are used as constraints while cost and topology size are optimised. The second phase of the approach is described as a mapping of software components to real-time tasks. The quality evaluations in both phases are based on mathematical functions which use parameters annotated to the architectural elements. Two abstraction levels are used in the two phases, and different algorithms (Genetic Algorithm and Simulated Annealing) have been adopted to search through the design spaces. The high-level model presented in this chapter takes inspiration from and supports the architecture optimisation problem addressed by Nicholson. The *transformation operator* entity in this model captures the topology and mapping changes to find optimal architectures. The use of mathematical functions for quality evaluation is inherently supported by the *quality evaluation model* entity of SCOUT's model.

- Blickle *et al.* [61] described a system-level synthesis approach that addresses three important architectural decisions, *i.e.* allocation of processing hardware resources, mapping of tasks to hardware and scheduling. The approach accommodates both hardware-and software-related aspects of the architectural decisions. A generic model of universal dependence graphs has been presented, which is then used to model hardware and software specifications. The approach admits multiple objectives which are defined as functions from complete architecture solutions to numerical values. A multi-objective genetic algorithm is used to traverse through the design space that consists of alterations in the three architectural decisions mentioned before. The model devised in this SCOUT approach can cater for the contributions presented by Blickle *et al.* as the three design decisions can be viewed as different *transformation operators*. The multiple objectives in the approach is captured by the *quality attribute* entity in the new high-level model.

- Diaz-Pace *et al.* [30, 121] formulated a novel reasoning framework and design assistant called *ArchE*. The framework models software architecture, architectural changes (called *tactics*) and architecture based quality evaluation. It also accommodates the use of third party quality models. ArchE integrates the concept of modifications in different views of the architecture (please refer to Chapter 2 for details), and use a complete architecture with annotated parameters for quality evaluation. In this work, the quality attributes are evaluated using simple additive functions of parameters annotated to individual elements. The *tactics* in ArchE corresponds to the *transformation operator* entity of the SCOUT's high-level model. Similar to previous descriptions, simple additive functions for quality evaluation are inherently supported by *quality evaluation model* of SCOUT's framework.

- Papadopoulos *et al.* [317] introduced a semi-automated methodology which can be used in embedded systems architecture design. A component selection phase has been presented which selects software components to satisfy

functional requirements while optimising cost-profit trade-offs using a genetic algorithm. Secondly, models are created for the architecture-based evaluation of safety and reliability for the candidate architectures. A redundancy allocation phase is then executed to find the best possible cost-reliability trade-offs with a genetic algorithm. In this approach, two abstraction levels of the architecture are modified in the two optimisation phases. In the first phase, the functional breakdown within components is considered while in the second phase, components are treated as high level entities. Parameters of individual elements in the architecture are used to derive the parameters for the Fault-Trees and FMEA based quality evaluation. The model presented in this work supports the Papadopoulos *et al.* approach by understanding the two phases as two *transformation operators*. The *architecture candidate* can be used to capture aspects in different abstraction levels. Hence, component selection becomes one transformation operator and secondly, the redundancy allocation. The objectives of optimisation in the two phases comply with the *quality attribute* entity and *quality evaluation model* instances can be created for Fault tree and FMEA models. Hence, the high-level model is inspired by and is compatible with the approach presented by Papadopoulos *et al.*.

- Fredriksson *et al.* [159] have presented an optimisation methodology for the allocation of components to real-time tasks minimising resource usage while satisfying real-time properties. Worst-case execution times (WCET) are computed using the properties of tasks and components. The authors presented a component-based development framework called *SAVEComp* [160], which supports designing safety-critical embedded systems. In SAVEComp, components and interfaces are defined in a restrictive manner enabling their composition to satisfy functional and non-functional goals. The approach considers models at both design-time and run-time, and integrates model-based quality evaluation and quality verification. In comparison, the SCOUT approach lies at a higher level of modelling, and components and tasks can be seen as reali-

sations of *architectural element*s. WCET and other non-functional goals are represented by the *quality attribute* abstraction. Hence, the model presented in this chapter has been able to cater for optimising the SAVEComp models.

- Pimentel *et al.* [325] has introduced a framework called *SESAME* for the exploration of the embedded system's architecture design space. The framework supports architecture design on multiple abstraction levels. SESAME allows for the use of both analytical and simulation methods for the architecture-based quality evaluation, with a specific focus on performance attributes. The optimisation problem has been modelled as a mapping of components, and connections to a platform model. Design space exploration on multiple abstraction levels is carried out by employing third-party optimisation engines. In comparison with *SESAME*, the presented approach shares the same goals in terms of different architecture decisions. The presented high-level model supports different quality goals and interface to optimisation algorithms. In addition to the support for third party optimisation algorithms, the framework given in this chapter also has tailored three stochastic algorithms into the high-level framework.

- In his PhD thesis, Bondarev [64] formulates an abstract model for architecture optimisation of embedded real-time systems. He defines high-level entities for architecture, quality attribute, quality attribute analyser *etc.* and formulates relationships among them. The abstract model is then adopted for the description of his novel performance optimisation framework. However, due to the different focus in the work, the abstract model has not explicitly included many important relationships in entities such as architecture, probabilistic quality evaluation models, input parameters and design decisions which require consideration in robust architecture optimisation with stochastic algorithms. Therefore, sharing similar goals with Bondarev, this work has chosen the latest international standard as the starting point of the model, and leveraged architecture optimisation-related entities followed by integration of concepts

related to uncertainty analysis.

- Grunske [189] presented a generic framework for early quality prediction in designing component-based embedded systems. The framework is formulated based on a comprehensive study on architecture based quality evaluation methods, and abstracts the common aspects between different approaches. The framework consists of four elements, *i.e.* 1) Encapsulated Evaluation Models, 2) Operational /usage profile, 3) Composition Algorithms and 4) Evaluation Algorithms. The combination of these four elements enables the construction of a complete probabilistic model of a quality attribute of interest, which is then used for the quantitative evaluation of the attribute. The *quality evaluation model* in the framework presented in this chapter can be regarded as a high level composite of the all four elements presented by Grunske [189].

- Koziolek *et al.* [235] proposed to integrate design space exploration into component-based software engineering. The original work focused on optimising software architectures modelled with *Palladio Component Model (PCM)* [44, 45] which provides modelling and evaluation support for software components, component assembly, allocation of components to resource platform as well as usage. In more recent work, Koziolek *et al.* [236] conceptualised the architecture optimisation on a higher level, and formulated a generic framework for architecture optimisation of component-based systems (CBS). In this meta-model, possible alterations to the CBS architectures are identified as *degrees of freedom*. A flexible and extensible formulation was presented which can be used for the design space exploration of any CBS model, for any number of quality properties and an arbitrary number of degrees of freedom. Sharing similar objectives in terms of architecture optimisation, the SCOUT approach is aimed at supporting multiple design decisions and multiple quality attributes together with uncertainty involved in architecture optimisation. Hence, the model of this work relaxes the dependency on any component model, and formulates a model of conceptual entities involved in

architecture optimisation of software-intensive systems using the collective knowledge extracted from numerous related work. Therefore, the SCOUT's high-level model devises framework for robust optimisation while taking the inspiration from PCM.

The high-level model presented earlier in this chapter complies with all the approaches discussed before. Specific aspects addressed in each of these approaches can be seen as instances of *architecture candidate*, *transformation operator*, *quality attribute*, *quality evaluation model* and *constraint* entities. In addition, different approaches use different optimisation techniques. However, none of these approaches consider the uncertainty in the input parameters. They are based on the assumption that the input parameters of the architecture optimisation problem are given as point estimates or are not uncertain. Hence, while being able to capture the diverse aspects covered in the above meta-approaches, this approach is able to cater for uncertainties in the input parameters. In contrast to the existing modelling frameworks, this work has integrated the uncertainty-related entities into a high-level model derived from the essence of the existing approaches. Furthermore, the compatibility with the latest ISO standard has been ensured, and used it as the foundation of the model.

Another orthogonal classification criterion of architecture optimisation is the optimisation strategy which ranges from exact methods like linear programming [51, 264] to heuristic algorithms and metaheuristics like evolutionary algorithms [304], tabu search [242] and ant colony optimisation [259]. A summary has been presented in Table 2.3. However, architecture optimisation problems are NP-hard [110] and the appropriate use of transformation operators and optimisation techniques have been highlighted as the key challenges, especially when multiple objectives are considered. Considering the performance and effectiveness of the different algorithms in architecture optimisation problems, this work has adapted several stochastic algorithms to allow for robust optimisation. The specific components that allow for the robust optimisation as described in Section 6.2 are in a high-level, and therefore,

applicable across many problems and application domains.

6.5.2 Optimisation under Uncertainty

Robust optimisation and optimisation with uncertain parameters has been an active research area in the last years, especially in engineering domains such as mechanical engineering [132] and antenna design [127]. The term *robust optimisation* refers to the process of obtaining optimal designs taking the factors of uncertainty into account [55]. Good surveys can be found in Beyer *et al.* [55], Ben-Tal *et al.* [46]. The *robustness* can be categorised with respect to the following aspects of uncertainty [55],

- Type A : changing environment and operating conditions
- Type B : realisation of design parameters
- Type C : uncertainties of system output (correctness of the models)
- Type D : feasibility uncertainties (imprecision of the design goals)

In addition to the application of conventional optimisation concepts, robust optimisation considers at least one of the aforementioned notions of uncertainty in the optimisation process. Apart from the work presented in this thesis that deals with Type A uncertainties, only a few approaches can be found in the software engineering research literature that consider parameter uncertainties into the optimisation.

Coit *et al.* [94] presented an optimisation approach for the redundancy allocation problem with uncertainty, using analytical means to deal with the estimates. In this approach, the mean and variance of reliability estimates of components are analytically propagated to the mean and variance of the of system reliability. A genetic algorithm is then used to optimise the two objectives, *i.e.* the mean and variance of system reliability. Wattanapongskorn *et al.* [405] have extended the scope of the redundancy allocation to both hardware and software, but they also take an analytical approach. Instead of an analytic approach, the SCOUT approach follows the direction of Marseguerra *et al.* [274] using Monte Carlo simulations to achieve better scalability [179]. Furthermore, the above-mentioned approaches consider the

mean and variance of one quality attribute as two objectives, thus they increase the dimensionality of the problem. This increased dimensionality makes it harder to interpret the results and select a design alternative.

In this context, the SCOUT approach has integrated uncertainty to be considered with multiple optimisation goals. In contrast to optimising the mean and variance of the same objective as conflicting goals, the SCOUT approach proposes to use flexible percentiles of quality attributes as objectives, which overcomes the dimensionality issue by keeping the number of objectives low while incorporating the support for diverse levels of tolerance. SCOUT accommodates diverse sources and characteristics of uncertainty in input parameters without limiting to assumptions on distributions. Complementary to the robust optimisation approaches which assume optimisation goals as linear mathematical functions of input parameters, the SCOUT approach provides the support for mathematically complex probabilistic models with possibly multiple conflicting goals. Rather than using a problem-specific algorithm to solve a specific architecture optimisation problem, the SCOUT approach has taken a generic modelling approach which can accommodate suitable stochastic algorithm for robust optimisation. In response to refined research question RQ 2.3, three prominent optimisation algorithms have been successfully adopted to the framework.

6.6 Summary and Conclusions

This chapter presents the optimisation aspect of the SCOUT approach that develops a framework for robust architecture optimisation with respect to uncertainties in the input parameters of the reliability evaluation models. In order to pursue an architecture optimisation approach which can perform under uncertainty, the SCOUT approach first formulates a high-level model for the architecture optimisation and entities related to parameter uncertainties. The model is devised based on a common understanding extracted from prevalent architecture optimisation approaches. The new contributions of the previous chapter on specification, evaluation and automated quantification of uncertainty related aspects are integrated into the high-level

model. Stochastic optimisation algorithms have shown profound success in NP-hard and combinatorial design space exploration problems in software engineering. The integration of three widespread stochastic optimisation algorithms into the devised model is presented, leading to a high-level framework for architecture optimisation under uncertainty.

Evidence can be collected from related literature towards the validity of the framework and the integration of optimisation algorithms. A derivative modelling approach is taken with a focus on justification in each step. The abstract model presented in ISO 42010 international standard is used as the starting point in systematically devising a high-level model to capture architecture optimisation related aspects. The validity of the model is also justified by bringing into the context of prominent modelling approaches that address specific aspects in the domain. The new model's ability to cater for the requirements in those modelling frameworks is discussed in order to pursue the uncertainty integration in the approach as a generic extension to the current methods. Finally, the applicability of the framework to a broader setting and its contribution in addressing the research questions listed earlier is justified with a detailed comparison with prevailing approaches. The following chapter continues the presentation of the overall validity approach including the implementation of the approach, application on the automotive case study and extended experiments on a series of problem instances in different architecture optimisation problems.

Chapter 7

Experimental Evaluation

Having described the theoretical concepts that contribute to the robust optimisation approach in the previous chapters, this chapter presents the implementation and validation of the overall SCOUT approach. The chapter is organised into three sections. The first part of the chapter briefly introduces the implementation of the approach, followed by an application of the robust optimisation method to the automotive ABS/ACC deployment architecture optimisation problem. Results produced for the case study by the new approach are discussed illustrating the extended decision support. In the third section, we conduct a series of experiments investigating various aspects of the overall research question.

7.1 ArcheOpterix Framework

Design and development of today's embedded systems is a complex and time-consuming task. This requires powerful computer aided tools and automated decision support. Industry standard tools such as Rational Rose, ARTiSAN and LinuxLink are already used in practice to assist the architects. However, in the context where the task is to find architecture alternatives which rank highly on the scale of conflicting quality goals, the design space exploration adds further complexity to the problem in focus. As it has already been presented in Chapter 2, the architecture

optimisation of embedded systems has many dimensions to consider while being a complex problem. The candidate designs have to justify the requirements applicable the software architecture as well as the hardware topology. Different types of problems have to be solved such as the deployment of software components to a hardware platform, allocating redundancies for software and hardware elements, selecting components from a repository of different alternatives. Many of these problems have proven to be NP-hard problems, making the automated design space exploration a crucial requirement. Being motivated with the results obtained from the systematic literature review on architecture optimisation methods, we have developed a tool and a framework called ArcheOpterix which aims to help software architects with this difficult task. This section presents the framework and the implementation of ArcheOpterix.

7.1.1 Technical Details

The ArcheOpterix tool provides a generic platform which can be used to specify, evaluate and optimise the architecture of component-based software systems. The framework is based on the high-level model introduced in Chapter 6. The architecture of the framework is presented in Figure 7.1 which has a modular structure, with an entry module responsible for interpreting and extracting a system description from a specification, recognising standard elements like components, services, processors, buses, *etc.*, specified in ADL like AADL or XML. It is possible to plug in different quality function evaluators, constraint validators and optimisation algorithms, making the tool a flexible and customisable for multi-criteria quality optimisation of embedded systems. The general features of the tool are listed below followed by descriptions of its main elements.

The ArcheOpterix tool is implemented as a Java library. It consists of a command line functionality to support batch execution and script-based invocations. Eclipse [382] wrapper of the tool provides a graphical user interface, and can be directly used as a plug-in for the Open Source AADL Tool Environment OSATE [146].

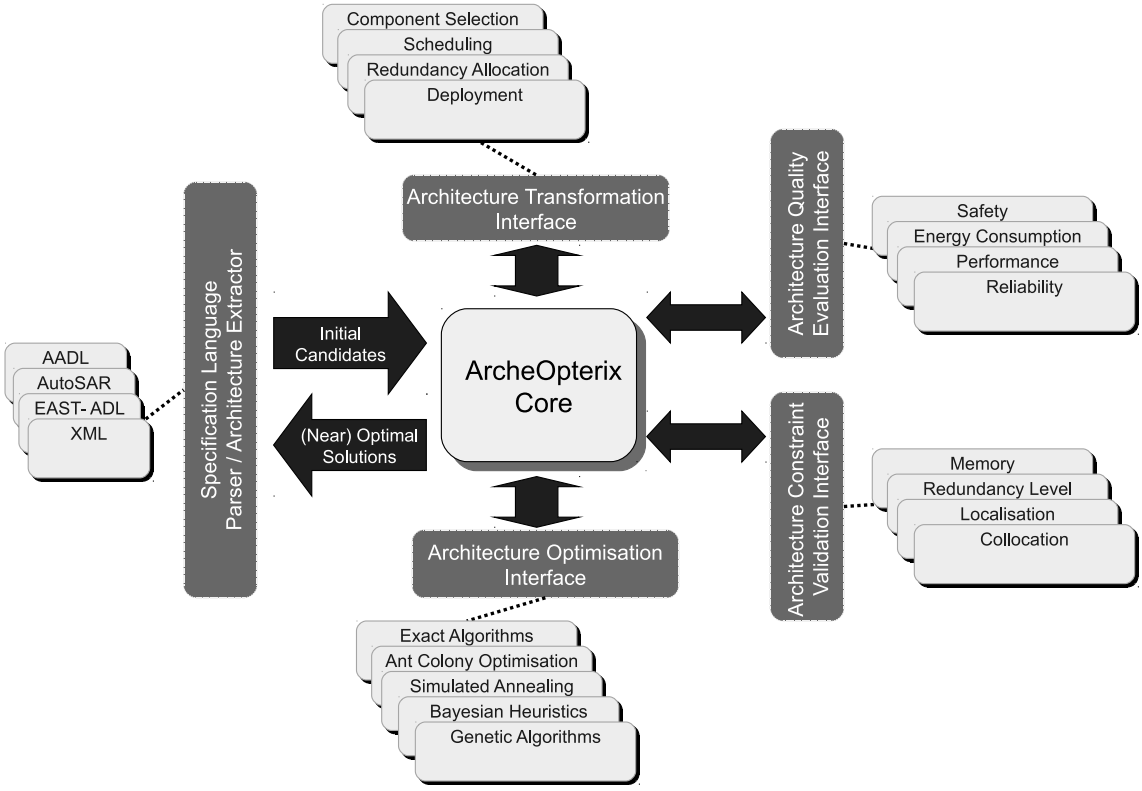


Figure 7.1: ArcheOpterix overview diagram

Specification Parser

Architecture optimisation of component-based software systems originates in a context of defined elements of interest in a candidate architecture and a possible set of changes which can be applied to them. Hence, the input for an architecture optimisation is the specification of candidate elements of the architecture. This includes entities such as software components, electronic control units, bus systems and component interactions, along with parameters annotated to these entities. In practice, this input information is often given using an architecture description language like AADL. To provide the flexibility to use different input ADLs, the ArcheOpterix framework defines an interface where a ADL-specific parser can be integrated. The interface is abstractly defined and different specification languages can be plugged in by implementing the corresponding parser to extract architectural elements and element parameters according to the high-level model presented in Figure 6.2. The

tool currently contains built-in parsers for AADL, XML and textural specifications.

One important extension in this parser is that it integrates the specification of uncertain parameters described earlier in the thesis. The extended ArcheOpterix parser provides an interface for extracting information from architecture elements and annotations. Annotations to the architecture elements can be numbers, character strings as well as a probability density function or probability mass function. Furthermore, the parameter annotations can also be an array (or list) of any data type.

In addition to the extraction of relevant information for optimisation, ArcheOpterix also provides the feature of transforming the produced solutions back into an application domain as AADL specifications or XML. For example, if the tool has been used for optimising deployment architectures, the output will be a set of AADL specifications with deployment relevant code.

ArcheOpterix Core

The core of the ArcheOpterix framework governs the preliminary control flow of the tool, integrating the functions implemented in the interfaces. Besides the execution governance of the tool, the core also acts as a central repository of the framework. The information extracted from the parser interface is kept in abstract data structures so that the modules implementing the interfaces can access as required. For example, evaluation of reliability of a deployment architecture requires information such as hardware failure rates, transition probabilities and failure data associated with algorithmic software components. This data is given as the annotations to the corresponding entities, and kept in the ArcheOpterix core such that a reliability evaluation module can query the required parameters.

Architecture Transformation Operators Interface

The optimisation of embedded systems involves making changes in a defined set of design decisions and search for better alternative architectures. These possible changes to an architecture is called architecture transformation operators [64, 186, 216]. To support various transformation operators, the ArcheOpterix framework provides an abstract interface. An implementation of this interface will be given a set of initial architectures as an input and returns one or more architectures. This many-to-many relationship in the transformation operators also reflects the characteristics that have been explained in the high-level model in Chapter 6. For example, when two deployment architectures are given as the input, swapping certain allocations of software components results in two new deployment architectures. However, the transformation operator can also be unary, for instance the change of a parameter in a given architecture. The framework also leaves the flexibility to use suitable solution representations to support these operators. The current implementation contains several representations of component deployment, redundancy allocation, scheduling and component selection operators. The concrete implementations of these operators have been conducted in relation to specific problems. For example, in the context of deployment architecture optimisation, solution representation is selected as an array containing the processor allocation of each software component, and deployment change is implemented as a change of allocation value in the array. Possible implementations of transformation operators are further illustrated in the experiments to follow.

Architecture Quality Evaluation Interface

In an automated architecture optimisation, it is essential to quantify the quality of individual candidates. The desired quality can be defined in terms of multiple attributes such as reliabilities of different services, performance metrics, energy consumption, cost and scheduling length. As also discussed in the background

chapter, architecture-based evaluation of these quality attributes require the use of probabilistic models. A multitude of models have been developed for specific quality attributes and quality metrics, for instance the DTMC-based reliability model to compute service reliability. To support diverse quality attributes and quality models, the ArcheOpterix framework provides an abstract interface with `Evaluate(Architecture, Problem)` and returns a data structure of `Attribute`. The implementation of the quality evaluation model and computation can be implemented as an extended module, specific to the quality attribute. During the evaluation of an architecture, the required information on input entities can be queried from the ArcheOpterix core via the `problem` object. We have already implemented and used many quality attributes and evaluation models including the following. For further details, please refer to our related publications cited below.

- service reliability [289, 291]
- response time [287]
- data communication overhead [6, 7, 8]
- cost [287]
- data transmission reliability [6, 7, 8]
- energy consumption [288]
- scheduling length [300]

An important aspect of this model is the integration of architecture evaluation under uncertainty, which we have presented in a prior chapter. We have provided the ability to incorporate probability distributions into the annotations, and the quality evaluation model is capable of identifying the uncertain parameters associated with the probabilistic model. In order to quantify the quality attribute (*e.g.* service reliability) with a desired tolerance against uncertainty, the new evaluation method presented in Chapter 5 will be used. The Monte Carlo simulation and Dynamic Stopping Criterion have been implemented as separate modules where quality evaluation modules can be used wherever required.

Architecture Constraint Validation Interface

In the design of embedded systems architecture, it is often necessary to satisfy numerous constraints. In the process of the search for better architecture alternatives, these constraints have to be checked before reporting a solution as a candidate. Certain constraints are related to specific elements in the architecture (*e.g.* localisation constraints) where as some require information on the overall architecture (*e.g.* safety constraints) to check satisfaction. The *architecture constraint validation interface* in the ArcheOpterix provides a generic means to specify and check the validity of architectural constraints. A `Constraint` is treated as a criterion to be checked for a given architecture, and the validation can be implemented in external modules. For example, in an embedded software deployment problem, primary and redundant software components should not be deployed to the same hardware. This can be specified as a *collocation* constraint (please refer to Chapter 2) and checking is implemented in `collocation validation` module. This interface is also useful in implementing constraint handling mechanisms. The `validate` method can be used for *prohibit/death penalty* strategy where as the `quantify violations` method caters for implementing *penalty* strategies. Appropriate solution *repair* mechanisms can be implemented using the problem-specific knowledge and solution representation. The current implementations of the tool contains constraint validation modules including the following where references are made for our related publications;

- localisation [6, 7, 8, 289, 300]
- collocation [6, 7, 289, 300]
- memory [6, 7, 8, 289, 300]
- redundancy levels [287, 288]

Another specific characteristic of constraint management in ArcheOpterix is the possibility of having built-in constraints in the transformation operators. It can be more efficient and meaningful to check and ensure the validity of certain constraints in the application of decision changes. For example, if a software component has to communicate with a specific sensor, the access to that sensor can be checked before deploying that component into a processing hardware unit. This feature can be

embedded into `change deployment` operator, hence not requiring an explicit checking as a constraint. To support this facility, the constraint validation interface also provide a method to dynamically enable or disable specific constraints depending on the problem in focus.

Optimisation Algorithm Interface

A key contribution of ArcheOpterix is to enable the use of sophisticated algorithms which are abstract and domain independent by nature, for the purpose of architecture optimisation. The *Architecture Optimisation Interface* (AOI) of the framework is the link to cater for this purpose, coupling architecture evaluation with abstract optimisation algorithms. Knowledge of the application context and the system attributes are hidden from the algorithms with the AOI by enforcing two generic functions to communicate with; namely `Evaluate` and `Validate`. The interface is implemented with a standard Strategy design pattern [162] and therefore different algorithmic optimisation strategies can be linked with the framework by plugging them into the AOI. However, as the central repository is accessible to the optimisation algorithm, it also provides the freedom to use any problem specific characteristics. Consequently, the ArcheOpterix AOI supports black-box metaheuristics as well as problem specific heuristics or exact methods. Different algorithms may achieve optimisation using their own approaches. However, all algorithms have to check whether the newly generated architecture is valid with respect to its constraints and fulfils its quality requirements. To achieve these two objectives, AOI communicates with the ArcheOpterix core using the two generic functions mentioned earlier. The output will be a set of architectures that can be optimal, near-optimal, Pareto optimal or near-Pareto optimal depending on the optimisation technique. For instance, if only one objective is used and an exact method is implemented for optimisation, an optimal architecture will be produced where as when a metaheuristic (*e.g.* NSGA) is applied with multiple objective, a (near-) Pareto optimal set of candidates will be found. The following algorithms have already been implemented and available

to use. For further details, please refer to our related publications cited below.

- Multi-Objective Genetic Algorithm (MOGA) [6, 7, 289]
- Non-dominated Sorting Genetic Algorithm (NSGA-II) [9, 288, 291, 300]
- Pareto Ant Colony Algorithm (P-ACO) [7, 287]
- Simulated Annealing (SA)
- Hill Climbing
- Bayesian Heuristic for Component Deployment Optimisation (BHCDO) [8]
- Random Search Algorithm [291]
- Brute-Force Algorithms [291]

7.1.2 Maturity and Availability

The source code and binary distribution of ArcheOpterix is available here: `http://mercury.it.swin.edu.au/g_archeopterix/`. The current distribution of the tool consists of a set of solutions representations, architecture design problems, architecture-based quality evaluation models, constraint validation models and optimisation algorithms listed in this chapter. The description of the tool accompanied with examples, case studies, a user guide and a development guide for the use of the tool as a library. The problem files of the case study and experiments presented in the thesis can also be found in the web site.

7.1.3 Comparison to Related Tools and Frameworks

A considerable number of tools have been developed over the past decade to provide support in finding optimal and near-optimal architectures with respect to different non-functional attributes. In this section, the ArcheOpterix tool is compared with the tools that are publicly available for architecture optimisation.

ArcheOpterix shares similar goals with the optimisation frameworks such as DeSi [296], ArchE [31, 121], DAnCE [118] and the RACE [355]. The DeSi tool, developed by Mikic-Rakic *et al.* [296] presents a tailorable environment for specification, manipulation, visualisation and attribute evaluation of deployment architec-

tures. The tool has been developed as a stand-alone Eclipse application and allows run-time deployment optimisation via monitoring of live systems. The tool is very mature, but has a limited focus on constrained deployment problems and requires model specifications in a special format. ArcheOpterix currently implements similar analysis capabilities; however the scope of ArcheOpterix extends the coverage of use in context of architecture optimisation beyond deployment decision making. Furthermore, ArcheOpterix incorporates the use of complex probabilistic models for quality evaluation as well as diversity of optimisation algorithms in contrast to DeSi. The ArchE tool [31] is a design assistant that helps software architects to make decisions with respect to relevant quality attributes. ArchE contains a rule-based expert systems that search the design space with reasoning frameworks. Each reasoning framework supports one quality domain and currently a reasoning framework is implemented for modifiability [31]. To allow the support of other quality domains ArchE provides a well defined interface to generate reasoning frameworks for other quality attributes [121] as well. The RACE framework presented by Shankaran *et al.* [355] and the DAnCE implementation by Deng *et al.* [118] addresses the needs of deployment decisions at real-time. Both these approaches have been found successful in industry, but limitations exist for the applicability for design-time architecture optimisation domain. They require implementations of specific system models with limited flexibility to the changes in architecture, and lack of support for complex probabilistic model analysis. Becker *et al.* [44, 45] has introduced a new meta-model for component-based software engineering, called *Palladio Component Model (PCM)*. PCM consists of modelling support for software components, component assembly, allocation of components to resource platform as well as usage. The dependencies are resolved in the scope of *architecture* where the complete assembly is considered. However, the current implementation of PCM is aimed at the design-time evaluation of component-based systems rather than finding optimal design candidates. In his PhD thesis, Bondarev [64] presents CARAET toolkit for design-time performance analysis and performance optimisation using third party

optimisation tools. All these tool provide comparable capabilities to ArcheOpterix. However, these tools need specialised formats for the input architectures and its quality annotations. Often these input formats are very restricted to specific quality attribute in focus. In contrast, ArcheOpterix defines an abstract interface to extract annotations of architectural elements in the high-level model depicted in Figure 6.2 and supports established architecture description languages including AADL.

The ArcheOpterix tooling framework can also be compared with validation techniques used in certain research contributions. Most of these approaches have also been implemented as tools, however they are either only for demonstration and experiment purposes or they are not available to the public. Papadopoulos and Grante [316] provide a novel technique for safety and reliability analysis in automotive software. Their approach consists of system modelling in Matlab Simulink [383], the approach combines fault tree analysis and genetic algorithms to support the decisions of "whether" and "when" redundancies are needed. Being able to link the system with Matlab Simulink, they provide the opportunity for standard analysis capabilities. ArcheOpterix gains a similar advantage from the modelling perspective by using the standard ADL or XML. Since our tool offers the attribute evaluation module in the same tool, the presented approach extends the [316] work from a semi-automated process into one integrated, automated process.

Fredriksson *et al.* [159] present a framework for allocating components to real time tasks, focusing on minimising the resource usage such as CPU time and memory. A model is given using components and tasks, and formulations are provided to derive memory consumption, CPU overhead for task deployment. The approach uses existing scheduling and optimisation algorithms with real-time analysis to ensure feasible allocation. This approach has been specifically tailored for a domain and a problem, where as the abstract ArcheOpterix framework provides similar capabilities in addition to the applicability of broader range of problems in the context of architecture optimisation in embedded systems.

Deployment architecture decision making for highly constrained environments

has been addressed by Kichkalyo *et al.* [230]. Their approach is to use AI planning techniques to find a feasible solution. The formalised general model of a Component Placement Problem (CPP) has been analysed by their Sekitei algorithm. A major novelty of ArcheOpterix in relation to [159, 230] is the ability to provide a set of non-dominated solutions instead of obtaining one single feasible solution, which is common to optimisation approaches that use a weighted sum function to translate a multi-objective optimisation problem into one with a single objective. Furthermore, ArcheOpterix's optimisation framework is applicable to architecture design problems other than component deployment in the above two approaches.

7.2 Example Application of the SCOUT Approach

This section demonstrates how the SCOUT approach can be applied on the automotive deployment architecture optimisation problem described in Chapter 3. The use of high-level modelling framework to model the specific problem is illustrated first followed by a discussion on the extended decision support given in the results of the new method.

7.2.1 Mapping the Problem to the High-level Model

The robust optimisation framework presented in this chapter has been applied to the deployment architecture design problem for the automotive subsystems ABS and ACC described in Chapter 3. According to the description of SCOUT's high-level model, the abstract entities can be mapped to the concrete problem-related elements as follows.

- Candidate Architecture

  Since the architectural decision of the considered automotive problem is deployment, the *candidate architecture* abstraction can be mapped to a deployment representation with all other architectural aspects fixed. A deployment architecture candidate is encoded into an array of atomic deployment decisions. Each element in the array represents an allocation of a component $c_i \in C$ to an ECU $u_j = d(c_i) \in U$.

- Architectural Elements

  Four types of architectural elements are involved in this case study. On the software level, software components and inter-component interactions are architectural elements. ECUs and network connections (ECU to ECU using buses) are hardware level manifestations of architectural elements.

- Parameters

  Parameters of the respective architectural elements reflect the concept of element parameters. With the SCOUT approach, it is possible to specify parameter-related uncertainties. The following parameters and the respective uncertainties are considered during the robust optimisation process;

  - fr : Failure rate of ECUs and buses, which are annotated as beta distributions with the same mean values as shown in Table 3.1.

  - q_0,p : Execution initialization probability and transition probabilities, which are annotated as normal distributions with the same mean as shown in Table 3.1 and with 0.01 variance. The complete parameter specification including their uncertainty aspects for the case study is given in Table 7.1.

- Quality Attributes

  The goal of the multi-objective optimisation problem in the case study is to find the approximate set of component deployment solutions that represent a robust trade-off between the conflicting objectives in $Q : A \rightarrow \mathbb{R}^2$, *i.e.* the reliabilities of the ABS and ACC services. For reliability evaluation from the architecture, the same DTMC-based reliability model is used as described in Chapter 3.

- Constraints

  Constraints in the deployment optimisation problem reflect criteria for the feasibility of a deployment. Two constraints are used in this case study.

  - Memory: the memory requirement for all software components allocated to an ECU should not exceed its capacity (cp).
  - Localisation: Software components 2,6,7 can only be deployed to ECUs

which have access to LIN bus 1, and components 4 and 5 require access to LIN bus 2.

Death-penalty strategy is used as a constraint handling technique, where newly generated solutions with constraint violations are disregarded and prohibited from including into the population.

- Robust Metric

  This entity of the high-level model is mapped to the desired level of tolerance in the produced solutions to uncertainty in input parameters. In this case study, the objectives of the robust optimisation have been fixed to the 5^{th} percentile which indicates 95% confidence level in the reliabilities of each service. The uncertainty analyser has configured with window size $(k) = 10$ and threshold error $(w_r)= 0.05$.

- Transformation Operators

  In the context of deployment architecture optimisation, the transformation operators refer to possible changes to the candidate deployment architecture. These changes can be mapped to the operators of the optimisation algorithm in use. According to the adoption of three optimisation algorithms to the framework as presented in Chapter 6, deployment changes can be realised differently in each solution representation. This illustration uses two transformation operator mappings to NSGA-II operators. The NSGA-II operators crossover and mutation can be mapped into deployment changes as follows.

  Crossover is defined as a recombination of two deployment alternatives $d_i = [u_{i_1}, u_{i_2}, ..., u_{i_n}]$ and $d_j = [u_{j_1}, u_{j_2}, ..., u_{j_n}]$ in generating a new deployment solution, *i.e.* for a random k: $d'_i = [u_{i_1}, ..., u_{i_{k-1}}, u_{j_k}, ..., u_{j_n}]$ and $d'_j = [u_{j_1}, ..., u_{j_{k-1}}, u_{i_k}, ..., u_{i_n}]$.

  Mutation produces a new solution d'_i from existing d_i by switching the atomic deployment of two components, *i.e.* for randomly selected k, l: $d'_i = [u_{i_1}, ..., u_{i_l}, ..., u_{i_k}, ..., u_{i_n}]$ while the original is $d_i = [u_{i_1}, ..., u_{i_k}, ..., u_{i_l}, ..., u_{i_n}]$.

Comp. ID	sz (KB)	q_0 ABS	q_0 ACC	wl_{ABS} (MI)	wl_{ACC} (MI)
0	512	NORMAL,0.1,0.01	0	1.2	1.2
1	128	0	0	0.6	0
2	128	NORMAL,0.3,0.01	NORMAL,0.2,0.01	0.4	0.4
3	256	0	0	1	0
4	128	NORMAL,0.3,0.01	0	0.4	0
5	128	NORMAL,0.3,0.01	0	0.4	0
6	128	0	0	0.4	0
7	128	0	0	0.4	0
8	128	0	NORMAL,0.4,0.01	0	0.8
9	256	0	0	0	1.1
10	512	0	0	0	1.2
11	512	0	0	0	2.1
12	256	0	0	0	2.1
13	256	0	NORMAL,0.2,0.01	0	0.9
14	256	0	NORMAL,0.2,0.01	0	0.9
15	128	0	0	0	0
16	256	0	0	0	0
17	512	0	0	0	0
18	256	0	0	0	0
19	128	0	0	0	0

(a) Software Components

ECU ID	cp (KB)	ps (MIPS)	fr (h^{-1})
0	512	40	$BETA\_SHD, 1.33 \cdot 10^{-4}, 4. \cdot 10^{-3}, 10, 2$
1	512	22	$BETA\_SHD, 1.33 \cdot 10^{-4}, 4. \cdot 10^{-3}, 10, 2$
2	512	45	$BETA\_SHD, 6.67 \cdot 10^{-5}, 2. \cdot 10^{-3}, 10, 2$
3	512	22	$BETA\_SHD, 6.67 \cdot 10^{-6}, 2. \cdot 10^{-4}, 10, 2$
4	1024	22	$BETA\_SHD, 3.33 \cdot 10^{-5}, 1. \cdot 10^{-3}, 10, 2$
5	512	22	$BETA\_SHD, 3.33 \cdot 10^{-6}, 1. \cdot 10^{-4}, 10, 2$
6	1024	110	$BETA\_SHD, 2.67 \cdot 10^{-6}, 8. \cdot 10^{-5}, 10, 2$
7	512	110	$BETA\_SHD, 6.67 \cdot 10^{-7}, 2. \cdot 10^{-5}, 10, 2$
8	512	22	$BETA\_SHD, 3. \cdot 10^{-6}, 9. \cdot 10^{-5}, 10, 2$

(b) ECUs

Trans $c_i \rightarrow c_j$	$p(c_i, c_j)$ ABS	$p(c_i, c_j)$ ACC	ds_{ABS} (KB)	ds_{ACC} (KB)
$0 \rightarrow 6$	NORMAL,0.5,0.01	0	2	0
$0 \rightarrow 7$	NORMAL,0.5,0.01	0	2	0
$1 \rightarrow 3$	1	0	2	0
$2 \rightarrow 1$	1	0	2	0
$2 \rightarrow 9$	0	1	0	2
$3 \rightarrow 0$	1	0	2	0
$4 \rightarrow 0$	NORMAL,0.7,0.01	0	1	0
$4 \rightarrow 3$	NORMAL,0.3,0.01	0	2	0
$5 \rightarrow 0$	NORMAL,0.7,0.01	0	1	0
$5 \rightarrow 3$	NORMAL,0.3,0.01	0	2	0
$8 \rightarrow 9$	0	1	0	1
$9 \rightarrow 0$	0	NORMAL,0.1,0.01	0	4
$9 \rightarrow 11$	0	NORMAL,0.4,0.01	0	2
$9 \rightarrow 12$	0	NORMAL,0.5,0.01	0	1
$10 \rightarrow 8$	0	NORMAL,0.4,0.01	0	1
$10 \rightarrow 9$	0	NORMAL,0.6,0.01	0	2
$11 \rightarrow 12$	0	1	0	3
$13 \rightarrow 10$	0	NORMAL,0.5,0.01	0	1
$13 \rightarrow 11$	0	NORMAL,0.5,0.01	0	1
$14 \rightarrow 10$	0	NORMAL,0.5,0.01	0	2
$14 \rightarrow 11$	0	NORMAL,0.5,0.01	0	2
$15 \rightarrow 17$	0	0	0	0
$16 \rightarrow 17$	0	0	0	0
$16 \rightarrow 18$	0	0	0	0
$17 \rightarrow 18$	0	0	0	0
$17 \rightarrow 19$	0	0	0	0
$18 \rightarrow 17$	0	0	0	0
$18 \rightarrow 19$	0	0	0	0

(c) Component Interactions

BUS ID	dr (KBPS)	fr (h^{-1})
0	128	$BETA\_SHD, 3. \cdot 10^{-6}, 3. \cdot 10^{-4}, 10, 2$
1	64	$BETA\_SHD, 1.2 \cdot 10^{-5}, 1.2 \cdot 10^{-3}, 10, 2$
2	64	$BETA\_SHD, 4. \cdot 10^{-6}, 4. \cdot 10^{-4}, 10, 2$

(d) Buses

Table 7.1: Parameter specification with uncertainty for ABS and ACC case study

7.2.2 Results of Robust Optimisation

As a motivation for the robust optimisation approach, Chapter 3 first presented the solutions obtained from a conventional optimisation technique. It could be recalled that the solutions obtained using state-of-the-art point estimate-based methods lack the decision support *w.r.t.* the uncertainties in the input parameters. In Figure 3.3 the results indicate that the reliabilities of the solutions obtained from conventional point-estimate-based techniques become significantly different when failure rates of some ECUs increased by only 5 times. However, the failure rates are estimated under uncertainty, and they can vary by a factor up to 40 times and other parameters such as usage profile can also be different from the estimates. Consequently, the actual service reliabilities of those solutions could be even worse. In contrast, with the SCOUT approach, the uncertainty associated with many parameters including ECU failure rates has been incorporated into the parameter specification as presented in Table 7.1.

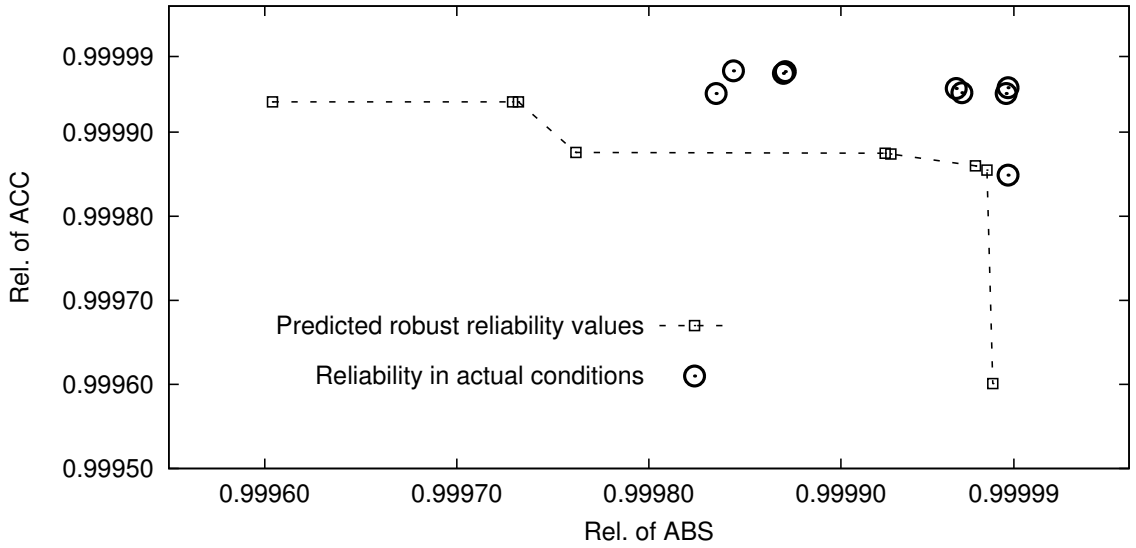


Figure 7.2: Comparison of reliabilities of the solutions obtained from robust optimisation and their values in considered actual conditions

Solution	Reliability of ABS				Reliability of ACC			
	μ	σ^2	95% Conf.	Robust. Test	μ	σ^2	95% Conf.	Robust. Test
1	0.999832	$1.6 \cdot 10^{-8}$	0.999597	$\surd$	0.999972	$2.3 \cdot 10^{-10}$	0.999946	$\surd$
2	0.999871	$7.7 \cdot 10^{-9}$	0.999706	$\surd$	0.999970	$3.5 \cdot 10^{-10}$	0.999934	$\surd$
3	0.999872	$7.8 \cdot 10^{-9}$	0.999704	$\surd$	0.999970	$3.4 \cdot 10^{-10}$	0.999935	$\surd$
4	0.999841	$1.1 \cdot 10^{-8}$	0.999660	$\surd$	0.999947	$9.1 \cdot 10^{-10}$	0.999893	$\surd$
5	0.999958	$6.1 \cdot 10^{-10}$	0.999913	$\surd$	0.999951	$1.3 \cdot 10^{-9}$	0.999881	$\surd$
6	0.999960	$5.8 \cdot 10^{-10}$	0.999916	$\surd$	0.999948	$1.4 \cdot 10^{-9}$	0.999876	$\surd$
7	0.999986	$2.8 \cdot 10^{-11}$	0.999976	$\surd$	0.999946	$1.4 \cdot 10^{-9}$	0.999875	$\surd$
8	0.999986	$2.9 \cdot 10^{-11}$	0.999976	$\surd$	0.999946	$1.4 \cdot 10^{-9}$	0.999876	$\surd$
9	0.999986	$2.8 \cdot 10^{-11}$	0.999977	$\surd$	0.999834	$1.7 \cdot 10^{-8}$	0.999581	$\surd$

Table 7.2: Accuracy of robustness in the solutions obtained from the new method

The new robust optimisation approach with the NSGA-II algorithm has been executed on the given configurations, and the obtained solutions are analysed *w.r.t.* the uncertain input parameters. The dashed line in Figure 7.2 depicts the 95% confidence estimates of the reliabilities of non-dominated robust solutions obtained from the method. In order to compare them with the results obtained from the

conventional point-estimate-based optimisation, the same variation as described in Section 3.4 (*i.e.* increase of the failure rates of ECU0,1,2 by 5 times) has been applied to the solutions obtained from the robust optimisation. The points depicted with circles in Figure 7.2 represent the updated reliability values if higher ECU failure rates are observed in actual conditions. It is evident that none of the solutions under-performs their predicted bounds in the robust optimisation, whereas in the conventional evaluation results, the actual values differ significantly from the predicted reliabilities (Figure 3.3). The lack of robustness in the solutions obtained from existing methods is due to the fact that their optimisation process is based on point estimated parameters and does not have the ability to consider different levels of tolerance.

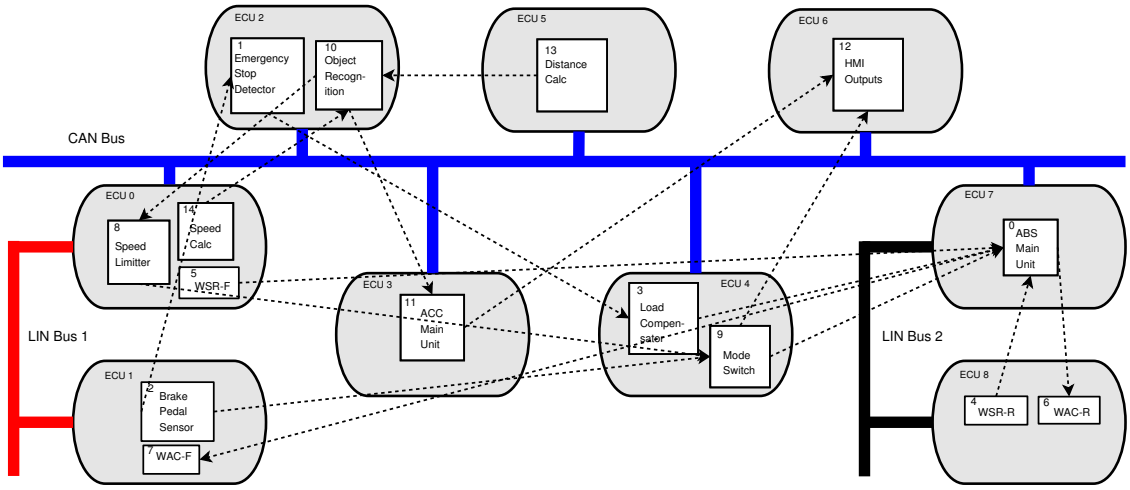


Figure 7.3: A sample deployment solution produced by the robust optimisation. Dotted lines represent communications among software compoments and solid lines represent hardware buses

Figure 7.3 illustrates the deployment details of one solution produced by the robust optimisation. The solution satisfies the given deployment constraints (*i.e.* memory and localisation) and indicates reliability values of 0.999597 and 0.999946 for ABS and ACC services respectively, with 95% confidence of immunity under uncertainties in the given input parameters. In order to test the robustness of the solutions produced by the method presented in this work, an experiment was conducted investigating the placement of robust reliability value (percentile) in their

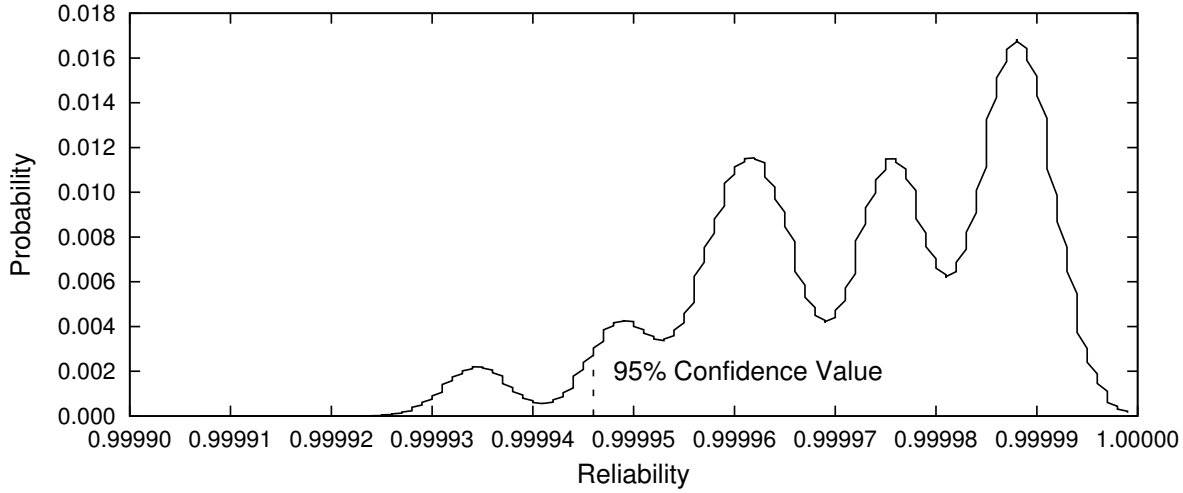


Figure 7.4: Histogram of an objective (Reliability of ACC for solution 1) constructed from MC runs

respective distributions. The actual distribution of reliability of a solution is approximated by constructing the histogram from reliability values for a large number of random samples taken from the input parameter distributions (Law of Large Numbers). Figure 7.4 depicts the histogram that corresponds to the service reliability of above solution, obtained from the random variations of input parameters. Table 7.2 further illustrates the accuracy of the use of the 95% confidence value as the robust objective in the optimisation process. The mean and variance obtained from 10,000 random cases in the robust solutions are compared with the 95% confidence reliability value provided at robust optimisation. When large numbers of samples are taken, the mean of the distribution can be assumed as normal (Central Limit Theorem), and hence the 95% confidence interval of a variable can be computed from $\mu \pm \frac{1.96\sigma}{\sqrt{n}}$ [299]. As a robustness test of the architecture solutions, we compare the 95% confidence estimate provided from the robust optimisation with the lower bound of the confidence interval. From the statistics of the 10,000 random tests, satisfaction of the relationship

$$\mu - \frac{1.96\sigma}{\sqrt{n}} > 95\% \text{ Conf. value}$$

for all solutions indicates that the solutions obtained from the presented robust optimisation approach are capable of tolerating the uncertainty.

Overall, the new robust optimisation approach has been able to include and use uncertain information available with regards to the architecture optimisation problem in focus. The approach has provided a set of solutions whose reliabilities are robust *w.r.t.* the uncertainty associated with the input parameters. The architect is given the flexibility to specify uncertain input parameters, and level of required tolerance against the uncertainty. The new deployment architectures produced from new approach includes additional decision support with (near-) optimality and a confidence on robustness of the solutions, which has not been possible with the conventional architecture optimisation methods. This example application provides evidence that the SCOUT approach has been able to answer the architecture-based reliability evaluation aspect (RQ 1) as well as the modelling and optimisation aspects (RQ 2) of the research problem in the context of the case study.

7.3 Experiments

The building blocks of the robust optimisation have been presented in the previous chapters. Each chapter contains confined evidence on the validity of the corresponding aspects of the contribution. The overall applicability of the SCOUT approach has been explained and illustrated with the case study of automotive ABS/ACC system. Supplementing to the validations presented before, this section further investigates various aspects of the overall contribution towards addressing the research questions formulated in Chapter 4.

7.3.1 Evaluation Criteria

From the point of view of the software engineering research problem formulation presented in Chapter 4, the case study can be considered as only one instance of the architecture optimisation problem. Even though the case study experiments have shown that the proposed approach provides good solutions to the specific problem instance, the scope of the research questions requires investigating whether consistent results can be obtained for problems with different settings and characteristics than the case study. Studies on software engineering research methods confirm that an *evaluation* of the proposed solution is an accepted form of validation for the *method or means of development* type research settings which provide *procedure or technique* [362,363] solutions. This section presents an *evaluation* of the SCOUT approach using experiments on a series of systematically designed problem instances. The following evaluation criteria are formulated to evaluate the SCOUT approach in addressing the corresponding research questions.

EC 1. Application to the case study has shown that the SCOUT approach is capable of including the uncertain parameters in a specific problem instance. However, the nature of uncertainty associated with different parameters of software architecture can be heterogeneous. One criterion considered in designing the experiments is to provide evidence of the SCOUT approach's ability to capture various uncertain characteristics codified in RQ 1.1.

EC 2. The research question of architecture-based reliability evaluation under uncertainty in model parameters is formulated as RQ 1.2. This work presents a method for reliability model evaluation which utilises quantitative metrics for finding candidate architectures which can tolerate possible variations of input parameters. Hence, the experiments are designed to investigate whether the solutions produced by the SCOUT approach are able to tolerate various uncertain conditions without violating the predicted quality.

EC 3. The ability of a candidate architecture to tolerate input parameter uncertainty is treated differently in different application domains. The SCOUT approach proposes to include the notion of immunity to uncertain parameters into quality goals, and provides support to use statistical indecisive of conventional quality metrics, particularly percentiles of reliability. This evaluation criterion seeks the validity of the proposed solution in addressing RQ 1.3.

EC 4. The modelling aspect of the overall research goal has been formulated as RQ 2.1. The experiments on the case study have illustrated an application of the approach to solve a deployment problem. Based on RQ 2.1, this evaluation criterion investigates whether the proposed solution has been able to cater for various aspects involved in architecture optimisation including design problems other than deployment.

EC 5. RQ 2.2 has formulated the question of providing the ability to use different stochastic optimisation algorithms in the context of designing software-intensive systems for search of solutions which gives better trade-offs in terms of quality while being robust to uncertainties in the input parameters. The SCOUT has presented a high-level modelling framework and adaptation of three stochastic optimisation algorithms. This evaluation criterion pursues experimental evidence to support the validity of the solution in addressing RQ 2.2.

EC 6. The overall SCOUT approach presents a method to architecture optimisation under uncertainty which contains an inner loop for Monte Carlo simulation-based uncertainty analysis in addition to the outer loop of architecture optimisation. It is important to investigate the time complexity of the approach with respect to different architecture optimisation problem sizes and instances. This evaluation criterion is considered to design the experiments with a focus of scalability evaluation of the SCOUT approach.

7.3.2 Experiment Setup

In order to represent diverse characteristics of the architecture optimisation problem with uncertain input parameters, a series of problem instances have been generated as follows.

Problem Types and Instances

In order to illustrate the SCOUT approach's ability to cater for diversity of architecture optimisation problems, problem instances are included that belong to different types of architecture decisions and optimisation objectives conflicting with reliability. Firstly, a set of deployment problems are considered which follow the same pattern of problem as the automotive ABS/ACC case study. A series of experiments are also conducted with another two types of architecture design problems, namely *redundancy allocation* and *component selection* which are widely investigated problems in the literature.

Deployment

The first set of problem instances are created in the form of deployment architecture optimisation problems.

■ *System Model.* The problem definition of the these experiments follows the same, but generalised formulation to the automotive ABS/ACC case study. The

experiment considers the general notion of deployment architecture in software-intensive systems which refers to the allocation of software components to hardware hosts and the assignment of inter-component communications to network links. Let $C = \{c_1, c_2, ..., c_n\}$, $n \in \mathbb{N}$, denote the set of all software components, and $U = \{u_1, u_2, ..., u_m\}$, $m \in \mathbb{N}$ be the set of hardware hosts. A complete deployment of all software components into hosts is denoted d, and the set of all possible d is $D = \{d \mid d : C \rightarrow U\}$. Since both C and U are finite, D is also finite.

■ *Generation of Deployment Problem Instances.* Five problem instances are created with different numbers of hosts and software components to be deployed. The interactions among software components are randomly defined, but the number of connections and nature of the interactions are adjusted to realistic problem settings.The network connections that connect the hosts are also configured randomly. The sizes and configurations of the problem instances are listed in Table 7.3.

Index	**Case ID**	**Components**	**ECUs**	**Buses**
1	DEP_H8_C20	20	8	1
2	DEP_H16_C40	40	16	2
3	DEP_H32_C80	80	32	2
4	DEP_H64_C160	160	64	2
5	DEP_H128_C320	320	128	3

Table 7.3: Configurations of deployment problem instances

Redundancy Allocation

Software and hardware level redundancy allocation is one of the key techniques used in improving reliability of software-intensive systems. However, adding redundant elements often makes adverse affects to the designed system such as increased cost or energy consumption. Hence, the Redundancy Allocation Problem (RAP) is formulated in the form of finding (near-) optimal set of allocations in terms of reliability and other conflicting objectives, and addressed as a combinatorial multiobjective optimisation problem [245, 246, 386, 387].

■ *System Model.* In these experiments, a model of embedded systems is used that is structured into interacting components (system elements), called special purpose microprocessors (MPs). The MPs are treated as self-contained micro-computers along with the software, dedicated to fulfil a specific functionality. They have only one entry and one exit points, and behave the same for each execution (visit of the MP by system control flow). For the redundancy allocation, the hot spare design topology is used with N-Modular Redundancy (NMR) extension [269]. In hot sparing, each component in the system has a number of replicas (possibly zero), all of which are active at the same time, mimicking the execution of the original component. For the purpose of describing the experiments, following paragraphs briefly outline the RAP model.

Let $C = \{c_1, c_2, ..., c_n\}$, $n \in \mathbb{N}$, denote the set of all components (before replication), and $\mathcal{I} = \{1, 2, ..., n\}$ the index set for components in C. A complete assignment of the redundancy levels for all the components is denoted a, and the set of all possible a is denoted $A = \{a \mid a : C \rightarrow L\}$, where $L = \{l \mid 0 \leq l \leq max\ redundancy, l \in \mathbb{N}_0\}$ delimits the redundancy level of a component[1]. Note that, since C and L are finite, A is also finite. A component c_i together with its redundancies form a *subsystem* S_i, which can be uniquely determined by c_i.

The interaction among components without replication can be formalised in terms of an absorbing DTMC, where nodes represent system components $c_i \in C$, and transitions represent the transfer of execution from one component to another (together with the probability of the transfer). Equivalently, the system with replication can be formalised as an absorbing DTMC where nodes represent the subsystems (components with their replicas), and the transitions represent the transfer of execution among subsystems. Note that since the replicated links are combined into transitions in the DTMC, the transfer probabilities remain unchanged with respect to redundancy allocation. In both cases, the DTMC represents execution flow of the system (its reaction to an external event), with possibly many execution scenarios

[1]The original component is not counted as redundant, hence at least the one is always present in a subsystem.

(initiated in different nodes of the DTMC, based on the triggering event).

■ *Generation of RAP Instances.* A set of redundancy allocation problem instances is generated to represent different scales of the problems in practice. The components are randomly generated symbolising different component characteristics (failure rate, execution initialisation probability, execution time, cost *etc.*) which may be present in the practical problems. The interactions among subsystems are randomly defined, but the number of connections and connection probabilities are adjusted to resemble actual architectures. Four problem instances are created with different number of components as listed in Table 7.4.

Index	**Case ID**	**Components**
1	RA_C20	20
2	RA_C40	40
3	RA_C80	80
4	RA_C160	160

Table 7.4: Configurations of redundancy allocation problem instances

Component Selection

As another variant of practical architecture optimisation problems, a set of experiments are created representing the selection of components from a possible set of alternatives in designing a software-intensive system. *Component Selection* (CS) is one of the key design decisions which have to made by an architect during the early stages of the design, and may involve both hardware and software. In the software domain, systems can be built by combining different alternative software components which implement similar functionality with different characteristics [104, 280, 304]. The architect has to make decisions on which ones to use, considering non-functional attributes such as reliability, performance as well as other factors like cost.

■ *System Model.* For the component selection experiments, a model of a software-intensive system is considered that is structured into interacting tasks, which are dedicated to fulfil a specific functionality of the system. Once a task is triggered,

some processing activity takes place which may produce results, or the output is passed to another task. The task can be implemented in different software components, which are often produced by different development teams or vendors.

Let $T = \{t_1, t_2, ..., t_n\}$, $n \in \mathbb{N}$, denote the set of all tasks, and $\mathcal{I} = \{1, 2, ..., n\}$ the index set for tasks in T. For each task t_i, there is a set of candidate components $C_i = \{c_{i_1}, c_{i_2}, c_{i_3}, ..., c_{i_m}\}$, where the number of options m can be different for different tasks. The selection of components for all tasks is denoted s, and the set of all possible s is denoted $S = \{s \mid s : T \to O\}$, where $O = \{o_i \mid 1 \leq o \leq m_i, o \in \mathbb{N}_0, \forall i \in n\}$. Note that, since T and O are finite, S is also finite.

The interactions among tasks can be formalised in terms of an absorbing DTMC, where the nodes represent tasks $t_i \in T$, and the transitions symbolise the transfer of execution from one task to another with a probability of transfer. A path from a starting node to an absorbing node in the DTMC represents a single execution trace of the system (its reaction to an external event).

■ *Generation of Instances.* Considering the nature of the component selection problem, a series of problem instances are created to represent diverse practical settings. Similar to the deployment and redundancy allocation problem generation, the interactions among tasks are randomly defined, but the number of connections and connection probabilities are adjusted to resemble realistic conditions. Four problem instances are generated with different number of tasks as listed in Table 7.5. Number of components for each task in each of the problem instance is randomly set within the range of $[2, 10]$.

Index	**Case ID**	**Tasks**
1	CS_T20	20
2	CS_T40	40
3	CS_T80	80
4	CS_T160	160

Table 7.5: Configurations of component selection problem instances

Input Parameter Setting

In order to validate the support of the SCOUT framework in integrating diverse sources and characteristics of uncertainty associated with estimated parameters, a set of parameters of the generated problem instances were allowed to take heterogeneous probability distributions. The distributions are restricted to spread only within realistic ranges of the corresponding parameter. As described in Chapter 5, not necessarily all the parameters are subject to uncertainty. Illustrating the ability of the framework to include both variable and fixed parameters in any combination, certain parameters are also allowed to be precise values. All of these parameters are randomly generated within realistic ranges, and configuration on whether they are represented as a distribution or fixed value is also changed in different instances.

■ *Deployment Problem Instances.* Since we follow the same deployment formulation as in the case study, the same architecture model and parameters for the generated problem instances are reused. Please refer to Chapter 3 for details on model parameters related to deployment instances. The parameter generation of the deployment problem instances has been carried out as presented in Table 7.6. The $\sim$ sign denotes randomly selecting a value within a range. For example $2^{[4\sim9]}$ represents selecting an integer (x) from the range $[4, 9]$ and obtaining the numerical value of 2^x.

■ *Redundancy Allocation Instances.* The following parameters are generated and annotated to each of the elements in the generated redundancy allocation problem instances. Note that some parameters are specified as probability distributions to represent certain amount of uncertainty associated with the estimation.

- *Failure Rate* (fr), given h^{-1}. The mean failure rate of a component is generated $\lambda_{mean} = [1 \sim 5] \cdot 10^{[-1\sim-3]}$. The failure rate parameter of a component is set to a random-discrete distribution over $[0.1 \times \lambda_{mean}, 100 \times \lambda_{mean}]$.
  Eg. $fr(c_1) = DISCRETE, 3 \cdot 10^{-4}, 0.2, 3 \cdot 10^{-3}, 0.3, 3 \cdot 10^{-2}, 0.3, 0.3, 0.2h^{-1}$
- *Sojourn time* (st), given in ms. The mean sojourn time for a component is randomly generated from the $[1 \sim 6] \cdot 2.0$, and the parameter for each component is set as a random-discrete distribution around this value.

Parameter	Units	Description
ECUs		
Failure Rate (fr_{ecu})	h^{-1}	Mean failure rate of an ECU is generated $\lambda_{mean} = [1 \sim 5] \cdot 10^{[-3 \sim -5]}$. Then the failure rate parameter of an ECU is set to a random-discrete distribution over $[0.1 \times \lambda_{mean} \sim 100 \times \lambda_{mean}]$. Eg. $fr_{ecu}(1) = DISCRETE, 3 \cdot 10^{-4}, 0.2, 3 \cdot 10^{-3}, 0.3, 3 \cdot 10^{-2}, 0.3, 0.3, 0.2h^{-1}$
Memory Capacity (cp)	KB	A randomly generated, fixed value in the rage $2^{[9 \sim 10]}$. Eg. 1024KB
Processing Speed (ps)	MIPS	Randomly generated, fixed value in the range $20 \times [1 \sim 6]$. Eg. 40MIPS
Software Components		
Memory Requirement (sz)	KB	Randomly generated, fixed value in the range $2^{[6 \sim 8]}$. Eg. 512KB
Processing Load (wl)	MI	The processing workload of a software component can be different for each service. Since we generate two services for each case, workloads are randomly generated in the range $[1 \sim 6] \times 0.4$. Eg. $wl_{service1} = 0.8MI, wl_{service2} = 1.2MI$
Execution Initialisation Probability (q_0)		Generated for each service, and ensure that sum of all execution initialisation probabilities for a service across components is 1. Mean execution initialisation probability is randomly generated, and the parameter is set as a normally distributed around that mean, with a random variance. Eg. $q_0 = NORMAL, 0.1, 0.05$
Component Interactions		
Transition Probability (p_{ij})		A random transition probability matrix is generated for each service. The behavioural constraint of *sum of all outgoing transitions from a node should be 1* is ensured during the generation of the DTMC. Then the actual values of the transition probabilities are let to have gamma distributed with those random means. Eg. $p_{c_0,c_1}(service1) = GAMMA, 0.3$, $p_{c_0,c_1}(service2) = GAMMA, 0.8$
Data Size (ds)	KB	A randomly generated, fixed value for each service within the range $[1 \sim 4]$. Eg. $ds_{c_0,c_1}(service1) = 2KB$
Buses		
Data Rate (dr)	KBPS	
Failure Rate (fr_{bus})	h^{-1}	Mean failure rate of a bus is generated $\lambda_{mean} = [2 \sim 9] \cdot 10^{[-2 \sim -4]}$. Then the failure rate parameter of a bus is set to a uniform distribution over $[0.1 \times \lambda_{mean} \sim 100 \times \lambda_{mean}]$. Eg. $fr_{bus}(1) = UNIFORM, 4 \cdot 10^{-4}, 4 \cdot 10^{-2}h^{-1}$

Table 7.6: Parameter generation for the deployment problem instances

Eg. $st(c_1) = DISCRETE, 7.2 \cdot 10^{-2}, 0.1, 8.0 \cdot 10^{-2}, 0.3, 1.2 \cdot 10^{-2}, 0.3, 1.6 \cdot 10^{-2}, 0.2$

- *Cost* (*cst*) of a component is randomly generated $[1 \sim 6] \cdot 5$, and assumed to be precise.
- *Execution Initialisation Probability* (q_0). The mean execution initialisation probability is randomly generated, and the parameter is set as a normally distributed around that mean, with a random variance.
  Eg. $q_0(c_1) = NORMAL, 0.1, 0.05$
- Transition Probability (p_{ij}), A random transition probability matrix is generated for each problem. The behavioural constraint of *the sum of all outgoing transitions from a node should be 1* is ensured during the generation of the DTMC. The actual values of the transition probabilities are let to have gamma distributed with those random means.
  Eg. $p_{c_0,c_1} = GAMMA, 0.3$

■ *Component Selection Instances.* The following parameters are generated and given as annotations to each of the elements in the generated component selection problem instances. Note that some parameters are specified as probability distributions to represent certain amount of uncertainty associated with the estimation.

- *Number of Options* for each task is randomly set from the range $[2 \sim 10]$. Each component option is individually configured with random reliability, execution time and cost.
- *Reliability* (*rel*), generated for each selectable component. The reference reliability class of a component is randomly generated *s.t.* $r_c = [1 \sim 150]$. Considering the uncertainty associated with component reliability estimation, the reliability parameter of a component is set to a distribution over $[0.86, 0.999]$ by creating a random discrete distribution in the range $[e^{-0.00005 \cdot r_c}, e^{-0.001 \cdot r_c}]$.
  Eg. $rel(t_1) = DISCRETE, 0.879, 0.2, 0.962, 0.3, 0.987, 0.3, 0.993, 0.2$
- *Sojourn time* (*st*), given in *ms*. The mean sojourn time for each component option for each task is randomly generated from the $[1 \sim 6] \cdot 2.0$, and the

parameter for each component is set as a random-discrete distribution around this value.

Eg. $st(c_1) = DISCRETE, 7.2 \cdot 10^{-2}, 0.1, 8.0 \cdot 10^{-2}, 0.3, 1.2 \cdot 10^{-2}, 0.3, 1.6 \cdot 10^{-2}, 0.2$

- *Cost* (*cst*) of a component option is randomly generated $[1 \sim 6] \cdot 5$, and assumed to be certain.
- *Execution Initialisation Probability* (q_0). Mean execution initialisation probability is randomly generated for each task, and the parameter is set as a normally distributed around that mean, with a random variance.

  Eg. $q_0(t_1) = NORMAL, 0.1, 0.05$
- Transition Probability (p_{ij}). Similar to the other architecture design problems, a random transition probability matrix is generated for each problem. The behavioural constraint of *sum of all outgoing transitions from a node should be 1* is ensured during the generation of the DTMC. Then the actual values of the transition probabilities are let to have gamma distributed with those random means.

  Eg. $p_{t_0,t_1} = GAMMA, 0.3$

Optimisation Objectives

■ *Deployment Problem Instances.* The robust architecture optimisation approach presented in this thesis is inherently multi-objective. However, as also mentioned in the thesis scope section, the main interest in terms of quality attributes has been restricted to reliability. Illustrating the support for multi-objective optimisation, two services are configured in each of the generated problem instances. Regarding the deployment decisions of software components to a fixed hardware platform, the reliability values of the two services become conflicting [289]. Therefore, the goals to be optimised have been set to the reliabilities of two services. The behaviour of a service is assumed to be a terminating process, *i.e.* the execution starts from one software component and transfers the execution through other components using message

passing, and terminates the execution at a component. The execution can start from a possible set of components, each having a different execution initialisation probability. The probabilistic transitions among software components are modelled using a DTMC which is randomly generated. During the generation of DTMCs for each service, the primitive DTMC constraints are enforced (*i.e.* existence of at least one absorbing state and sum of all outgoing transitions of non-absorbing states should be 1 [175, 391]). In order to show the SCOUT's support for different robustness levels, the level of desired tolerance of service reliabilities against the uncertainties associated with input parameters are varied in different experiments. The first set of experiments is conducted by setting the goals to near-pessimistic estimates having 95% confidence reliability estimation. Therefore, the optimisation is guided using the 5^{th} percentile of reliabilities of each service. Secondly, the experiments are conducted with different tolerance level setting to the first quartile, median and third quartile.

■ *Redundancy Allocation.* Reliability and cost trade-off is considered as the goal of redundancy allocation experiments. During the generation of redundancy allocation problem instances, different parameters are included with a certain amount of uncertainty. Hence, the reliability goal is reliability with a 90% confidence tolerance to uncertainty. In evaluating the reliability of RAP candidates, the expected number of visits of DTMC nodes (represent subsystems) can be computed in the same manner as presented for deployment evaluation (please refer to Section 3.2 in Chapter 3 for details). With the component-annotated failure rate (fr) and processing time (st), the *reliability of a component c_i per visit* can be computed as [361];

$$R_c(c_i) = e^{-fr(c_i)\cdot st(c_i)} \tag{7.1}$$

When the redundancy levels are employed, the *reliability of a subsystem S_i* (with identical replicas connected in parallel) for the architectural alternative a can be computed as:

$$R_c^a(c_i) = 1 - (1 - R_c(c_i))^{a(i)+1} \tag{7.2}$$

Having the reliabilities of individual system elements per visit, the reliability of the system execution can be computed using the expected number of visits [182,241]:

$$R^a \approx \prod_{i \in \mathcal{I}} (R_c^a(c_i))^{v_c(c_i)} \tag{7.3}$$

The cost of a redundancy allocation is computed as the sum of all costs involved in the candidate architecture.

$$C^a = \sum_{i \in \mathcal{I}} cst(c_i) \cdot (a(i) + 1) \tag{7.4}$$

■ *Component Selection.* Given that the scope of the thesis in terms of quality attributes is limited to reliability, these experiments are also used to briefly mention the potential accommodation of other quality attributes in the SCOUT framework. In the component selection problem instances, the tri-objective trade-off among reliability, response time and cost is considered. Different parameters with a certain amount of uncertainty have been included during the generation of problem instances. Hence, the reliability goal is set for reliability with a 90% confidence tolerance to the uncertainty and response time is set to a median tolerance *w.r.t.* to uncertainty. Illustrating the support for the mix of quality objectives which are certain and uncertain, the cost is represented as a point value. In evaluating the reliability and response time of component selection candidates, the expected number of visits of DTMC nodes (represent tasks) can be computed in the same manner presented for deployment evaluation (please refer to Section 3.2). When a component is selected to perform a certain task, the reliability of the task can be computed as;

$$R^s(t_i) = rel(S^{-1}(t_i)) \tag{7.5}$$

where S^{-1} is the inverse of the component selection representation which can be used to obtain the component selected for a task. Having the reliabilities of individual tasks per a single visit, the reliability of the system execution can be

computed using the expected number of visits [182, 241]:

$$R^s \approx \prod_{i \in \mathcal{I}} (R^s(t_i))^{v(t_i)} \tag{7.6}$$

Similarly, response time of the system can be computed as the sum of all execution times in the given component selection as follows.

$$RT^s = \sum_{i \in \mathcal{I}} (st^s(t_i)) \cdot v(t_i) \tag{7.7}$$

Cost of a component selection candidate is computed as the sum of all costs of selected components.

$$C^s = \sum_{i \in \mathcal{I}} cst(S^{-1}(t_i)) \tag{7.8}$$

Design Constraints

The generation of problem instances also considers a set of constraints to be satisfied in all candidate architectures. The following three constraints are enforced throughout the deployment experiments.

■ *Memory.* Memory capacity of an ECU and memory requirement of software components are set randomly, but lie within realistic ranges. Therefore, the amount of software components that can be allocated to an ECU is limited.

■ *Localisation.* As the generation of the deployment problem instances is based on practical embedded architecture design where certain software components commonly require access to hardware-specific capabilities, localisation constraints are embedded into problem generation. For example, some software components require reading from sensors and some need control of certain hardware actuators. In these cases, the deployment of the software components is restricted to ECUs that can access the specific feature. To represent this realistic nature, some of the components (randomly selected) are configured with localisation restrictions to specific ECUs. In reality, the ECUs and software components only implement a subset of

communication protocols, and hence they have an inherent localisation restriction. Mimicking this nature, ECUs and components are allocated to buses and define localisation constraints appropriately.

■ *Collocation.* In a practical embedded software design problem, not every two software components can be deployed to the same hardware. For instance, as per the *N-version software engineering paradigm* [269], the tolerance for hardware failures will not be satisfied if redundant software units are allocated to a single hardware unit. Therefore, collocation constraints are enforced in most of the practical deployment problems. To illustrate the support of the robust optimisation framework, the problem generation configures randomly selected pairs of software components with negative and positive collocation constraints. Checking of this constraint is enforced in validating a candidate deployment architecture.

In redundancy allocation experiments, the maximal redundancy level of subsystems is treated as an optimisation constraint, and use the value of 3 for all the subsystems. For component selection problems, number of available options for a component is restricted in the problem generation.

Optimisation Algorithm Settings

NSGA-II

The NSGA-II optimisation is configured to 50 iterations, crossover rate of 0.8, mutation rate 0.2 and population size of 20. Initial population of architecture candidates are randomly generated for each problem instance, sealing the satisfaction of design constraints. The aforementioned robust quality attributes are used as the optimisation goals in each experiment. The robustness-based selection operator implementation described in Chapter 6 is used for all the experiments with NSGA-II. In order to use the SCOUT framework for a specific architecture optimisation problem, the transformation operators in the problem in focus have to be mapped to the operators in the optimisation algorithm. The crossover and mutation oper-

ators are implemented for each solutions representation corresponding to different architecture optimisation problems.

■ *Deployment.*

Crossover is defined as a recombination of two deployment alternatives $d_i = [u_{i_1}, u_{i_2}, ..., u_{i_n}]$ and $d_j = [u_{j_1}, u_{j_2}, ..., u_{j_n}]$ in generating a new deployment solution, *i.e.* for a random k: $d'_i = [u_{i_1}, ..., u_{i_{k-1}}, u_{j_k}, ..., u_{j_n}]$ and $d'_j = [u_{j_1}, ..., u_{j_{k-1}}, u_{i_k}, ..., u_{i_n}]$.

Mutation produces a new solution d'_i from existing d_i by switching the atomic deployment of two components, *i.e.* for randomly selected k, l: $d'_i = [u_{i_1}, ..., u_{i_l}, ..., u_{i_k}, ..., u_{i_n}]$ while the original is $d_i = [u_{i_1}, ..., u_{i_k}, ..., u_{i_l}, ..., u_{i_n}]$.

■ *Redundancy Allocation.* For the set of experiments with redundancy allocation problem instances, two redundancy allocation changes are mapped to the genetic operators in NSGA-II.

Crossover is implemented as a creation of new redundancy allocation solutions $a'_i, a'_j \in A$ from two parents solutions $a_i = [l_{i_1}, l_{i_2}, ..., l_{i_n}]$ and $a_j = [l_{j_1}, l_{j_2}, ..., l_{j_n}]$ coming from existing population by recombining the redundancy levels of components, *i.e.* for a random k: $a'_i = [l_{i_1}, ..., l_{i_{k-1}}, l_{j_k}, ..., l_{j_n}]$ and $a'_j = [l_{j_1}, ..., l_{j_{k-1}}, l_{i_k}, ..., l_{i_n}]$.

Mutation is mapped to produces a new redundancy allocation solution a'_i from existing solution a_i by changing the redundancy level of a randomly selected component. *i.e.* for randomly selected component k, the redundancy level is changed to l'_{i_k} such that $a'_i = [l_{i_1}, ..., l_{i_l}, ..., l'_{i_k}, ..., l_{i_n}]$.

■ *Component Selection.* The component selection architecture transformations are mapped into the genetic operators in NSGA-II as follows.

Crossover operator is implemented as the creation of new component selection solutions $s'_i, s'_j \in S$ from two parents $s_i = [o_{i_1}, o_{i_2}, ..., o_{i_n}]$ and $s_j = [o_{j_1}, o_{j_2}, ..., o_{j_n}]$ coming from existing population by recombining the component selections of tasks, *i.e.* for a random k: $s'_i = [o_{i_1}, ..., o_{i_{k-1}}, o_{j_k}, ..., o_{j_n}]$ and $s'_j = [o_{j_1}, ..., o_{j_{k-1}}, o_{i_k}, ..., o_{i_n}]$.

Mutation produces a new component selection solution s'_i from existing s_i by changing the selected component of randomly selected task, *i.e.* for randomly selected k: $s'_i = [o_{i_1}, ..., o'_{i_k}, ..., o_{i_n}]$ where the original is $s_i = [o_{i_1}, ..., o_{i_k}, ..., o_{i_n}]$ and o'_{i_k}

denotes a different selection option for task t_k.

Ant Colony Optimisation

All of the problem instances are executed with the version of Pareto Ant Colony Optimisation algorithm described in Chapter 6. Each problem instance is solved with ACO configurations ants per iteration $(n) = 30$, pheromone evaporation rate $(\rho) = 0.1$, transition rule $(q_{th}) = 0.4$, initial pheromone value $(\tau_0) = 1$, best of each attribute $(\Delta\tau_b) = 10$, second-best of each objective $(\Delta\tau_{sb})$ =5. The pheromone model is implemented as a vector of pheromone maps corresponding to each objective. The pheromone map and the ant's choices are implemented in different architecture design problems.

■ *Deployment.* The pheromone map representation in deployment architecture optimisation problems is implemented as a $n \times m$ matrix, where n denotes the number of components and m is the number of hosts. Each cell (τ_{ij}) in the matrix is filled with pheromone values corresponding to the allocation of c_i into host u_j. Construction of a candidate architecture is implemented as a series of steps that manifests the deployment of software components to hosts, where each step makes a decision on deploying a specific component.

■ *Redundancy Allocation.* In applying ACO to solving redundancy allocation problem instances, the pheromone model has to capture the effects of redundancy level decisions for each component. A $n \times m$ matrix is kept for each objective as the pheromone model, where n denotes the number of components and m is the maximum redundancy level of the problem instance. The solution construction of the model is mapped to iterative construction of a complete redundancy allocation for a problem instance, where each step of the walk represents allocating redundancy level l_i to the component c_i.

■ *Component Selection.* According to the formulation of the component selection problem, the number of options for tasks can be different from each other. Hence, the

pheromone model is defined in the vector form rather than the matrix formulation in the above two problems. For each objective, a vector is maintained with a length corresponding to the number of tasks (n). Individual elements of this vector are also vectors of the variable number of candidate components (m_i) for the task t_i. The solution construction is denoted by n steps, each step representing a selection of a component for task t_i which can have different number of options (m_i) in the steps. A complete path (of length n) across the pheromone model constitutes a component selection solution.

Simulated Annealing

As the third stochastic algorithm used in the experimental evaluation of the approach, the Pareto simulated annealing algorithm presented in Chapter 6 is applied to all the problem instances described before. The algorithm parameters are set with Active period (α) =0.75, initial temperature (T_0) = average normalised quality of all solutions / $ln(2)$, epoch length (L) =100 and cooling factor (β)= $-1 \cdot \sqrt[L*\alpha]{0.00001 \cdot T_0}$. An initial solution for each problem is randomly created and neighbourhood moves are repeatedly executed on the solution to generate the initial solution archive in burn-in. The neighbourhood move is implemented separately for each architecture design problem, similar to the mutation operator implementation described in the previous paragraph titled NSGA-II.

7.3.3 Results

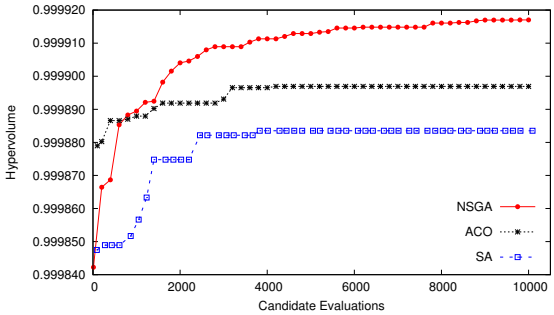
The generated problem instances have been specified in XML format, and a series of experiments were conducted using the ArcheOpterix tool. The results of the experiments are presented in two phases. Considering the evaluation criteria EC 1, EC 4 and EC 5, experimental results of problem instances of different architecture design problems (*i.e.* deployment, redundancy allocation and component selection) in all three optimisation algorithms (*i.e.* NSGA, ACO and SA) are presented. A detailed analysis is presented for the deployment problem instances in response to

EC 2, EC 3 and EC 6. The other problem instances produce similar results.

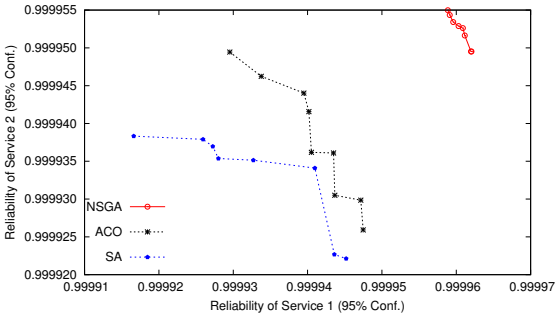
Iterative Provision of Robust-and-Reliable Solutions

Providing evidence on the SCOUT approach's ability to solve architecture optimisation problems of different types (EC 4) with heterogeneous nature of uncertainty (EC 1), results of three problem instances in different problem types are presented. Figure 7.5 illustrates the results of the robust optimisation using the three stochastic algorithms of one deployment problem instance (DEP_H32_C80) one redundancy allocation problem (RA_C40) and one component selection instance (CS_T20). As a measure of the quality of solutions produced in each iteration, the *hypervolume* of the front of non-dominated solutions is used. The *hypervolume indicator* is recommended by Zitzler *et al.* [423] for performance measurement in multi-objective optimisation problems. It indicates both the distance from a reference point in the objective space (normally, the point where all the objective values are at the best) and the spread of solutions within in the Pareto front. The indicator measures the hypervolume between the approximation set surface and a fixed point in the result space. For maximisation problems, the most intuitive reference point is zero in each dimension, which is the value used in reporting results of this experiments. The solutions produced by SCOUT approach were used to establish the hypervolume (area in two dimensions) between (0.0, 0.0) and the robust objective values of the non-dominated set.

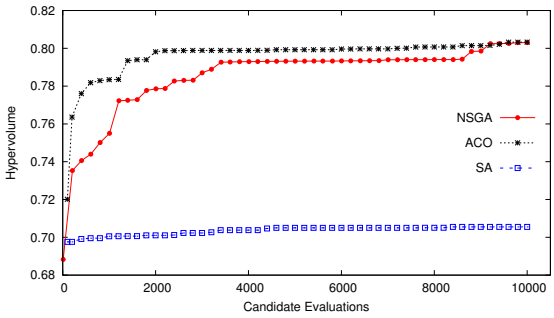
The graphs on the left side of Figure 7.5 present the hypervolume of the solutions produced by different algorithms in sequence of number of candidate evaluations. The figures show that the hypervolume values grow progressively in all problem types and all algorithms. This indicates improvement of the quality of solutions over the iterations illustrating stochastic algorithms' contribution in search for better solutions in each iteration *w.r.t.* robust quality goals (EC 5). The graphs show different performance characteristics of the stochastic algorithms in the problem settings. However, detailed performance analysis of stochastic algorithms or prox-



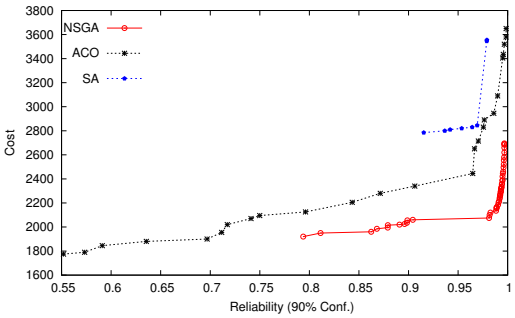
(a) Hypervolume growth of DEP_H32_C80



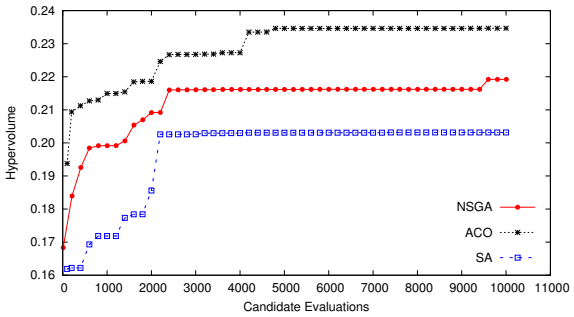
(b) Final Pareto-front of DEP_H32_C80



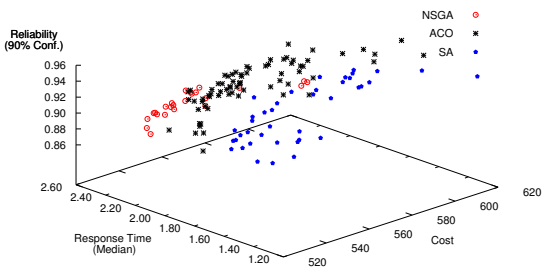
(c) Hypervolume growth of RA_C40



(d) Final Pareto-front of RA_C40



(e) Hypervolume growth of CS_T20



(f) Final Pareto-front of CS_T20

Figure 7.5: Optimisation results of the some problem instances

imity of the produced solutions to the true Pareto front has not been conducted and is considered out of the scope of this work. The final solutions produced after exploring 10,000 candidate architectures (candidate evaluations) are depicted in the graphs on the right. The results indicate that the SCOUT approach has been capable of capturing diverse problem characteristics described before, and producing non-dominated set of architecture solutions with improved quality for all the cases in all problem instances.

The results of the experiments with problem instances representing different architecture optimisation problems and different scales indicate that the SCOUT approach has been able to cater for their diverse modelling requirements (EC 4). The approach has been able to capture various uncertain input parameters (EC 1), different architecture transformation operators, different tolerance levels, different number of optimisation goals, various constraints as well as different quality evaluation models. The results suggest that the SCOUT framework has provided modelling support for these different problem types and characteristics and, has produced architecture candidates with improved quality in terms of robust quality attributes.

Robustness of the Produced Solutions

During the first experiments with deployment instances, 5^{th} percentiles of service reliability (95% pessimistic) is used as the robust goals to be optimised. To investigate the suitability of using these goals as a measure of the candidate architecture's tolerance for uncertainty (EC 2), the final solutions were further tested for large amount of possible variations of the uncertain parameters. The mean and variance of the two service reliability metrics are computed using repeated evaluation of reliability models for 10,000 input parameter samples. Similar to the robustness test conducted for the solutions of the automotive case study, the 95% confidence interval of the normal distribution which is obtained using the sample mean (Computed μ) and variance (Computed σ^2) can be compared with the robust reliability produced

from the SCOUT approach by the following relationship [299]:

$$\mu - \frac{1.96\sigma}{\sqrt{n}} > 95\% \text{ Conf. value}$$

This test has been conducted for all the final solutions and for each service reliability. Table 7.7 illustrates the results of the robustness test. The *robustness test* columns indicate whether the robust reliability estimate of produced solutions is less than the lower bound of standard interval. Having a $\surd$ for the all solutions indicates that the solutions obtained from the presented robust optimisation approach are capable of tolerating the uncertainty, and the 95% confidence reliability estimate can be used as a pessimistic robustness measure (EC 3).

Performance of the Dynamic Stopping Criterion

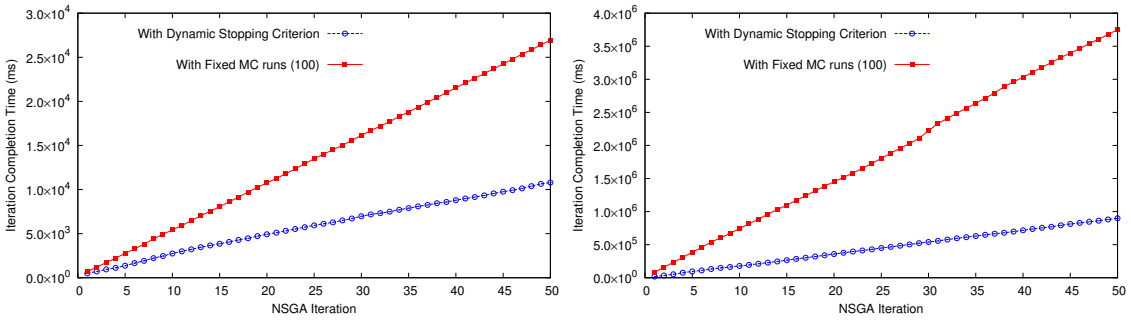
During the robust optimisation experiments, the new approach applies the statistical significance testing-based dynamic stopping criterion which was introduced in Chapter 5. Table 7.8 lists the final solutions produced by the robust optimisation approach for the case DEP_H16_C40. The robust reliability values have been obtained using MC simulation with the dynamic stopping criterion, and the corresponding columns contain the number of MC runs conducted until the stop criterion is reached. The second column of the table provides the details of the deployment of software components to the ECUs in respective solutions as an array of ECU ids to the length of components. The table shows that the different solutions require different number of MC samples to estimate the reliabilities accurately with respect to the same level of uncertainty in the input parameters. It can also be seen that certain service reliability evaluations have been completed in 19 runs, while the estimation of some solutions has required up to 149 evaluations. To verify the accuracy of the dynamic stopping criterion in this context (EC 2) further, the estimation produced by the robust optimisation is compared with empirical results obtained using large numbers of MC runs. Considering the sizes of the problems, 10,000 runs

	Reliability of Service 1				Reliability of Service 2			
Sol. Idx.	**Estimated 95% Conf.**	**Computed μ**	**Computed σ^2**	**Robustness Test**	**Estimated 95% Conf.**	**Computed μ**	**Computed σ^2**	**Robustness Test**
DEP_H8_C20								
1	0.999923	0.999956	$3.74 \cdot 10^{-10}$	√	0.999900	0.999931	$9.59 \cdot 10^{-10}$	√
2	0.999937	0.999952	$4.46 \cdot 10^{-10}$	√	0.999857	0.999918	$1.61 \cdot 10^{-9}$	√
3	0.999933	0.999951	$4.67 \cdot 10^{-10}$	√	0.999891	0.999919	$1.43 \cdot 10^{-9}$	√
4	0.999934	0.999956	$3.7 \cdot 10^{-10}$	√	0.999879	0.999930	$1.01 \cdot 10^{-9}$	√
DEP_H16_C40								
1	0.999966	0.999974	$1.3 \cdot 10^{-10}$	√	0.999849	0.999931	$2.04 \cdot 10^{-9}$	√
2	0.999962	0.999974	$1.23 \cdot 10^{-10}$	√	0.999881	0.999932	$2.02 \cdot 10^{-9}$	√
3	0.999901	0.999961	$6.15 \cdot 10^{-10}$	√	0.999926	0.999952	$6.58 \cdot 10^{-10}$	√
4	0.999942	0.999943	$2.4 \cdot 10^{-9}$	×	0.999909	0.999944	$8.27 \cdot 10^{-10}$	√
5	0.999905	0.999961	$6.03 \cdot 10^{-10}$	√	0.999913	0.999952	$6.76 \cdot 10^{-10}$	√
DEP_H32_C80								
1	0.999941	0.999948	$7.65 \cdot 10^{-11}$	√	0.999933	0.999943	$1.15 \cdot 10^{-10}$	√
2	0.999942	0.999949	$7.41 \cdot 10^{-11}$	√	0.999925	0.999944	$1.13 \cdot 10^{-10}$	√
3	0.999934	0.999949	$7.42 \cdot 10^{-11}$	√	0.999934	0.999944	$1.09 \cdot 10^{-10}$	√
DEP_H64_C160								
1	0.999871	0.999877	$1.95 \cdot 10^{-10}$	√	0.999826	0.999869	$5.37 \cdot 10^{-10}$	√
2	0.999861	0.999881	$1.82 \cdot 10^{-10}$	√	0.999871	0.999883	$1.99 \cdot 10^{-10}$	√
3	0.999848	0.999881	$1.83 \cdot 10^{-10}$	√	0.999871	0.999883	$2.03 \cdot 10^{-10}$	√
DEP_H128_C320								
1	0.999889	0.999911	$1.38 \cdot 10^{-10}$	√	0.999911	0.999914	$9.62 \cdot 10^{-11}$	√
2	0.999898	0.999911	$1.37 \cdot 10^{-10}$	√	0.999906	0.999916	$8.63 \cdot 10^{-11}$	√
3	0.999893	0.999911	$1.38 \cdot 10^{-10}$	√	0.999910	0.999916	$8.52 \cdot 10^{-11}$	√
4	0.999916	0.999922	$6.05 \cdot 10^{-11}$	√	0.999893	0.999913	$1.29 \cdot 10^{-10}$	√
5	0.999916	0.999922	$5.81 \cdot 10^{-11}$	√	0.999883	0.999912	$1.31 \cdot 10^{-10}$	√
6	0.999898	0.999914	$1.07 \cdot 10^{-10}$	√	0.999907	0.999917	$8.35 \cdot 10^{-11}$	√
7	0.999915	0.999923	$5.6 \cdot 10^{-11}$	√	0.999901	0.999912	$1.35 \cdot 10^{-10}$	√

Table 7.7: Robustness test results of the solutions produced in optimisation

is set as a considerably large number and the relative error of actual 95% confidence estimate (a^*) and 95% confidence estimate given at the reach of stop criterion ($\hat{a}$) is computed. Table 7.9 presents the Monte Carlo accuracy test results for all solution for 5 cases. All the *relative error* values in the table having lower error than threshold relative error (w_r) configuration given in the experiments confirms that the dynamic stopping criterion is accurate.

To illustrate the computation gain produced by the use of the dynamic stopping criterion, the same set of experiments has been conducted again with identical configurations except the MC simulation. Instead of using the dynamic stopping criterion, fixed number of 100 MC runs ware conducted for each robust reliability evaluation. Figure 7.6 illustrates the results of the comparison for smallest and largest deployment problem instances. The graphs clearly show that time taken for iterations when dynamic stop criterion is significantly lower than the fixed runs. It should be noted that the fixed runs have only set to 100 where as dynamic stopping criterion can go for further runs whenever required (*e.g.* row 1 in Table 7.8). Results of these experiments suggest that the dynamic stopping criterion provides significant computational gain for the robust optimisation, while maintaing the defined level of accuracy in reliability metric estimation.



(a) Case DEP_H8_C20 (b) Case DEP_H128_C320

Figure 7.6: Computational efficiency of dynamic stopping criterion in comparison to fixed MC runs

ID	Deployment	Service 1		Service 2	
		Reliability (95% Conf.)	MC Runs	Reliability (95% Conf.)	MC Runs
1	[1, 1, 9, 14, 11, 0, 1, 3, 9, 3, 4, 3, 0, 2, 1, 13, 11, 2, 1, 5, 13, 15, 4, 5, 2, 13, 13, 13, 3, 13, 12, 0, 8, 12, 13, 14, 0, 7, 9, 1]	0.999966	19	0.999849	149
2	[1, 1, 9, 14, 2, 0, 1, 3, 9, 3, 4, 3, 0, 2, 1, 13, 11, 2, 1, 5, 13, 15, 4, 5, 2, 13, 13, 13, 3, 13, 12, 0, 8, 12, 13, 14, 0, 7, 9, 1]	0.999962	29	0.999881	29
3	[1, 1, 9, 14, 2, 0, 1, 3, 9, 3, 4, 3, 0, 2, 1, 13, 11, 2, 1, 5, 13, 15, 4, 5, 2, 13, 13, 13, 3, 13, 12, 0, 8, 12, 13, 12, 11, 14, 9, 1]	0.999901	19	0.999926	49
4	[1, 1, 9, 14, 2, 0, 1, 3, 9, 3, 4, 3, 0, 2, 1, 13, 5, 2, 1, 5, 13, 15, 4, 5, 2, 13, 13, 13, 3, 13, 12, 0, 11, 12, 15, 12, 0, 7, 9, 1]	0.999942	19	0.999909	92
5	[1, 1, 9, 14, 11, 0, 1, 3, 9, 3, 4, 3, 0, 2, 1, 13, 11, 2, 1, 5, 13, 15, 4, 5, 2, 13, 13, 13, 3, 13, 12, 0, 8, 12, 13, 12, 11, 14, 9, 1]	0.999905	52	0.999913	29

Table 7.8: Non-dominated solutions produced for the case DEP_H16_C40

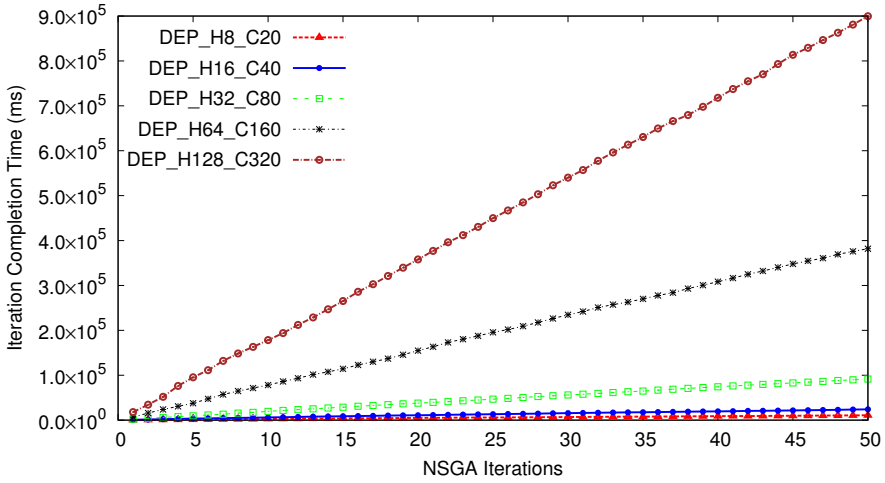
Scalability of the Approach

One aim of this experiment setup is to investigate the scalability of the robust optimisation approach presented in the thesis. In addition to the diversity of problem characteristics of generated instances, they also represent different scales of architecture design problems (EC 6). The number of ECUs, software components, tasks *etc.* have been deliberately selected to represent small-scale academic case study to large real world setting of 128 ECUs and 320 software components. The time complexity of the NSGA-II algorithm with the different problem sizes is illustrated in Figure 7.7. Figure 7.7a shows that, for all cases, the total algorithm execution time after each iteration grows linearly, indicating a fixed iteration time to a case. Similar results can be obtained with the other two stochastic algorithms and other two problem types. On the scalability aspect, Figure 7.7b indicates the time needed for fixed number of iterations considering the variable inner-loop of Monte Carlo simulation, grows much faster (near exponentially) with respect to the problem size. It should be noted that this growth has a strong relationship with the quality

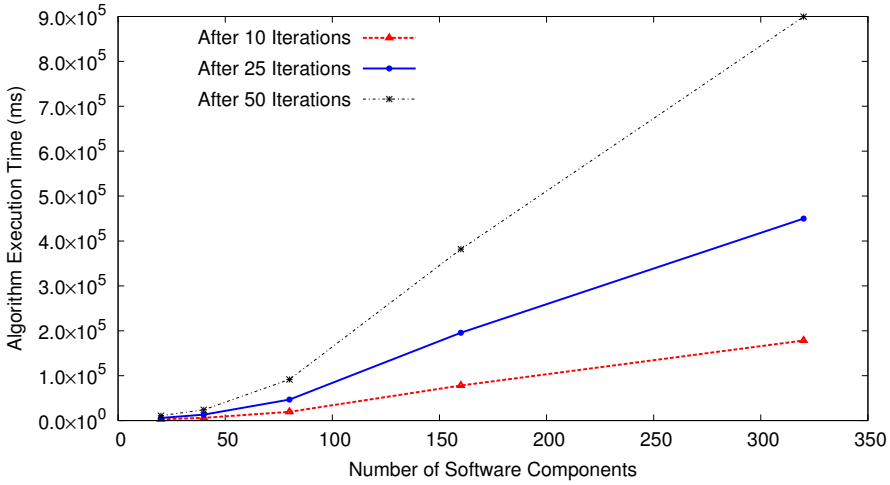
	95% Conf. Reliability of Service 1				**95% Conf. Reliability of Service 2**			
Sol. Idx.	**Reported.** e_{dyn}	**Actual** e_{act}	w_r $(\frac{\lvert e_{act}-e_{dyn}\rvert}{e_{act}})^2$	$w_r < w_{rTh}$	**Reported.** e_{dyn}	**Actual** e_{act}	w_r $(\frac{\lvert e_{act}-e_{dyn}\rvert}{e_{act}})^2$	$w_r < w_{rTh}$
DEP_H8_C20								
1	0.999923	0.99992	$1.04 \cdot 10^{-11}$	√	0.9999	0.999875	$6.1 \cdot 10^{-10}$	√
2	0.999937	0.999913	$5.35 \cdot 10^{-10}$	√	0.999857	0.999844	$1.66 \cdot 10^{-10}$	√
3	0.999933	0.999911	$5.01 \cdot 10^{-10}$	√	0.999891	0.99985	$1.71 \cdot 10^{-9}$	√
4	0.999934	0.999921	$1.76 \cdot 10^{-10}$	√	0.999879	0.999873	$3.73 \cdot 10^{-11}$	√
DEP_H16_C40								
1	0.999966	0.999952	$1.94 \cdot 10^{-10}$	√	0.999849	0.999841	$7.27 \cdot 10^{-11}$	√
2	0.999962	0.999953	$8.12 \cdot 10^{-11}$	√	0.999881	0.999843	$1.49 \cdot 10^{-9}$	√
3	0.999901	0.999909	$7.76 \cdot 10^{-11}$	√	0.999926	0.999904	$5.25 \cdot 10^{-10}$	√
4	0.999942	0.999841	$1.03 \cdot 10^{-8}$	√	0.999909	0.999891	$3.33 \cdot 10^{-10}$	√
5	0.999905	0.99991	$2.34 \cdot 10^{-11}$	√	0.999913	0.999903	$9.81 \cdot 10^{-11}$	√
H32_C80								
1	0.999941	0.999932	$8.68 \cdot 10^{-11}$	√	0.999933	0.999923	$1.05 \cdot 10^{-10}$	√
2	0.999942	0.999933	$9.03 \cdot 10^{-11}$	√	0.999925	0.999924	$1.28 \cdot 10^{-12}$	√
3	0.999934	0.999934	$3.53 \cdot 10^{-13}$	√	0.999934	0.999925	$7.95 \cdot 10^{-11}$	√
H64_C160								
1	0.999871	0.999853	$3.44 \cdot 10^{-10}$	√	0.999826	0.999824	$3.39 \cdot 10^{-12}$	√
2	0.999861	0.999857	$1.68 \cdot 10^{-11}$	√	0.999871	0.999858	$1.7 \cdot 10^{-10}$	√
3	0.999848	0.999857	$8.68 \cdot 10^{-11}$	√	0.999871	0.999857	$2.05 \cdot 10^{-10}$	√
DEP_H128_C320								
1	0.999889	0.99989	$9.65 \cdot 10^{-13}$	√	0.999911	0.999897	$1.97 \cdot 10^{-10}$	√
2	0.999898	0.99989	$6.76 \cdot 10^{-11}$	√	0.999906	0.9999	$3.85 \cdot 10^{-11}$	√
3	0.999893	0.99989	$7.2 \cdot 10^{-12}$	√	0.99991	0.9999	$9.28 \cdot 10^{-11}$	√
4	0.999916	0.999909	$4.54 \cdot 10^{-11}$	√	0.999893	0.999893	$2.23 \cdot 10^{-13}$	√
5	0.999916	0.999909	$4.67 \cdot 10^{-11}$	√	0.999883	0.999892	$7.98 \cdot 10^{-11}$	√
6	0.999898	0.999895	$5.12 \cdot 10^{-12}$	√	0.999907	0.999901	$4.0 \cdot 10^{-11}$	√
7	0.999915	0.99991	$1.88 \cdot 10^{-11}$	√	0.999901	0.999892	$7.71 \cdot 10^{-11}$	√

Table 7.9: Accuracy of estimates produced using dynamic stopping criterion

evaluation functions as well. When the problem gets bigger, complex quality evaluation functions like Markov Chain-based reliability evaluations, require exponentially growing computation time.



(a) Time complexity of different problem instances



(b) Algorithm execution time at fixed iteration milestones

Figure 7.7: Scalability of robust optimisation approach

Diversity of Tolerance Levels

Different types of architecture design problems in the initial set of experiments have used various tolerance levels for uncertain objective functions. In the deployment instances, the level of desired tolerance against uncertain input parameters has been

set to 95% for the two reliability attributes, requiring very high level of robustness. The robust optimisation approach supports different tolerance levels demanded in diverse application domains as well as for different quality metrics (EC 3). To further illustrate this capability, a set of experiments have been re-conducted for deployment instances, with different tolerance level settings for the two reliability attributes. In the controlled experiments, the confidence value is considered the only variable, and all the other factors were maintained fixed as described in Section 7.3.2. Tables 7.10, 7.11 and 7.12 illustrate the results for the case H32_C80, where deployment details column lists the allocated hosts in sequence of component indices. One representative case (median sized) is presented as all the others share the same pattern.

From the results presented in Tables 7.10, 7.11 and 7.12, it is evident that the number of solutions produced for different tolerance settings of the goals are not the same, even all the other factors of the experiments were fixed. Careful observation of deployment details of individual solutions shows that that the solutions corresponding to each tolerance level settings are not necessarily the same. This observation further emphasises the complexity of the design problem in focus, and it is often impractical to identify solutions manually. The experimental results suggest that the robust optimisation approach introduced in the thesis is able to cater for diverse levels of tolerance requirements against uncertainty, and the overall approach has been able to produce non-dominated set of solutions for different settings with non-obvious architectural differences.

7.3.4 Discussion of Results

Complementary to the illustration of the approach on the automotive ABS/ACC case study and validations of different elements presented in earlier chapters of the thesis, this chapter presents experiments conducted using the complete robust optimisation framework. The experiments were designed to cover different aspects of the robust optimisation as formulated in the research questions given in Chapter 4.

Id	Deployment Details	Rel. of Service 1	Rel. of Service 2
1	[29, 0, 11, 2, 0, 15, 18, 21, 4, 0, 7, 23, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 21, 25, 29, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 30, 28, 5, 2, 28, 21, 13, 9, 8, 19, 23, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 24, 28, 18, 2, 13, 24]	0.999955	0.999958
2	[29, 19, 3, 2, 0, 15, 18, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 29, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 14, 0, 21, 17, 14, 7, 5, 11, 8, 1, 31, 21, 28, 14, 31, 14, 24, 21, 5, 3, 19, 19, 3, 12, 31, 16, 29, 28, 8, 2, 11, 4]	0.999961	0.999954
3	[29, 15, 3, 12, 0, 4, 3, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 31, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 14, 0, 21, 17, 14, 7, 5, 11, 8, 1, 26, 21, 28, 14, 31, 14, 24, 21, 2, 30, 25, 19, 27, 12, 31, 2, 24, 28, 18, 2, 13, 24]	0.999957	0.999956
4	[29, 15, 3, 12, 0, 4, 3, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 31, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 28, 21, 13, 9, 8, 19, 21, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 24, 28, 18, 2, 13, 24]	0.999956	0.999956
5	[29, 15, 3, 12, 0, 19, 3, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 29, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 14, 0, 21, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 29, 28, 8, 2, 11, 4]	0.999953	0.999958
6	[29, 15, 3, 12, 0, 4, 3, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 31, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 14, 0, 21, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 24, 28, 18, 2, 13, 24]	0.999957	0.999956
7	[29, 15, 3, 12, 0, 4, 18, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 29, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 8, 19, 21, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 24, 28, 18, 2, 13, 24]	0.999959	0.999955
8	[29, 15, 3, 12, 0, 19, 3, 21, 15, 0, 7, 16, 19, 16, 19, 20, 22, 3, 3, 7, 2, 25, 4, 22, 2, 3, 23, 25, 31, 0, 9, 14, 19, 0, 12, 20, 31, 18, 12, 7, 6, 7, 23, 2, 8, 21, 1, 17, 14, 0, 21, 17, 3, 7, 5, 11, 17, 20, 12, 2, 28, 14, 21, 1, 24, 21, 2, 30, 25, 19, 27, 25, 31, 2, 29, 28, 8, 2, 11, 4]	0.999959	0.999955

Table 7.10: Solutions produced for DEP_H32_C80 with 25% tolerance setting

Id	Deployment Details	Rel. of Service 1	Rel. of Service 2
1	[15, 19, 27, 31, 29, 0, 14, 19, 0, 29, 20, 21, 26, 0, 4, 11, 1, 28, 14, 18, 2, 21, 23, 13, 0, 10, 4, 2, 21, 2, 18, 1, 19, 23, 2, 13, 25, 22, 2, 22, 7, 22, 19, 24, 18, 31, 7, 30, 22, 29, 4, 28, 8, 6, 12, 14, 11, 10, 5, 4, 17, 1, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 1, 27, 16, 18, 31]	0.999953	0.999944
2	[15, 19, 27, 31, 29, 21, 14, 0, 0, 29, 6, 4, 24, 0, 19, 11, 1, 28, 14, 18, 2, 21, 23, 13, 2, 10, 4, 2, 21, 2, 18, 1, 19, 23, 2, 13, 25, 22, 2, 22, 7, 22, 19, 24, 18, 31, 7, 30, 22, 29, 4, 28, 8, 6, 12, 14, 11, 10, 5, 4, 17, 1, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 1, 27, 16, 18, 31]	0.999948	0.999952
3	[15, 19, 27, 31, 29, 0, 14, 0, 0, 29, 6, 21, 24, 0, 19, 11, 1, 28, 14, 18, 2, 21, 23, 13, 2, 10, 4, 2, 21, 2, 18, 1, 19, 23, 2, 13, 25, 22, 2, 22, 7, 22, 19, 24, 18, 31, 7, 30, 22, 29, 4, 28, 8, 6, 12, 14, 11, 10, 5, 4, 17, 1, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 1, 27, 16, 18, 31]	0.999951	0.999949
4	[15, 19, 27, 31, 29, 2, 14, 19, 0, 29, 20, 21, 26, 0, 4, 20, 6, 6, 6, 1, 0, 21, 0, 6, 19, 28, 24, 2, 12, 2, 18, 18, 19, 23, 2, 13, 25, 22, 2, 22, 7, 22, 19, 24, 18, 31, 7, 30, 22, 29, 4, 20, 8, 14, 12, 14, 11, 10, 5, 4, 1, 13, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 30, 27, 16, 1, 31]	0.999951	0.999946
5	[15, 19, 27, 31, 29, 21, 14, 0, 0, 29, 6, 4, 24, 0, 19, 11, 1, 28, 14, 18, 2, 19, 23, 13, 2, 10, 4, 2, 21, 2, 18, 1, 16, 23, 2, 13, 25, 22, 2, 22, 7, 22, 0, 24, 18, 31, 7, 30, 22, 29, 4, 28, 8, 6, 12, 14, 11, 10, 5, 4, 17, 1, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 1, 27, 5, 18, 31]	0.999948	0.99995
6	[15, 21, 27, 31, 29, 0, 14, 19, 0, 29, 20, 21, 24, 0, 19, 11, 1, 28, 14, 18, 2, 21, 23, 13, 19, 28, 24, 2, 12, 2, 3, 1, 19, 23, 2, 13, 25, 22, 2, 22, 7, 22, 19, 24, 18, 31, 7, 30, 22, 29, 4, 20, 8, 6, 12, 14, 11, 10, 5, 4, 17, 1, 24, 27, 4, 25, 12, 11, 23, 16, 14, 0, 5, 12, 16, 1, 27, 16, 18, 31]	0.999952	0.999946

Table 7.11: Solutions produced for DEP_H32_C80 with 50% tolerance setting

Id	Deployment Details	Rel. of Service 1	Rel. of Service 2
1	[4, 5, 3, 12, 0, 16, 8, 19, 15, 19, 9, 0, 2, 2, 19, 20, 13, 3, 9, 7, 5, 12, 12, 22, 2, 17, 2, 2, 24, 2, 1, 28, 31, 21, 21, 7, 0, 14, 23, 8, 14, 20, 0, 31, 30, 21, 27, 28, 22, 2, 4, 9, 14, 22, 25, 18, 1, 30, 26, 16, 7, 3, 0, 14, 15, 24, 21, 8, 16, 19, 13, 31, 4, 29, 31, 28, 7, 4, 14, 21]	0.999947	0.999948
2	[4, 21, 3, 12, 0, 16, 8, 19, 15, 19, 9, 0, 2, 2, 19, 20, 13, 3, 9, 7, 5, 12, 12, 22, 2, 17, 2, 2, 24, 2, 1, 28, 31, 21, 21, 7, 5, 14, 23, 8, 14, 20, 0, 31, 30, 21, 27, 28, 22, 2, 4, 9, 14, 22, 25, 18, 1, 30, 26, 16, 7, 3, 0, 14, 15, 24, 21, 8, 16, 19, 13, 31, 4, 29, 31, 28, 7, 4, 14, 21]	0.999941	0.99995
3	[4, 5, 3, 12, 0, 21, 8, 19, 15, 19, 9, 0, 2, 2, 19, 20, 13, 3, 9, 7, 16, 12, 12, 22, 2, 17, 2, 2, 24, 2, 1, 28, 31, 21, 21, 7, 0, 14, 23, 8, 14, 20, 0, 31, 30, 21, 27, 28, 22, 2, 4, 9, 14, 22, 25, 18, 1, 30, 26, 16, 7, 3, 0, 14, 15, 24, 21, 8, 16, 19, 13, 31, 4, 29, 31, 28, 7, 4, 14, 21]	0.999949	0.999946
4	[4, 5, 3, 12, 0, 21, 8, 19, 15, 19, 9, 0, 2, 2, 19, 20, 13, 3, 9, 7, 16, 12, 12, 22, 2, 17, 2, 2, 24, 2, 1, 28, 31, 21, 21, 7, 0, 14, 23, 8, 14, 20, 0, 31, 30, 21, 27, 28, 22, 2, 4, 9, 14, 22, 25, 18, 1, 30, 26, 16, 7, 3, 0, 18, 15, 24, 21, 8, 16, 19, 13, 31, 4, 29, 31, 28, 7, 4, 14, 21]	0.999947	0.999947

Table 7.12: Solutions produced for DEP_H32_C80 with 75% tolerance setting

The experiments over random problem instances illustrate the capability of the framework to cater for different architecture design problems. The relevant transformation operators for specific architecture design problem have been modelled and the framework was used to search for robust and (near-)optimal architecture candidates. The results of the first part of the experiments clearly showed that the approach is able to capture the different problem instances of various characteristics, successfully evaluates the service reliability attributes, and iteratively produce a set of non-dominated solutions with improved quality. Prevalent growth of hypervolume over the optimisation iterations indicates a promising contribution from the stochastic algorithm, and lead towards solutions which are robust as well as better in terms of quality attributes. The application of different optimisation algorithms over all problem instances shows the capability of the framework in adopting different solvers to the search of better solutions in the presences of uncertainty. The hypervolume growth results of different problem instances indicate that certain algorithms perform better when solving specific problem instances. ACO and

NSGA-II perform competitively in many instances while the selected variant of simulated annealing algorithm's performance is comparatively lower. One key reason behind this observation would be the difference in handling the search space where SA's search technique is based on neighbourhood moves on a single solution while the other two algorithms use a collection of solutions. However, the focus of these experiments with different algorithms is to show the approach's capability to use various optimisation strategies, and the detailed analysis of the performance of different algorithms over the different problem characteristics is considered out of the scope of this work. Overall, the first group of experiments validate the approach's capability to cater for assorted parameter and problem characteristics as well as the use of different stochastic algorithms.

The solutions produced by the new approach were tested for robustness. By comparing the estimated 95% confidence service reliability values of the produced solutions against standard lower bound of very large number of variation samples, the experiments illustrated the suitability of 95% percentile as a pessimistic robustness indicator.

As the robust optimisation approach comprises a novel dynamic stopping criterion for probabilistic quality evaluation in the presence of uncertainty, the next set of experiments were dedicated to investigating its suitability, accuracy and computational benefit to the approach. The analysis of solutions produced from the approach showed that the number of Monte Carlo simulations conducted for different solutions and for different service reliability evaluations are not the same. A huge variability of the number of Monte Carlo runs could be observed. The results indicate that the dynamic stopping mechanism automatically controls the sampling process according to a given accuracy goal. To further verify the accuracy of the stopping mechanism, the estimated service reliability values produced by the dynamic-stopping Monte Carlo are compared with (near-) actual ones obtained from a very large number of samples. The accuracy test for all the solutions and for all the cases satisfied the threshold error tolerance, which is an input of the dynamic

stopping criterion. From these results, it can be concluded that the novel dynamic stopping criterion is accurate and has been successfully integrated into the robust optimisation approach. In addition, its performance has also been compared against experiments with fixed Monte Carlo runs. Note that the fixed number of MC runs has only been set to 100 while the dynamic stopping criterion had reported certain samples over 150 runs. However, the results clearly show that, the dynamic stopping criterion provides significant computational gain over the fixed Monte Carlo configuration while maintaining a configured level of accuracy, for all the case.

The next part of the validation was to investigate the support for diverse tolerance goals in the optimisation. Having fixed all the other settings, the experiments were conducted with different tolerance goals for the service reliability. For the same problem instance, the robust optimisation approach has produced different results for different desired tolerance levels. It is to be noted that the degree of uncertainty in the input parameters was the same for all, only the tolerance levels of the optimisation goals are different. By mere fact of the solutions being different, it illustrates that the approach is capable of search solutions that are specifically tailored for the desired level of robustness. When higher tolerance is required, the produced solutions are less sensitive to the uncertain input parameters where as when moderate tolerance is sufficient, the approach produces sensitive, but suitable enough trade-offs.

Threats to Validity

The generation of experiments has been aimed at capturing the diversity of characteristics of the architecture optimisation problem and uncertainty in the input parameter space. Instead of generating an instance for every possible combination of problem dimensions, the different aspects to be addressed have been abstracted, and instances were randomly generated from that abstract formulation. As explained in the experiment setup, the input parameters of the problem instances were generated randomly, but securing the realistic ranges for respective parameter. To represent diverse source and characteristics of uncertainty associated with a subset of pa-

rameters, certain generated parameters are specified as heterogeneous probability distributions. Problem instances were generated from different architecture design problem types in order to illustrate the approach's applicability over diverse design problems. In addition to extensive experiments on deployment optimisation problems, a set of instances have been generated to represent other common architecture design problems. Redundancy allocation and component selection are important and widely used design problems in the context of component-based software engineering. In the deployment problem instance, the objectives to be optimised were set to service reliabilities, where a service is considered a randomly generated transition matrix over the software components. The cost-reliability trade-off under uncertainty is analysed with a series of redundancy allocation problem instances and a set of component selection problem instances which were created with the goal of a trade-off among cost, reliability and response time. When generating the instances, due measures were taken to ensure that the architectural assumptions and constraints are satisfied. By allowing randomness over defined abstract characteristics, the experiment design relaxes the assumptions from an instance-specific setting to much broader aspect.

The experiments have been designed with the intention to validate the framework's capability in capturing heterogeneous and diverse characteristics of uncertainty. However, the estimation uncertainties in the practical architecture design problems may have different characteristics in comparison to the ones which were included in the experiments. As a precautionary measure, a mechanism has been used to create random distributions, even with random parameters to specify uncertainty, and due care has been made to ensure the ranges of the distributions. By selecting the distributions at random and including the random-discrete distributions, the experiments were encouraged to cover the scope of practical problem instances.

For accuracy testing of the solutions produced by the SCOUT approach, we have selected 10,000 Monte Carlo samples as a *significantly large* number which can

resemble the actual characteristics of uncertainty in the quality metrics (*e.g.* service reliability). It could be argued that the comparisons depend on the sample size, and the accuracy may change if even larger sample sizes are used. However, it could be seen that the robust reliability values with different tolerance levels are produced by the dynamic stopping criterion within sample ranges of 20-200. Furthermore, the variation of the percentile estimates have been individually observed even conducting various number of samples (by observing graphs similar to Figure 5.7c in Chapter 5). Considering the experience with the experiments on various problem sizes, 10,000 samples is a large enough sample to represent actual distributions.

Another threat to validity of the results is that the applicability of the contribution may be limited to problems of specific size or type. To reduce the risk regarding this threat, a series of problem instances were created with a variety of sizes. As opposed to manually setting specific problem characteristics, the common model of the problems has been abstracted out and instances are parametrically generated as presented before. By doing that, the experiments set themselves apart from an instance-specific setting to much broader applicability.

The validity of the presented experiments can be questioned on the grounds that the applicability to architecture design problems other than deployment, redundancy allocation and component selection. Even though the framework has been able to cater for the specific problem types and instances considered in the experiments, there is a threat that the framework may not be applicable to other problems. However, the framework is inherently generic and we have already compared the underlying model with prevailing methods in Chapter 6. Despite the validity to other problems may require experiments on specific problem types, we suggest that the SCOUT framework can be used for other architecture optimisation problems in component based software systems.

The experimental results presented in the chapter has a considerable dependency on the implementation. It is possible that implementation is erroneous or bias towards favourable results. However, in several occasions, we have consulted

experienced programmers and followed regular code-review sessions. Due measures were taken to cross check the implementation and the conceptual design. By taking these precautions, we have minimised the possibility of having erroneous or misleading results in the experiments.

Chapter 8

Conclusions

This thesis has focused on the issues arising from uncertainty in design-time parameter estimates that are used in architecture-based quality evaluation and optimisation of software-intensive systems. The baseline of the research has been set with a background study on software architecture, architecture-based quality evaluation and methods on design-time architecture optimisation. The work has specifically focused on reliability as a probabilistic quality attribute which is evaluated from the software architecture descriptions, and investigated the impact of design-time parameter estimates on the architectural decisions that are made based on them. The research goal was set to finding a (near-)optimal set of architectures in terms of a set of probabilistic quality attributes with a confidence guarantee against the heterogeneous uncertainties in design-time parameter estimates (type A uncertainties [55]). Specific research questions on capturing the uncertainty, propagation through the probabilistic quality model construction, treatment in architecture-based reliability evaluation and architecture optimisation have been emphasised with an analysis of existing work and experiments on industrial case study.

Contributions

The work presented in this thesis has made a contribution to reducing the impact of inaccurate design-time estimates for parameters, such as hardware and software

failure rates, call probabilities of different services and usage profile in an architecture optimisation process. A novel robust optimisation approach has been presented that extends the current practice of using point estimates for the model parameters in architecture optimisation by allowing to include uncertain information available at the early design phase, and combine with optimisation techniques to find better architectures that can tolerate uncertainty. In summary, the work presented in this thesis contains following novel contributions.

- The conventional parameter specification in the architecture element specification has been extended to incorporate heterogeneous characteristics of uncertain information related to parameter estimation into the parameter specification of the architecture optimisation problem. A generalised probabilistic specification has been presented which can capture both the extent and the characteristics of uncertain parameters. The conventional assumptions on input parameter distributions have been relaxed with the support to include heterogeneous variability features of the parameters estimated in software architecture design.
- An automated and scalable model evaluation technique has been introduced which can be used to evaluate complex probabilistic quality models in the presence of uncertain model parameters. A Monte Carlo (MC) simulation technique has been adapted to uncertainty analysis to cater for probabilistic quality evaluation models which are hard to solve analytically for variable input parameters. A novel dynamic stopping criterion has been introduced that automatically finds the trade-off between time complexity and accuracy of MC simulation. Principles of sequential statistical testing have been combined with continuous accuracy monitoring mechanism in the MC simulation.
- A robust optimisation framework has been devised which can be used to model and optimise various architecture decisions involved in embedded systems design. The framework contains a model that integrates architectural elements, quality objectives, design constraints, probabilistic quality evaluation models and optimisation algorithms. The support has been provided to integrate various stochastic

algorithms into the framework for evaluation-expensive, multi-objective optimisation problems. The adaptation of three algorithms was presented, namely Non-dominated Sorting Genetic Algorithm-II, Ant Colony Optimisation and Simulated Annealing. Supporting the different application domains and requirements, a method has been presented for optimising architectures with a desired level of tolerance to uncertainty in the parameter estimation.

The novel robust optimisation approach has been implemented and validated with an illustrative case study from the automotive industry, as well as with a series of experiments with generated problem instances. The data obtained from these experiments have shown that the approach leads to robust designs, while also explaining the accuracy and scalability of the approach.

Goal Achievement and General Applicability of the Research

As a solution to the implications arising from the uncertainty associated with design-time parameter estimates, this work proposed to include uncertain characteristics into the specification as an extension to the current practice of providing them as point estimates. A probabilistic specification approach has been taken, relaxing the conventional assumptions on the distributions. The proposed approach has given the flexibility to use any probability distribution and in any combination to characterise uncertainty associated with each parameter. With the generalisation of the distribution and the ability to include discrete-interval distributions, the extension provided in this approach also subsumes the powers of current complementary techniques such as mean-variance and fuzzy uncertainty specification techniques. A set of guidelines have been given on using diverse uncertain information associated with each parameter to a formal specification as a distribution function. Due to this generic setting, our probabilistic approach to capturing uncertainties is theoretically capable of catering for any stationary variation. The scope of the thesis limits to the type-A uncertainties (model parameters) in architecture-based reliability models in the context of embedded systems design. The validity has been justified by spec-

ifying possible variations in an industrial case study as well as random variations in a series of generated problem instances. The empirical evidence and theoretical foundation presented in the thesis suggest that the SCOUT approach can be used to specify the uncertain characteristics associated with the parameters in practical problems within the scope of the thesis.

The architecture-based quality evaluation aspect of the research problem has been addressed considering the heterogeneous characteristics of uncertainty in the input parameter specification. Due to the use of non-linear and mathematically complex models such as DTMCs in architecture-based quantification of probabilistic properties, analytical methods used to propagate irregular variations through the models are less applicable. A Monte Carlo simulation technique is proposed in this approach in which the probabilistic model evaluation behaves as a black-box during the sampling. Even with the limited scope of reliability properties as the quality attributes, it has been shown that the variation of probabilistic properties under uncertain inputs does not follow a normal distribution. Hence, the SCOUT approach has adopted a non-parametric estimation method which can cater for probabilistic property estimation without prior knowledge of the distribution that the property may exhibit under uncertain inputs. Amongst the other descriptive figures, the author suggests that percentiles can be used as flexible indicators that can cover both pessimistic, optimistic and moderate ends of the property *w.r.t.* to cumulative effect of all the uncertainty in input parameter estimates. This extends the applicability of the approach to quantifying requirements in various application domains, including safety-critical systems as well as non-critical applications where moderate estimates are sufficient. The new method is further equipped with a dynamic stopping criterion which can control the number of Monte Carlo simulation automatically. With that, the approach has relaxed the assumptions on specific quality evaluation model or input characteristics, and brought the approach to a generic setting. It has been shown that the approach is able to cater for the industrial case study as well as the series of instances generated from a generic problem setting. Experiments with

all the problem instances have shown that the dynamic stopping criterion finds the appropriate threshold automatically and is able to provide accurate results with a given confidence. This further suggests that the architecture-based quality evaluation and dynamic stopping criterion can be applied to a variety of problem instances considering their various characteristics and sources of uncertainty.

In addressing the optimisation dimension of the research problem, this work has developed a framework for robust architecture optimisation which considers uncertainties in the input parameters. The requirements and different dimensions of a prospective framework were first analysed, and a high-level model has been devised based on a common understanding extracted from prevalent architecture optimisation approaches. The prior contributions on specification, evaluation and automated quantification of uncertainty-related aspects have been integrated into the high-level model. Resting on the fact that stochastic optimisation algorithms have shown consistent success in solving NP-hard and combinatorial design space exploration problems in software engineering, the framework was developed with the support for different stochastic algorithms. The integration of three widespread stochastic optimisation algorithms into the devised model has been described, leading to a high-level framework for architecture optimisation under uncertainty. During the development of the framework, a derivative modelling approach has been taken with a focus on justification in each step. The abstract model presented in the ISO 42010 international standard has been used as the starting point in systematically devising a high-level model to capture architecture optimisation related aspects. The validity of the model was also justified by bringing it into the context of related modelling approaches that address specific aspects in the domain. The new model's ability to cater for the requirements in those modelling frameworks is discussed in order to pursue the uncertainty integration in the approach as a generic extension to the current methods. The applicability of the framework to a broader setting and its contribution to addressing the overall research problem has been justified with a detailed comparison with prevailing approaches. Furthermore, the approach was fully

implemented, and the capabilities have been illustrated by applying to an industrial case study and a variety of problem instances. Hence, the results indicate that the robust optimisation framework is able to cater for diversity of architecture design and optimisation problems that are in the scope of the thesis.

Future Work

With the probabilistic input specification scheme introduced in this work, a Monte Carlo simulation is used to estimate the quality of candidates considering the uncertainties. This requires probabilistic models to be re-evaluated for each MC run. As described in Chapter 2, the probabilistic model evaluation is a computationally expensive process. This has been further explained in the scalability analysis experiments given in Chapter 5 and Chapter 7. However, in many practical situations, not all parameters are subject to uncertainty and specified as probability distributions. On the other hand, the same quality evaluation model has to be re-evaluated with individual parameter samples in each MC run. Considering these characteristics of the problem, the performance of the Monte Carlo simulation can be further improved. With this motivation, we are currently working on efficient algorithms to probabilistic model evaluation for small changes, which we called *Delta evaluation* [290]. Furthermore, the recent work of Filieri *et al.* [150] on constructing close form formulas for probabilistic model evaluation and work of Kwiatkowska *et al.* [248] on incremental verification probabilistic properties would potentially provide a computational benefit to MC simulation. These approaches could easily be transferred and would improve the performance of the robust optimisation approach.

As it has been emphasised in the scope description, this work is specifically focused on reliability-related attributes and the scope of probabilistic models have been limited to the architecture-based reliability models. However, we suggest that the approach is inherently generic and could be extended to other probabilistic quality attributes and evaluation models. We have included two other quality attributes (response time and cost) in the experiments presented in Chapter 7 as an

indication of the extensibility. However, the generalisability of the approach to the spectrum of probabilistic quality attributes and their evaluation models require further research on specific challenges. It would be interesting to investigate further on the relationship of design-time parameter estimates, uncertainty in estimation and architecture-based quality evaluation models in more general context of probabilistic quality attributes.

Another interesting research direction continuing from this work is to apply robust optimisation at run-time. With the emerging trends on keeping quality evaluation models at run-time (KAMI) [139] and run-time monitoring of software-intensive systems [194], more efficient run-time optimisation techniques are being developed. The current approaches propose to monitor a running system and apply Bayesian estimation techniques to obtain the updated parameter values. However, the values are still treated as point estimates. Due to the stochastic nature of statistics in the run-time monitoring, it is desirable to have better run-time decisions that can tolerate variation while providing better quality of service. Analogies can be seen between this problem and the research presented in the thesis on robust optimisation. Instead of using the point estimates updated at run-time, the SCOUT approach can be adapted to use the probability distribution of the parameter. Further identification and addressing the specific issues related run-time could lead developing the approach into run-time robust optimization.

The term *uncertainty* is associated with many aspects in the general context of systems design. According to Beyer *et al.* [55], the uncertainties that a system designer has to face can be put into four categories, namely (A) Changing environmental and operating conditions, (B) Production tolerances and actuator imprecision, (C) Uncertainties in the system output, (D) Feasibility uncertainties. In this work, the scope was limited to the uncertainties associated with model parameters which in the form of Type A. However, the *robustness* of a software intensive system needs consideration of all four types. This work has identified and addressed specific issues within the limited scope of type A uncertainties. Extending the scope to consider

other uncertain aspects involved in software intensive design and optimisation would be a valuable research effort, which potentially benefit the industrial community.

Bibliography

[1] T. F. Abdelzaher and K. G. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *IEEE Real-Time Systems Symposium*, pages 162–171, 1995.

[2] A. Abraham, H. Liu, and M. Zhao. Particle swarm scheduling for work-flow applications in distributed computing environments. In F. Xhafa and A. Abraham, editors, *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, volume 128 of *Studies in Computational Intelligence*, pages 327–342. Springer, 2008.

[3] M. Agarwal, SudhanshuAggarwal, and V. K. Sharma. Optimal redundancy allocation in complex systems. *Journal of Quality in Maintenance Engineering*, 16:413–424, 2010.

[4] W. Ahmed and D. Myers. Concept-based partitioning for large multidomain multifunctional embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 15(3):221–229, 2010.

[5] M. Åkerholm, J. Fredriksson, K. Sandström, and I. Crnkovic. Quality attribute support in a component technology for vehicular software. In *Fourth Conference on Software Engineering Research and Practice in Sweden*, 2004.

[6] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of AADL models. In *ICSE 2009 Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2009*, pages 61–71. IEEE Computer Society, 2009.

[7] A. Aleti, L. Grunske, I. Meedeniya, and I. Moser. Let the ants deploy your software - an ACO based deployment optimisation strategy. In *International Conference on Automated Software Engineering (ASE'09)*, pages 505–509. IEEE Computer Society, 2009.

[8] A. Aleti and I. Meedeniya. Component deployment optimisation with bayesian learning. In *Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering, CBSE 2011, part of Comparch '11 Federated Events on Component-Based Software Engineering and Software Architecture, Boulder, CO, USA, June 20-24, 2011*, pages 11–20. ACM, 2011.

[9] A. Aleti and I. Moser. Predictive parameter control. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 561–568, 2011.

[10] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple UAVs. In *Proceedings of the American Control Conference*, volume 6, pages 5311–5316. IEEE, 2003.

[11] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue. Cost models with explicit uncertainties for electronic architecture trade-off and risk analysis. In *In Proc. 16th International Symposium of the International Council on Systems Engineering, Orlando, Florida, July, 2006*, pages 1–15. IEEE Computer Society, 2006.

[12] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue. Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In *Quantitative Evaluation of Systems (QEST'09)*, pages 299–308. IEEE Computer Society, 2009.

[13] S. Amari and G. Dill. Redundancy optimization problem with warm-standby redundancy. In *Reliability and Maintainability Symposium (RAMS), 2010 Proceedings*, pages 1 –6, 2010.

[14] V. Ambriola and A. Kmiecik. Architectural transformations. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering (SEKE'02)*, pages 275–278, 2002.

[15] J. Andrews and L. Bartlett. A branching search approach to safety system design optimisation. *Reliability Engineering & System Safety*, 87(1):23–30, 2005.

[16] J. D. Andrews and L. M. Bartlett. Using statistically designed experiments for safety system optimization. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, 218(1):53—-63, 2008.

[17] Y. P. Aneja, R. Chandrasekaran, and K. P. K. Nair. Minimal-cost system reliability with discrete-choice sets for components. *IEEE Transactions on Reliability*, 53(1):71–76, 2004.

[18] B. R. Arafeh, K. Day, and A. Touzene. A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems. *Journal of Systems Architecture - Embedded Systems Design*, 54(5):530–548, 2008.

[19] D. Ardagna, G. Giunta, N. Ingraffia, R. Mirandola, and B. Pernici. QoS-driven web services selection in autonomic grid environments. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2006*, volume 4276 of *Lecture Notes in Computer Science*, pages 1273–1289. Springer, 2006.

[20] D. Ardagna and R. Mirandola. Per-flow optimal service selection for web services based processes. *The Journal of Systems and Software*, 83(8):1512–1523, Aug. 2010.

[21] D. Ardagna and B. Pernici. Global and local QoS constraints guarantee in web service selection. In *Proc. of the IEEE International Conference on Web Services (ICWS 2005)*, pages 805–806. IEEE Computer Society, 2005.

[22] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, 2007.

[23] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *Dependable Systems and Networks (DSN'04)*, pages 347–356. IEEE Computer Society, 2004.

[24] AutoSAR Consortium. AUTOSAR development partnership. www.autosar.org.

[25] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, 1985.

[26] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[27] J. Axelsson, J. Fröberg, H. Hansson, C. Norström, K. Sandström, and B. Villing. A comparative case study of distributed network architectures for different automotive applications. In *The Industrial Information Technology Handbook*, pages 1–20. CRC Press, 2005.

[28] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. G. Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *IEEE Real-Time Systems Symposium*, pages 95–105. IEEE Computer Society, 2001.

[29] A. Azaron, C. Perkgoz, H. Katagiri, K. Kato, and M. Sakawa. Multi-objective reliability optimization for dissimilar-unit cold-standby systems using a genetic algorithm. *Computers & OR*, 36(5):1562–1571, 2009.

[30] F. Bachmann, L. Bass, and M. Klein. Preliminary design of archE: A software architecture design assistant. Technical report, Carnagie-Mellon University Pittsburg PA, Software Engineering Institute, 2003.

[31] F. Bachmann, L. J. Bass, M. Klein, and C. P. Shelton. Experience using an expert system to assist an architect in designing for modifiability. In *4th Working IEEE / IFIP Conference on Software Architecture (WICSA 2004)*, pages 281–284. IEEE Computer Society, 2004.

[32] R. Bahsoon and W. Emmerich. Evaluating software architectures: development, stability and evolution. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia*, pages 47–56. IEEE Computer Society Press, 2003.

[33] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng*, 29(6):524–541, 2003.

[34] C. Baier and M. Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *DISTCOMP: Distributed Computing*, 11, 1998.

[35] J. Balasubramanian, A. S. Gokhale, A. Dubey, F. Wolf, C. Lu, C. D. Gill, and D. C. Schmidt. Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems. In M. Caccamo, editor, *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 69–78. IEEE Computer Society, 2010.

[36] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[37] S. Balsamo and M. Simeoni. Deriving performance models from software architecture specifications. In *In Proc. European Simulation Multiconference(ESM'01),Prague, June 2001*, pages 6–9. Citeseer, 2001.

[38] S. Banerjee and N. Dutt. Efficient search space exploration for hw-sw partitioning. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 122–127, New York, NY, USA, 2004. ACM.

[39] O. Barais, L. Duchien, and A.-F. L. Meur. A framework to specify incremental software architecture transformations. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 62–69. IEEE Computer Society, 2005.

[40] M. R. Barbacci, M. H. Klein, and C. B. Weinstock. Principles for evaluating the quality attributes of a software architecture. Technical report, Carnagie-Mellon University Pittsburg PA, Software Engineering Institute, 1997.

[41] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*, volume 54. Addison-Wesley Longman Publishing Co., Inc., 2003.

[42] M. Basseur and E. Zitzler. A preliminary study on handling uncertainty in indicator-based multiobjective optimization. In *Applications of Evolutionary Computing*, pages 727–739. Springer, 2006.

[43] G. Bauer and H. Kopetz. Transparent redundancy in the time-triggered architecture. In *Proceeding International Conference on Dependable Systems and Networks (DSN'00)*, pages 5–13. IEEE Computer Society, 2000.

[44] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the palladio component model. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *Proceedings of the 6th International Workshop on Software and Performance, WOSP 2007, Buenes Aires, Argentina, February 5-8, 2007*, pages 54–65. ACM, 2007.

[45] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[46] A. Ben-Tal, S. Boyd, and A. Nemirovski. Extending scope of robust optimization: Comprehensive robust counterparts of uncertain problems. *Mathematical Programming*, 107(1):63–89, 2006.

[47] M. Benazouz, O. Marchetti, A. Munier-Kordon, and P. Urard. A New Method for Minimizing Buffer Sizes for Cyclo-Static Dataflow Graphs. In *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, pages 11–20, 2010.

[48] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst*, 8(3):299–316, 2000.

[49] L. Benini, R. Hodgson, and P. Siegel. System-level power estimation and optimization. In A. Chandrakasan and S. Kiaei, editors, *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 173–178. ACM, 1998.

[50] L. Benini and G. D. Micheli. System-level power optimization: techniques and tools. *ACM Trans. Design Autom. Electr. Syst*, 5(2):115–192, 2000.

[51] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and S. Quer. Power optimization of core-based systems by address bus encoding. *IEEE Trans. VLSI Syst*, 6(4):554–562, 1998.

[52] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Heuristics for QoS-aware web service composition. In *Proccedings of the IEEE International Conference on Web Services (ICWS 2006)*, pages 72–82. IEEE Computer Society, 2006.

[53] G. Berry, A. Bouali, X. Fornari, E. Ledinot, E. Nassor, and R. D. Simone. ESTEREL: a formal method applied to avionic software development. *Science of Computer Programming*, 36(1):5–25, 2000.

[54] S. L. Beux, G. Bois, G. Nicolescu, Y. Bouchebaba, M. Langevin, and P. G. Paulin. Combining mapping and partitioning exploration for noC-based embedded systems. *Journal of Systems Architecture - Embedded Systems Design*, 56(7):223–232, 2010.

[55] H.-G. Beyer and B. Sendhoff. Robust optimization - a comprehensive survey. *Computer Methods in Applied Mechanics and Eng.*, 196(33-34):3190 – 3218, 2007.

[56] A. K. Bhunia, L. Sahoo, and D. Roy. Reliability stochastic optimization for a series system with interval component reliability via genetic algorithm. *Applied Mathematics and Computation*, 216(3):929–939, 2010.

[57] A. Billionnet. Redundancy allocation for series-parallel systems using integer linear programming. *IEEE Transactions on Reliability*, 57(3):507–516, 2008.

[58] P. Binns, M. Englehart, M. Jackson, and S. Vestal. Domain-specific software architectures for guidance, navigation and control. *International Journal of Software Engineering and Knowledge Engineering*, 6(2):201–227, 1996.

[59] A. Birolini. *Reliability Engineering: Theory and Practice, Fourth Edition*. Springer-Verlag, 6th edition edition, 2010.

[60] T. Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[61] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. Technical Report TIK Report-Nr. 16, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), 1996.

[62] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker. Compositional dependability evaluation for STATEMATE. *IEEE Transactions on Software Engineering*, 35(2):274–292, 2009.

[63] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[64] E. Bondarev. *Design-Time Performance Analysis of Component-Based Real-Time Systems*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[65] B. Boone, S. Van Hoecke, G. Van Seghbroeck, N. Jonckheere, V. Jonckers, F. D. Turck, C. Develder, and B. Dhoedt. SALSA: QoS-aware load balancing for autonomous service brokering. *Journal of Systems and Software*, 83(3):446–456, 2010.

[66] J. Bosch and P. Molin. Software architecture design: Evaluation and transformation. In *6th Symposium on Engineering of Computer-Based Systems (ECBS'99), 7-12 March 1999, Nashville, TN, USA. IEEE Computer Society, 1999*, pages 4–10. IEEE Computer Society, 1999.

[67] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Parameterized reliability prediction for component-based software architectures. In *Research into Practice - Reality and Gaps, 6th International Conference on the Quality of Software Architectures, QoSA 2010, Prague, Czech Republic, June 23 - 25, 2010. Proceedings*, volume 6093 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2010.

[68] F. Brosch and B. Zimmerova. Design-Time Reliability Prediction for Software Systems. In *Proceedings of the International Workshop on Software Quality and Maintainability (SQM'09)*, pages 70–74, 2009.

[69] M. Broy. Challenges in automotive software engineering. In *International Conference on Software Engineering (ICSE'06)*, pages 33–42. ACM, 2006.

[70] M. Broy. Architecture based specification and verification of embedded software systems (work in progress). In *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings*, volume 17 of *Communications in Computer and Information Science*, pages 1–13. Springer, 2008.

[71] M. Broy, I. Krüger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356 –373, 2007.

[72] S. Burmester, H. Giese, E. Münch, O. Oberschelp, F. Klein, and P. Scheideler. Tool support for the design of self-optimizing mechatronic multi-agent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3):207–222, June 2008.

[73] P. G. Busacca, M. Marseguerra, and E. Zio. Multiobjective optimization by genetic algorithms: application to safety systems. *Reliability Engineering & System Safety*, 72(1):59–74, Apr. 2001.

[74] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM.

[75] G. Canfora, M. D. Penta, R. Esposito, F. Perfetto, and M. L. Villani. Service composition (re)binding driven by application-specific QoS. In A. Dan and W. Lamersdorf, editors, *Proceedings of the 4th International Conference on Service-Oriented Computing - ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2006.

[76] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81(10):1754–1769, 2008.

[77] L. Cao, J. Cao, and M. Li. Genetic algorithm utilized in cost-reduction driven web service selection. In Y. Hao, J. Liu, Y. Wang, Y. ming Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, editors, *Proceedings of the International Conference on Computational Intelligence and Security, CIS 2005*, volume 3802 of *Lecture Notes in Computer Science*, pages 679–686. Springer, 2005.

[78] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium*, pages 131–140. ACM, 2009.

[79] V. Cardellini, E. Casalicchio, V. Grassi, and R. Mirandola. A framework for optimal service selection in broker-based architectures with multiple QoS classes. In *Services computing workshops (SCW'06)*, pages 105–112. IEEE computer society, 2006.

[80] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti. Flow-based service selection forweb service composition supporting multiple QoS classes. In *2007 IEEE International Conference on Web Services (ICWS 2007)*, pages 743–750. IEEE Computer Society, 2007.

[81] M. Ceriani, F. Ferrandi, P. L. Lanzi, D. Sciuto, and A. Tumeo. Multiprocessor systems-on-chip synthesis using multi-objective evolutionary computation. In M. Pelikan and J. Branke, editors, *Genetic and Evolutionary Computation Conference, GECCO 2010,*, pages 1267–1274. ACM, 2010.

[82] R.-S. Chang, J.-S. Chang, and P.-S. Lin. An ant algorithm for balanced job scheduling in Grids. *Future Generation Computer Systems*, 25(1):20–27, Jan. 2009.

[83] Y.-S. Chen, C.-S. Shih, and T.-W. Kuo. Processing element allocation and dynamic scheduling codesign for multi-function socs. *Real-Time Systems*, 44(1-3):72–104, 2010.

[84] M.-S. Chern. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, 11(5):309 – 315, 1992.

[85] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik. Early prediction of software component reliability. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 111–120. ACM, 2008.

[86] R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980.

[87] P. H. Chou, R. B. Ortega, and G. Borriello. Interface co-synthesis techniques for embedded systems. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, 1995*, pages 280–287. IEEE Computer Society, 1995.

[88] P. C. Clements. Active reviews for intermediate designs. Technical report, Carnagie-Mellon University Pittsburg PA, Software Engineering Institute, 2000.

[89] L. Cloth, M. R. Jongerden, and B. R. Haverkort. Computing battery lifetime distributions. In *Proceedings 2005 International Conference on Dependable Systems and Networks (DSN'07)*, pages 780–789. IEEE Computer Society, 2007.

[90] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking markov reward models with impulse rewards. In *Proceedings 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 722–731. IEEE Computer Society, 2005.

[91] D. W. Coit. Cold-standby redundancy optimization for nonrepairable systems. *IIE Transactions*, 33:471–478, 2001.

[92] D. W. Coit. Maximization of system reliability with a choice of redundancy strategies. *IIE Transactions*, 35(6):535–543, 2003.

[93] D. W. Coit and T. Jin. Multi-Criteria Optimization: Maximization of a System Reliability Estimate & Minimization of the Estimate Variance. In *Proceedings of the 2001 European Safety & Reliability International Conference (ESREL)*, pages 1–8, 2001.

[94] D. W. Coit, T. Jin, and N. Wattanapongsakorn. System optimization with component reliability estimation uncertainty: a multi-criteria approach. *IEEE Transactions on Reliability*, 53(3):369–380, 2004.

[95] D. W. Coit and A. Konak. Multiple weighted objectives heuristic for the redundancy allocation problem. *IEEE Transactions on Reliability*, 55(3):551–558, 2006.

[96] D. W. Coit and J. Liu. System reliability optimization with k-out-of-n subsystems. *Int Journal of Reliabilty Quality and Safety Engeneering*, 7(2):129–142, 2000.

[97] D. W. Coit and A. E. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, 45(2):225–266, 1996.

[98] D. W. Coit and A. E. Smith. Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach. *Computers & OR*, 23(6):515–526, 1996.

[99] D. W. Coit and A. E. Smith. Redundancy allocation to maximize a lower percentile of the system time-to-failure distribution. *Reliability, IEEE Transactions on*, 47(1):79 –87, Mar. 1998.

[100] D. W. Coit and A. E. Smith. Genetic algorithm to maximize a lower-bound for system time-to-failure with uncertain component weibull parameters. *Computers & Industrial Engineering*, 41(4):423 – 440, 2002.

[101] P. Conmy, M. Nicholson, and J. McDermid. Safety assurance contracts for integrated modular avionics. In P. Lindsay and T. Cant, editors, *Eighth Australian Workshop on Safety Critical Systems and Software (SCS'03)*, volume 33 of *CRPIT*, pages 69–78. ACS, 2003.

[102] V. Cortellessa, I. Crnkovic, F. Marinelli, and P. Potena. Experimenting the automated selection of COTS components based on cost and system requirements. *J. UCS*, 14(8):1228–1255, 2008.

[103] V. Cortellessa and V. Grassi. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In *Component-Based Software Engineering CBSE'07*, volume 4608, pages 140–156, 2007.

[104] V. Cortellessa, F. Marinelli, and P. Potena. Automated selection of software components based on cost/reliability tradeoff. In V. Gruhn and F. Oquendo, editors, *Software Architecture, Third European Workshop, EWSA 2006*, volume 4344 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 2006.

[105] V. Cortellessa, F. Marinelli, and P. Potena. An optimization framework for "build-or-buy" decisions in software architecture. *Computers & OR*, 35(10):3090–3106, 2008.

[106] V. Cortellessa and R. Mirandola. Deriving a queueing network based performance model from UML diagrams. In *Workshop on Software and Performance*, pages 58–70, 2000.

[107] V. Cortellessa and P. Potena. Path-based error propagation analysis in composition of software services. In *Software Composition, 6th International Symposium, SC 2007, Braga, Portugal, March 24-25, 2007*, volume 4829 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2007.

[108] V. Cortellessa and P. Potena. How can optimization models support the maintenance of component-based software? In *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*, pages 97–100, Washington, DC, USA, 2009. IEEE Computer Society.

[109] V. Cortellessa, H. Singh, and B. Cukic. Early reliability assessment of UML based software models. In *Workshop on Software and Performance*, pages 302–309, 2002.

[110] P. Crescenzi and V. Kann. A compendium of NP optimization problems. www.csc.kth.se/ viggo/wwwcompendium/, 2000.

[111] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Distributed Applications and Interoperable Systems (8th DAIS'08)*, volume 5053 of *Lecture Notes in Computer Science (LNCS)*, pages 253–268. Springer-Verlag (New York), Oslo, Norway, 2008.

[112] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Reiser, M.-O, D. Servat, R. Tavakoli Kolagari, and D. Chen. Developing Automotive Products Using the EAST-ADL2, an AUTOSAR Compliant Architecture Description Language. *Ingéniurs de lAutomobile*, 793:58–64, 2008.

[113] Y.-S. Dai and G. Levitin. Optimal resource allocation for maximizing performance and reliability in tree-structured grid services. *IEEE Transactions on Reliability*, 56(3):444–453, 2007.

[114] B. P. Dave and N. K. Jha. COHRA: Hardware-software co-synthesis of hierarchical distributed embedded system architectures. In *VLSI Design*, pages 347–354. IEEE Computer Society, 1998.

[115] B. P. Dave, G. Lakshminarayana, and N. K. Jha. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. *IEEE Trans. VLSI Syst*, 7(1):92–104, 1999.

[116] P. de Oliveira Castro, S. Louise, and D. Barthou. Reducing memory requirements of stream programs by graph transformations. In W. W. Smari and J. P. McIntire, editors, *Proceedings of the 2010 International Conference on High Performance Computing & Simulation, HPCS 2010*, pages 171–180. IEEE, 2010.

[117] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm:NSGA-II. *IEEE-Evolutionary Computation*, 6:182–197, 2002.

[118] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. S. Gokhale. DAnCE: A QoS-Enabled Component Deployment and Configuration Engine. In *Component Deployment*, volume 3798 of *LNCS*, pages 67–82. Springer, 2005.

[119] Department of Defense. *MIL-HDBK-217,Military Handbook, Reliability Prediction of Electronic Equipment*. Department of Defense, United States of America, 1990.

[120] M. Di Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. Di Nitto. Ws binder: a framework to enable dynamic binding of composite web services. In *Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 74–80, New York, NY, USA, 2006. ACM.

[121] A. Díaz Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann. Integrating quality-attribute reasoning frameworks in the ArchE design assistant. In *International Conference on the Quality of Software Architecture (QoSA'08)*, pages 171–188. Springer, 2008.

[122] R. P. Dick and N. K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, 1998.

[123] L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Trans. Software Eng*, 28(7):638–653, 2002.

[124] K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection. *Annals of Operations Research*, 131(1–4):79–99, 2004.

[125] A. Dogan and F. Özgüner. Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 48(3):300–314, May 2005.

[126] J. Dolbec and T. Shepard. A component based software reliability model. In *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '95, pages 19–. IBM Press, 1995.

[127] I. Doltsinis and Z. Kang. Robust design of structures using optimization methods. *Computer Methods in Applied Mechanics and Engineering*, 193(23-26):2221 – 2237, 2004.

[128] W.-L. Dong and H. Yu. Optimizing web service composition based on QoS negotiation. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006)*, page 46. IEEE Computer Society, 2006.

[129] Dorigo and Blum. Ant colony optimization theory: A survey. *TCS: Theoretical Computer Science*, 345, 2005.

[130] L. dos Santos Coelho. An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications. *Reliability Engineering & System Safety*, 94(4):830 – 837, 2009.

[131] L. dos Santos Coelho. Reliability-redundancy optimization by means of a chaotic differential evolution approach. *Chaos, Solitons & Fractals*, 41(2):594 – 602, 2009.

[132] X. Du and W. Chen. A methodology for managing the effect of uncertainty in simulation-based design. *American Institute of Aeronautics and Astronautics (AIAA)*, 38(8):1471–1478, 2000.

[133] V. K. Dubey and D. A. Menascé. Utility-based optimal service selection for business processes in service oriented architectures. In *IEEE International Conference on Web Services, ICWS 2010*, pages 542–550, 2010.

[134] M. B. Dwyer, J. Hatcliff, R. Robby, C. S. Pasareanu, and W. Visser. Formal software analysis emerging trends in software model checking. In *Future of Software Engineering (FOSE'07)*, pages 120–136. IEEE Computer Society, 2007.

[135] U. Eklund, Ö. Askerdal, J. Granholm, A. Alminger, and J. Axelsson. Experience of introducing reference architectures in the development of automotive electronic systems. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6, 2005.

[136] H. El-Sayed, D. Cameron, and C. M. Woodside. Automation support for software performance engineering. In *Proceedings of the Joint International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS/Performance)*, pages 301–311. ACM, 2001.

[137] C. Elegbede and K. Adjallah. Availability allocation to repairable systems with genetic algorithms: a multi-objective formulation. *Reliability Engineering & System Safety*, 82(3):319 – 330, 2003.

[138] P. Emberson. *Searching For Flexible Solutions To Task Allocation Problems*. PhD thesis, University of York, UK, 2009.

[139] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering (ICSE'09)*, pages 111–121. IEEE Computer Society, 2009.

[140] C. Erbas, S. Cerac-Erbas, and A. D. Pimentel. Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design. *IEEE Transactions on Evolutionary Computation*, 10(3):358–374, June 2006.

[141] R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers*, 15(2):45–54, 1998.

[142] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Des. Test*, 10:64–75, October 1993.

[143] K. Etminani and M. Naghibzadeh. A Min-Min Max-Min selective algorihtm for grid task scheduling. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1–7. IEEE, 2007.

[144] I. D. Falco, A. D. Cioppa, U. Scafuri, and E. Tarantino. Multiobjective differential evolution for mapping in a grid environment. In *High Performance Computing and Communications (3rd HPCC'07)*, volume 4782 of *Lecture Notes in Computer Science (LNCS)*, pages 322–333. Springer-Verlag (New York), Houston, TX, USA, Sept. 2007.

[145] M. Farnsworth, E. Benkhelifa, A. Tiwari, and M. Zhu. A novel approach to multi-level evolutionary design optimization of a MEMS device. In *Evolvable Systems: From Biology to Hardware - 9th International Conference, ICES 2010,*, volume 6274 of *Lecture Notes in Computer Science*, pages 322–334. Springer, 2010.

[146] P. Feiler and A. Greenhouse. *Plug-in Development for the Open Source AADL Tool Environment*. Available from http://la.sei.cmu.edu/aadlinfosite/OSATEPlug-inDevelopmentPresentationSerie.html#Topic19, 2005.

[147] P. H. Feiler. Evolution of an avionics system. Technical report, Carnagie-Mellon University Pittsburg PA, Software Engineering Institute, 2007.

[148] P. H. Feiler and A.-E. Rugina. Dependability Modeling with the Architecture Analysis and Design Language (AADL). Technical report, CMU/SEI-2007-TN-043, 2007.

[149] A. Filieri, C. Ghezzi, V. Grassi, and R. Mirandola. Reliability analysis of component-based systems with multiple failure modes. In *Component-Based Software Engineering, 13th International Symposium, CBSE'10, Prague, Czech Republic, June 23-25, 2010. Proceedings*, volume 6092 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2010.

[150] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011*, pages 341–350. ACM, 2011.

[151] L. Fiondella and S. S. Gokhale. Software reliability with architectural uncertainties. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–5. IEEE Computer Society, 2008.

[152] N. FitzRoy-Dale and IhorKuz. Towards automatic performance optimisation of componentised systems. In *Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems*, pages 31–36, New York, NY, USA, 2009. ACM.

[153] B. Florentz and M. Huhn. Embedded systems architecture: Evaluation and analysis. In *QoSA:Quality of Software Architectures, Second International Conference on Quality of Software Architectures, (QoSA'06)*, volume 4214, pages 145–162. Springer, 2006.

[154] B. Florentz and M. Huhn. Architecture potential analysis: A closer look inside architecture evaluation. *Journal of Software*, 2(4):43–56, 2007.

[155] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulatiin, discussion and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93)*, pages 416–423, San Mateo, California, 1993. Morgan Kaufmann Publishers.

[156] M. Förster and M. Trapp. Fault tree analysis of software-controlled component systems based on second-order probabilities. In *ISSRE '09*, pages 146–154. IEEE Computer Society, 2009.

[157] R. B. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 37–54, 2007.

[158] J. Fredriksson, T. Nolte, M. Nolin, and H. Schmidt. Contract-based reusable worst-case execution time estimate. In *The International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 39–46, 2007.

[159] J. Fredriksson, K. Sandström, and M. Åkerholm. Optimizing resource usage in component-based real-time systems. In *Component-Based Software Engineering, 8th International Symposium (CBSE'05)*, volume 3489 of *LNCS*, pages 49–65. Springer, 2005.

[160] J. Fredriksson, M. Tivoli, and I. Crnkovic. A component-based development framework for supporting functional and non-functional analysis in control system design. In *ASE 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pages 368–371, 2005.

[161] B. Galván, G. Winter, D. Greiner, D. Salazar, and M. Méndez. New evolutionary methodologies for integrated safety system design and maintenance optimization. In

G. Levitin, editor, *Computational Intelligence in Reliability Engineering*, volume 39 of *Studies in Computational Intelligence*, pages 151–190. Springer, 2007.

[162] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 2007.

[163] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. Journ. Robotic Res*, 23(9):939–954, 2004.

[164] J. D. Gibbons and S. Chakraborti. *Nonparametric statistical inference*. CRC Press, 2003.

[165] L. D. Giovanni and F. Pezzella. An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. *European Journal of Operational Research*, 200(2):395 – 408, 2010.

[166] A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *J. Parallel Distrib. Comput*, 69(3):326–336, 2009.

[167] M. Glaß, M. Lukasiewycz, C. Haubelt, and J. Teich. Lifetime Reliability Optimization for Embedded Systems: A System-Level Approach. In *Proceedings of RASDAT '10*, pages 17–22, 2010.

[168] R. L. Glass. The software-research crisis. *IEEE Software*, 11(6):42–47, 1994.

[169] P. L. Goddard. Software fmea techniques. In *Proceedings of Reliability and Maintainability Symposium*, pages 118 –123, 2000.

[170] S. S. Gokhale. Cost constrained reliability maximization of software systems. In *Reliability and Maintainability, 2004 Annual Symposium - RAMS*, pages 195 – 200, 2004.

[171] S. S. Gokhale. Software application design based on architecture, reliability and cost. In *Symposium on Computers and Communications (ISCC'06)*, pages 1098–1103. IEEE Computer Society, 2004.

[172] S. S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions Dependable and Secure Computing*, 4(1):32–40, 2007.

[173] S. S. Gokhale, M. R. Lyu, and K. S.Trivedi. Reliability simulation of component-based software systems. In *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pages 192 –201, 1998.

[174] S. S. Gokhale and K. S. Trivedi. Structure-based software reliability prediction. In *In Proc. of Fifth International Conference on Advanced Computing (ADCOMP '97)*, pages 447–452, 1997.

[175] S. S. Gokhale and K. S. Trivedi. Reliability prediction and sensitivity analysis based on software architecture. In *International Symposium of Software Reliability Engineering (ISSRE'02)*, pages 64–78. IEEE Computer Society, 2002.

[176] S. S. Gokhale, W. E. Wong, J. R. Horgan, and K. S. Trivedi. An analytical approach to architecture-based software performance and reliability prediction. *Performance Evaluation*, 58(4):391–412, 2004.

[177] K. Goševa-Popstojanova and M. Hamill. Architecture-based software reliability: Why only a few parameters matter? In *IEEE Computer Software and Applications Conference*, pages 423–430. IEEE Computer Society, 2007.

[178] K. Goševa-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *International Symposium on Software Reliability Engineering(ISSRE'05)*, pages 43–52. IEEE Computer Society, 2005.

[179] K. Goševa-Popstojanova, M. Hamill, and X. Wang. Adequacy, accuracy, scalability, and uncertainty of architecture-based software reliability: Lessons learned from large empirical case studies. In *International Symposium of Software Reliability Engineering (ISSRE'06)*, pages 197–203. IEEE Computer Society, 2006.

[180] K. Goševa-Popstojanova and S. Kamavaram. Assessing uncertainty in reliability of component-based software systems. In *ISSRE*, pages 307–320. IEEE Computer Society, 2003.

[181] K. Goševa-Popstojanova and S. Kamavaram. Software reliability estimation under uncertainty: Generalization of the method of moments. In *High-Assurance Systems Engineering (HASE'04)*, pages 209–218. IEEE Computer Society, 2004.

[182] K. Goševa-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001.

[183] D. Greiner, B. Galván, and G. Winter. Safety Systems Optimum Design by Multicriteria Evolutionary Algorithms. In *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 722–736. Springer. Lecture Notes in Computer Science. Volume 2632, 2003.

[184] K. Grimm. Software technology in an automotive company: major challenges. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 498–505, Piscataway, NJ, 2003. IEEE Computer Society.

[185] C. Groß, H. Hermanns, and R. Pulungan. Does clock precision influence zigbee's energy consumptions? In E. Tovar, P. Tsigas, and H. Fouchal, editors, *Principles of Distributed Systems (11th OPODIS'07)*, volume 4878 of *Lecture Notes in Computer Science (LNCS)*, pages 174–188. Springer-Verlag (New York), 2007.

[186] L. Grunske. Using a graph transformation system to improve the quality characteristics of UML-RT specifications. In H. Yang, editor, *Paradigm Gems 2*, pages 21–48. IDEA Group Publishing, 2005.

[187] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *28th International Conference on Software Engineering (ICSE 2006)*, pages 849–852. ACM, 2006.

[188] L. Grunske. Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. In *Quality of Software Architectures, QoSA'06,*, volume 4214 of *LNCS*, pages 199–213. Springer, 2006.

[189] L. Grunske. Early quality prediction of component-based systems - A generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[190] L. Grunske, R. Colvin, and K. Winter. Probabilistic Model-Checking Support for FMEA. In *Proc. 4th International Conference on the Quantitative Evaluation of Systems, QEST 07*, pages 119–128. IEEE Computer Society, 2007.

[191] L. Grunske and J. Han. A comparative study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models. In *IEEE High Assurance Systems Engineering Symposium, (HASE'08)*, pages 283–292. IEEE Computer Society, 2008.

[192] L. Grunske, B. Kaiser, and Y. Papadopoulos. Model-driven safety evaluation with state-event-based component failure annotations. In *8th Int. Symp. on Component-Based Software Engineering, CBSE 2005, Proc.*, pages 33–48, 2005.

[193] L. Grunske, P. A. Lindsay, E. Bondarev, Y. Papadopoulos, and D. Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In R. de Lemos, C. Gacek, and A. B. Romanovsky, editors, *Architecting Dependable Systems*, volume 4615 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2006.

[194] L. Grunske and P. Zhang. Monitoring probabilistic properties. In *Proceedings of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 183–192. ACM, 2009.

[195] R. Gulati. A modular approach to static and dynamic fault tree analysis. Master's thesis, The University of Virginia, Department of Electrical Engineering, 1996.

[196] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du. ANGEL: Optimal Configuration for High Available Service Composition. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 280–287. IEEE Computer Society, 2007.

[197] R. K. Gupta. *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Kluwer Academic Publishers, Norwell, USA, 1995.

[198] J. E. Haddad, M. Manouvrier, and M. Rukoz. TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition. *IEEE T. Services Computing*, 3(1):73–85, 2010.

[199] A. B. Hadj-Alouane, J. C. Bean, and K. G. Murty. A hybrid genetic/optimization algorithm for a task allocation problem. *Journal of Scheduling*, 2(4), 1999.

[200] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. In *SPIN:Model Checking Software, 16th International SPIN Workshop, Grenoble, France, June 26-28, 2009. Proceedings*, volume 5578 of *LNCS*, pages 88–106. Springer, 2009.

[201] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskander. Component selection strategies based on system requirements' dependencies on component attributes. In *Systems Conference, 2008 2nd Annual IEEE*, pages 1 –5, 2008.

[202] M. Hashemi and S. Ghiasi. Throughput-driven synthesis of embedded software for pipelined execution on multicore architectures. *ACM Trans. Embedded Comput. Syst*, 8(2), 2009.

[203] A. B. Hassine, S. Matsubara, and T. Ishida. A constraint-based approach to horizontal web service composition. In *5th International Semantic Web Conference, ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 130–143. Springer, 2006.

[204] X. He, Z. Gu, and Y. Zhu. Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming. *Comput. J*, 53(7):1071–1091, 2010.

[205] M. P. E. Heimdahl. Safety and software intensive systems: Challenges old and new. In *Future of Software Engineering (FOSE '07)*, pages 137–152, Washington, DC, USA, 2007. IEEE Computer Society.

[206] G. Heiner and T. Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *International Symposium on Fault-Tolerant Computing (FTCS'98)*, pages 402–407, 1998.

[207] J. Henkel, R. Ernst, U. Holtmann, and T. Benner. Adaptation of partitioning and high-level synthesis in hardware/software co-synthesis. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design, 1994*, pages 96–100. IEEE Computer Society, 1994.

[208] C. A. R. Hoare. Communicating sequential processes. *Comm.ACM*, 21(8):666–677, 1978.

[209] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 18(12):1702–1714, 1999.

[210] C.-J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *Proceedings of the Real-Time Systems Symposium - 1992*, pages 146–156. IEEE Computer Society Press, 1992.

[211] H.-Z. Huang, J. Qu, and M. J. Zou. Genetic-algorithm-based optimal apportionment of reliability and redundancy under multiple objectives. *IIE Transactions*, 41(4):287–298, Apr. 2009.

[212] H. P. Huynh and T. Mitra. Runtime reconfiguration of custom instructions for real-time embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'2009)*, pages 1536–1541. IEEE, 2009.

[213] IEC. IEC 61025 – Fault-Tree-Analysis (FTA), 1990.

[214] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and System Modeling*, 7(1):49–65, 2008.

[215] S. Islam, R. Lindstrom, and N. Suri. Dependability driven integration of mixed criticality SW components. In *ISORC 2006*, pages 485–495. IEEE Computer Society, 2006.

[216] S. Islam and N. Suri. A multi variable optimization approach for the design of integrated dependable real-time embedded systems. In *Embedded and Ubiquitous Computing, International Conference, EUC 2007*, volume 4808 of *Lecture Notes in Computer Science*, pages 517–530. Springer, 2007.

[217] ISO/IEC. International Standard 9126 Software Engineering – Product Quality, 1991.

[218] ISO/IEC. IEEE International Standard 1471 2000 - Systems and software engineering Recommended practice for architectural description of software-intensive systems, 2000.

[219] ISO/IEC. Final Comittee Draft 42010 Architecture Description, 2010.

[220] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *DATE 2005*, pages 864–869. IEEE Computer Society, 2005.

[221] N. Jafarpour and M. R. Khayyambashi. Qos-aware selection of web service composition based on harmony search algorithm. In *Proceedings of the 12th international conference on Advanced communication technology (ICACT'10)*, pages 1345–1350. IEEE Press, 2010.

[222] A. Jhumka, M. Hiller, and N. Suri. Assessing inter-modular error propagation in distributed software. In *Proceedings of the 20th Symposium on Reliable Distributed Systems (20th SRDS'01)*, pages 152–161, 2001.

[223] A. Joshi and M. P. E. Heimdahl. Behavioral fault modeling for model-based safety analysis. In *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*, pages 199–208. IEEE Computer Society, 2007.

[224] B. Kaiser, C. Gramlich, and M. Förster. State/event fault trees - safety analysis model for software-controlled systems. *Reliability Engineering and System Safety*, 92(11):1521–1537, 2007.

[225] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *8th Australian Workshop on Safety-Critical Systems and Software proceedings (SCS'03)*, volume 33, pages 37–46. Australian Computer Society, 2003.

[226] R. C. Kastner. *Synthesis techniques and optimizations for reconfigurable systems.* PhD thesis, University of California, Los Angeles, 2002.

[227] K. Kaya and B. Uçar. Exact algorithms for a task assignment problem. *Parallel Processing Letters*, 19(3):451–465, 2009.

[228] R. Kazman, L. Bass, M. Webb, and G. Abowd. SAAM: a method for analyzing the properties of software architectures. In *International Conference on Software Engineering (ICSE'94)*, pages 81–90. ACM, 1994.

[229] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. *IEEE International Conference on Engineering of Complex Computer Systems*, 0:0068, 1998.

[230] T. Kichkaylo, A.-A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'03)*, pages 1–10, 2003.

[231] Y.-J. Kim and T. Kim. A HW/SW partitioner for multi-mode multi-task embedded applications. *VLSI Signal Processing*, 44(3):269–283, 2006.

[232] A. Kishor, S. P. Yadav, and S. Kumar. Application of a multi-objective genetic algorithm to solve reliability optimization problem. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007) - Volume 01*, pages 458–462, Washington, DC, USA, 2007. IEEE Computer Society.

[233] J. M. Ko, C. O. Kim, and I.-H. Kwon. Quality-of-service oriented web service composition algorithm and planning architecture. *Journal of Systems and Software*, 81(11):2079–2090, 2008.

[234] H. Kopetz and G. Grunsteidl. TTP – A time-triggered protocol for fault-tolerant real-time systems. In *Digest of Papers: 23rd Annual International Symposium on Fault-Tolerant Computing Systems (FTCS'93)*, pages 524–533, Toulouse, France, 1993. IEEE Computer Society.

[235] A. Koziolek, H. Koziolek, and R. Reussner. PerOpteryx: automated application of tactics in multi-objective software architecture optimization. In *Joint proceedings of the Seventh International ACM SIGSOFT Conference on the Quality of Software Architectures and the 2nd ACM SIGSOFT International Symposium on Architecting Critical Systems (QoSA-ISARCS 2011)*, pages 33–42. ACM, 2011.

[236] A. Koziolek and R. Reussner. Towards a generic quality optimisation framework for component-based system models. In *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, CBSE '11, pages 103–108. ACM, 2011.

[237] H. Koziolek and F. Brosch. Parameter dependencies for component reliability specifications. In *6th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA'09)*, volume 253 of *Electronic Notes in Theoretical Computer Science*, pages 23 – 38. Elsevier, 2009.

[238] S. Krishnamurthy and A. Mathur. On the estimation of reliability of a software system using reliabilities of its components. *Software Reliability Engineering, International Symposium on*, 0:146, 1997.

[239] I. Krka, G. Edwards, L. Cheung, L. Golubchik, and N. Medvidovic. A comprehensive exploration of challenges in architecture-based reliability estimation. *Architecting Dependable Systems VI*, 5835:202–227, 2008.

[240] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[241] P. Kubat. Assessing reliability of modular software. *Operations Research Letters*, 8(1):35–41, 1989.

[242] S. Kulturel-Konak and A. E. S. D. W. Coit. Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*, 35(6):515–526, 2003.

[243] S. Kulturel-Konak, D. W. Coit, and F. Baheranwala. Pruned pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives. *Journal of Heuristics*, 14(4):335–357, Aug. 2008.

[244] S. Kulturel-Konak, A. E. Smith, and D. W. Coit. Efficiently solving the redundancy allocation problem using tabu search. *Iet Software/iee Proceedings - Software*, 2002.

[245] W. Kuo and V. R. Prasad. An annotated overview of system-reliability optimization. *Reliability, IEEE Transactions on*, 49(2):176 –187, June 2000.

[246] W. Kuo and R. Wan. Recent Advances in Optimal Reliability Allocation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering. Evolutionary Techniques in Reliability Analysis and Optimization*, pages 1–36. Springer, Heidelberg, 2007.

[247] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: case studies with prism. *SIGMETRICS Performance Evaluation Review*, 32(4):16–21, 2005.

[248] M. Z. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for markov decision processes. In *Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011)*, pages 359–370. IEEE, 2011.

[249] Y. Laalaoui, H. Drias, A. Bouridah, and R. Ahmed. Ant colony system with stagnation avoidance for the scheduling of real-time tasks. In *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, pages 1 –6, 2009.

[250] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput*, 19(1):93–109, 2007.

[251] H. le Guen, R. A. Marie, and T. Thelin. Reliability estimation for statistical usage testing using markov chains. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE04)*, pages 54–65. IEEE Computer Society, 2004.

[252] C. Lee, S. Kim, and S. Ha. A systematic design space exploration of MPSoC based on synchronous data flow specification. *Signal Processing Systems*, 58(2):193–213, 2010.

[253] E. A. Lee. Embedded software. *Advances in Computers*, 56(1):1–34, 2002.

[254] N. G. Leveson. An approach to designing safe embedded software. In *Embedded Software, Second International Conference, EMSOFT 2002, Grenoble, France, October 7-9, 2002, Proceedings*, volume 2491 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2002.

[255] N. G. Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Trans. Dependable Sec. Comput*, 1(1):66–86, 2004.

[256] H. Li, G. Casale, and T. N. Ellahi. SLA-driven planning and optimization of enterprise applications. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering, 2010*, pages 117–128. ACM, 2010.

[257] Y.-C. Liang and Y.-C. Chen. Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm. *Reliability Engineering & System Safety*, 92(3):323 – 331, 2007.

[258] Y.-C. Liang, M.-H. Lo, and Y.-C. Chen. Variable neighbourhood search for redundancy allocation problems. *IMA Journal of Management Mathematics*, 18(2):135–155, April 2007.

[259] Y.-C. Liang and A. E. Smith. An ant system approach to redundancy allocation. In *Congress on Evolutionary Computation*, pages 1478–1484. IEEE, 1999.

[260] P. Limbourg. *Dependability modelling under uncertainty: An imprecise probabilistic approach*. Springer Verlag, 2008.

[261] P. Limbourg and H.-D. Kochs. Multi-objective optimization of generalized reliability design problems using feature models - A concept for early design stages. *Reliability Engineering & System Safety*, 93(6):815–828, June 2008.

[262] B. Littlewood. A software reliability model for modular program structure. *IEEE Transactions on Reliability*, R-28:241–246, 1979.

[263] B. Littlewood and L. Strigini. Software reliability and dependability: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*, pages 175–188, 2000.

[264] Y. H. Lu, T. Simunic, and G. D. Micheli. Software controlled power management. In *Proceedings of the 7th International Workshop on Hardware/Software Codesign (CODES'99)*, pages 157–161, 1999.

[265] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *ASP-DAC 2008*, pages 691–696. IEEE, 2008.

[266] M. Lukasiewycz, M. Glaß, and J. Teich. Robust design of embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE'10, pages 1578–1583. European Design and Automation Association, 2010.

[267] L. Lundberg, J. Bosch, D. Hggander, and P.-O. Bengtsson. Quality attributes in software architecture design. In *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, pages 353–362. Citeseer, 1999.

[268] R. R. Lutz. Software engineering for safety: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*, pages 213–226. ACM, 2000.

[269] M. R. Lyu. *Handbook of software reliability engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

[270] M. R. Lyu. Software reliability engineering: A roadmap. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 153–170, 2007.

[271] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny. QoS-aware service composition in dynamic service oriented environments. In *Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 123–142. Springer, 2009.

[272] S. Malek. *A User-Centric Approach For Improving A Distributed Software System's Deployment Architecture*. PhD thesis, Faculty of The Graduate School University Of Southern California, 2007.

[273] B. S. Manoj, A. Sekhar, and C. S. R. Murthy. A state-space search approach for optimizing reliability and cost of execution in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 69(1):12–19, Jan. 2009.

[274] M. Marseguerra, E. Zato, and L. Podofillini. Genetic Algorithms and Monte Carlo Simulation for the Optimization of System Design and Operation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering. Evolutionary Techniques in Reliability Analysis and Optimization*, pages 101–150. Springer, Heidelberg, 2007.

[275] M. Marseguerra, E. Zio, and S. Martorell. Basics of genetic algorithms optimization for rams applications. *Reliability Engineering & System Safety*, 91(9):977 – 991, 2006.

[276] M. Marseguerra, E. Zio, and L. Podofillini. A multiobjective genetic algorithm approach to the optimization of the technical specifications of a nuclear safety system. *Reliability Engineering & System Safety*, 84(1):87 – 99, 2004.

[277] M. Marseguerra, E. Zio, and L. Podofillini. Multiobjective spare part allocation by means of genetic algorithms and monte carlo simulation. *Reliability Engineering & System Safety*, 87(3):325 – 335, 2005.

[278] M. Marseguerra, E. Zio, L. Podofillini, and D. W. Coit. Optimal design of reliable network systems in presence of uncertainty. *IEEE Transactions on Reliability*, 54(2):243–253, 2005.

[279] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola, and R. Reussner. A hybrid approach for multi-attribute QoS optimisation in component based software systems. In *6th International Conference on the Quality of Software Architectures (QoSA'10)*, volume 6093 of *LNCS*, pages 84–101. Springer, 2010.

[280] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering, 2010*, pages 105–116. ACM, 2010.

[281] S. Martorell, A. Snchez, S. Carlos, and V. Serradell. Alternatives and challenges in optimizing industrial safety using genetic algorithms. *Reliability Engineering & System Safety*, 86(1):25 – 38, 2004.

[282] A. Marzia, A. F. Francesca, A. D. Ardagna, B. Luciano, B. Carlo, C. Cinzia, C. Marco, C. Marco, D. P. F. Maria, F. Chiara, G. Simone, L. P, M. Andrea, M. Stefano, P. Barbara, R. Claudia, and T. Francesco. The mais approach to web service design. In *10th International Workshop on Exploring Modelling Methods in Systems Analysis and Design, June 13-17, Porto, Portugal*, pages 387–398, 2005.

[283] N. Medvidovic and S. Malek. Software deployment architecture and quality-of-service in pervasive environments. In *Workshop on the Engineering of Software Services for Pervasive Environements, (ESSPE'07)*, pages 47–51. ACM, 2007.

[284] N. Medvidovic, S. Malek, and M. Mikic-Rakic. Software architectures and embedded systems. In *In Proceedings of the Monterey Workshop on Software Engineering for Embedded Systems (SEES 2003)*, pages 65–71, 2003.

[285] N. Medvidovic, M. Mikic-Rakic, and N. R. Mehta. Improving dependability of component-based systems via multi-versioning connectors. *Architecting Dependable Systems*, 2677:37–60, 2003.

[286] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transaxtions on Software Engineering*, 26(1):70–93, 2000.

[287] I. Meedeniya, A. Aleti, and B. Bühnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium of Avionics/Automotive Systems Engineering (SAASE'09), San Diego, CA*, 2009.

[288] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *6th International Conference on the Quality of Software Architectures, QoSA 2010*, volume 6093 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2010.

[289] I. Meedeniya, B. Bühnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011.

[290] I. Meedeniya and L. Grunske. An Efficient Method for Architecture-Based Reliability Evaluation for Evolving Systems with Changing Parameters. In *IEEE International*

Symposium on Software Reliability Engineering (ISSRE'10), pages 229–238. IEEE, 2010.

[291] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske. Architecture-based reliability evaluation under uncertainty. In *7th International Conference on the Quality of Software Architectures, QoSA 2011 and 2nd International Symposium on Architecting Critical Systems, ISARCS 2011. Boulder, CO, USA, June 20-24, 2011, Proceedings*, pages 85–94. ACM, 2011.

[292] D. A. Menascé, E. Casalicchio, and V. Dubey. A heuristic approach to optimal service selection in service oriented architectures. In A. Avritzer, E. J. Weyuker, and C. M. Woodside, editors, *Proceedings of the 7th International Workshop on Software and Performance, WOSP 2008*, pages 13–24. ACM, 2008.

[293] D. A. Menascé and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *Proceedings of the International Conference on Web Services ICWS '07*, pages 422–430, 2007.

[294] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malek, and J. P. Sousa. A framework for utility-based service oriented design in SASSY. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering, 2010*, pages 27–36. ACM, 2010.

[295] D. A. Menascé, H. Ruan, and H. Gomaa. QoS management in service-oriented architectures. *Perform. Eval.*, 64(7-8):646–663, 2007.

[296] M. Mikic-Rakic, S. Malek, N. Beckman, and N. Medvidovic. A tailorable environment for assessing the quality of deployment architectures in highly distributed settings. In *International Working Conference on Component Deployment (CD'04)*, volume 3083 of *LNCS*, pages 1–17. Springer, 2004.

[297] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving availability in large, distributed component-based systems via redeployment. In *Component Deployment, Third International Working Conference, CD 2005*, volume 3798 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2005.

[298] M. Mikic-Rakic and N. Medvidovic. Architecture-level support for software component deployment in resource constrained environments. In *Component Deployment*, volume 2370, pages 31–50. Springer, 2002.

[299] D. C. Montgomery and G. C. Runger. *Applied statistics and probability for engineers*. New York; John Wiley & Sons, 2011.

[300] J. Montgomery and I. Moser. Parallel constraint handling in a multiobjective evolutionary algorithm for the automotive deployment problem. In *e-Science Workshops, 2010 Sixth IEEE International Conference on*, pages 104 –109, 2010.

[301] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the 7th ACM & IEEE International conference on Embedded software, EMSOFT 2007*, pages 57–66. ACM, 2007.

[302] I. Moser and S. Mostaghim. The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[303] B. Naderi, S. M. T. F. Ghomi, and M. Aminnayeri. A high performing metaheuristic for job shop scheduling with sequence-dependent setup times. *Appl. Soft Comput*, 10(3):703–710, 2010.

[304] M. Nicholson. *Selecting a Topology for Safety-Critical Real-Time Control Systems*. PhD thesis, Department of Computer Science, University of York, 1998.

[305] M. Nicholson, A. Burns, and Y. Dd. Emergence of an architectural topology for safety-critical real-time systems. Technical report, Department of Computer Science,University of York, 1997.

[306] M. Nicholson and D. Prasad. Design synthesis using adaptive search techniques and multi-criteria decision analysis. In *ICECCS 1996*, pages 522 – 529. IEEE Computer Society, 1996.

[307] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1(1):48–65, 2004.

[308] W. L. Oberkampf, J. C. Helton, C. A. Joslyn, S. F. Wojtkiewicz, and S. Ferson. Challenge problems: uncertainty in system response given uncertain parameters. *Reliability Engineering & System Safety*, 85(1-3):11 – 19, 2004.

[309] Object Management Group (OMG). Unified modelling language specification, supersturcture specification : version 2.2, 2008.

[310] H. Oh and S. Ha. A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling. In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign, CODES 1999*, pages 183–187. ACM, 1999.

[311] F. Ortmeier and W. Reif. Safety optimization: A combination of fault tree analysis and optimization techniques. In *International Conference on Dependable Systems and Networks (DSN 2004), Performance and Dependability Symposium*, pages 651–658. IEEE Computer Society, 2004.

[312] M. Ouzineb, M. Nourelfath, and M. Gendreau. Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems. *Reliability Engineering & System Safety*, 93(8):1257 – 1272, 2008.

[313] M. Ouzineb, M. Nourelfath, and M. Gendreau. An efficient heuristic for reliability design optimization problems. *Computers & OR*, 37(2):223–235, 2010.

[314] L. Painton and J. Campbell. Genetic algorithms in optimization of system reliability. *IEEE Transactions on Reliability*, 44(2):172 –178, 1995.

[315] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 6(2):149–206, Jan. 2001.

[316] Y. Papadopoulos and C. Grante. Techniques and tools for automated safety analysis & decision support for redundancy allocation in automotive systems. In *Computer Software and Applications Software (COMPSAC'03)*, pages 105–110. IEEE Computer Society, 2003.

[317] Y. Papadopoulos and C. Grante. Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software*, 76(1):77–89, 2005.

[318] Y. Papadopoulos and M. Maruhn. Model-based synthesis of fault trees from matlab-simulink models. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*, pages 77–82, Washington - Brussels - Tokyo, 2001. IEEE.

[319] D. L. Parnas and D. M. Weiss. Active design reviews: Principles and practices. *Journal of Systems and Software*, 7:259–265, 1987.

[320] A. S. Parrish, B. Dixon, and D. Cordes. A conceptual foundation for component-based software deployment. *Journal of Systems and Software*, 57(3):193–200, 2001.

[321] R. L. Pattison and J. Andrews. Genetic algorithms in optimal safety design. *Proceedings of the Institution of Mechanical Engineers, Part E : Journal of Process Mechanical Engineering*, 213(3):187–197, 1999.

[322] M. Perillo and W. Heinzelman. Optimal sensor management under energy and reliability constraints. In *Wireless Communications and Networking, (WCNC 2003)*, volume 3, pages 1621 –1626 vol.3. IEEE, 2003.

[323] J. E. Pezoa, S. Dhakal, and M. M. Hayat. Maximizing service reliability in distributed computing systems with random node failures: Theory and implementation. *IEEE Trans. Parallel Distrib. Syst*, 21(10):1531–1544, 2010.

[324] J. E. Pezoa and M. M. Hayat. Task reallocation for maximal reliability in distributed computing systems with uncertain topologies and non-markovian delays. Technical report, Electrical Engineering Department of the Universidad de Concepcin, 2009.

[325] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Computers*, 55(2):99–112, 2006.

[326] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and E. F. Deprettere. Exploring embedded-systems architectures with artemis. *IEEE Computer*, 34(11):57–63, 2001.

[327] F. Pop, C. Dobre, and V. Cristea. Genetic algorithm for dag scheduling in grid environments. In *Intelligent Computer Communication and Processing,ICCP 2009*, pages 299 –305, 2009.

[328] P. Potena. Composition and tradeoff of non-functional attributes in software systems: research directions. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 583–586. ACM, 2007.

[329] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71. IEEE Computer Society, 2007.

[330] X. Qin and H. Jiang. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *J. Parallel Distrib. Comput*, 65(8):885–900, 2005.

[331] Q. Qiu and M. Pedram. Dynamic power management based on continuous-time markov decision processes. In *DAC 1999*, pages 555–561, 1999.

[332] Q. Qiu, Q. Wu, and M. Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *DAC 2000*, pages 352–356, 2000.

[333] O. Räihä, K. Koskimies, and E. Mäkinen. Genetic synthesis of software architecture. Technical Report D-2008-4, Department of Computer Science, University of Tampere, 2008.

[334] O. Räihä, K. Koskimies, and E. Mäkinen. Scenario-based genetic synthesis of software architecture. In *The Fourth International Conference on Software Engineering Advances, ICSEA 2009*, pages 437–445. IEEE Computer Society, 2009.

[335] O. Räihä, E. Mäkinen, and T. Poranen. Using simulated annealing for producing software architectures. In *GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference*, pages 2131–2136. ACM, 8-12 July 2009.

[336] Y. Ren and J. Bechta Dugan. Design of reliable systems using static and dynamic fault trees. *IEEE Transactions on Reliability*, 47(3):234 –244, Sept. 1998.

[337] R. Reussner, H. W. Schmidt, and I. Poernomo. Reliability prediction for component-based software architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.

[338] J. Riauke and L. M. Bartlett. An offshore safety system optimization using an spea2-based approach. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(3):271–282, 2008.

[339] G. N. Rodrigues, D. S. Rosenblum, and S. Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *Fundamental Approaches to Software Engineering (FASE'05)*, pages 111–126. Springer, 2005.

[340] C. B. A. Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *CSURV: Computing Surveys*, 35(3):268–308, 2003.

[341] F. Rosenberg, M. B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar. Metaheuristic optimization of large-scale QoS-aware service compositions. In *IEEE SCC 2010*, pages 97–104. IEEE Computer Society, 2010.

[342] V. Roshanaei, B. Naderi, F. Jolai, and M. Khalili. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Comp. Syst*, 25(6):654–661, 2009.

[343] R. Roshandel, S. Banerjee, L. Cheung, N. Medvidovic, and L. Golubchik. Estimating software component reliability by leveraging architectural models. In *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 853–856. ACM, 2006.

[344] R. Roshandel, N. Medvidovic, and L. Golubchik. A bayesian model for predicting reliability of software systems at the architectural level. *Software Architectures, Components, and Applications*, 4880:108–126, 2007.

[345] R. Y. Rubinstein and D. Kroese. *Simulation and the Monte Carlo Method*. Wiley-interscience, 2008.

[346] SAE. Architecture Analysis and Design Language (AADL). *SAE standards*, AS5506(1), 2004.

[347] D. Salazar, C. M. Rocco, and B. J. Galvan. Optimization of constrained multiple-objective reliability problems using evolutionary algorithms. *Reliability Engineering & System Safety*, 91(9):1057–1070, 2006.

[348] D. E. Salazar and C. M. Rocco. Solving advanced multi-objective robust designs by means of multiple objective evolutionary algorithms (MOEA): A reliability application. *Reliability Engineering & System Safety*, 92(6):697–706, June 2007.

[349] A. Sanchez, S. Carlos, S. Martorell, and J. F. Villanueva. Addressing imperfect maintenance modelling uncertainty in unavailability and cost based optimization. *Reliability Engineering & System Safety*, 94(1):22 – 32, 2009.

[350] C. Seo, G. Edwards, S. Malek, and N. Medvidovic. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In *Working IEEE/IFIP Conference on Software Architecture (WICSA'08)*, pages 277–280. IEEE Computer Society, 2008.

[351] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed java-based systems. In *22nd IEEE/ACM International Conference on Automated Software Engineering ASE 2007*, pages 421–424. ACM, 2007.

[352] C. Seo, S. Malek, and N. Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. In *Component-Based Software Engineering, 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, volume 5282 of *LNCS*, pages 97–113. Springer, 2008.

[353] C. Serban, A. Vescan, and H. F. Pop. A new component selection algorithm based on metrics and fuzzy clustering analysis. In *Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS 2009*, volume 5572 of *Lecture Notes in Computer Science*, pages 621–628. Springer, 2009.

[354] S. Shan and G. G. Wang. Reliable design space and complete single-loop reliability-based design optimization. *Reliability Engineering & System Safety*, 93(8):1218 – 1230, 2008.

[355] N. Shankaran, J. Balasubramanian, D. C. Schmidt, G. Biswas, P. J. Lardieri, E. Mulholland, and T. Damiano. A framework for (re)deploying components in distributed real-time and embedded systems. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pages 737–738. ACM, 2006.

[356] V. S. Sharma and M. Agarwal. Ant colony optimization approach to heterogeneous redundancy in multi-state systems with multi-state components. In *Reliability, Maintainability and Safety, ICRMS 2009*, pages 116 –121, 2009.

[357] V. S. Sharma and P. Jalote. Deploying software components for performance. In *Component-Based Software Engineering, 11th International Symposium, CBSE 2008*, volume 5282 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2008.

[358] V. S. Sharma, P. Jalote, and K. S. Trivedi. Evaluating performance attributes of layered software architecture. In *Component-Based Software Engineering, 8th International Symposium (CBSE'05)*, volume 3489 of *LNCS*, pages 66–81. Springer, 2005.

[359] V. S. Sharma and K. S. Trivedi. Architecture based analysis of performance, reliability and security of software systems. In *Proceedings of the 5th international workshop on Software and performance (WOSP'05)*, pages 217–227. ACM, 2005.

[360] V. S. Sharma and K. S. Trivedi. Quantifying software performance, reliability and security: An architecture-based approach. *Journal of Systems and Software*, 80(4):493–509, 2007.

[361] S. M. Shatz, J.-P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Computers*, 41(9):1156–1168, 1992.

[362] M. Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, 2002.

[363] M. Shaw. Writing good software engineering research paper. In *Proceedings of the 25th International Conference on Software Engineering, ICSE'03, Portland, Oregon, USA*, pages 726–737, 2003.

[364] M. Shaw and P. C. Clements. The golden age of software architecture. *IEEE Software*, 23(2):31–39, 2006.

[365] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, 2000,*, pages 365–368. IEEE, 2000.

[366] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM-00)*, pages 11–19. ACM Press, 2000.

[367] T. Simunic, L. Benini, and G. D. Micheli. Energy-efficient design of battery-powered embedded systems. *IEEE Trans. VLSI Syst*, 9(1):15–28, 2001.

[368] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A bayesian approach to reliability prediction and assessment of component based systems. In *International Symposium of Software Reliability Engineering (ISSRE'01)*, pages 12–21. IEEE Computer Society, 2001.

[369] O. Skroch. Multi-criteria service selection with optimal stopping in dynamic service-oriented systems. In *Distributed Computing and Internet Technology (6th ICDCIT'10)*, volume 5966 of *Lecture Notes in Computer Science (LNCS)*, pages 1–12 110–121. Springer-Verlag (New York), 2010.

[370] C. U. Smith and L. G. Williams. Performance evaluation of a distributed software architecture. In *In Proceedings of the 1st International Workshop on Software and Performance*, pages 164–177, 1998.

[371] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[372] M. Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78:176–198, 2002.

[373] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *DAC 07*, pages 777–782. IEEE, 2007.

[374] N. Suri, S. Ghosh, and T. Marlowe. A framework for dependability driven software integration. In *International Conference on Distributed Computing Systems (ICDCS'98)*. IEEE, 1998.

[375] N. Suri, A. Jhumka, M. Hiller, A. Pataricza, S. Islam, and C. Sârbu. A software integration approach for designing and assessing dependable embedded systems. *The Journal of Systems and Software*, 83(10):1780–1800, 2010.

[376] H. A. Taboada, F. Baheranwala, D. W. Coit, and N. Wattanapongsakorn. Practical solutions for multi-objective optimization: An application to system reliability design problems. *Reliability Engineering & System Safety*, 92(3):314 – 322, 2007.

[377] H. A. Taboada and D. W. Coit. Moea-dap: A new multiple objective evolutionary algorithm for solving design allocation problems. Technical report, Department Industrial & Systems Engineering, Rutgers University, 2006.

[378] H. A. Taboada and D. W. Coit. Data clustering of solutions for multiple objective system reliability optimization problems. *Quality Technology & Quantitative Management Journal*, pages 35–54, 2007.

[379] H. A. Taboada, J. F. Espiritu, and D. W. Coit. MOMS-GA: A multi-objective multi-state genetic algorithm for system reliability optimization design problems. *IEEE Transactions on Reliability*, 57(1):182–191, 2008.

[380] S.-A. Tahaee and A. H. Jahangir. A polynomial algorithm for partitioning problems. *ACM Transactions on Embedded Computing Systems*, 9(4):34–44, Mar. 2010.

[381] M. Tang and L. Ai. A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[382] The Eclipse Foundation. *Eclipse Ganymede Documentation*. Available from http://help.eclipse.org/ganymede/index.jsp, 2005.

[383] The MathWorks. *Simulink - Simulation and Model-Based Design*. Available from http://www.mathworks.com/.

[384] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration of network processor architectures. In *1st Workshop on Network Processors at the 8th International Symposium on High-Performance Computer Architecture (HPCA8)*, Cambridge MA, USA, Feb. 2002.

[385] Z. Tian, G. Levitin, and M. J. Zuo. A joint reliability-redundancy optimization approach for multi-state series-parallel systems. *Reliability Engineering & System Safety*, 94(10):1568 – 1576, 2009.

[386] F. A. Tillman, C.-L. Hwang, and W. Kuo. Determining Component Reliability and Redundancy for Optimum System Reliability. *IEEE Transactions on Reliability*, R-26(3):162–165, 1977.

[387] F. A. Tillman, C.-L. Hwang, and W. Kuo. Optimization Techniques for System Reliability with Redundancy – A Review. *IEEE Transactions on Reliability*, R-26(3):148–155, Aug. 1977.

[388] K. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[389] J. L. Tokar. Architecting dependable systems with the SAE architecture analysis and description language (AADL). In *Architecting Dependable Systems IV*, volume 4615 of *LNCS*, pages 1–13. Springer-Verlag (New York), 2007.

[390] A. C. Torres-Echeverria, S. Martorell, and H. A. Thompson. Design optimization of a safety-instrumented system based on RAMS plus C addressing IEC 61508 requirements and diverse redundancy. *Reliability Engineering & System Safety*, 94(2):162–179, Feb. 2009.

[391] K. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. Wiley-India, 2009.

[392] Y. Vanrompay, P. Rigole, and Y. Berbers. Genetic algorithm-based optimization of service composition and deployment. In *Proceedings of the 3rd international workshop on Services integration in pervasive environments*, SIPE '08, pages 13–18, New York, NY, USA, 2008. ACM.

[393] A. Vescan. A metrics-based evolutionary approach for the component selection problem. In *Proceedings of the UKSim 2009: 11th International Conference on Computer Modelling and Simulation*, pages 83–88, Washington, DC, USA, 2009. IEEE Computer Society.

[394] A. Vescan and C. Grosan. A hybrid evolutionary multiobjective approach for the component selection problem. In *Hybrid Artificial Intelligence Systems, Third International Workshop, HAIS 2008*, volume 5271 of *Lecture Notes in Computer Science*, pages 164–171. Springer, 2008.

[395] A. F. Vescan. *Construction Approaches for Component-Based Systems*. Phd, Babes-Bolyai University, 2008.

[396] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. U. S. Nuclear Regulatory Commission, NUREG-0492, Washington DC, 1981.

[397] N. Vijaykrishnan, M. T. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *27th Annual International Symposium on Computer Architecture (ISCA '00)*, pages 95–106, 2000.

[398] H. Wada, P. Champrasert, J. Suzuki, and K. Oba. Multiobjective optimization of SLA-aware service composition. In *IEEE Congress on Services (SERVICES 2008)*, pages 368–375, 2008.

[399] S. A. Wadekar and S. S. Gokhale. Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pages 104–114, Washington, DC, USA, 1999. IEEE Computer Society.

[400] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[401] P. Wallin and J. Axelsson. A case study of issues related to automotive E/E system architecture development. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*, pages 87–95. IEEE Computer Society, 2008.

[402] G. Wang, W. Gong, and R. Kastner. A new approach for task level computational resource bi-partitioning. In *15th International Conference on Parallel and Distributed Computing and Systems, PDCS2003*, 2003.

[403] S. Wang and K. G. Shin. An architecture for embedded software integration using reusable components. In *Proc. of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'00)*, pages 110–118, 2000.

[404] W.-L. Wang, Y. Wu, and M.-H. Chen. An architecture-based software reliability model. In *Pacific Rim International Symposium on Dependable Computing (PRDC'99),*, pages 143–150. IEEE Computer Society, 1999.

[405] N. Wattanapongskorn and D. W. Coit. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty. *Reliability Engineering & System Safety*, 92(4):395–407, 2007.

[406] T. Wiangtong, P. Y. K. Cheung, and W. Luk. Tabu search with intensification strategy for functional partitioning in hardware-software codesign. In *The 10th Annual*

IEEE Symposium on. Field-Programmable Custom Computing Machines (FCCM '02), pages 297–298. IEEE Computer Society, 2002.

[407] W. Wiesemann, R. Hochreiter, and D. Kuhn. A stochastic programming approach for QoS-aware service composition. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*, pages 226–233. IEEE Computer Society, 2008.

[408] L. G. Williams and C. U. Smith. Performance evaluation of software architectures. In *Proceedings of the 1st international workshop on Software and performance (WOSP'98)*, pages 164–177. ACM, 1998.

[409] C. M. Woodside. Software resource architecture and performance evaluation of software architectures. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS'01)*, 2001.

[410] S. M. Yacoub, B. Cukic, and H. H. Ammar. A scenario-based reliability analysis approach for component-based software. *IEEE Transactions on Reliability*, 53(4):465–480, 2004.

[411] E. Yang, A. T. Erdogan, T. Arslan, and N. Barton. Multi-objective evolutionary optimizations of a space-based reconfigurable sensor network under hard constraints. In *Symposium on Bio-inspired, Learning, and Intelligent Systems for Security (BLISS'07)*, pages 72–75. IEEE Computer Society, 2007.

[412] W.-C. Yeh and T.-J. Hsieh. Solving reliability redundancy allocation problems using an artificial bee colony algorithm. *Computers & Operations Research*, 38(11):1465–1473, 2010.

[413] L. Yin, M. Smith, and K. Trivedi. Uncertainty analysis in reliability modeling. In *Proceedings of Annual Reliability and Maintainability Symposium*, pages 229 –234, 2001.

[414] H. Youness, M. Hassan, K. Sakanushi, Y. Takeuchi, M. Imai, A. Salem, A.-M. Wahdan, and M. Moness. Optimization method for scheduling length and the number of processors on multiprocessor systems. In *International Conference on Computer Engineering Systems (ICCES'09)*, pages 231 –236, 2009.

[415] M. F. Younis, K. Akkaya, and A. Kunjithapatham. Optimization of task allocation in a cluster-based sensor network. In *Symposium on Computers and Communications (ISCC'03)*, pages 329–334. IEEE Computer Society, 2003.

[416] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[417] C. Zhang, S. Su, and J. Chen. DiGA: Population diversity handling genetic algorithm for QoS-aware web services selection. *Computer Communications*, 30(5):1082–1090, 2007.

[418] W. Zhang, Y. Yang, S. Tang, and L. Fang. QoS-driven service selection optimization model and algorithms for composite web services. In *31st Annual International Computer Software and Applications Conference (COMPSAC'07)*, pages 425–431. IEEE Computer Society, 2007.

[419] R. Zhao and B. Liu. Redundancy optimization problems with uncertainty of combining randomness and fuzziness. *European Journal of Operational Research*, 157(3):716–735, 2004.

[420] T. Zheng and C. M. Woodside. Heuristic optimization of scheduling and allocation for distributed systems with soft deadlines. In *Computer Performance Evaluations, Modelling Techniques and Tools. 13th International Conference, TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 2003.

[421] T. Zheng, C. M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Trans. Software Eng*, 34(3):391–406, 2008.

[422] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Nov. 1999.

[423] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.