
Faculty of Informatics
Masaryk University

# Quality-Driven Architecture Design of Software Systems

Habilitation Thesis
(Collection of Articles)

Barbora Bühnová

January 2016
Brno, Czech Republic

# Abstract

Over the years, software-intensive systems have become an inseparable part of our lives, gradually supporting critical business and industrial processes, such as in enterprise information systems, e-business applications, or industrial control systems. Along the trend of automation of critical software processes, the quality of these software solutions became the key concern to many research teams worldwide. Although techniques for formal quality assessment have long been available, the size and complexity of the present day software systems introduced new challenges motivating the emergence of novel approaches to tackle the complexity via decomposition of the problem along the architecture of the software solutions.

In this thesis, I elaborate on the role of architecture-based quality assessment during system development process. In particular, I focus on the system design phase, when critical architectural design decisions are being made, which strongly affect the software quality of the final product. This text details my contributions towards the quality-driven design of a software system from three different perspectives, which are the functional correctness, reliability and performance, and discusses the techniques to support this aim. Besides the techniques to model and evaluate the quality of a certain system design alternative, specific attention is being paid to challenges and techniques of architecture-based design decision making and automated architecture design optimization.

The thesis is structured as a collection of articles accompanied with a commentary putting my contributions in the context of the state of the art in the area, and linking them to an integrated view on the quality-driven architecture design of software systems.



**Keywords:** Software architecture, software system design, quality, reliability, performance, architecture optimization.

# Abstrakt

V průběhu let se softwarové systémy staly nedílnou součástí našich životů, kdy se stále častěji dostávají na pozadí klíčových obchodních a průmyslových procesů, ať už v rámci podnikových informačních systémů, systémů pro elektronické obchodování či systémů pro řízení výroby. Spolu s trendem automatizace kritických softwarových procesů se do popředí zájmu mnoha výzkumných týmů dostává kvalita těchto softwarových řešení. Ačkoli formální přistupy k řízení kvality jsou k dispozici již řadu let, velikost a složitost dnešních softwarových systémů přináší nové otázky motivující vznik technik pro analýzu takto složitých systémů za pomocí dekompozice problému na základě architektury systémů.

Tato práce se věnuje roli analýzy kvality systému na základě jeho architektury v průběhu procesu vývoje systému. V práci se konkrétně zaměřuji na fázi návrhu systému, kdy jsou přijímána klíčová návrhová rozhodnutí s dopadem na architekturu a následně i kvalitu finálního softwarového produktu. Tato práce přibližuje mé přínosy k procesu návrhu softwarového systému s ohledem na jeho kvalitu, a to ze tří perspektiv, kterými jsou funkční korektnost, spolehlivost a výkonnost, a diskutuje techniky pro podporu tohoto cíle. Vedle technik pro modelování a analýzu kvality konkrétního návrhu systému, je zvláštní pozornost věnována problémům a technikám spojeným s návrhovými rozhodnutími na základě architektury systému a automatické optimalizaci návrhu architektury.

Tato práce je souborem publikovaných vědeckých prací doprovozených komentářem, který mé výsledky zasazuje do kontextu aktuálního stavu výzkumu v této oblasti, a propojuje je do integrovaného pohledu na kvalitou řízený návrh softwarového systému s ohledem na jeho architekturu.

**Klíčová slova:** Softwarová architektura, návrh softwarového systému, kvalita, spolehlivost, výkonnost, optimalizace architektury.

## Acknowledgements

I would like to express my appreciation to all the tremendous mentors I have had along my journey—to Jiří Sochor and Ivana Černá for guiding me during my Ph.D. studies, to Ralf Reussner for helping me to grow as a scientist after I graduated and for becoming one of my greatest inspirations, and to Tomáš Pitner for welcoming me warmly in the LaSArIS lab after I joined FI MU as an assistant professor. I would like to thank all my colleagues, co-authors, and LaSArIS members for being such a great team to work with.

Finally and foremost, I wish to thank my parents, husband and children for being my home, my love, my joy.

*Barbora Bühnová*

# Contents

# Part I

# Commentary

# Chapter 1

# Introduction

The idea of system quality assessment has been around since the 1960s, when research on rigorous methods for analysing complex concurrent systems was initiated. The field has evolved very dynamically and today, there are many methods and techniques available for either manual or automated program analysis and verification [120, 38].

In the context of large-scale business and industrial systems, however, rigorous quality assessment methods started to set up quite recently, together with the emergence of the concept of architecture-based reasoning, which suggests to decompose the complexity of the quality assessment problem along the architecture of the software system [84, 7].

The importance of software architecture within the design and quality assessment of software systems started to be recognized around 1990, when the increasing industrial demand towards software systems with high complexity and challenging quality requirements became apparent. At that time, the emergence of the Component-Based Development (CBD) paradigm [126, 71] initiated discussions on the separation of concerns with respect to the wide-ranging functionality available throughout a given software system. Although the CBD, which suggests to build software systems via composition of existing autonomous components similarly to manufacturing, did not find its way into the mainstream of software engineering, it provided the basis for overly very popular concept of Service-Oriented Architecture (SOA), which relies on very similar foundations. Along with this transformation, the complexity of the systems further increased into the Systems of Systems considerations, and the concept of software architecture became an integral part of both industrial and academic reasoning.

In industry, architecture specifications and models [74] not only structure complex software systems, but also provide a blueprint that is the foundation for later software engineering activities. Thanks to architecture specifications, which identify system components, their communication and mapping to the underlying infrastructure, software engineers are better supported in reasoning about the desired quality attributes of the developed systems [13]. This is where industry meets academia, which aims at developing techniques and tools that can assist software architects in this endeavour.

Initially, the research focused on the compositional reasoning targeting the functional quality of large software systems, which was to large extent reusing and adapting existing results from the formal verification domain [33, 133]. However, it soon became apparent that the attention must be turned also to the non-functional quality criteria, such as reliability, availability or performance, which became widely studied within the software architecture context mainly in the last decade [73, 79].

## 1.1 Focus of the thesis

This thesis summarizes my contributions to the progress within the field of architecture-based quality assessment of software systems and its role in the development process. Over the years when I studied software quality from different perspectives (e.g. functional correctness, reliability, performance, energy consumption) and over different application domains (e.g. enterprise software systems, large control systems, embedded systems), I became to understand that critical decisions influencing the quality of the software product are being made early during the architecture design, which is when software engineers should be supported with adequate techniques and tools that help them to identify the right design alternative, despite high uncertainty about the context that is yet unknown at that time.

This text discusses the techniques that are in line with this goal, aimed at the quality assessment of large-scale systems early during system design, and the support of architecture design based on their results. Specific attention is being paid to three quality perspectives – functional correctness, reliability and performance – and the role of uncertainty within this endeavour. All this is closely linked to my contributions in this field. This text is a follow up of my doctoral thesis [137], which however only targeted the functional perspective on system quality, and understood system design in a narrow scope of component composition, as distinct from the high diversity of architecture-based design activities discussed here.

## 1.2 Thesis structure

An essential precondition for early assessment of software system quality based on its architecture is an adequate model of the analysed software system and its architecture in particular. In Section 2 we therefore outline the state of the art and my contributions within the domain of software architecture models from the three perspectives outlined above, i.e. functionality, reliability and performance. In Section 3, we study the actual techniques for architecture-based quality assessment of individual design alternatives for the particular system. Having the basis to model and evaluate individual design alternatives, Section 4 elaborates on the challenges and techniques of architecture-based design decision making and later also the automated architecture design optimization, which is the ultimate goal within the targeted research area.

Each section presents the state of the art within the particular problem domain, my contributions to its progress, and lists selected articles I have co-authored that are attached to this text to exemplify my contributions. Some of the articles are mentioned more than once, as they contribute to multiple aspects studied in this text. The overall collection of articles is listed in Part II of this thesis.

# Chapter 2

# Software architecture models

Benefits of compositional reasoning are widely understood since the works of Lamport, Pnueli and others in early 80s [83, 110, 11], which is why compositional system models are not new either. Initially, system models have been decomposed into processes, modelled with different variations of Labelled Transition Systems (LTS) and Process Algebras (PA), which became the basis for architecture-based modelling as well.

However, the emergence of Component-Based Development (CBD) [126, 71] was accompanied with new views and concepts, which is why a plethora of modelling approaches has been built since the concept of a software component became more clear and commercial component frameworks, such as Microsoft's COM [102], Sun's EJB [125] and OMG's CCM [108], appeared. A component in this work is understood as a self-contained, replaceable part of a software system that fulfils a clear function or a group of related functions in the context of software architecture [73]. We distinguish two types of components: basic and composite. Each basic component is assumed to be a collection of services provided to other components in the system, each composite component is understood as an assembly of components (basic or composite).

In its early stages, component-based methods targeted mainly the functional quality of software systems under consideration. In order to leverage architecture-based reasoning to build correct and dependable software systems, researchers have developed various formal and semiformal component models, which concentrate on different yet related aspects of component modelling [109, 31, 91, 5, 52]. However, soon after that, architecture-based modelling notations highlighting the non-functional aspects of software system quality (such as reliability, availability, performance or energy consumption) started to be introduced [64, 9, 79, 22, 67].

## 2.1 Functionality

Within the context of functional correctness of systems with component-based architecture, the most popular modelling approaches rely on process algebra, automata-based approaches and architecture description languages, which are briefly described in this section.

**Process algebra.** Process algebras [8, 23, 50] allow a high-level view on interacting systems. The interacting components are regarded as processes, and the interaction among them is defined with a set of operators within an algebraic theory. Process algebras have their operational semantics defined in terms of labelled transition systems. Since the 1980s when the extensive work in the field was initiated by the works of Milner [103] and Hoare [72], a number of different process algebras has been developed. A similarity among them is the definition of the system communication behaviour in terms of process expressions, and their interaction in terms of synchronization primitives. Among the best known representatives, there are Milner's CCS [104] studying interaction on complementary input/output actions; Hoare's CSP [72], offering an alternative view on component communication via multi-synchronization; Bergstra's and Klop's ACP [8] with even a more general communication scheme; and Milner's Pi-calculus [105], based on CCS with support of mobile processes. The most influential representative of the process algebras in the domain of component-interaction modelling, is with no doubt the Calculus of Communicating Systems (CCS) [104].

**Automata-based languages.** The basic constructs defined in CCS, namely the transition interpretation of process semantics and communication on complementary actions, became the fundamental basis of Input/Output-automata-based languages. These languages produce system models directly in terms of transition systems labelled with three types of actions: *input*, *output* and *internal*. As distinct to process algebras, the automata-based languages are usually used in a strictly compositional manner, where only the basic processes (representing the basic components as described above) are constructed explicitly (as transition systems); components on higher levels (i.e. composite components) are created solely via composition of existing automata, in a hierarchical manner. The languages often add a number of restrictions on the components entering the composition, to better adhere to the type of systems the languages are defined for. The best-known languages in this field are the I/O automata [89, 88], its extension Team automata [16, 47], and modification Interface automata [42, 43].

**Architecture description languages.** Concurrently with the evolution of I/O-automata-based modelling languages, the software engineering community put effort in the design of interaction-specification languages within emerging Architecture Description Languages (ADLs). The origination of ADLs was motivated by the practical need of description and formalization of software architectures at the beginning of the 1990s, and was

followed by a second advent in the 2000s triggered by the growing popularity of CBD. Architecture description languages allow us to specify both static (system architecture, bindings among components) and dynamic (component behaviour, interactions) aspects of hierarchical component-based systems, hence they have wider scope than the languages discussed above. In fact, they define specific sub-languages for component-interaction modelling. Among the best known, there are SOFA Behavior protocols [109], Darwin Tracta [91], Wright [6] or Fractal pNets [12].

**Component-Interaction Automata.** In my doctoral thesis [137], I aimed at connecting the formal body of knowledge available within the automata-based languages and the specifics of real-world component-based systems, reflected within architecture description languages, which resulted into the introduction of *Component-Interaction automata* [26, 34, 139], which are an automata-based specification language for modelling component interactions in component-based software systems. In Component-Interaction automata, each component is modelled as an automaton with actions representing points in its communication with the environment, and the system is modelled via composition of the automata representing the components it consists of. The language builds on a simple, yet powerful, composition operator that can be parameterized to simulate several communication strategies used in various component models. In this manner, Component-Interaction automata can be instantiated to a particular component model by fixing the composition operator and semantics of actions, which helped them to become a recognized notation employed by a number of authors worldwide [87, 75, 86].

## Contributions

As the follow up of my doctoral thesis [137], which introduced the Component-Interaction automata notation [26] and a set of equivalence relations that can be employed for the evaluation of safe component integration and replacement [35], my contribution within the context of this text is in the examination of the applicability of the Component-Interaction automata notation in the context of real business and industrial systems [139, 128, 21], and the challenges that emerge during this effort [138, 129, 127].

My aim in this context have been to build a bridge for existing verification techniques to verify realistically large business and industrial systems, and to show how the intricate specifics of component based systems can be modelled with simple labelled transition systems notation, such as the Component-Interaction automata.

The initial set of component-based system specifics, which served as the test base for this effort, was derived from the common component modelling example [115], designed to comprise a large number of various aspects and modelling issues that can be identified in different types of component-based systems. Together with a detailed specification of the example in terms of Java source code (125 Java classes in total), the example originally

served as a common modelling example for the CoCoME (Common Component Modelling Example) Modelling Contest [114], whose aim was to compare practical applicability of existing component modelling approaches and component models on a common component-based system [115, 139].

Specifically, we have defined a mapping of Java methods and events to the actions in the model, and we have identified different types of communication among components that we have realized by the parameterizable composition operator that the Component-Interaction automata provide. During the modelling process, we have faced many modelling issues, which have been detailed both in [139] and a number of our subsequent works [138, 128, 129].

Namely, we studied the modelling of exception handling, including advanced concepts like forwarding of exceptions among method calls [139], and examined the modelling of the internal persistent state of a component and its influence on component's behaviour, as well as the role of the global shared state within the system [139]. We discuss modelling of not only the synchronous communication, but also two common types of asynchronous communication schemes, the point-to-point and publish-and-subscribe schemes [128, 139]. We elaborated on how creation and destruction of component instances can be modelled within a finite-state formalism [138, 127]. And we also touched the restriction of the system model to a given usage profile and its impact on system analysis [21, 139]. In summary, we were successful in demonstrating that even a simple automata-based language, such as the Component-Interaction automata, has the power to faithfully model a number of non-trivial situations common in real component-based software systems.

## Articles in collection

[139] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*, chapter Component-Interaction Automata Approach (CoIn), pages 146–176. Springer, 2008

I devised the modelling process, designed the solutions to the modelling challenges and created all the models. I contributed to the verification process and wrote most of the text. Contribution 60%.

[128] P. Vařeková and B. Zimmerova. Challenge Problem: Subject-Observer Specification with Component-Interaction Automata. In *Proceedings of the ESEC/FSE Conference on Specification and Verification of Component-Based Systems (SAVCBS'07)*, pages 75–81. ACM Press, September 2007

I was responsible for the modelling part of the paper and participated in devising the verification approach. I wrote most of the text. Contribution 50%.

[138] B. Zimmerova and P. Vařeková. Reflecting Creation and Destruction of Instances in CBSs Modelling and Verification. In *Proceedings of the Doctoral Workshop on Mathem-*

*atical and Engineering Methods in Computer Science (MEMICS'07)*, pages 257–264, October 2007

I devised the modelling and part of the verification approach and wrote the paper. Contribution 70%.

[19] N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. The CoIn Tool: Modelling and Verification of Interactions in Component-Based Systems. In *Preproceedings of the International Workshop on Formal Aspects of Component Software (FACS'08)*, pages 221–225. Department of Computer Science, University of Malaga, 2008

I participated in the design of the tool and its algorithms, and paper writing. Contribution 15%.

## 2.2 Reliability

In the realm of software systems, the reliability characterizes the likelihood of failure-free behaviour of a software system, because even a high-quality software development process generally cannot eliminate every fault in the system. Some faults are hard to localize, because of having complex activation and propagation patterns, which may only be revealed at runtime [65]. The role of reliability engineering within the software development process is hence to track the expected system reliability and guide design decisions that reduce the potential negative effect of the faults that may still be present in the system.

**Reliability definition.** The failure behaviour of the system or its components is usually defined as a failure rate, time-dependent failure intensity, or probability of failure on demand [57]. Failure rate is the frequency in which the failures occur. Time-dependent failure intensity is defined as the rate of change of the expected number of failures with respect to time [90]. The probability of failure on demand is understood as $1 - r$ where $r$ is the probability of failure-free operation of a software system for a specified period of time in a specified environment [73]. In general, the failure behaviour can be specified in greater detail by additional parameters, such as the failure dependencies, their durations or latency [24], although such parameters are utilized very rarely.

When discussing the reliability-relevant system properties in this text, we refer to system failures and faults. The term failure is used for any deviation of the system's behaviour from its intended functionality as perceived by a system user. Fault on the other hand refers to any defect in system's behaviour that may but needs not to lead to a failure.

**Software reliability models.** The fundamental approaches in the domain of software reliability engineering are concerned mainly with system tests and reliability growth models,

treating systems as black boxes [107, 90]. Recently, architecture-based reliability approaches, which take advantage of the compositional structure of component-based software systems, have been introduced and their limitations studied by different authors [73, 57, 62].

The architecture of a system is most commonly being modelled with either a discrete time Markov chain (DTMC) [37, 122, 131, 58] or a UML-like modelling notation that is later transformed to a Markovian model [40, 59, 136, 117, 111]. Next to DTMCs, which are together with Markov decision processes (MDP) being employed for the analysis of system reliability under a given terminating usage scenario, continuous time Markov chains (CTMC) and semi-Markov processes (SMP) are better suited for continuously operating software applications, such as real-time control systems [62].

**The role of execution environment.** While the approaches detailed above focus on the software-level modelling only, some authors are beginning to recognize that the reliability of a software system is critically dependent on the reliability of the underlying execution environment, in which the system is deployed, and hence include this aspect in their models to increase the plausibility of the reliability analysis [119, 63, 41, 44].

Execution environment in software systems typically comprises of hardware resources and network infrastructure. The reliability of the hardware resources is being understood via their availability, which may be compromised by resource breakdowns due to hardware wear out. Typically, a broken-down resource, which might be a CPU, memory or storage device, for instance, is eventually repaired or replaced with a new resource. The reliability of such resources is therefore expressed in terms of Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) [121]. Reliability in networks is being defined either as a probability of successful communication between a specified pair of nodes within the network, or as the ratio of correctly delivered data. The reliability is often modelled either with a random variable or with a fixed constant [135, 136].

Although the reliability of the execution environment is well understood, the reasoning about the software reliability in the presence of unreliable execution environment is non-trivial and not yet well resolved, since the nature of the software and hardware reliability is very different and the interplay of the two depends on the probability that the software operation uses a certain hardware resource, which depends on many aspects (e.g. the implementation of involved services, user inputs). This is especially true for systems with complex architecture and deployment settings.

## Contributions

Over the last years, I have contributed to numerous works on software reliability analysis, which relied on architecture-based models of reliability-relevant system properties [29, 27, 97, 28, 30, 17, 96, 95].

Together with Brosch et al., we have identified that two major aspects that disturb the results of reliability analysis in large software systems, are the missing consideration of the execution environment, and overlooked usage profile modelling [29, 28]. Based on these findings, we have developed an approach that integrates these two views into reliability modelling and assessment for large software systems.

Our work builds upon the Palladio Component Model (PCM) [15], originally designed for performance modelling and prediction, and incorporates the notion of software failures, communication link failures, and unavailable hardware into its modelling and analytical capabilities. In the work, we employed a UML-like notation that details the reliability effects on different levels of system architecture, starting with component services and their internal computation, and propagating it through the reliability of individual components to the system level. We considered the influence of system usage and explicitly model the propagation of the usage profile throughout the architecture. Furthermore, the execution environment and software component allocation is also modelled in detail (in a UML-like fashion too), so that the effect of physical-resource breakdowns and network problems can be propagated to the software reliability assessment. Tool support for an automated transformation of the UML-like models into Markov chains and space-effective evaluation of these chains is implemented using the Eclipse Modeling Framework (EMF) within the Palladio Component Model (PCM) Bench tool set [2].

In the approach, we have associated individual software actions and communication links with failure probabilities, and hardware resources with Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) values. Taking these into account, we assume that the execution of a system scenario fails if a software failure happens during component-internal processing, a communication link fails in propagating a call between two components, or a hardware resource is unavailable when required by service execution.

**Fault tolerance.** In [27] we further extended the modelling capabilities of our approach to include various fault-tolerance mechanisms, which are commonly employed to mask faults in software systems and prohibit them to result in a failure. Fault-tolerance mechanisms may be established on different abstraction levels, such as exception handling on the source code level, watch dog and heart beat as design patterns, and replication on the architecture level [113, 106]. This allowed us to reason about the effect of various fault-tolerance mechanisms on the reliability of the system and hence navigate the developer to use them wisely [27], which was a novel contribution, not yet considered by related work [57, 61, 73].

**Estimation of failure model parameters.** While some parameters of the models can be obtained straightforwardly (e.g. MTTF specified by hardware vendor or benchmarked [121]), there are parameters that need to be based on estimates (e.g. failure probabilities of component-internal actions), which is why reliability model accuracy may be questioned, because some parameters are hard to estimate before system implementation. In [32] we

have therefore surveyed existing techniques for estimation and collection of failure parameters for architecture-based reliability assessment models, and presented the findings that can be learned from their detailed analysis. To accomplish this goal, we studied possible reliability-model parameters, failure data sources, methods and analytical capabilities of existing techniques. The techniques have then been analysed and classified according to devised classification scheme.

**Reliability models for embedded systems.** Because of certain specifics of embedded systems comparing to the information software systems targeted in the above work, we have further developed multiple reliability models for embedded systems with complex architectures [96, 95, 97]. Embedded systems, such as avionic or automotive control systems, rely on different concepts, typically driven by sensors and actuators. They are deployed on compact self-contained computational units (containing both processing and memory part), known as Electronic Control Units (ECUs), interconnected into the architecture through communication buses. The software layer of an embedded system consists of a high number of lightweight components, representing the logical blocks of system functionality (typically in a low-level programming language). In our work [96, 95, 97], we have defined modelling abstraction of such systems, presenting the software and hardware architecture models, along with the model of failure behaviour with respect to various reliability assessment problems.

## Articles in collection

[29] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Architecture-based reliability prediction with the palladio component model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, 2012. ISSN 0098-5589

I participated in devising the approach, its evaluation and paper writing. I was responsible for the formal correctness of the whole approach. Contribution 30%.

[27] F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner. Reliability prediction for fault-tolerant software architectures. In *Proceedings of the joint ACM SIGSOFT conference QoSA and ACM SIGSOFT symposium ISARCS on Quality of software architectures (QoSA) and architecting critical systems (ISARCS)*, pages 75–84. ACM, 2011

I participated in devising the approach and defining the fault tolerance mechanisms included in the model, for which I then designed the trafsformation into the formal model and its evaluation. I participated in paper writing. Contribution 30%.

[32] B. Buhnova, S. Chren, and L. Fabrikova. Failure data collection for reliability prediction models: A survey. In *Proceedings of the 10th International ACM Sigsoft Conference on the Quality of Software Architectures*, pages 83–92. ACM, 2014

I participated in all phases of work on the paper, including paper writing. Contribution 35%.

[97] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212

I was mainly responsible for the problem statement and quality assessment model definition and evaluation. I participated in paper writing. Contribution 35%.

## 2.3 Performance

Software performance is one of the essential quality attributes of software systems nowadays, being understood as a collection of more specific attributes, namely the response time, throughput and resource utilization.

Although classical performance models such as queuing networks [85, 46, 99], execution graphs [48, 69], and discrete-time Markov chains [123] are applicable to model performance also in the context of software architectures, specialized models are emerging that take advantage of the benefits of the compositional structure and interplay between both software and hardware components in component-based systems.

These approaches [9, 79] take advantage of the knowledge of the system architecture and model it first with a UML-like notation, which is later transformed to one of the classical performance prediction notations (e.g. queuing networks or Markov chains) so that the available techniques can be used to compute the expected performance. UML is a very good basis for system modelling, as it allows modelling the system architecture with component diagrams, and for each component also its behaviour with sequence, activity, and communication diagrams. Component allocation can be described with deployment diagrams. While UML only supports functional specifications, its extension mechanisms (profiles, consisting of stereotypes, constraints, and tagged values) allow modelling performance attributes such as timing values and workload parameters [79].

The architecture-based performance engineering approaches aim at understanding the performance (i.e., response time, throughput, resource utilisation) of the software system based on the performance properties of individual components and their deployment context [14]. Within the state of the art, well surveyed in [9, 79], one can identify numerous challenges that have been addressed by the community in the recent years. First of all, the performance of a component-based system is influenced by a number of factors, including component implementation, deployment, environmental context and system usage. The component developer has to provide a parameterized specification (i.e. a function over the different factors), which makes the influences by these external factors explicit.

Specifically, for each provided service of a component, the component developer should provide a performance specification that besides the functional control flow (similar to the functional and reliability models) includes the details of resource demand. The demanded

resources are typically CPUs or storage devices, but can also include memory buffers, threads from a pool, etc., and since the execution often demands changing portion of the resource time, it may be specified by different means (e.g. with a mean value, distribution function, best/worst case).

Additionally, similarly to the functional and reliability models, it is critical to well incorporate the usage model into the system model, so that one can reason about various execution scenarios of the system. For the usage profile to be effectively integrated into the performance model, it is important to well resolve the information propagation in the model. Another challenge is that only a few methods deal with modelling the internal state of a component at runtime (analogous to a global state of a system as such), although the internal state can have severe effects on component's performance [79, 70].

## Contributions

I have been involved in numerous research activities related to performance modelling for early performance assessment, in the context of large enterprise systems [70, 77, 76], cloud-based applications [53] and automotive embedded systems [95]. The most relevant to the aim of this text are the works extending an already established performance modelling approach [14] with new performance-relevant details without increasing the model size and complexity over the limit that would hinder its analysis. These included the modelling of component internal state [70, 77] and the automation of model enrichment with low-level performance characteristics, called model completions [76].

**Internal state modelling.** In [70, 77] we studied the dependence of system performance on a configuration, context or history related state of the system, typically reflected with a (persistent) system attribute, and implying the existence of stateful services, stateful components and stateful systems as such. More specifically, we targeted the issue of appropriate balance between the expressiveness and complexity of performance models via an effective abstraction of state modelling. We identified and classified stateful information in component-based software systems, studied the performance impact of the individual state categories, and analysed the costs of their modelling in terms of the increased model size.

**Model completions.** To provide accurate performance predictions, performance models need to include many low-level details that affect the performance (e.g., performance-relevant middleware configurations). Such details can be integrated into architectural models through model refinements (realised by model-driven transformations), called completions [132]. Completions introduce quality-relevant details into the model, for example defined as abstractions of design patterns for a specific purpose, such as concurrency. However, it may be intricate to employ completions automatically, as they may influence each other. In [76] we studied the possible conflicts among performance completions and based on

it proposed a systematic method of conflict localization and resolution reflecting the quality effects of different transformation orders (completion chains).

**Performance models for embedded systems.** Within the scope of automotive embedded systems research, we have been concerned with performance modelling in [95], within which we relied on the specific architectural characteristics of embedded systems (of hardware-software co-design within so called Electronic Control Units, ECUs), which enabled us to study the effect of redundancy allocation (deployment of redundant ECUs to increase system reliability) on system performance. For this purpose, we described the system behaviour as a Markov model with probabilities of execution transfer between components together with the probabilities of execution initialisation at each component, and used the approach of Kubat [82] to estimate the performance.

## Articles in collection

[77] L. Kapova, B. Buhnova, A. Martens, J. Happe, and R. Reussner. State dependence in performance evaluation of component-based software systems. In *Proceedings of the Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE'10)*, pages 37–48. ACM, 2010

I was an author of the main idea, I contributed significantly to the problem statement, state of the art analysis and realized the modelling part of the approach. I wrote the text of most of the paper. Contribution 30%.

[70] L. Happe, B. Buhnova, and R. Reussner. Stateful component-based performance models. *Software and Systems Modeling*, 13(4):1319–1343, 2014. ISSN 1619-1374

I contributed to several parts of the paper, I was responsible for the state identification, modelling and the definition of the heuristics. I wrote most of the text. Contribution 40%.

[76] L. Kapova and B. Buhnova. Performance-driven stepwise refinement of component-based architectures. In *Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems*, pages 1–7. ACM, 2010

I participated in all phases of work on the paper, and was responsible mainly for the employed techniques and the overall correctness. I participated in paper writing. Contribution 40%.

[95] I. Meedeniya, A. Aleti, and B. Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium on Automotive/Avionics Systems Engineering (SAASE'09)*, pages 1–16, 2009

I participated in devising the approach, its models and its formulation in terms of an optimization problem. I participated in paper writing. Contribution 30%.

# Chapter 3

# Architecture quality assessment

Having an architecture-based model of a software system emphasizing a certain quality perspective, the next step is to quantify the quality of the design alternative represented by the model, to allow its comparison with alternative designs.

The methods for quality quantification are typically based on either mathematical analysis of the model or on the simulation of an executable version of the model, and can have many forms, combining different techniques, which can be best understood from available surveys, e.g. for reliability [64], performance [9, 79], energy consumption [22], and safety [67].

## 3.1 Functionality

The functional correctness is being typically understood in terms of functional requirements satisfaction. In case of the architecture-based models, two separate aspects need to be considered, which are the correctness of each basic component, and the correctness of a composite system. The first point of view refers to the functionality that is encapsulated in the basic components, which are the building blocks of the system. The components are usually delivered by third parties without source code or implementation details, which may come with a serious risk of system faultiness. Current solutions to this problem are based mainly on third party certification and third party testing, which results in trusted components [100]. The second point of view, the correctness of a composite system, refers to the fact that even if we assume that each individual component is trusted and can be supposed to be correct, there is no guarantee that the components cooperate correctly to deliver expected results. Thus what remains to be assessed is the interconnection logic among components, which we refer to as component interactions.

**Correctness of a basic component.** In component-based development, components are developed independently of their deployment context. In such settings, the correctness of a component can be hardly defined, because a component can behave correctly in one context but incorrectly in another. Existing techniques [101, 100] instead propose to assign each component a certified description of its properties, including sequences of allowed service calls and their effects, so that each user of a component can decide whether the component behaves correctly in their context. Or alternatively, components come with a set of property qualities that are guaranteed in all the contexts that have an expected behaviour [134, 39].

**Correctness of a composite system.** Whenever the components are assembled into a composite system, automated formal methods can be employed to analyse the composite system as a whole. Formal analysis can include verification of coordination errors, like deadlocks, computational progress or fairness, or checking various temporal properties on the system, such as the interaction among specific components [38, 120]. We can for instance ask whether a component always serves given calls in a given order, or if two components can access a given service at the same time. We can also focus on verification of component-specific errors, like local deadlocks of selected components [21]. The verification can be carried out with traditional verification methods, like model checking or theorem proving [38, 120], or can take advantage of component-specific characteristics of systems, like the compositionality in the assume-guarantee manner [134, 54]. Compositional verification in general aims at decomposing a global system property specification into local properties that hold for small sub-parts of the system [33, 133]. In case of component-based systems, a system-level requirements are decomposed into local component requirements, which are verified on the components in isolation [25, 134]. The isolated validity then guarantees the validity of the system-level property on the whole system.

**Assessment complexity.** In practice, real systems are composed of a large number of concurrent components, which are often independent of each other. In such a case, automated verification becomes challenging due to model size and complexity. This motivates the search of component-specific attributes, which can be exploited in order to make the verification feasible. One of the techniques successfully employed to state-space reduction is the partial order reduction. This technique is able to identify redundancies in the model during the verification process, commonly caused by interleaving of independent actions. This allows the technique to omit the generation of some of them while at least one representative of each equivalence class remains part of the actually verified model [55].

## Contributions

Having a realistic architecture-based model of a large software systems modelled in Component-Interaction automata [139], as discussed in Section 2.1, the aim of our sub-

sequent work was to devise techniques for efficient verification of a reasonable set of properties that would be capable of describing realistic functional requirements related to system architecture and component interaction in the modelled system.

**Correctness properties.** For the specification of system requirements that shall be verified on system models, we have introduced an extended version of the action-based linear time logic LTL, called CI-LTL [21], together with the identification of a set of properties defining correctness issues in component-based systems, their formalization in terms of CI-LTL temporal logic, and demonstration of the feasibility and efficiency of their automated verification in parallel settings [139, 21]. In [18], we have further experimented with the state/event LTL [36, 1], which reflects that although communication among components proceeds via events, which represent message passing, service calls, delivery of return values, etc., components may at the same time preserve also persistent state information about current values of their attributes. For instance in [21] we defined and analysed two basic properties describing the broadcasting ability of the event-channel components, three properties concerning the possibility of a local deadlock, two properties addressing the component blocking problem, and two properties dealing with the problems caused by cycles in the model. Besides these we discuss how the verification helped us to check the validity of the model during modelling.

**Verification and model complexity.** For the verification itself we have used the automata-based model checking algorithms implemented in the model checking tool DiVinE [10, 45] and our custom model-checking tool CoIn Tool [19] designed specifically for formal verification of interactions among components in hierarchical component-based systems. As distinct from existing verification frameworks, the CoIn Tool is able to analyse complex hierarchical models on the fly, and to verify linear temporal properties involving both state and action propositions. DiVinE, on the other hand, implements parallel model checking algorithms, which are effective in case of tremendous model size given by concurrency of components in the system. Furthermore, besides parallel verification, we have experimented with other approaches to tackle the complexity of the verification tasks.

**Partial order reduction.** One of the crucial observations we made during the verification is that the correctness attributes often highlight interaction among specific components, which form only a small part of the system. Even if the rest of the system is also important as it may coordinate these components, with appropriate reduction techniques a large portion of its complexity can be abstracted away during verification. In [18, 20] we have developed a partial order reduction method for state/event LTL, and for action-based LTL as its special case. The core of the partial order reduction is a novel notion of stuttering equivalence, which we call state/event stuttering equivalence. The positive attribute of the equivalence is that it can be resolved with existing methods for partial order reduction.

**High number of component instances.** Another reason for enormous model size is the occurrence of large number of component instances of the same type in the software architecture. To tackle this issue, we have developed a verification techniques for checking LTL-like interaction properties on the specific type of systems with a certain component occurring in many (possibly unknown number of) instances. The method is based on computing a cut-off on the number of component instances under study, such that if the system is proved to be correct for every number of instances up to the cut-off, it is guaranteed to be correct for any larger number of component instances [129, 127].

## Articles in collection

[129] P. Vařeková, B. Zimmerova, P. Moravec, and I. Černá. Formal Verification of Systems with an Unlimited Number of Components. *IET Software journal*, 2(6):532–546, 2008. ISSN 1751-8814

I participated in devising the approach and formulating the proofs, together with its evaluation and considerable part of paper writing. Contribution 25%.

[18] N. Beneš, L. Brim, B. Buhnova, I. Černá, J. Sochor, and P. Vařeková. Partial order reduction for State/Event LTL with application to Component-Interaction Automata. *Science of Computer Programming*, 76(10):877–890, 2011. ISSN 0167-6423

I participated in devising the approach, its evaluation and paper writing. Contribution 20%.

[139] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*, chapter Component-Interaction Automata Approach (CoIn), pages 146–176. Springer, 2008

I devised the modelling process, designed the solutions to the modelling challenges and created all the models. I contributed to the verification process and wrote most of the text. Contribution 60%.

[21] N. Beneš, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. A Case Study in Parallel Verification of Component-Based Systems. In *Proceedings of the Workshop on Parallel and Distributed Methods in verifiCation (PDMC'08)*, ENTCS, pages 35–51. Elsevier Science Publishers, March 2008

I was responsible for the modelling part of the case study and participated in the design of the properties for verification. I wrote relevant parts of the paper. Contribution 25%.

## 3.2 Reliability

The initial works on software reliability assessment [107, 90] were based on system tests and reliability growth models, treating systems as black boxes. As opposed to them, architecture-based reliability assessment approaches revealed the internal system architecture and introduced new techniques that take advantage of it [61, 57, 73, 118, 116].

Thanks to the so called white-box view, the architecture-based reliability analysis can be successfully employed not only for (early) reliability prediction, but more importantly to identify critical software components, quantify their influence on the overall system reliability, and optimise future testing activities [81]. Early quantification of expected system reliability allows software engineers to assess the reliability impact of their design decisions (e.g. the effect of different fault-tolerance mechanisms) and increases the maturity of the software engineering process.

Therefore, although the actual quantification of expected software reliability is not very novel, as it relies on existing algorithms for computing the probability of reaching a certain set of states (the failure states, in fact) within a Markovian model (which is the prevalent notation in this case, as discussed in Section 2.2), there is significant body of knowledge in the state of the art in transforming various reliability-related research questions into the Markovian settings [61, 57, 73, 118].

Moreover, the challenge in this case is to make the reliability assessment models as accurate as possible, without making them too complex for the analysis. That is why many approaches neglect the importance the execution environment or system usage, as discussed in Section 2.2, or consider very coarse-grained system descriptions, assuming the reliability of individual software components as given, and only considering the system level [37, 130], which strongly impairs their accuracy.

### Contributions

Our work on the architecture-based reliability assessment targeted mainly an efficient computation of the failure-free system operation in the presence of many system details that are included in the reliability models (as discussed in Section 2.2), and the identification of the critical system elements with respect to reliability.

**Execution environment and usage profile.** Since our main aim was to explicitly consider and integrate the execution environment and usage profile into an efficient software reliability assessment, we developed a technique that offers usage-profile separation and propagation through the concept of parameter dependencies [80] and accounts for hardware unavailability through reliability evaluation of service execution under different hardware availability states. The combined consideration of the software, hardware, and network di-

mensions within our approach enables the reflection of their interplay in the context of the overall architecture and system usage profile, which respects the following:

- Unavailable hardware resources and failing communication links only impact the system's reliability if they are actually required by the service execution.
- Software faults only lead to failures if the respective parts of the implementation are actually executed under a given usage profile.
- The failure potential of all three dimensions depends on the control and data flow through the architecture, which is captured by the component behaviour specifications and the architectural model.

**Reliability assessment.** The reliability solver, which has been implemented as an Eclipse plug-in integrated into the Palladio Component Model (PCM) Bench [2], takes a fully specified model instance as an input and calculates system reliability as an output. Optionally, a more detailed output differentiates the determined failure potential according to user-defined failure types. Furthermore, stop conditions can be defined for fast evaluation of systems with many hardware resources, at the cost of prediction accuracy. Such stop conditions define a maximal number of evaluated physical system states, a maximal solving time, or a minimal reached numerical accuracy.

**Sensitivity analysis.** We have studied techniques to determine the software components that are strongly affecting system reliability, and examined the techniques to understand the sensitivity of the system reliability to the reliability of such components. For this purpose, we have also enriched our tool support with the sensitivity analyses over varying user-defined parameters of the models [29, 28, 27].

**User-defined reliability.** As far as the reliability is understood solely as the probability of avoiding failure states in the system model execution, the simple Markov-based computation techniques are sufficient. In [17], we elaborated how the situation changes if we want to include different user-defined reliability attributes, such as that *no critical and at most one non-critical fault* occurs during system execution. Our approach in [17] was based on the probabilistic model checking of Markov decision processes (MDP), and the reliability assessment criteria formalized in the probabilistic linear temporal logic (PLTL). To increase the usability of the approach, we introduced the intermediate modelling notation of communicating automata with probabilistic transitions (CAWPTs) and discuss the incorporation of various types of behavioural uncertainties into CAWPT models.

**Reliability assessment for embedded systems.** The reliability assessment problems we have studied in case of embedded systems included the problem of balancing the reli-

ability of various system services [97], which in the case of automotive application studied in [97] included the Anti-lock Brake System (ABS), Adaptive Cruise Control (ACC) and the Airbag service, or to balance the reliability (being increased by introduction of redundant components) with the energy consumption of the system [96], its performance and cost [95], which are all impaired along the reliability improvement.

## Articles in collection

[29] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Architecture-based reliability prediction with the palladio component model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, 2012. ISSN 0098-5589

I participated in devising the approach, its evaluation and paper writing. I was responsible for the formal correctness of the whole approach. Contribution 30%.

[30] F. Brosch and B. Zimmerova. Design-time reliability prediction for software systems. In *Proc. International Workshop on Software Quality and Maintainability (SQM'09)*, pages 1–5, 2009

The paper presents the first version of the approach, on which I participated with a fair share. Contribution 50%.

[17] N. Benes, B. Buhnova, I. Cerna, and R. Oslejsek. Reliability analysis in component-based development via probabilistic model checking. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 83–92. ACM, 2012

I was an author of the idea and participated strongly in all phases of the work, with an emphasis on properties definition and uncertainty encoding into the model. Contribution 40%.

[97] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212

I was mainly responsible for the problem statement and quality assessment model definition and evaluation. I participated in paper writing. Contribution 35%.

[96] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *Proceedings of the Sixth International Conference on the Quality of Software Architectures*, volume 6093 of *LNCS*, pages 52–67. Springer, 2010

I contributed significantly to the problem statement, model formalization and design alternative evaluation. I participated in paper writing. Contribution 30%.

## 3.3 Performance

Since 2000, researchers have proposed many approaches for evaluating system performance (i.e., response time, throughput, resource utilisation) of software systems with component-based architectures [79, 9]. The central goal of these approaches is besides the performance measurement of an existing system, the performance prediction for a system under design, i.e. before its actual implementation. The benefits of the design-time performance prediction include not only the prevention of costly software architecture re-design, but also the possibility to examine system performance under various hardware configurations, which can be simulated or described mathematically instead of being physically deployed.

In the recent surveys [79, 9], one can see that the overall trend of early performance assessment is very similar for many approaches, which only differ in their modelling capabilities and prediction effectiveness and efficiency. The systems under study are first modelled with a UML-like notation that describes both the software and hardware architecture of the system, labelled with performance-relevant characteristics. Such a model is then transformed into a formal model, mostly a kind of queuing networks or Markov chains [9, 79], which are then solved with standard methods. One of the most complete representatives of early performance prediction approaches for component-based architectures is the Palladio approach, based on the Palladio Component Model (PCM) [15], which we describe in more detail.

**The Palladio approach.** The Palladio approach [15] to architecture-based design-time performance prediction is tailored for a separation of four developer roles, who contribute to building the performance models. Component developers model the performance properties of component services with annotated control flow graphs, which include resource demands and required service calls, being parameterized with input and output parameter values. Resource demands can be specified using general distribution functions. Software architects compose these component performance models into component assemblies. System deployers model the deployment environment as well as the allocation of components to hardware resources. Finally, domain experts model the usage profile (including workload, user flow, and input parameters).

The PCM-Bench tool [2], implemented as a set of Eclipse plugins, allows graphical modelling of various performance-relevant aspects of a software architecture in a UML-like notation. Model transformations are then employed to automatically enrich the models with low-level performance-relevant middleware features (e.g., connector protocols, message channel properties), and the full model is then mapped into formal performance models based on queuing networks, which are solved by simulation or numerical analysis. There is also support to generate code stubs and performance prototypes from the models.

## Contributions

Besides my contributions to the Palladio Component Model (PCM) in terms of reliability extensions, described in Section 3.2, I have participated also in its performance extensions related to component internal state inclusion in the performance analysis [70, 77], and the automated model completion with low-level performance characteristics [76], both discussed already in Section 2.3.

**Component internal state.** Besides the modelling approach to include state information into performance prediction models, discussed in Section 2.3, we have implemented an extension to the Palladio approach, which allowed us to experimentally evaluate the increase in performance prediction accuracy and model size costs associated with including state information in performance models [70, 77]. Based on these experiments, we have introduced and evaluated a number of heuristics summarising the advices to software engineers, helping them to competently decide on the appropriate abstraction of state modelling, because the state information inclusion is only recommended if the increase in prediction accuracy justifies the increase in model complexity.

**Model completions.** Besides the study of possible conflicts among the model completions (enriching performance models with low-level details) as discussed in Section 2.3, we have in [76] also introduced a method of conflict localization and resolution. The approach optimises the completion order, based on its validity and performance-relevant impact. The introduced method addresses not only the automatic unambiguous transformation order specification for individual model elements, but also the order of processing the elements. This way, we provide a software architect with a performance semantics to decide on the optimal execution order of given completions, with respect to their performance impact and implied model validity. To verify the introduced method, we implemented the method upon the Palladio Component Model (PCM) and its capabilities of performance prediction.

## Articles in collection

[77] L. Kapova, B. Buhnova, A. Martens, J. Happe, and R. Reussner. State dependence in performance evaluation of component-based software systems. In *Proceedings of the Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE'10)*, pages 37–48. ACM, 2010

I was an author of the main idea, I contributed significantly to the problem statement, state of the art analysis and realized the modelling part of the approach. I wrote the text of most of the paper. Contribution 30%.

[70] L. Happe, B. Buhnova, and R. Reussner. Stateful component-based performance models. *Software and Systems Modeling*, 13(4):1319–1343, 2014. ISSN 1619-1374

I contributed to several parts of the paper, I was responsible for the state identification, modelling and the definition of the heuristics. I wrote most of the text. Contribution 40%.

[76] L. Kapova and B. Buhnova. Performance-driven stepwise refinement of component-based architectures. In *Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems*, pages 1–7. ACM, 2010

I participated in all phases of work on the paper, and was responsible mainly for the employed techniques and the overall correctness. I participated in paper writing. Contribution 40%.

# Chapter 4

# Design process support

The design of a software system, which is in case of a component-based system essentially based on the architecture design and its refinement, is considered to be one of the most important activities in a software engineering project [13]. The decisions made during architecture design have significant implications for economic and quality goals, which is why software architects should be supported with adequate techniques and tools that help them to make the right design decisions. Examples of architecture-level decisions include the selection of software and hardware components, their replication, the mapping of software components to available hardware nodes, and the overall system topology.

After the study of design-time quality assessment approaches in Chapters 2 and 3, which summarized techniques for modelling and evaluation of individual design alternatives, this section details the challenges and techniques of architecture-based design decision making and automated architecture design optimization, which is the ultimate goal within this research area.

## 4.1 Design decision making

To support fundamental architectural design decisions early in the development process, model-based quality assessment techniques discussed earlier can be employed to both evaluate the quality of system design from the perspective of a specific quality attribute (such as reliability or performance), and to identify the critical elements in the architecture that strongly affect the quality attribute. Multiple surveys of these approaches exist, for instance [61, 57, 73, 118] for reliability or [79, 9] for performance.

Then starting from an initial architecture model, the software architect can evaluate multiple options for architecture improvement, possibly from different perspectives, and choose the most beneficial one. This process may be repeated to stepwise improve the architec-

ture until a sequence of changes converts into another architecture that satisfies existing quality and cost goals. When considering architectural models discussed in Chapter 2, the improvement can for instance be modelled with a topological change of the architecture, by changing component deployment, by replacing or replicating a specific software or hardware component, or by adjusting values used for model annotation (e.g. by decreasing a failure rate of a certain action and agreeing that the action will be implemented in a way that confirms to it).

**Conflicting quality views.** Along the decision process, it is critically important that the software architect is supported with evaluation methods for multiple quality views that need to be incorporated in the design, because if for instance the design was only evaluated from the reliability perspective, the performance of the design could easily drop below acceptable threshold, because of conflicts among these quality attributes (i.e. reliability-increasing techniques usually decrease performance). That is why many techniques try to evaluate multiple quality criteria at once, being then confronted with the issue of choosing the right design from a set of hardly comparable alternatives (e.g. if one is better in performance, the other in reliability, it is hard to say which one is more beneficial overall) [4].

**Quality improvement tactics.** To better guide the software architect along the design process and prevent an evaluation of an enormous number of design alternatives, various architectural tactics have been introduced [13, 78]. Generally, these tactics are designed to improve a specific quality attribute, but often declare the additional cost in terms of degrading the architecture with respect to other quality attributes. It is the task of the software architect to evaluate various solutions and determine a good trade-off between all existing quality and cost goals.

**Model changes and parameterization.** One of the essential prerequisites of an approach to be usable for the design decision support is an efficient support for model changes. The notations need to be designed in a way that allows easy model configuration and parameterization that is then automatically propagated to the formal model behind it. That is why it is very popular to model the system in a UML-like notation first and then run an automated transformation into a formal model [79]. In practice, it for instance means that various system information is propagated throughout the architecture model in terms of parameters and in effect influences different formal-model annotations (such as transition probabilities) that are in case of a parameter change generated automatically and do not need to be adjusted manually.

**Model uncertainties.** One of the reasons why it is challenging to identify the right design alternative early during system development is that at design time, uncertainties about the exact functionality of the system as well as the expected system context and

usage significantly affect the accuracy of the quality assessment. The prevalent approach to acknowledge uncertainties in quality assessment models is to replace point estimates of uncertain model parameters with confidence intervals, to determine the risk associated with the estimate [60, 56, 49]. Meedeniya et al. [98] additionally introduce a simulation-based approach that accommodates diverse parameter-range distributions, and reports the desired percentiles of the reliability values.

## Contributions

Along the work I contributed to (whether in terms of functionality, reliability or performance), we always put high emphasis on the compositional structure of the models, with model parameters encapsulating critical design changes. This allowed us to make our techniques ready for the design process support, which we explicitly discussed in numerous research contributions [29, 28, 27, 17, 139]. These works elaborate on the evaluation of various quality perspectives, whether in terms of quality attributes, such as functional correctness [139, 21], reliability [29, 97], performance [70, 76], or energy consumption [96], but also on varying aspects of the quality attributes, such as in [17] where we studied different custom-defined reliability properties of the system. Besides the assessment of system quality, we also focused on the identification of the critical architectural elements with high impact on system quality, so that they can be given special attention during system development [29], or evaluated the effect of quality improvement constructs [27]. Furthermore, the Palladio Component Model (PCM) [15], which we used as the basis for much of our work, allowed us to support distributed design process with multiple developer roles who can independently specify their respective parts of the architecture and contribute with different views to autonomous model parts, which can then be integrated automatically for the purpose of quality assessment [29].

**Model changes and parameterization.** Within much of our work, mainly related to reliability and performance [29, 28, 27, 70, 77], we relied on a highly parameterized UML-like model based on the PCM, which facilitated transparent evaluation of architectural design options, as it covers the propagation of the system usage profile throughout the architecture, and the impact of the execution environment, which are neglected in most of the existing approaches. Before analysis, the model is automatically transformed into a formal model in order to support effective analytical techniques to be employed. The extensive parameterization of our model allowed for sensitivity analysis in a straightforward way, which for instance helped us to examined the sensitivity of system reliability to individual failure probabilities, variations in the system-level usage profile, and changing hardware availability due to wear out effects [29]. This capability not only enabled us to identify quality-critical model parameters, but also allowed us to detect the parameters, whose uncertainty needs to be reduced by further investigation to prevent the distortion of quality assessment results [28, 29].

**Model accuracy.** There are many different types of parameters commonly present in quality assessment models. In [32] we have surveyed possible parameter types in architecture-based reliability assessment models and methods of their estimation, together with the artefacts that can be used for this purpose (e.g. previous system versions, similar systems, design-time artefacts). Moreover, to make the models more precise, we in [76] studied so called model completions, which are meant for automated enrichment of quality prediction models with low-level details, which would be too laborious to add manually.

**Model uncertainties.** Besides the employment of sensitivity analysis to evaluate the role of uncertainty in architectural design decisions [29], we have in [17] investigated the employment of Markov decision processes (MDP) as the formalism well suited for the incorporation of model uncertainties in terms of nondeterministic (scheduler-decided) choices in the MDP to model the uncertainty in component behaviour on various levels (choices in possible behaviour and structural generalization of uncertain behavioural patterns).

## Articles in collection

[28] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Parameterized reliability prediction for component-based software architectures. In *Proc. 6th Int. Conf. on the Quality of Software Architectures (QoSA'10)*, volume 6093 of *LNCS*, pages 36–51. Springer, 2010

My contribution was mainly in the inclusion of failure parameters in the model and its transformation into the formal model, together with its evaluation. Besides that I participated in the whole approach, as well as in paper writing. Contribution 30%.

[29] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Architecture-based reliability prediction with the palladio component model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, 2012. ISSN 0098-5589

I participated in devising the approach, its evaluation and paper writing. I was responsible for the formal correctness of the whole approach. Contribution 30%.

[27] F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner. Reliability prediction for fault-tolerant software architectures. In *Proceedings of the joint ACM SIGSOFT conference QoSA and ACM SIGSOFT symposium ISARCS on Quality of software architectures (QoSA) and architecting critical systems (ISARCS)*, pages 75–84. ACM, 2011

I participated in devising the approach and defining the fault tolerance mechanisms included in the model, for which I then designed the trafsformation into the formal model and its evaluation. I participated in paper writing. Contribution 30%.

[17] N. Benes, B. Buhnova, I. Cerna, and R. Oslejsek. Reliability analysis in component-based development via probabilistic model checking. In *Proceedings of the 15th ACM*

*SIGSOFT symposium on Component Based Software Engineering*, pages 83–92. ACM, 2012

I was an author of the idea and participated strongly in all phases of the work, with an emphasis on properties definition and uncertainty encoding into the model. Contribution 40%.

[32] B. Buhnova, S. Chren, and L. Fabrikova. Failure data collection for reliability prediction models: A survey. In *Proceedings of the 10th International ACM Sigsoft Conference on the Quality of Software Architectures*, pages 83–92. ACM, 2014

I participated in all phases of work on the paper, including paper writing. Contribution 35%.

## 4.2 Architecture optimization

Due to increasing system complexity, software architects are often facing an overwhelming number of design alternatives when searching for an optimal architecture design with respect to defined quality attributes and constraints. This results in a selection problem that may be beyond human capabilities and makes the architectural design very challenging [68]. The need for automated exploration of the design alternative space has already been recognized in existing literature [112] and numerous architecture optimization approaches have been developed for this purpose. This section summarizes our findings on the architecture optimization state of the art, which resulted from the systematic literature review on architecture optimization that we performed on 188 different approaches [4], and outlines our contributions within this context.

**Design changes.** To handle the complexity of the task, the optimization approaches restrict the variability of architectural decisions, optimizing the architecture by modifying one of its specific aspects, where the allocation, hardware replication, and hardware selection are the most intensively studied design changes [4]. However as the popularity of these design changes depends strongly on the purpose (what quality attribute is being optimized), research approaches to their optimization are scattered across many research communities, with similar approaches being proposed by multiple research teams without being aware of each other [4].

**Quality attributes.** It became evident during the study [4] that some quality attributes are addressed more frequently than others. Examples of frequently addresses quality attributes are performance, cost, and reliability. Other quality attributes that are harder to quantify, such as security, are not considered very often. Since quality attributes are often in conflict with each other, many approaches consider multiple quality attributes during the optimization. Among the quality attributes studied together, the combinations reliability/performance, reliability/cost, availability/cost, and cost/energy consumption seem to

receive the greatest attention. The majority of architecture level optimization approaches have been applied in the embedded systems domain, while a comparatively low number of approaches have been applied to enterprise information systems.

**Constraints.** A major influencer on the architecture of software systems are constraints that need to be satisfied in order for the system to be accepted (such as the cost or performance limits). However, our study [4] revealed that a high number of papers solve the architecture optimization problem without considering any constraints. It is important to note that constraint satisfaction is a crucial aspect of optimization, especially in the design of embedded system. However, it is true that constraints add more complexity to the problem, which may be the reason for ignoring them, even if their later reflection can be very laborious.

**Optimization strategy.** Both approximate and exact approaches are being employed for the architecture optimization problem. When the search time and resources used to perform the optimization process are limited, then approximate algorithms are the right optimization approach, although they reveal only near-optimal solutions. On the other hand, if the goal is to find the optimal solutions and the resources and time are unlimited, then one may choose exact optimization algorithms. Due to the complexity of the problem in realistic settings, the vast majority of approaches use approximate methods (mostly metaheuristics) as an optimization technique [4].

## Contributions

Besides the architecture-optimization survey [4], which provided a consolidated view on existing research efforts in the area of architecture optimization and helped researchers to identify future research directions, we have focused on numerous specific architecture optimization problems [97, 95, 96], some of which are described here.

In all the three works [97, 95, 96], we have applied architecture optimization within the domain of embedded systems, which have certain specifics. Namely, they consist of a high number of lightweight components, representing the logical blocks of system functionality (typically in a low-level programming language), deployed on compact self-contained computational units, known as Electronic Control Units (ECUs). For the architecture optimization approaches defined for the enterprise information systems upon the Palladio Component Model (PCM) see [92, 93].

**Reliability optimization via deployment changes.** In [97] we addressed the problem of reliability-driven component deployment for embedded systems, and the automotive domain in particular. In the paper, we have defined a modelling framework to express the important attributes of embedded systems with respect to reliability, and designed a method

to find the set of near-optimal deployments, optimizing the conflicting objectives of system services (like the ABS, the ACC, or the Airbag service in an automotive case study). First, we have formalized the propagation of hardware level and software level properties to the reliability evaluation model. To solve the optimization problem, we employ one of the most popular algorithms for addressing the deployment problem, Genetic Algorithm (GA) [94, 51, 66].

**Performance/reliability/cost optimization via redundancy allocation changes.** Redundancy allocation, which is in fact the employment of redundant components in the system, is a widely used method for the reliability improvement in complex embedded systems. The allocation of redundant components can however affect other non-functional quality attributes of the system, which are as important as reliability and very often conflicting with each other. In [95] we have developed a multi-objective optimization method based on the Ant Colony Optimisation algorithm, to identify the right design options balancing reliability together with system cost and response time.

**Reliability/energy consumption optimization via redundancy allocation changes.** In [96] we have studied the effects of redundancy allocation in the context of trade-off between reliability and energy consumption in embedded systems, which are both affected significantly by the allocation of redundant components in the system. To achieve this aim, we first identified the main reliability- and energy-relevant attributes of distributed embedded systems with respect to the redundancy allocation problem. Then we formalized the system model in terms of an annotated Markov Reward Model, formulated the optimization problem, and designed an algorithm to resolve it. Specifically, we employed the Non-dominated Sorting Genetic Algorithm (NSGA) [124], which has shown to be robust and have good performance in the settings related to ours.

**ArcheOpterix framework.** The tool support for all the three discussed approaches [97, 95, 96] has been implemented as a part of the ArcheOpterix framework [3], developed with Java and Eclipse. The ArcheOpterix tool provides a generic platform that can be used to specify, evaluate and optimize embedded system architectures. ArcheOpterix provides a set of features to specify the embedded-system model, to check for constraint satisfaction, evaluate the model, and find the set of near-optimal deployments using various approximate optimization algorithms.

## Articles in collection

[4] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 5(39):658–683, 2013. ISSN 0098-5589

I participated in all phases of work on the paper, including paper writing. Contribution 20%.

[95] I. Meedeniya, A. Aleti, and B. Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium on Automotive/Avionics Systems Engineering (SAASE'09)*, pages 1–16, 2009

I participated in devising the approach, its models and its formulation in terms of an optimization problem. I participated in paper writing. Contribution 30%.

[97] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212

I was mainly responsible for the problem statement and quality assessment model definition and evaluation. I participated in paper writing. Contribution 35%.

[96] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *Proceedings of the Sixth International Conference on the Quality of Software Architectures*, volume 6093 of *LNCS*, pages 52–67. Springer, 2010

I contributed significantly to the problem statement, model formalization and design alternative evaluation. I participated in paper writing. Contribution 30%.

# Chapter 5

# Conclusion

In this text, I have presented my research contributions to the progress within the area of quality-driven architecture design of software systems, and put these in the context of the state of the art in the area. In this commentary, the individual research contributions were presented as pieces of one puzzle, which spans across numerous steps leading to the architecture-design support driven by various quality perspectives, which are the functional correctness, reliability and performance in case of my work. The individual research contributions were accompanied with selected representative articles I have co-authored, which are also attached to this text[1].

[1] The fulltexts of the articles are excluded from the public version of this text to avoid copyright violation.

# Bibliography

[1] Concurrent software verification with states, events, and deadlocks. *Formal Aspects of Computing*, 17(4):461–483, 2005.

[2] PCM: Palladio Component Model. URL http://www.palladio-approach.net, May 2011. Last retrieved 2016-01-01.

[3] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *Model-based Methodologies for Pervasive and Embedded Software (MOMPES)*, pages 61–71. ACM and IEEE Digital Libraries, 2009.

[4] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 5(39):658–683, 2013. ISSN 0098-5589.

[5] R. J. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon University, School of Computer Science, USA, May 1997.

[6] R. J. Allen and D. Garlan. The Wright Architectural Specification Language. Technical Report CMU-CS-96-TBD, Carnegie Mellon University, School of Computer Science, USA, September 1996.

[7] C. Atkinson, C. Bunse, H. G. Gross, and C. Peper, editors. *Component-Based Software Development for Embedded Systems – An Overview of current Research Trends*. Springer-Verlag, 2005. ISBN 3-540-30644-7.

[8] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, UK, 1995. ISBN 0-521-40043-0.

[9] 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.

[10] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimecek. DiVinE – A Tool for Distributed Verification. In *Proceedings of the Computer Aided Verification conference (CAV'06)*, pages 278–281. LNCS Berlin, Germany, August 2006.

[11] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proc. of the 16th ACM Symposium on Theory of Computing*, pages 51–63. ACM, 1984.

[12] T. Barros. *Formal Specification and Verification of Distributed Component Systems.* PhD thesis, Université de Nice – Sophia Antipolis, France, November 2005.

[13] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison-Wesley, second edition, 2003.

[14] S. Becker, L. Grunske, R. Mirandola, and S. Overhage. Performance prediction of component-based systems. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 169–192. Springer, 2006.

[15] 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, January 2009.

[16] M. Beek, C. Ellis, J. Kleijn, and G. Rozenberg. Synchronizations in Team Automata for Groupware Systems. *Computer Supported Cooperative Work — The Journal of Collaborative Computing*, 12(1):21–69, February 2003.

[17] N. Benes, B. Buhnova, I. Cerna, and R. Oslejsek. Reliability analysis in component-based development via probabilistic model checking. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 83–92. ACM, 2012.

[18] N. Beneš, L. Brim, B. Buhnova, I. Černá, J. Sochor, and P. Vařeková. Partial order reduction for State/Event LTL with application to Component-Interaction Automata. *Science of Computer Programming*, 76(10):877–890, 2011. ISSN 0167-6423.

[19] N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. The CoIn Tool: Modelling and Verification of Interactions in Component-Based Systems. In *Pre-proceedings of the International Workshop on Formal Aspects of Component Software (FACS'08)*, pages 221–225. Department of Computer Science, University of Malaga, 2008.

[20] N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. Partial order reduction for state/event ltl. In *Proceedings of the International Conference on Integrated Formal Methods (IFM'09)*, volume 5423 of *LNCS*, pages 307–321. Springer, 2009.

[21] N. Beneš, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. A Case Study in Parallel Verification of Component-Based Systems. In *Proceedings of the Workshop on Parallel and Distributed Methods in verifiCation (PDMC'08)*, ENTCS, pages 35–51. Elsevier Science Publishers, March 2008.

[22] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration Systems*, 8(3):299–316, 2000.

[23] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Publishers, 2001. ISBN 0-444-82830-3.

[24] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software & Systems Modeling*, 10(3):313–336, 2011.

[25] C. Blundell, D. Giannakopoulou, and C. S. Pasareanu. Assume-Guarantee Testing. In *Proceedings of the conference on Specification and Verification of Component-Based Systems (SAVCBS'05)*, pages 7–14. ACM Press, September 2005.

[26] L. Brim, I. Černá, P. Vařeková, and B. Zimmerova. Component-Interaction Automata as a Verification-Oriented Component-Based System Specification. *ACM SIGSOFT Software Engineering Notes*, 31(2):1–8, March 2006. SESSION: Specification and Verification of Component-Based Systems (SAVCBS 2005), Article No. 4.

[27] F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner. Reliability prediction for fault-tolerant software architectures. In *Proceedings of the joint ACM SIGSOFT conference QoSA and ACM SIGSOFT symposium ISARCS on Quality of software architectures (QoSA) and architecting critical systems (ISARCS)*, pages 75–84. ACM, 2011.

[28] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Parameterized reliability prediction for component-based software architectures. In *Proc. 6th Int. Conf. on the Quality of Software Architectures (QoSA'10)*, volume 6093 of *LNCS*, pages 36–51. Springer, 2010.

[29] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Architecture-based reliability prediction with the palladio component model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, 2012. ISSN 0098-5589.

[30] F. Brosch and B. Zimmerova. Design-time reliability prediction for software systems. In *Proc. International Workshop on Software Quality and Maintainability (SQM'09)*, pages 1–5, 2009.

[31] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and its Support in Java. *Software: Practice and Experience*, 36(11-12):1257–1284, August 2006.

[32] B. Buhnova, S. Chren, and L. Fabrikova. Failure data collection for reliability prediction models: A survey. In *Proceedings of the 10th International ACM Sigsoft Conference on the Quality of Software Architectures*, pages 83–92. ACM, 2014.

[33] M. Caporuscio, P. Inverardi, and P. Pelliccione. Compositional Verification of Middleware-Based Software Architecture Descriptions. In *Proceedings of the International Conference on Software Engineering (ICSE'04)*, pages 221–230. IEEE Computer Society, May 2004.

[34] I. Černá, P. Vařeková, and B. Zimmerova. Component-Interaction Automata Modelling Language. Technical Report FIMU-RS-2006-08, Masaryk University, Faculty of Informatics, Brno, Czech Republic, October 2006.

[35] I. Černá, P. Vařeková, and B. Zimmerova. Component Substitutability via Equivalencies of Component-Interaction Automata. *Electronic Notes in Theoretical Computer Science*, 182:39–55, June 2007. ISSN 1571-0661.

[36] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *Proceedings of Integrated Formal Methods'04*, volume 2999 of *LNCS*, pages 128–147. Springer Berlin Heidelberg, 2004.

[37] R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980.

[38] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, USA, January 2000. ISBN 0-262-03270-8.

[39] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning Assumptions for Compositional Verification. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 331–346. Springer-Verlag, January 2003.

[40] V. Cortellessa, H. Singh, and B. Cukic. Early reliability assessment of UML based software models. In *Proc. 3rd Int. Workshop on Software and Performance (WOSP'02)*, pages 302–309, New York, NY, USA, 2002. ACM.

[41] O. Das and C. M. Woodside. Dependability modeling of self-healing client-server applications. In *Proc. Workshop on Software Architectures for Dependable Systems (WADS'03)*, volume 3069 of *LNCS*, pages 266–285. Springer, 2003.

[42] L. de Alfaro and T. A. Henzinger. Interface Automata. In *Proceedings of the 9th Annual Symposium on Foundations of Software Engineering (FSE'01)*, pages 109–120. ACM Press, September 2001.

[43] L. de Alfaro and T. A. Henzinger. Interface-based Design. In *Proceedings of the 2004 Marktoberdorf Summer School*, pages 1–25. Kluwer, The Netherlands, 2005.

[44] S. Distefano and A. Puliafito. Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees. *IEEE Trans. on Dependable and Secure Computing*, 6(1):4–17, 2009.

[45] DiVinE – Distributed Verification Environment. Last retrieved 2016-01-01.

[46] H. El-Sayed, D. Cameron, and C. M. Woodside. Automation support for software performance engineering. In *SIGMETRICS/Performance*, pages 301–311. ACM, 2001.

[47] C. Ellis. Team Automata for Groupware Systems. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP'97)*, pages 415–424. ACM Press, November 1997.

[48] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test*, 10:64–75, October 1993.

[49] L. Fiondella and S. Gokhale. Importance measures for modular software with uncertain parameters. *Software Testing, Verification and Reliability*, 20(1):63–85, March 2010.

[50] W. Fokkink. *Introduction to Process Algebra*. Springer-Verlag, 2000. ISBN 3-540-66579-X.

[51] 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)*, volume 3489 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 2005.

[52] D. Garlan, R. T. Monroe, and D. Wile. *Foundations of Component-Based Systems*, chapter Acme: Architectural Description of Component-Based Systems. Cambridge University Press, USA, 2000. ISBN 0-521-77164-1.

[53] D. Gesvindr and B. Buhnova. Performance challenges, current bad practices and hints in PaaS cloud application design. *ACM SIGMETRICS Performance Evaluation Review*, pages 1–10, to appear in 2016. ISSN 0163-5999.

[54] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh. Assume-Guarantee Verification of Source Code with Design-Level Assumptions. In *Proceedings of the International Conference on Software Engineering (ICSE'04)*, pages 211–220. IEEE Computer Society, May 2004.

[55] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proceedings of CAV'90*, volume 531 of *LNCS*, pages 176–185. Springer, 1991.

[56] S. S. Gokhale. Quantifying the variance in application reliability. In *Proc. 10th IEEE Int. Symp. on Dependable Computing (PRDC'04)*, pages 113–121. IEEE Computer Society, 2004.

[57] S. S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Computing*, 4(1):32–40, January-March 2007.

[58] S. S. Gokhale and K. S. Trivedi. Reliability prediction and sensitivity analysis based on software architecture. In *Proc. 13th Int. Symp. on Software Reliability Engineering (ISSRE'02)*, pages 64–78. IEEE Computer Society, 2002.

[59] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. Ammar, and A. Mili. Architectural-level risk analysis using uml. *IEEE Transactions on Software Engineering*, 29(10):946–960, October 2003.

[60] K. Goseva-Popstojanova and S. Kamavaram. Assessing uncertainty in reliability of component-based software systems. In *Proc. 14th Int. Symp. on Software Reliability Engineering*, pages 307–320. IEEE Computer Society, 2003.

[61] K. Goseva-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, May 2001.

[62] K. Goseva-Popstojanova and K. S. Trivedi. Architecture-based approaches to software reliability prediction. *Computers & Mathematics with Applications*, 46(7):1023–1036, October 2003.

[63] K. K. Goswami and R. K. Iyer. Simulation of software behavior under hardware faults. In *Proc. 23rd Int. Symp. on Fault-Tolerant Computing*, pages 218–227, 1993.

[64] K. Goševa-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001.

[65] M. Grottke and K. S. Trivedi. Software faults, software aging and software rejuvenation. *Journal of the Reliability Engineering Association of Japan*, 27(7):425–438, 2005.

[66] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *International Conference on Software Engineering, ICSE*, pages 849–852. ACM, 2006.

[67] 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.

[68] 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 *WADS*, volume 4615 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2006.

[69] R. K. Gupta. *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Kluwer Acad. Publishers, Norwell, USA, 1995.

[70] L. Happe, B. Buhnova, and R. Reussner. Stateful component-based performance models. *Software and Systems Modeling*, 13(4):1319–1343, 2014. ISSN 1619-1374.

[71] G. T. Heineman and W. T. Councill. *Component Based Software Engineering – Putting the Pieces Together*. Addison-Wesley, USA, May 2001. ISBN 0-201-70485-4.

[72] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. ISBN 0-13-153271-5 hardback, ISBN 0-13-153289-8 paperback.

[73] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Journal on Softw. Syst. Model.*, 7(1):49–65, February 2008.

[74] International Standard Organization. ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems. *ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15*, pages c1 –24, 6 2007.

[75] Y. Jia, Z. Li, and Z. Zhang. Timed component-interaction automata for specification and verification of real-time reactive systems. In *Proceedings of the International Conference on Computer Science and Software Engineering*, pages 135–138. IEEE, 2008.

[76] L. Kapova and B. Buhnova. Performance-driven stepwise refinement of component-based architectures. In *Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems*, pages 1–7. ACM, 2010.

[77] L. Kapova, B. Buhnova, A. Martens, J. Happe, and R. Reussner. State dependence in performance evaluation of component-based software systems. In *Proceedings of the Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE'10)*, pages 37–48. ACM, 2010.

[78] S. Kim, D.-K. Kim, L. Lu, and S. Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8):1211–1231, August 2009.

[79] H. Koziolek. Performance evaluation for component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010.

[80] H. Koziolek and F. Brosch. Parameter dependencies for component reliability specifications. In *Proc. 6th Int. Workshop on Formal Engineering Approaches to Software Components and Architecture (FESCA'09)*, volume 253 of *ENTCS*, pages 23–38. Elsevier, 2009.

[81] H. Koziolek, B. Schlich, and C. Bilich. A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis. In *Proc. 21st IEEE Int. Symp. on Software Reliability Engineering (ISSRE'10)*, pages 279–288. IEEE Computer Society, 2010.

[82] P. Kubat. Assessing reliability of modular software. *Operations Research Letters*, 8(1):35–41, 1989.

[83] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(2):190–222, 1983.

[84] G. T. Leavens and M. Sitaraman, editors. *Foundations of Component-Based Systems*. Cambridge University Press, UK, 2000. ISBN 0-521-77164-1.

[85] 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, 2010.

[86] M. Lumpe. Action prefixes: reified synchronization paths in minimal component interaction automata. In *Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009)*, volume 263 of *ENTCS*, pages 179–195. Elsevier, 2010.

[87] M. Lumpe. Partition refinement of Component Interaction Automata. *Science of Computer Programming*, 78(1):27–45, 2012.

[88] N. A. Lynch and M. R. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 137–151. ACM Press, August 1987.

[89] N. A. Lynch and M. R. Tuttle. An Introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[90] M. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill, Inc., 1996.

[91] J. Magee, J. Kramer, and D. Giannakopoulou. Behaviour Analysis of Software Architectures. In *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA'99)*, pages 35–50. Kluwer, The Netherlands, February 1999.

[92] A. Martens, F. Brosch, and R. Reussner. Optimising multiple quality criteria of service-oriented software architectures. In *Proc. 1st Int. workshop on Quality of service-oriented software systems (QUASOSS'09)*, pages 25–32. ACM, 2009.

[93] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. 1st Joint WOSP/SIPEW Int. Conf. on Performance Engineering (ICPE'10)*, pages 105–116. ACM, 2010.

[94] 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*, pages 47–51. ACM, 2007.

[95] I. Meedeniya, A. Aleti, and B. Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium on Automotive/Avionics Systems Engineering (SAASE'09)*, pages 1–16, 2009.

[96] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *Proceedings of the Sixth International Conference on the Quality of Software Architectures*, volume 6093 of *LNCS*, pages 52–67. Springer, 2010.

[97] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212.

[98] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske. Architecture-based reliability evaluation under uncertainty. In *Proc. of joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 85–94. ACM, 2011.

[99] 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.

[100] B. Meyer. The Grand Challenge of Trusted Components. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 660–667. IEEE Computer Society, May 2003.

[101] B. Meyer, C. Mingins, and H. Schmidt. Providing Trusted Components to the Industry. *Computer*, 31(5):104–105, May 1998.

[102] Microsoft Corporation. COM: Component Object Model Technologies. URL https://www.microsoft.com/com/. Last retrieved 2016-01-01.

[103] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980. ISBN 3-540-10235-3.

[104] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989. ISBN 0-13-115007-3.

[105] R. Milner. *Communicating and Mobile Systems: the $\pi$-calculus*. Cambridge University Press, UK, 1999. ISBN 0-521-65869-1.

[106] H. Muccini and A. Romanovsky. Architecting fault tolerant systems. Technical Report CS-TR-1051, University of Newcastle upon Tyne, September 2007.

[107] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, Inc., New York, NY, USA, 1987.

[108] Object Management Group. CORBA Component Model 4.0 Specification. Technical Report formal/06-04-01, Object Management Group, April 2006.

[109] F. Plášil and S. Višňovský. Behavior Protocols for Software Components. *IEEE Transactions on Software Engineering*, 28(11):1056–1076, November 2002.

[110] A. Pnueli. In transition from global to modular reasoning about programs. In *Proc. of Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series*, 1985.

[111] P. Popic, D. Desovski, W. Abdelmoez, and B. Cukic. Error propagation in the reliability analysis of component based systems. In *Proc. 16th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'05)*, pages 53–62. IEEE Computer Society, 2005.

[112] 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.

[113] B. Randell. System structure for software fault tolerance. In *Proc. Int. Conf. on Reliable software*, pages 437–449. ACM, 1975.

[114] A. Rausch, R. Reussner, R. Mirandola, and F. Plášil. Modelling Contest: Common Component Modelling Example (CoCoME). URL http://www.cocome.org/. Last retrieved 2016-01-01.

[115] A. Rausch, R. Reussner, R. Mirandola, and F. Plášil, editors. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*. Springer.

[116] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo. Reliability prediction for component-based software architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.

[117] G. N. Rodrigues, D. S. Rosenblum, and S. Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *Proc. Fundamental Approaches to Software Engineering (FASE'05)*, volume 3442 of *LNCS*, pages 111–126. Springer, 2005.

[118] R. Roshandel, N. Medvidovic, and L. Golubchik. A Bayesian model for predicting reliability of software systems at the architectural level. In *Proc. Int. Conf. on Quality of Software Architectures, QoSA*, volume 4880 of *LNCS*, pages 108–126. Springer, 2007.

[119] N. Sato and K. S. Trivedi. Accurate and efficient stochastic reliability analysis of composite services using their compact markov reward model representations. In *Proc. IEEE Int. Conf. on Services Computing (SCC'07)*, pages 114–121. IEEE Computer Society, 2007.

[120] K. Schneider. *Verification of Reactive Systems – Formal Methods and Algorithms*. Springer-Verlag, 2004. ISBN 3-540-00296-0.

[121] B. Schroeder and G. A. Gibson. Disk failures in the real word: What does an MTTF of 1,000,000 hours mean to you? In *Proc. 5th USENIX Conf. on File and Storage Technologies (FAST'07)*, 2007.

[122] V. Sharma and K. Trivedi. Quantifying software performance, reliability and security: An architecture-based approach. *Journal of Systems and Software*, 80:493–509, August 2007.

[123] 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, 2008.

[124] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[125] Sun Microsystems. Enterprise JavaBeans 3.0 Specification, May 2006.

[126] C. Szyperski. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley, USA, 2. edition, 2002. ISBN 0-201-74572-0.

[127] P. Vařeková, P. Moravec, I. Černá, and B. Zimmerova. Effective Verification of Systems with a Dynamic Number of Components. In *Proceedings of the ESEC/FSE Conference on Specification and Verification of Component-Based Systems (SAVCBS'07)*, pages 3–13. ACM Press, September 2007.

[128] P. Vařeková and B. Zimmerova. Challenge Problem: Subject-Observer Specification with Component-Interaction Automata. In *Proceedings of the ESEC/FSE Conference on Specification and Verification of Component-Based Systems (SAVCBS'07)*, pages 75–81. ACM Press, September 2007.

[129] P. Vařeková, B. Zimmerova, P. Moravec, and I. Černá. Formal Verification of Systems with an Unlimited Number of Components. *IET Software journal*, 2(6):532–546, 2008. ISSN 1751-8814.

[130] W. Wang, D. Pan, and M. Chen. An architecture-based software reliability model. In *Proceedings of PRDS '99*, pages 143–150. IEEE Computer Society, 1999.

[131] W.-L. Wang, D. Pan, and M.-H. Chen. Architecture-based software reliability modeling. *Journal of Systems and Software*, 79(1):132–146, January 2006.

[132] X. Wu and M. Woodside. Performance Modeling from Software Components. *SIGSOFT Softw. Eng. Notes*, 29(1):290–301, 2004.

[133] F. Xie and J. C. Browne. Verified systems by composition from verified components. In *Proceedings of the European Software Engineering Conference (ESEC'03)*, pages 277–286. ACM Press, September 2003.

[134] G. Xie. Decompositional Verification of Component-based Systems – A Hybrid Approach. In *Proceedings of the IEEE International Conference on Automated Software Engineering (ASE'04)*, pages 414–417. IEEE Computer Society, September 2004.

[135] L. Xing. Fault-tolerant network reliability and importance analysis using binary decision diagrams. In *Proc. of 2004 Annual Symposium on Reliability and Maintainability, RAMS*, pages 122–128. IEEE, 2004.

[136] 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.

[137] B. Zimmerova. *Modelling and Formal Analysis of Component-Based Systems in View of Component Interaction*. PhD thesis, Masaryk University, Czech Republic, 2008.

[138] B. Zimmerova and P. Vařeková. Reflecting Creation and Destruction of Instances in CBSs Modelling and Verification. In *Proceedings of the Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'07)*, pages 257–264, October 2007.

[139] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*, chapter Component-Interaction Automata Approach (CoIn), pages 146–176. Springer, 2008.

# Part II

# Collection of Articles

# Appendix A

# Journal articles and chapters

This appendix together with Appendix B and C list (in the chronological order) the total of 20 research articles that were selected as the representatives of my contributions within the studied research field. The fulltexts of the articles are inserted into the corresponding appendixes of the printed version of this thesis[1] and referenced via the article numbers assigned in the list below (replacing page numbers). The same holds for Appendix B and C.

**Article A.1:** P. Vařeková, B. Zimmerova, P. Moravec, and I. Černá. Formal Verification of Systems with an Unlimited Number of Components. *IET Software journal*, 2(6):532–546, 2008. ISSN 1751-8814

**Article A.2:** B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153 of *LNCS*, chapter Component-Interaction Automata Approach (CoIn), pages 146–176. Springer, 2008

**Article A.3:** N. Beneš, L. Brim, B. Buhnova, I. Černá, J. Sochor, and P. Vařeková. Partial order reduction for State/Event LTL with application to Component-Interaction Automata. *Science of Computer Programming*, 76(10):877–890, 2011. ISSN 0167-6423

**Article A.4:** I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212

**Article A.5:** F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Architecture-based reliability prediction with the palladio component model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, 2012. ISSN 0098-5589

[1] The fulltexts of the articles are excluded from the publicly available electronic version of this text to avoid copyright violation.

**Article A.6:** A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 5(39):658–683, 2013. ISSN 0098-5589

**Article A.7:** L. Happe, B. Buhnova, and R. Reussner. Stateful component-based performance models. *Software and Systems Modeling*, 13(4):1319–1343, 2014. ISSN 1619-1374

# Appendix B

# Conference papers

**Article B.1:** P. Vařeková and B. Zimmerova. Challenge Problem: Subject-Observer Specification with Component-Interaction Automata. In *Proceedings of the ESEC/FSE Conference on Specification and Verification of Component-Based Systems (SAVCBS'07)*, pages 75–81. ACM Press, September 2007

**Article B.2:** B. Zimmerova and P. Vařeková. Reflecting Creation and Destruction of Instances in CBSs Modelling and Verification. In *Proceedings of the Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'07)*, pages 257–264, October 2007

**Article B.3:** F. Brosch and B. Zimmerova. Design-time reliability prediction for software systems. In *Proc. International Workshop on Software Quality and Maintainability (SQM'09)*, pages 1–5, 2009

**Article B.4:** I. Meedeniya, A. Aleti, and B. Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium on Automotive/Avionics Systems Engineering (SAASE'09)*, pages 1–16, 2009

**Article B.5:** F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Parameterized reliability prediction for component-based software architectures. In *Proc. 6th Int. Conf. on the Quality of Software Architectures (QoSA'10)*, volume 6093 of *LNCS*, pages 36–51. Springer, 2010

**Article B.6:** L. Kapova and B. Buhnova. Performance-driven stepwise refinement of component-based architectures. In *Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems*, pages 1–7. ACM, 2010

**Article B.7:** L. Kapova, B. Buhnova, A. Martens, J. Happe, and R. Reussner. State dependence in performance evaluation of component-based software systems. In *Proceedings of the Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE'10)*, pages 37–48. ACM, 2010

**Article B.8:** I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In *Proceedings of the Sixth International Conference on the Quality of Software Architectures*, volume 6093 of *LNCS*, pages 52–67. Springer, 2010

**Article B.9:** F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner. Reliability prediction for fault-tolerant software architectures. In *Proceedings of the joint ACM SIGSOFT conference QoSA and ACM SIGSOFT symposium ISARCS on Quality of software architectures (QoSA) and architecting critical systems (ISARCS)*, pages 75–84. ACM, 2011

**Article B.10:** N. Benes, B. Buhnova, I. Cerna, and R. Oslejsek. Reliability analysis in component-based development via probabilistic model checking. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 83–92. ACM, 2012

**Article B.11:** B. Buhnova, S. Chren, and L. Fabrikova. Failure data collection for reliability prediction models: A survey. In *Proceedings of the 10th International ACM Sigsoft Conference on the Quality of Software Architectures*, pages 83–92. ACM, 2014

# Appendix C

# Case studies and tool papers

**Article C.1:** N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. The CoIn Tool: Modelling and Verification of Interactions in Component-Based Systems. In *Pre-proceedings of the International Workshop on Formal Aspects of Component Software (FACS'08)*, pages 221–225. Department of Computer Science, University of Malaga, 2008

**Article C.2:** N. Beneš, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova. A Case Study in Parallel Verification of Component-Based Systems. In *Proceedings of the Workshop on Parallel and Distributed Methods in verifiCation (PDMC'08)*, ENTCS, pages 35–51. Elsevier Science Publishers, March 2008