Software Architecture Optimization Methods: A Systematic Literature Review

Aldeida Aleti, Barbora Buhnova, Lars Grunske, *Member, IEEE,* Anne Koziolek, *Member, IEEE,* and Indika Meedeniya

Abstract—Due to significant industrial demands toward software systems with increasing complexity and challenging quality requirements, software architecture design has become an important development activity and the research domain is rapidly evolving. In the last decades, software architecture optimization methods, which aim to automate the search for an optimal architecture design with respect to a (set of) quality attribute(s), have proliferated. However, the reported results are fragmented over different research communities, multiple system domains, and multiple quality attributes. To integrate the existing research results, we have performed a systematic literature review and analyzed the results of 188 research papers from the different research communities. Based on this survey, a taxonomy has been created which is used to classify the existing research. Furthermore, the systematic analysis of the research literature provided in this review aims to help the research community in consolidating the existing research efforts and deriving a research agenda for future developments.

Index Terms—Software Architecture Optimization, Systematic Literature Review, Optimization Methods, Problem Overview

1 Introduction

Architecture specifications and models [120] are used to structure complex software systems and to provide a blueprint that is the foundation for later software engineering activities. Thanks to architecture specifications, software engineers are better supported in coping with the increasing complexity of today's software systems. Thus, the architecture design phase is considered one of the most important activities in a software engineering project [24]. The decisions made during architecture design have significant implications for economic and quality goals. 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.

Problem Description and Motivation. Due to the increasing system complexity, software architects have to choose from a combinatorially growing number of design options when searching for an optimal architecture design with respect to a defined (set of) quality attribute(s) and constraints. This results in a design space search that is often beyond human capabilities and makes the architectural design a challenging task [105]. The need for automated design space exploration that improves an existing architecture specification has been recognized [191] and a plethora of architecture optimization approaches based on formal architecture specifications have been developed. 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 (allocation, replication, selection of architectural elements etc.). Hence the research activities are scattered across many research communities, system domains (such as embedded systems or information systems), and quality attributes. Similar approaches are proposed in multiple domains without being aware of each other.

Research Approach and Contribution. To connect the knowledge and provide a comprehensive overview of the current state of the art, this article provides a systematic literature review of the existing architecture optimization approaches. As a result, a gateway to new approaches of architecture optimization can be opened, combining different types of architectural decisions during the optimization or using unconventional optimization techniques. Moreover, new trade-off analysis techniques can be developed by combining results from different optimization domains. All this can bring significant benefits to the general practice of architecture optimization. In general, with the survey we aim to achieve the following objectives:

- Provide a basic classification framework in form of a taxonomy to classify existing architecture optimization approaches.
- Provide an overview of the current state of the art in the architecture optimization domain.
- Point out current trends, gaps, and directions for future research.

- *A. Aleti is with the Faculty of Information Technology, Monash University, Australia. E-mail: aldeida.aleti@monash.edu*
- *I. Meedeniya is with the Faculty of ICT, Swinburne University of Technology, Australia. E-mail: imeedeniya@swin.edu.au*
- *B. Buhnova is with the Faculty of Informatics, Masaryk University, Czech Republic. E-mail: buhnova@fi.muni.cz*
- *L. Grunske is with the Faculty of Computer Science, University of Kaiserslautern, Germany. E-mail: grunske@cs.uni-kl.de*
- *A. Koziolek is with the Department of Informatics, University of Zurich, Switzerland. E-mail: koziolek@ifi.uzh.ch*

We examined 188 papers from multiple research sub-areas, published in software-engineering journals and conferences. Initially, we derived a taxonomy by performing a formal content analysis. More specifically, based on the initial set of keywords and defined inclusion and exclusion criteria, we collected a set of papers, which we iteratively analyzed to identify the taxonomy concepts. The taxonomy was then used to classify and analyze the papers, which provided a comprehensive overview of the current research in architecture optimization. The data was then used to perform a cross analysis of different concepts in the taxonomy and derive gaps and possible directions for further research.

Related surveys. Architecture optimization can be categorized into the general research discipline of Search Based Software Engineering (SBSE) [110] as it applies efficient search strategies to identify an optimal or near-optimal architecture specification. SBSE is applied in all phases of the software engineering process including requirements engineering, project management, design, maintenance, reverse engineering, and software testing. A comprehensive survey of different optimization techniques applied to software engineering tasks is provided by Harman et al. [111]. The survey indicates that in the past years, a particular increase in SBSE activity has been witnessed, with many new applications being addressed. The paper identifies research trends and relationships between the search techniques and the applications to which they have been applied. The focus of Harman et al.'s survey is on the broad field of SBSE, especially on approaches in the software testing phase which are also covered in detailed surveys [156], [163]. However, the area of architecture optimization has not been investigated in detail. The SBSE survey lists several approaches to optimizing software design, but does not analyze properties of these approaches except naming the used optimization strategy.

Beside this general SBSE survey, other surveys describe sub-areas of architecture optimization and design-space exploration that are only concerned with a specific system domains, or a specific optimization method. For instance, the survey of Grunske et al. [105] is concerned with the domain of safety-critical embedded systems and compares 15 architecture optimization methods. Another example is the survey of Villegas et al. [230], which evaluates 16 approaches that target run-time architecture optimizations with a focus on self-adaptive systems. In the research sub-area of systems with high reliability demands, Kuo and Wan [140] have published a survey in 2007 comparing different redundancy allocation approaches. Finally, several surveys are concerned with the application of a specific optimization technique, typically related to Genetic Algorithms [4], [125] or metaheuristics in general [195].

Although these surveys provide a good overview of a specific application domain, optimization method, or even a design phase, none of them is suitable in giving a comprehensive overview of the existing research in the area of architecture optimization.

Organization. The rest of the paper is organized as follows. First, Section 2 outlines the research method and the underlying protocol for the systematic literature review. The first contribution of this article, a taxonomy for architecture optimization approaches that has been derived from an iterative analysis of the existing research literature is presented in Section 3. The second contribution, a classification of existing architecture optimization approaches according to this taxonomy, is presented in Section 4. This section contains both a classification into the categories of the taxonomy including some descriptive statistics as well as a cross-category analysis between the different taxonomy areas. Finally, Section 5 identifies future research directions based on the survey results and Section 6 presents the conclusions.

2 Research Method

Our literature review follows the guidelines proposed by Kitchenham [129], which structure the stages involved in a systematic literature review into three phases: planning, conducting, and reporting the review. Based on the guidelines, this section details the research questions, the performed research steps, and the protocol of the literature review. First, Section 2.1 describes the research questions underlying our survey. Then, Section 2.2 derives the research tasks we conducted, and thus describes our procedure. Section 2.3 then details the literature search step and highlights the inclusion and exclusion criteria. Finally, Section 2.4 discusses threats to the validity of our study.

2.1 Research Questions

Based on the objectives described in the introduction, the following research questions have been derived, which form the basis for the literature review:

- **RQ1** How can the current research on software architecture optimization be classified?
- **RQ2** What is the current state of software architecture optimization research with respect to this classification?
- **RQ3** What can be learned from the current research results that will lead to topics for further investigation?

2.2 Research Tasks

To answer the three research questions RQ1-3, four research tasks have been conducted: one task to set up the literature review, and three research tasks dedicated to the identified research questions. The tasks have been conducted in a sequential manner and interconnected through a number of artifacts generated by their subtasks. The overall research method is outlined in Figure 1 and detailed in the following text.

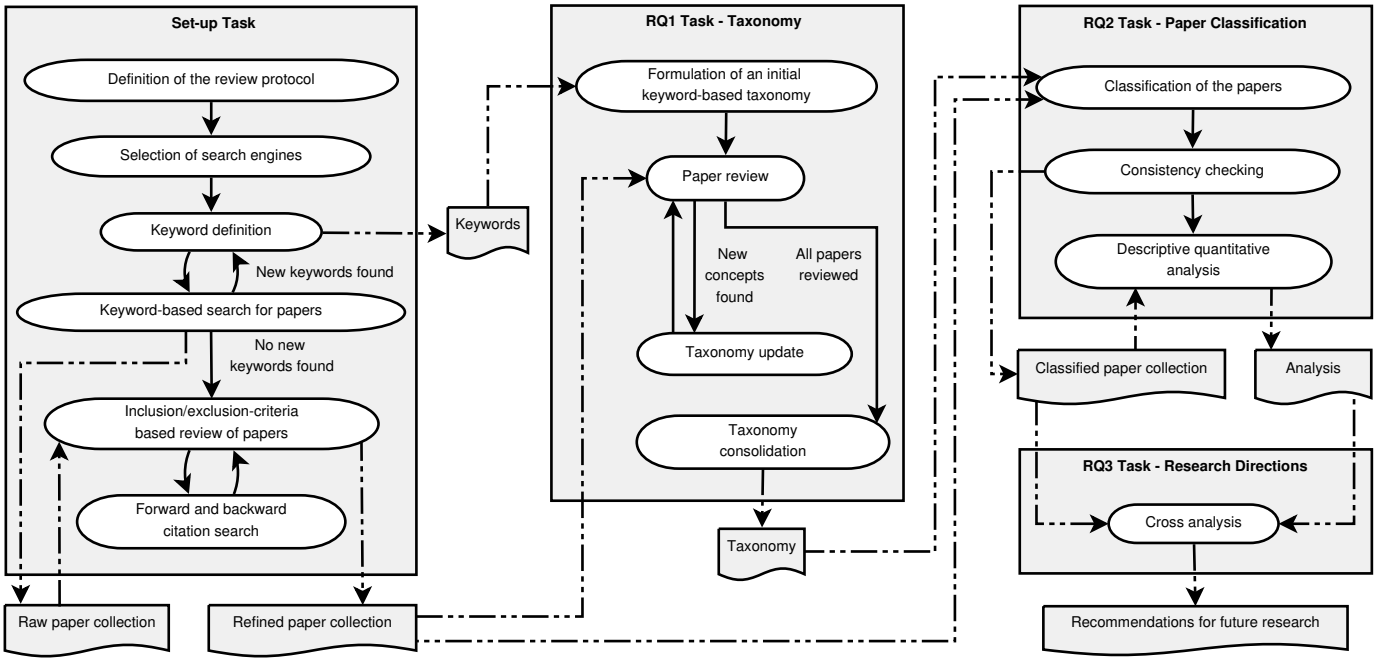

Fig. 1: The process model of our research method.

The set-up task includes the definition of the review protocol, the selection of search engines, the definition of a keyword list, a keyword-based collection of published architecture optimization papers, and a review filtering the papers according to a defined set of inclusion and exclusion criteria. The search step and the inclusion/exclusion review step are explained in more detail in Section 2.3.

Based on the set of selected papers, we performed a *content analysis* [135] of the papers in the first research task (RQ1). The goal was to derive a taxonomy to classify the current architecture optimization approaches. We used an iterative coding process to identify the main categories of the taxonomy. The coding process was based on the *grounded theory* [94] qualitative research method. First we analyzed each paper with the goal to identify new concepts for the taxonomy. Second, after all papers have been reviewed and the taxonomy updated with newly identified concepts, we consolidated the taxonomy terms, mainly by merging the synonyms and unifying the concepts on different levels of abstraction. Section 3 presents the findings.

In the second research task (RQ2), each paper collected in the set-up task was classified based on the taxonomy derived in the first research task. Within our team of authors, one person was nominated as a data extractor for each paper. Furthermore, one person was nominated as a data checker for each top-level taxonomy category. While the responsibility of the data extractors was to classify the papers, data checkers cross-checked the classification and discussed any inconsistencies with data extractors. Extracted data was stored in a database, which enabled a descriptive quantitative analysis. The aim of the data extraction and the resulting classification was to provide a significant overview of the current research effort and the archived results in this domain. Sections 4.1 to 4.3 present the findings.

In the third research task (RQ3), we cross-analyzed the survey results and synthesized possible directions for further research. The derivation of possible future research directions was specifically enabled by the variety of papers from multiple research sub-areas each of which has its own strengths. Consequently, the survey enables the knowledge transfer from one research subarea to another and thus aims at improving the overall research area. Section 4.4 presents the cross-analysis results, while Section 5 provides our recommendations for future research.

2.3 Literature Search Process

The search strategy for the review was primarily directed toward finding published papers in journals and conference proceedings via the widely accepted literature search engines and databases Google Scholar, IEEE Explore, ACM Digital Library, Springer Digital Library, and Elsevier ScienceDirect.

For the search we focused on selected keywords, based on the aimed scope of the literature review. Examples of the keywords are: automated selection of software components, component deployment optimization, energy consumption optimization, component selection optimization, automated component selection, reliability optimization, software safety optimization, redundancy allocation, optimal scheduling, hardware-software co-synthesis, search based software engineering, run-time and design-time architecture optimization, software engineering optimization, self-adaptive software systems.

The keywords were refined and extended during the search process. The final keyword list is available at the project website [6].

In the subsequent phase, we reviewed the abstracts (and keywords) of the collected papers with respect to the defined set of inclusion and exclusion criteria (Sections 2.3.1 and 2.3.2 below), and further extended the collection with additional papers based on an analysis of the cited papers and the ones citing it (forward and backward citation search). As a result, we included 188 peer-reviewed papers in the survey comprising of papers from 1992 to 2011, with more than 50% of the papers published in the last years between 2005 and 2010.

Although the selection process was primarily based on the review of paper abstracts and keywords, in the cases where these two were insufficient, we also considered parts of the introduction, contribution and conclusion sections.

2.3.1 Inclusion Criteria

The focus of this literature review is on software architecture optimization. We understand the architecture of a software system to be "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [120]. Software architecture optimization is understood as an automated method aiming to reach an optimal architecture design with respect to a (set of) quality attribute(s). The main criteria for inclusion were based on the *automation* of software architecture optimization, both at run time and at design time. To enable automated optimization of software architectures, three basic prerequisites need to be fulfilled:

1) A machine-processable representation of the software architecture must be available as an input for automated search (e.g. a UML model with agreed semantics, models described in any architecture description language, or representations in formalisms such as Markov chains). Such a representation may be an *architecture model* as defined in [120], but can also be another machine-processable representation such as a Markov chain.
2) A function or procedure that automatically evaluates an aspect of quality for a given software architecture is required (called quality evaluation function/procedure in this work). Different quality attributes used during the optimization process were included as long as they were quantifiable by such a quality evaluation function/procedure. Cost was also considered since it is a commonly addressed optimization objective in conjunction with quality attributes. Both single-objective and multi-objective problems were taken into account. Furthermore, papers that solved any type of constrained problem were included, not excluding the papers that did not include constraints.
3) A definition of the considered design space is required that describes how a given software architecture representation can be changed or enhanced by the optimization. We call this information "architectural degrees of freedom" [132] in this work as there is no other agreed term in the context of architecture optimization. Example architectural degrees of freedom are allocation, component selection, or hardware parameter change.

Papers that provide these three aspects are included in our review.

2.3.2 Exclusion Criteria

We excluded papers that: (a) optimize a single component without integrating context and interactions with other architectural elements, (b) focus on an architecture-irrelevant problem (e.g. requirements prioritization, compiler optimization, or task allocation to agents that cooperate in executing and finishing the tasks), (c) optimize hardware with no relation to software (e.g. FPGA optimization), or (d) solely optimize cost without considering any other quality attribute. Moreover, due to the goal of approach classification, we excluded the papers discussing an approach already included in the collection (recognized based on the author list and approach attributes) and we excluded non-reviewed papers. We did not exclude papers for quality reasons, because the quality of the papers was generally acceptable. Evidence for the quality of the papers can be found in a post selection analysis of the citations of each paper via Google Scholar, which in 2012 revealed that each of the papers has been cited at least once and the average citation count for the papers included in the survey was 76.5. The h-index and g-index of the included papers was 57 and 128 respectively.

2.4 Threats to Validity

One of the main threats to the validity of this systematic literature review is the incompleteness. The risk of this threat highly depends on the selected list of keywords and the limitations of the employed search engines. To decrease the risk of an incomplete keyword list, we have used an iterative approach to keyword-list construction. A well-known set of papers was used to build the initial taxonomy which evolved over time. New keywords were added when the keyword list was not able to find the state-of-the-art in the respective area of study. In order to omit the limitations implied by employing a particular search engine, we used multiple search engines. Moreover, the authors' expertise in different system domains, quality attributes, and optimization approaches reduced the search bias.

Another important issue is whether our taxonomy is robust enough for the analysis and classification of the papers. To avoid a taxonomy with insufficient capability to classify the selected papers, we used an iterative

content analysis method to continuously evolve the taxonomy for every new concept encountered in the papers. New concepts were introduced into the taxonomy and changes were made in the related taxonomy categories.

Furthermore, in order to make the taxonomy a better foundation for analyzing the selected papers, we allowed multiple abstraction levels for selected taxonomy concepts. As a result, one of the concepts (namely the used optimization strategy) has different levels of detail, where the highest level is abstract with few classes, whereas lower levels have more details with more classes used to classify the papers. The appropriate level was selected when presenting the results. In order to reduce the classification bias, paper classification results were checked by all the authors. The classification according to the remaining abstraction levels is recorded in the survey database, which can be accessed at [6].

3 Taxonomy

The quality of a literature review project highly depends on the selected taxonomy scheme, which influences the depth of knowledge recorded about each studied approach. In this article, an iterative coding process has been employed to identify the taxonomy categories (see Section 2 for details) and to provide an answer to the first research question (**RQ1**). The resulting taxonomy hierarchy is depicted in Figure 2.

The first level of the taxonomy hierarchy structures the existing work according to three fundamental questions characterizing the approaches. These are:

(1) What is the formulation of the optimization *problem* being addressed?
(2) What techniques are applied to the *solution* of the problem?
(3) How is the *validity* of the approach assessed?

We discuss each of these questions in detail, and define the implied taxonomy scheme. For each of the questions, we derive the sub-categories of the taxonomy related to the question. Each category has a number of possible values used to characterize the optimization approaches. For example, the category *Domain* has the three values *Embedded systems*, *Information systems* and *General*. We only briefly discuss the possible values of categories in the following, while the complete structured list of all the values is in Tables 1, 3 and 6 where full details can be found in the wiki page[1].

3.1 The Problem Category

The first category is related to the problem the approaches aim to solve in the real world. Generally speaking, the approaches try to achieve a certain optimization goal in a specific context. For example, an optimization goal is to minimize the response time of an architecture given costs constraints. An example context is to consider embedded systems at design time. While the context of the problem is determined by the sub-categories *domain* (i.e. the type of targeted systems) and *phase* (i.e. place in the development process) of the problem, the sub-categories related to the optimization goal include *quality attributes*, *constraints*, and the *dimensionality* of the optimization problem, which is governed by the question if the set of optimized quality attributes is aggregated into a single mathematical function or decoupled into conflicting objectives (single/multi-objective optimization).

In particular, the *domain* has three possible values: Information systems (IS) are business related systems operated on a general purpose computer that include for instance e-business applications, enterprize and government information systems. Embedded systems (ES) in contrast are realized on a dedicated hardware to perform a specific function in a technical system. They scale from small portable devices like mobile phones to large factories and power plants. If an approach is designed for both domains, the third possible value "general" is used. The *phase* category specifies whether the problem is occurring at design-time (DT) or run-time (RT). The main difference between the two is that while the setting of a design-time problem is known in advance, the setting of a run-time problem changes dynamically (e.g. new tasks can arrive during run-time scheduling). Again, the value "General" can be used here to denote approaches that address both DT and RT.

The goal of the optimization task is typically the maximization of the software-architecture quality under given constraints. Since the quality of a software system as a concept is difficult to define, due to its subjective nature, software experts do not define the quality directly but relate it to a number of system attributes, called quality attributes [119]. In this work, we only consider quantifiable quality attributes (cf. Section 2.3.1). Examples are performance, reliability, cost, availability, and other well established quality attributes (find the full list in Table 1 and at [6]). When categorizing quality attributes, we followed widely accepted definitions and quality attribute taxonomies [16], [24], [103], [241]. In our taxonomy, we distinguish quality attributes to be optimized (category *quality attributes*) from additional constraints on quality attributes or other system properties (category *constraints*). For example, reducing the response time and the costs of a system as much as possible is a setting with two quality attributes to be optimized. Increasing the availability while keeping the response time lower than 5 seconds and adhering to structural constraints is a setting with one quality attribute to be optimized (availability) and two constraints (for performance and structural).

Finally, the *dimensionality* category reflects if the approach addresses a single-objective optimization (SOO) or multi-objective optimization (MOO) problem. The SOO optimizes a single quality attribute only. The MOO

1. https://sdqweb.ipd.kit.edu/wiki/OptimizationSurvey

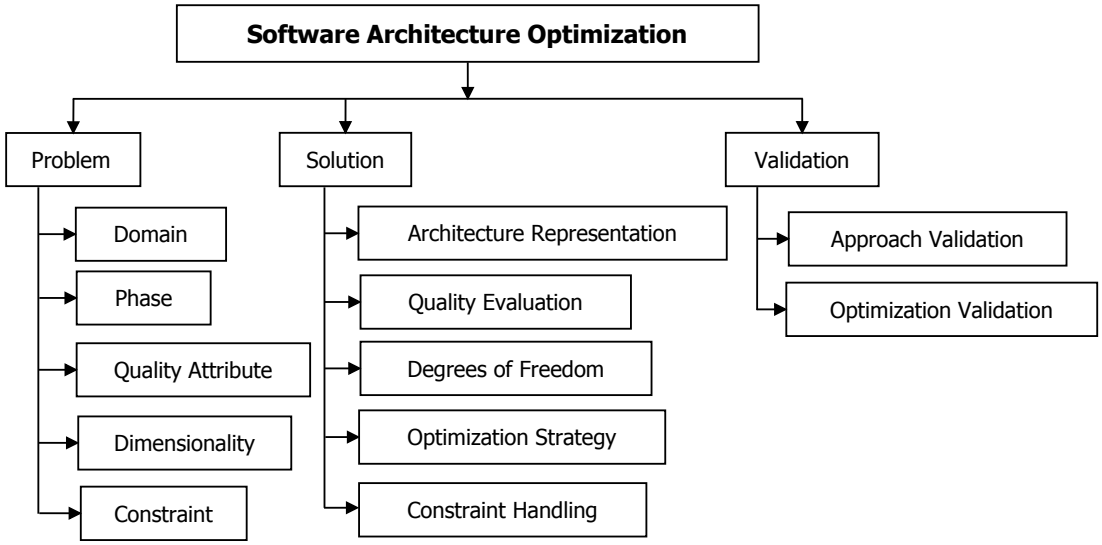

Fig. 2: The taxonomy for architecture optimization approaches, derived from the reviewed literature.

optimizes multiple quality attributes at once, so that the quality of every architecture model is a vector of values. As quality attributes often conflict, usually there is no single optimal result but a set of solutions non-dominated by the others from the point of view of the optimized qualities – i.e. solutions that are Pareto-optimal [70]. Since in MOO a decision has to be taken on the final architecture design selected from the set of resulting candidates, one can also use the multi-objective transformed to single-objective optimization (MTS) approaches, which encode the selection criteria following MOO into a single mathematical function (e.g. a weighted sum), which is then optimized as a single objective.

For a structured view on all the values of the discussed sub-categories see Table 1.

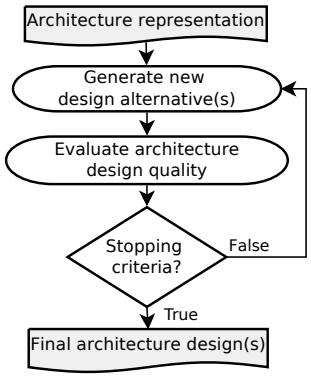

Fig. 3: Optimization process.

3.2 The Solution Category

The solution category classifies the approaches according to how they achieve the optimization goal and thus describes the main step of the optimization process, which is depicted in Figure 3. First, the sub-category *architecture representation* is the process input that describes the architecture to optimize. Second, the sub-category *degrees of freedom* describes what changes of the architecture are considered as variables in the optimization. Third, the sub-category *quality evaluation* describes the used quality evaluation procedures, which make up the objective function(s) of the optimization process. Furthermore, this category contains the techniques used to solve the formulated optimization problem: Sub-categories are the overall *optimization strategy* and *constraint handling*.

The *architecture representation* category classifies the approaches based on the information used to describe the software architecture. Any architecture optimization approach takes some representation of the system's architecture as an input (cf. Figure 3). This representation may be an architecture model [120] documenting the architecture by defining components and connectors. To predict more complex quality attributes, a quality evaluation model such as a Layered Queueing Network or a Markov chain may be derived from an architecture model or may be used directly as an input. Finally, in order to employ optimization techniques, the architecture and the design decisions have to be encoded into an optimization model describing the decision variables and the objective function. This optimization model may be derived from an architecture model or from a quality evaluation model, or may directly be required as an input. To assess the used architecture representation relevant for the user, we classify the approaches based on the input they require, so that the possible values are "architecture model" (an architecture model is used as the input), "quality evaluation model" (a quality evaluation model is used as the input, no architecture model is used), and "optimization model" (an optimization model is used as the input, no architecture model or quality evaluation model is used). Note that several approaches

that start with an architecture model also internally use a quality evaluation model, and that all approaches also internally use an optimization model. If quality evaluation models or optimization models are used as an input, it needs to be guaranteed that an optimal found solution can be traced back to a meaningful solution on the architecture level.

Furthermore, we drill into the used architecture models in more detail, also distinguishing the used modeling formalism as follows. "UML" denotes architecture models defined in any modeling formalism of the Unified Modeling Language. Other architecture description languages such as AADL [203] or PCM [25] are subsumed in the value "ADL". A specific form of architecture description for service-based systems is "workflow specifications". By "custom architecture models" (custom AM) we denote approaches that define a custom model to describe the architecture which, in contrast to ADLs, does not have the purpose to document the architecture but is more tailored towards a specific purpose (such as the architecture middleware PRISM-MW [154]). Finally, approaches that allow to exchange the used architecture model, e.g. by reasoning in the metamodel level or by offering plug-ins for handling different ADLs, are classified with the value "General".

The *quality evaluation* category differentiates the approaches in those formalizing the optimized criteria with a simple additive function (SAF), with a nonlinear mathematical function (NMF), or with a more complex evaluation function and model that, for example, cannot be expressed with closed formulas and are solved numerically or with simulations. We denote this latter case as model-based (MB). For example, consider the quality attribute performance. A simple additive function that calculates the response time of a specific function would sum up the response times of used individual services. A more complex nonlinear mathematical function is used if a queuing behavior of the system analyzed using exact queuing theory formulae. Finally, a model-based procedure is used if the system is represented as an extended queuing network and the performance is evaluated with approximative or simulation-based techniques. In essence, the optimization process aims at optimising the quality attribute(s), whose evaluation constitutes the objective function(s), also referred to as fitness function(s) in the optimisation domain.

The *architectural degrees of freedom* category defines how the architecture representation can be changed to make it optimal with respect to the optimization goal. Example architectural degrees of freedom are component selection, allocation, or hardware parameter change. Thus, this category describes the types of variables of the optimization, i.e. it describes the types of design decision that can be varied by the optimization and thus defines the considered subset of the design space [132]. Another synonymous term is "architecture transformation operators" [101], [105]. More general terms describing the same idea are "design decisions", or "dimensions of variation" [166]. The possible values are those found in the reviewed papers, grouped by synonyms, since no existing classification (such as for quality attributes) is available to use, hence, we explain them in more detail in the next paragraphs.

The *selection degrees of freedom* are concerned with selecting entities in the architecture. These entities can be software entities (such as modules) or hardware entities (such as servers or devices), resulting in "software selection" and "hardware selection" values. We explicitly distinguish "component selection", because some domains have a certain notion of a component. For example, in embedded systems design, component selection could mean deciding between a component realizing a functionality in hardware and a component with general-purpose hardware realizing functionality in software. Furthermore, we explicitly distinguish "service selection", because next to selecting the software to execute, selecting a service also includes selecting the service provider (thus including hardware aspects as well).

Replication degrees of freedom are concerned with changing the multiplicity of an architectural element. Under the term "hardware replication", we subsume all degrees of freedom that concern the number of a hardware entity's copies, while possibly also changing the multiplicity of software elements (e.g. software components deployed to the replicated servers). The popular term redundancy allocation is thus included in "hardware replication". Under the term "software replication", we subsume degrees of freedom that change the number of copies of software entities only. For brevity, we include both identical copies of the software and different implementations of the same functionality (e.g. n-version programming) in the term "software replication" in this paper.

Parameter degrees of freedom refer to other parameters of architectural elements. We distinguish "software parameters" (e.g number of threads of an application server) and "hardware parameters" (e.g. parameters for the hard disk drive). Hardware parameters may overlap with hardware selection, because the choice (e.g. of a CPU with different speed) can be modeled both as hardware selection or as a parameter of the hosting server. Here, we classified a paper based on the presentation of the degree of freedom in that paper.

Further common degrees of freedom are the following. "Scheduling" is concerned with deciding about the order of execution. "Service composition" changes how services are composed by changing service topology and/or the service workflow. "Allocation" (a broader term to "Deployment") changes the mapping of software entities (components or tasks) to processing elements, for example to servers. Other, less common degrees of freedom are architectural patterns, maintenance schedules, partitioning, and clustering. We do not explicitly name degrees of freedom that are used by fewer than 2 papers, but treat them commonly as "problem-specific degrees

of freedom". Some approaches allow for applying any degree of freedom and are classified as "general". Finally, some approaches do not explicitly present the considered degrees of freedom, and are classified under "not presented".

For the *optimization strategy*, we provide three levels of classification. First, we distinguish whether the used optimization strategy guarantees exact solutions (the best available architecture design with respect to the objective function) or only finds approximate solutions. As a subclassification, among the exact methods, we distinguish standard methods (such as standard mixed integer linear programming tools) and problem-specific methods (e.g. operating on graph representations of the problems and exploiting problem properties). Among the approximate methods, we distinguish methods that guarantee a lower bound of the solution, such as some branch-and-bound approaches, methods that require problem or domain specific information to perform the search, i.e. problem-specific heuristics, and methods that apply high-level strategies, i.e. metaheuristics, which are not problem-specific, but can use domain or problem-specific knowledge to guide the search [34], such as evolutionary algorithms. The lowest level of the optimization strategy category describes the concrete used optimization strategy, such as for example evolutionary algorithms, standard linear integer programming solvers, or problem-specific heuristics.

The *constraint handling* category describes the used strategies to handle constraints. Based on the insightful surveys of Michalewicz [172] and Coello Coello [50], possible strategies are encoded with the following values. "Penalty" refers to the strategy that converts the constrained optimization problem into a series of unconstrained problems by adding a penalty parameter to the objective function which reflects the violation of the constraints. "Prohibit" refers to the constraint handling strategy that discards solutions that violate constraints. A "repair" mechanism is employed during the optimization process to fix any violation of constraints before the solution is evaluated. Finally, "General" describes any or a variety of constraint handling techniques.

For a structured view on all the values of the discussed sub-categories see Table 3.

3.3 The Validation Category

For the validation classification of the taxonomy, two subcategories are considered, *approach validation* and *optimization validation*.

The *approach validation* describes techniques used to assess the practicality and accuracy of the approach. This includes specifically the effort spent on the modeling of quality prediction functions and evaluating their accuracy. Possible validation types found in the reviewed approaches include demonstration with a simple example, validation with dedicated benchmark problems or experiments with randomly generated problems, and validation with an academic or industrial case study. As industrial case studies we have classified systems that are used in practice with a clear commercial aim. An academic case study is different to a simple example in that it invents a somewhat realistic system with a clear purpose but without a commercial background, while a simple example describes an abstract small example (e.g. an architecture built from components C1 to C4). Besides these, the possible validation types also include mathematical proofs of the accuracy of the results, and comparison with related literature.

In contrast to the *approach validation* the *optimization validation* specifically validates the used optimization strategy. Such a validation may evaluate (1) how well an approach approximates the global optimimum and / or (2) the performance of an approach compared to other approaches. A possible type of an *optimization validation* for an approach that uses a heuristic is a comparison with a random search strategy, an exact algorithm or a baseline heuristic algorithm. Alternatively, internal comparison is typically employed in the reviewed papers that propose multiple optimization strategies. Then, only the proposed strategies are compared with each other. Some problem-specific approaches also use mathematical proofs to validate the correctness of the optimization strategy. For a structured view on all the values of the discussed sub-categories see Table 6.

4 Results

In this section we aim to answer the second research question **RQ2**. The 188 reviewed papers are classified based on the taxonomy described in Section 3. The quantitative results are presented in Tables 1, 3 and 6.

To provide an overview of the current state of the art in software architecture optimization and to guide the reader to a specific set of approaches that is of interest, the approaches in the different categories including references to the papers are presented in Tables 2, 4 and 5. The references in all the three tables have been structured according to a common characteristics (index) to simplify the orientation in the tables. Since the overall goal of any software architecture optimization approach is to identify candidate architectures with better quality, the *quality attribute* has been used as the index in the tables. For each of the top seven quality attributes, a row presents the references for the approaches addressing this quality attribute q. The total number of the papers is given in parenthesis in the first column. Each column provides the results for one taxonomy category t. Then, each cell (q,t) lists the papers that address quality attribute q, grouped by the values of t. To show the quality attributes that are optimized together, the "other quality attributes" column lists the quality attributes being optimized together with q, instead of presenting the quality-attribute taxonomy category itself (i.e. all combinations of quality attributes optimized together).

Papers may appear in multiple rows if they address several quality attributes. Because all of the reviewed

papers optimize at least one of the top seven quality attributes, all papers appear in the tables. Furthermore, for some other taxonomy categories such as constraints, papers may have multiple values and thus be listed several times. As a result, percentages in the tables may sum up to more than 100%.

The rest of this section presents the observations that can be derived from both Tables 1, 3, 6, as well as other views on the data distilled from the paper collection.

4.1 Problem

TABLE 1: Problem category - quantitative summary of the results.

Quality Attributes			**Constraints**		
Performance	84	44%	Not presented	49	26%
Cost	74	39%	Cost	32	17%
Reliability	71	37%	Performance	26	14%
Availability	25	13%	General	25	13%
General	22	12%	Weight	20	11%
Energy	18	9%	Physical	13	7%
Weight	5	3%	Timing	10	5%
Safety	4	2%	QoS Values	10	5%
Reputation	4	2%	Memory	9	5%
Modifiability	3	2%	Precedence	9	5%
Area	3	2%	Mapping	8	4%
Security	1	< 1 %	Reliability	7	4%
			Requirements	6	3%
			Volume	6	3%
Domain			Structural	5	3%
ES	100	53%	Area	3	2%
GENERAL	49	26%	Redundancy level	3	2%
IS	41	22%	Delivery time	3	2%
			Availability	2	1%
Phase			Throughput	2	1%
DT	128	67%	Processing power	1	< 1 %
RT	60	32%	Stability	1	< 1 %
GENERAL	3	2%	Path loss	1	< 1 %
			Functional correctness	1	< 1 %
Dimensionality			Design	1	< 1 %
SOO	75	39%	Dependability		
MOO	58	31%			
MTS	51	27%			
GENERAL	7	4%			

A summary of the problem-specific aspects that are extracted from the set of papers included in the survey are given in Table 1. In the following, we summarize the main results for each problem subcategory.

Quality Attributes: The architecture optimization approaches investigated in this literature review have covered diverse types of design goals. Based on the analysis of the existing approaches, it is evident 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, comprising less than 1% of the papers. An interesting result is the number of papers which use generic approaches to allow for the definition of customized quality functions, which was encountered in 22 papers (12% of the overall papers). Since quality attributes are often in conflict with each other, many approaches consider multiple quality attributes during the optimization. Details about the combination of the considered quality attributes can be found and extracted from the column "other quality attributes" in Table 2. Among the quality attributes studied together, the combinations reliability-performance, reliability-cost, availability-cost , and cost-energy-consumption have received the biggest attention.

Domain, Dimensionality, and Phase: It can be observed from Table 1 that the majority of architecture level optimization approaches have been applied in the embedded systems domain, comprising 53% of the overall set of papers collected for this literature review, while a comparatively low number of approaches (22%) have been applied to enterprise information systems. The remaining approaches (26%) are either generic (i.e. have not clearly specified an application domain) or, from the evidence provided in the papers, apply to systems from both domains.

Concerning the dimensionality of the optimization problem, the approaches are almost evenly distributed between single-objective (SOO 39%) and multi-objective optimization problems (MTS 27% and MOO 31%).

Concerning the phase, the number of research contributions for design-time architecture optimization (67%) is significantly larger than that of run-time contributions (32%). With respect to quality attributes, reliability and safety are widely addressed at design-time.

Constraints: One major influence on the architecture of software systems used in the industry are constraints that need to be satisfied in order for the system to be accepted. However, a high number of papers (26% of overall collected 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, constraints add more complexity to the problem. If constraints are not considered, designers might have to rework the architecture in order to satisfy the constraints after the optimization process, which affects the quality of the system.

In the papers that consider constraints, the main focus was on cost, comprising 17% of the papers. This is not a surprising result since cost is often an important concern of the system architect. Other popular constraints are performance (14%), weight (11%), and physical constraints (7%). Little importance is given to some critical constraints such as memory (5%) and reliability (only 4% of the papers).

4.2 Solution

A summary of the solution-specific aspects that are extracted from the set of papers included in the literature

TABLE 2: Problem category - problem specific categorization of the reviewed approaches.

QA	Domain	Phase	Other Quality Attributes	Dimensionality	Constraints
Performance(84)	**ES(32)** [32], [45], [67]–[69], [80], [83], [84], [90], [93], [109], [112], [114], [117], [121]–[123], [126], [152], [153], [174]–[177], [179], [188], [213], [218], [220], [233], [235], [237], **GENERAL(21)** [1]–[3], [10], [15], [23], [26], [29], [46], [73], [87], [88], [141], [145], [189], [196]–[198], [210], [222], [239], **IS(31)** [12]–[14], [35], [38]–[40], [42]–[44], [66], [74], [77], [82], [124], [130], [133], [143], [144], [151], [160], [161], [168]–[170], [200], [226], [231], [236], [241], [245]	**DT(50)** [1]–[3], [23], [26], [32], [45], [67]–[69], [73], [80], [83], [84], [88], [90], [93], [109], [112], [114], [117], [121]–[123], [126], [133], [141], [143], [145], [152], [160], [161], [175]–[177], [179], [188], [189], [196]–[198], [210], [213], [218], [220], [222], [233], [235], [239], [245], **GENERAL(1)** [66], **RT(33)** [10], [12]–[15], [29], [35], [38]–[40], [42]–[44], [46], [74], [77], [82], [87], [124], [130], [144], [151], [153], [168]–[170], [174], [200], [226], [231], [236], [237], [241]	**AREA(3)** [45], [218], [220], **AVAILABILITY(18)** [12], [14], [15], [29], [39], [40], [42], [44], [74], [77], [82], [124], [130], [151], [170], [226], [236], [241], **COST(38)** [12]–[15], [32], [38]–[40], [42]–[44], [67], [68], [74], [77], [82], [84], [123], [124], [130], [143]–[145], [151], [152], [160], [161], [168], [177], [188], [200], [218], [220], [226], [231], [236], [237], [241], **ENERGY(6)** [67], [68], [84], [153], [188], [237], **GENERAL(2)** [145], [153], **MODIFIABILITY(3)** [196]–[198], **RELIABILITY(25)** [3], [39], [40], [66], [67], [73], [83], [88], [93], [121]–[124], [130], [133], [151], [153], [160], [161], [175]–[177], [226], [236], [241], **REPUTATION(4)** [15], [82], [124], [241], **SECURITY(1)** [170]	**GENERAL(1)** [32], **MOO(24)** [3], [45], [66], [67], [73], [84], [88], [93], [121], [123], [133], [143], [145], [152], [160], [161], [175]–[177], [188], [200], [218], [220], [237], **MTS(33)** [12]–[15], [29], [35], [38]–[40], [42]–[44], [74], [77], [82], [83], [122], [124], [130], [151], [153], [168]–[170], [179], [196]–[198], [222], [226], [231], [236], [241], **SOO(26)** [1], [2], [10], [23], [26], [46], [68], [69], [80], [87], [90], [109], [112], [114], [117], [126], [141], [144], [174], [189], [210], [213], [233], [235], [239], [245]	**AREA(3)** [23], [218], [235], **COST(6)** [3], [123], [144], [168], [218], [233], **DEPENDABILITY(1)** [176], **FUNCTIONAL CORRECTNESS(1)** [43], **GENERAL(7)** [3], [32], [39], [40], [121], [145], [177], **MAPPING(5)** [45], [144], [153], [175], [176], **MEMORY(6)** [45], [153], [175], [176], [220], [226], **NOT PRESENTED(28)** [13], [29], [38], [46], [66], [77], [82], [87], [88], [90], [109], [112], [124], [126], [133], [143], [152], [160], [161], [170], [196]–[198], [200], [210], [231], [241], [245], **PATH LOSS(1)** [237], **PERFORMANCE(14)** [26], [35], [67]–[69], [83], [93], [122], [144], [168], [174], [179], [218], [220], **PHYSICAL(8)** [1], [2], [114], [117], [189], [222], [237], [239], **PRECEDENCE(8)** [1], [2], [26], [73], [117], [141], [189], [239], **PROCESSING POWER(1)** [226], **QOS VALUES(10)** [12], [14], [42]–[44], [74], [130], [151], [169], [236], **REDUNDANCY LEVEL(1)** [130], **REQUIREMENTS(1)** [15], **STABILITY(1)** [43], **STRUCTURAL(4)** [10], [15], [84], [188], **THROUGHPUT(2)** [174], [213], **TIMING(8)** [12], [14], [80], [114], [123], [141], [189], [222], **WEIGHT(1)** [3]
Cost(74)	**ES(35)** [9], [18], [32], [47], [49], [58], [67], [68], [72], [84], [100], [102], [108], [118], [123], [127], [128], [137], [142], [148], [152], [157], [177], [178], [180], [181], [188], [209], [215]–[218], [220], [221], [237], **GENERAL(13)** [15], [62]–[64], [81], [145], [147], [149], [190], [207], [227]–[229], **IS(26)** [12]–[14], [38]–[44], [74], [77], [82], [124], [130], [143], [144], [151], [160], [161], [168], [200], [226], [231], [236], [241]	**DT(47)** [9], [18], [32], [47], [49], [58], [62], [63], [67], [68], [72], [81], [84], [100], [102], [108], [118], [123], [128], [137], [142], [143], [145], [147]–[149], [152], [157], [160], [161], [177], [178], [180], [181], [188], [190], [207], [209], [215]–[218], [220], [221], [227]–[229], **GENERAL(1)** [127], **RT(26)** [12]–[15], [38]–[44], [64], [74], [77], [82], [124], [130], [144], [151], [168], [200], [226], [231], [236], [237], [241]	**AREA(2)** [218], [220], **AVAILABILITY(20)** [12], [14], [15], [39], [40], [42], [44], [74], [77], [81], [82], [100], [124], [130], [151], [157], [217], [226], [236], [241], **ENERGY(6)** [67], [68], [72], [84], [188], [237], **GENERAL(1)** [145], **PERFORMANCE(38)** [12]–[15], [32], [38]–[40], [42]–[44], [67], [68], [74], [77], [82], [84], [123], [124], [130], [143]–[145], [151], [152], [160], [161], [168], [177], [188], [200], [218], [220], [226], [231], [236], [237], [241], **RELIABILITY(24)** [18], [39], [40], [67], [102], [118], [123], [124], [128], [130], [137], [147]–[149], [151], [160], [161], [177], [178], [215], [216], [226], [236], [241], **REPUTATION(4)** [15], [82], [124], [241], **SAFETY(2)** [178], [180], **WEIGHT(5)** [137], [147], [215]–[217]	**GENERAL(2)** [32], [49], **MOO(31)** [18], [67], [72], [84], [100], [102], [118], [123], [128], [137], [143], [145], [147]–[149], [152], [157], [160], [161], [177], [188], [190], [200], [215]–[218], [220], [228], [229], [237], **MTS(26)** [12]–[15], [38]–[40], [42]–[44], [74], [77], [81], [82], [124], [130], [151], [168], [178], [180], [207], [226], [227], [231], [236], [241], **SOO(16)** [9], [41], [47], [58], [62]–[64], [68], [108], [127], [142], [144], [148], [181], [209], [221]	**AREA(1)** [218], **AVAILABILITY(2)** [181], [221], **COST(7)** [41], [72], [123], [144], [148], [168], [218], **DELIVERY TIME(3)** [63], [64], [190], **DESIGN(1)** [180], **FUNCTIONAL CORRECTNESS(1)** [43], **GENERAL(8)** [18], [32], [39], [40], [81], [145], [149], [177], **MAPPING(1)** [144], **MEMORY(2)** [220], [226], **NOT PRESENTED(19)** [13], [38], [77], [82], [100], [124], [128], [143], [152], [157], [160], [161], [178], [200], [215]–[217], [231], [241], **PATH LOSS(1)** [237], **PERFORMANCE(10)** [47], [49], [67], [68], [127], [142], [144], [168], [218], [220], **PHYSICAL(2)** [108], [237], **PROCESSING POWER(1)** [226], **QOS VALUES(9)** [12], [14], [42]–[44], [74], [130], [151], [236], **REDUNDANCY LEVEL(2)** [130], [147], **RELIABILITY(6)** [9], [58], [63], [64], [190], [209], **REQUIREMENTS(6)** [15], [62], [207], [227]–[229], **STABILITY(1)** [43], **STRUCTURAL(3)** [15], [84], [188], **TIMING(3)** [12], [14], [123], **VOLUME(2)** [118], [147], **WEIGHT(5)** [102], [118], [147], [148], [209]
Reliability(71)	**ES(40)** [18], [51]–[57], [59]–[61], [67], [83], [93], [102], [118], [121]–[123], [128], [136], [137], [146], [148], [153], [155], [164], [165], [175]–[178], [186], [199], [204], [214]–[216], [234], [244], **GENERAL(18)** [3], [7], [30], [31], [73], [75], [76], [88], [96], [97], [147], [149], [182], [183], [187], [192], [232], [238], **IS(13)** [39], [40], [66], [124], [130], [131], [133], [151], [160], [161], [226], [236], [241]	**DT(56)** [3], [7], [18], [30], [31], [51]–[57], [59], [60], [67], [73], [75], [76], [83], [88], [93], [96], [97], [102], [118], [121]–[123], [128], [133], [136], [137], [146]–[149], [155], [160], [161], [164], [165], [175]–[178], [182], [183], [199], [204], [214]–[216], [232], [234], [238], [244], **GENERAL(1)** [66], **RT(14)** [39], [40], [61], [124], [130], [131], [151], [153], [186], [187], [192], [226], [236], [241]	**AVAILABILITY(9)** [39], [40], [61], [124], [130], [151], [226], [236], [241], **COST(24)** [18], [39], [40], [67], [102], [118], [123], [124], [128], [130], [137], [147]–[149], [151], [160], [161], [177], [178], [215], [216], [226], [236], [241], **ENERGY(4)** [67], [153], [155], [165], **GENERAL(1)** [153], **PERFORMANCE(25)** [3], [39], [40], [66], [67], [73], [83], [88], [93], [121]–[124], [130], [133], [151], [153], [160], [161], [175]–[177], [226], [236], [241], **REPUTATION(2)** [124], [241], **SAFETY(1)** [178], **WEIGHT(4)** [137], [147], [215], [216]	**MOO(31)** [3], [18], [53], [54], [66], [67], [73], [88], [93], [102], [118], [121], [123], [128], [133], [137], [147]–[149], [160], [161], [164], [165], [175]–[177], [199], [214]–[216], [234], **MTS(15)** [39], [40], [61], [83], [122], [124], [130], [131], [151], [153], [155], [178], [226], [236], [241], **SOO(26)** [7], [30], [31], [51], [52], [55]–[57], [59], [60], [75], [76], [96], [97], [136], [146], [148], [182], [183], [186], [187], [192], [204], [232], [238], [244]	**COST(24)** [3], [7], [31], [51], [52], [55]–[57], [76], [96], [97], [123], [131], [136], [146], [148], [182], [183], [199], [204], [232], [234], [238], [244], **DEPENDABILITY(1)** [176], **GENERAL(12)** [3], [18], [30], [39], [40], [53], [54], [59], [60], [121], [149], [177], **MAPPING(3)** [153], [175], [176], **MEMORY(5)** [153], [164], [175], [176], [226], **NOT PRESENTED(13)** [66], [88], [124], [128], [133], [160], [161], [178], [186], [187], [215], [216], [241], **PERFORMANCE(5)** [67], [83], [93], [122], [214], **PHYSICAL(3)** [155], [192], [199], **PRECEDENCE(2)** [73], [192], **PROCESSING POWER(1)** [226], **QOS VALUES(3)** [130], [151], [236], **REDUNDANCY LEVEL(3)** [130], [147], [165], **RELIABILITY(1)** [61], **TIMING(2)** [123], [192], **VOLUME(6)** [7], [75], [76], [118], [147], [238], **WEIGHT(19)** [3], [7], [31], [51], [52], [55]–[57], [75], [76], [102], [118], [136], [146]–[148], [182], [199], [238]
Availability(25)	**ES(5)** [61], [100], [157], [173], [217], **GENERAL(3)** [15], [29], [81], **IS(17)** [12], [14], [39], [40], [42], [44], [74], [77], [82], [106], [124], [130], [151], [170], [226], [236], [241]	**DT(5)** [81], [100], [157], [173], [217], **RT(20)** [12], [14], [15], [29], [39], [40], [42], [44], [61], [74], [77], [82], [106], [124], [130], [151], [170], [226], [236], [241]	**COST(20)** [12], [14], [15], [39], [40], [42], [44], [74], [77], [81], [82], [100], [124], [130], [151], [157], [217], [226], [236], [241], **PERFORMANCE(18)** [12], [14], [15], [29], [39], [40], [42], [44], [74], [77], [82], [124], [130], [151], [170], [226], [236], [241], **RELIABILITY(9)** [39], [40], [61], [124], [130], [151], [226], [236], [241], **REPUTATION(4)** [15], [82], [124], [241], **SECURITY(1)** [170], **WEIGHT(1)** [217]	**MOO(3)** [100], [157], [217], **MTS(20)** [12], [14], [15], [29], [39], [40], [42], [44], [61], [74], [77], [81], [82], [124], [130], [151], [170], [226], [236], [241], **SOO(2)** [106], [173]	**COST(1)** [106], **GENERAL(3)** [39], [40], [81], **MAPPING(1)** [173], **MEMORY(2)** [173], [226], **NOT PRESENTED(9)** [29], [77], [82], [100], [124], [157], [170], [217], [241], **PROCESSING POWER(1)** [226], **QOS VALUES(8)** [12], [14], [42], [44], [74], [130], [151], [236], **REDUNDANCY LEVEL(1)** [130], **RELIABILITY(1)** [61], **REQUIREMENTS(1)** [15], **STRUCTURAL(1)** [15], **TIMING(2)** [12], [14]
General(22)	**ES(12)** [5], [33], [36], [79], [86], [91], [95], [138], [139], [150], [153], [224], **GENERAL(4)** [21], [145], [162], [205], **IS(6)** [71], [113], [202], [219], [242], [243]	**DT(10)** [5], [33], [79], [95], [138], [139], [145], [150], [205], [224], **RT(13)** [21], [36], [71], [86], [91], [95], [113], [153], [162], [202], [219], [242], [243]	**COST(1)** [145], **ENERGY(1)** [153], **PERFORMANCE(2)** [145], [153], **RELIABILITY(1)** [153]	**GENERAL(5)** [21], [36], [91], [162], [224], **MOO(7)** [5], [33], [95], [138], [139], [145], [150], **MTS(10)** [71], [79], [86], [113], [153], [202], [205], [219], [242], [243]	**COST(1)** [33], **GENERAL(10)** [71], [79], [86], [113], [138], [139], [145], [205], [224], [243], **MAPPING(3)** [5], [95], [153], **MEMORY(2)** [5], [153], **NOT PRESENTED(7)** [21], [36], [91], [162], [202], [219], [242], **PERFORMANCE(1)** [33], **STRUCTURAL(1)** [150]
Energy(18)	**ES(17)** [17], [28], [67], [68], [72], [84], [116], [153], [155], [165], [188], [193], [194], [211], [212], [237], [240], **GENERAL(1)** [206]	**DT(9)** [67], [68], [72], [84], [116], [155], [165], [188], [240], **GENERAL(1)** [206], **RT(8)** [17], [28], [153], [193], [194], [211], [212], [237]	**COST(6)** [67], [68], [72], [84], [188], [237], **GENERAL(1)** [153], **PERFORMANCE(6)** [67], [68], [84], [153], [188], [237], **RELIABILITY(4)** [67], [153], [155], [165]	**MOO(7)** [67], [72], [84], [165], [188], [206], [237], **MTS(2)** [153], [155], **SOO(9)** [17], [28], [68], [116], [193], [194], [211], [212], [240]	**COST(1)** [72], **GENERAL(1)** [194], **MAPPING(1)** [153], **MEMORY(1)** [153], **NOT PRESENTED(1)** [206], **PATH LOSS(1)** [237], **PERFORMANCE(8)** [17], [28], [67], [68], [116], [193], [211], [212], **PHYSICAL(3)** [155], [237], [240], **REDUNDANCY LEVEL(1)** [165], **STRUCTURAL(2)** [84], [188], **TIMING(1)** [240]
Weight (5)	**ES(4)** [137], [215]–[217], **GENERAL(1)** [147]	**DT(5)** [137], [147], [215]–[217]	**AVAILABILITY(1)** [217], **COST(5)** [137], [147], [215]–[217], **RELIABILITY(4)** [137], [147], [215], [216]	**MOO(5)** [137], [147], [215]–[217]	**NOT PRESENTED(3)** [215]–[217], **REDUNDANCY LEVEL(1)** [147], **VOLUME(1)** [147], **WEIGHT(1)** [147]
Safety (4)	**ES(4)** [178], [180], [184], [223]	**DT(4)** [178], [180], [184], [223]	**COST(2)** [178], [180], **RELIABILITY(1)** [178]	**MOO(2)** [184], [223], **MTS(2)** [178], [180]	**DESIGN(1)** [180], **NOT PRESENTED(3)** [178], [184], [223]

review is given in Table 3. In the following, we summarize the main results for each solution subcategory, observable in Tables 4 and 5.

TABLE 3: Solution category - quantitative summary of the results.

Degrees of Freedom			**Architecture Representation**		
Allocation	59	31%	Architecture model	43	23%
Hardware replication	40	21%	- UML	5	3%
Hardware selection	38	20%	- ADL	7	4%
Software replication	35	18%	- Custom arch. model	9	5%
Scheduling	33	17%	- Workflow language	17	9%
Component selection	30	16%	- General	5	3%
Service selection	28	15%	Quality eval. model	65	34%
Software selection	24	13%	Optimization model	68	36%
Other problem specific	18	9%			
Service composition	12	6%			
Software parameters	10	5%			
Clustering	5	3%	**Optimization Strategy**		
General	5	3%	Approximative	149	78%
Hardware parameters	4	2%	Exact	38	20%
Architectural pattern	3	2%	Not presented	9	5%
Not presented	3	2%	General	4	2%
Partitioning	2	1%			
Maintenance schedules	2	1%			
Quality Evaluation			**Constraint Handling**		
SAF	80	42%	Prohibit	84	44%
MB	60	32%	Not presented	61	32%
NMF	40	21%	Penalty	36	19%
Not presented	6	3%	Repair	9	5%
General	6	3%	General	1	$< 1\%$

Architecture Representation: We observe that most approaches directly use either a quality evaluation model (34%) or a optimization model (39%) as an input. Only 23% of the approaches take an architecture model as an input. Amongst them, most models are workflow languages for service-based systems (9%). UML, ADLs, and custom architecture models are used similarly often with 3%, 4%, and 5%, respectively. Some of the architecture model-based optimization approaches are general (3%), i.e. designed to be extendable to other than the mentioned modelling language.

Quality Evaluation: Quality evaluation is an important part of the architecture optimization process, since it provides a quantitative metric for the quality of the system based on the architecture specification, which in turn is used as an indicator of the fitness of the solutions produced by the optimization algorithm. The majority of the studies use a Simple Aggregation Function (SAF) (42%) a Model-Based (MB) technique (32%), or a Non-linear Mathematical Function (NMF) (21%). In comparison, SAF and NMF are easier to model and to integrate into the optimization problem. However, they often are not as accurate and as realistic as Model-Based (MB) techniques, since they omit details and dependencies.

For the model-based approaches, different quality evaluation techniques have been used, implied by the models used for specific quality attributes. As an example reliability block diagrams [57], [58], [63], [121], [136], [137], [149], [208], [217], discrete-time Markov chains [61], [96], [97], [165], [232], and fault trees [67], [184], [199] are used for reliability; queuing networks [35], [80], [143], [144], [168], [169], [171], [245], execution graphs [85], [107], [115], and discrete-time Markov chains [210] are used to evaluate performance; fault trees [8], [180], [184], [185], [201], [223] and binary decision diagrams [8], [185] are used for safety evaluation; continuous-time Markov chains [193], Markov decision processes [212], Petri-nets [194], and Markov reward models [165] are used for evaluation of a system's energy consumption. Quantitative metrics of the quality attributes are obtained by either mathematically analyzing or simulating the models. For an overview of the different evaluation models and techniques several surveys can be recommended, e.g. for reliability [99], performance [22], [134], energy consumption [27], and safety [104].

Degrees of Freedom: Allocation, hardware replication, and hardware selection are the most intensively studied degrees of freedom with 31%, 21%, and 20% of the overall papers, respectively. Other popular degrees of freedom are software replication (18%), scheduling (17%), component selection (16%) and service selection (15%). A small amount of papers (9%) presents a problem-specific degree of freedom, such as changing the transmission power in communicating embedded systems or decisions on whether to implement a certain functionality in software or hardware.

Optimization Strategy: When the search time and resources used to perform the optimization process are limited and near-optimal solutions are good enough for the given problem, then approximate algorithms are the right optimization tool. However, if the goal is to find the optimal solutions, and if the resources and time are unlimited then one should choose exact optimization algorithms. This is an important trade-off that should be made when choosing an optimization algorithm. Assuming problems of non-trivial size, the complexity of the problem is the most important factor that needs to be taken into account.

The majority of the approaches use approximate methods (mostly metaheuristics) as an optimization technique, comprising 78% of the overall approaches. The main reason for using approximate methods is the difficulty of the search-space, in which often an exhaustive search is not feasible in polynomial time. Moreover, the objective functions are usually computationally expensive and non-linear. Listing all possible solutions in order to find the best candidates is a non-deterministic

TABLE 4: Solution category - architecture representation, quality-evaluation and degree-of-freedom specific categorization of the reviewed approaches.

QA	Architecture Representation	Quality Evaluation	Degrees of Freedom
Performance(84)	**ADL(4)** [145], [160], [161], [170], **ANY(2)** [133], [153], **CUSTOMAM(5)** [90], [152], [196]–[198], **OPTIMPL(3)** [35], [109], [169], **OPTSTRUC(11)** [3], [45], [46], [87], [141], [143], [177], [189], [222], [226], [237], **QUALMM(42)** [1], [2], [10], [23], [32], [39], [43], [66]–[68], [73], [77], [80], [83], [84], [88], [93], [112], [114], [117], [121]–[124], [126], [144], [151], [168], [174]–[176], [179], [188], [210], [218], [220], [231], [233], [235], [236], [239], [245], **QUALSTD(3)** [26], [69], [213], **UML(1)** [241], **WFL(13)** [12]–[15], [29], [38], [40], [42], [44], [74], [82], [130], [200]	**GENERAL(1)** [3], **MB(27)** [10], [23], [26], [35], [45], [66]–[69], [80], [90], [117], [121], [123], [133], [143]–[145], [152], [160], [161], [168], [169], [176], [177], [210], [245], **NMF(11)** [46], [73], [83], [87], [88], [93], [122], [175], [189], [220], [239], **NOT PRESENTED(1)** [141], **SAF(44)** [1], [2], [12]–[15], [29], [32], [38]–[40], [42]–[44], [74], [77], [82], [84], [109], [112], [114], [124], [126], [130], [151], [153], [170], [174], [179], [188], [196]–[198], [200], [213], [218], [222], [226], [231], [233], [235]–[237], [241]	**ALLOCATION(37)** [2], [10], [23], [32], [45], [46], [80], [83], [84], [88], [112], [114], [117], [121]–[123], [126], [133], [144], [145], [152], [153], [160], [161], [175]–[177], [179], [188], [189], [210], [213], [220], [222], [233], [239], [245], **ARCHITECTURAL PATTERN(3)** [196]–[198], **CLUSTERING(4)** [67]–[69], [122], **COMPONENT SELECTION(6)** [45], [90], [109], [160], [161], [177], **HARDWARE PARAMETERS(3)** [143], [160], [161], **HARDWARE REPLICATION(3)** [66], [144], [177], **HARDWARE SELECTION(8)** [66], [133], [145], [160], [161], [176], [177], [220], **OTHER PROBLEM SPECIFIC(12)** [15], [29], [38], [39], [42], [44], [74], [130], [151], [170], [235], [237], **PARTITIONING(2)** [35], [218], **SCHEDULING(25)** [1], [2], [26], [32], [45], [46], [67]–[69], [73], [83], [87], [93], [114], [117], [123], [141], [174], [179], [189], [213], [220], [222], [239], [245], **SERVICE COMPOSITION(8)** [40], [77], [82], [124], [226], [231], [236], [241], **SERVICE SELECTION(20)** [12]–[15], [38], [39], [42]–[44], [74], [77], [82], [124], [130], [151], [168]–[170], [200], [241], **SOFTWARE PARAMETERS(2)** [26], [143], **SOFTWARE REPLICATION(3)** [3], [93], [177], **SOFTWARE SELECTION(1)** [133]
Cost(74)	**ADL(3)** [145], [160], [161], **CUSTOMAM(3)** [49], [102], [152], **OPTIMPL(2)** [178], [180], **OPTSTRUC(28)** [9], [18], [58], [62], [64], [81], [108], [118], [128], [137], [143], [147], [148], [157], [177], [181], [190], [207], [209], [215]–[217], [221], [226]–[229], [237], **QUALMM(23)** [32], [39], [41], [43], [47], [67], [68], [72], [77], [84], [123], [124], [127], [142], [144], [149], [151], [168], [188], [218], [220], [231], [236], **QUALSTD(1)** [100], **UML(2)** [63], [241], **WFL(12)** [12]–[15], [38], [40], [42], [44], [74], [82], [130], [200]	**GENERAL(1)** [147], **MB(19)** [47], [67], [68], [72], [100], [102], [123], [127], [143]–[145], [149], [152], [157], [160], [161], [168], [177], [180], **NMF(10)** [9], [18], [58], [81], [118], [128], [137], [148], [217], [220], **NOT PRESENTED(1)** [178], **SAF(44)** [12]–[15], [32], [38]–[44], [49], [62]–[64], [74], [77], [82], [84], [108], [124], [130], [142], [147], [151], [181], [188], [190], [200], [207], [209], [215], [216], [218], [221], [226]–[229], [231], [236], [237], [241]	**ALLOCATION(17)** [32], [47], [72], [84], [108], [123], [127], [142], [144], [145], [152], [160], [161], [177], [178], [188], [220], **CLUSTERING(2)** [67], [68], **COMPONENT SELECTION(19)** [9], [18], [62]–[64], [142], [147], [149], [157], [160], [161], [177], [178], [190], [207], [209], [227]–[229], **HARDWARE PARAMETERS(3)** [143], [160], [161], **HARDWARE REPLICATION(18)** [58], [81], [100], [102], [118], [137], [144], [147]–[149], [157], [177], [181], [209], [215]–[217], [221], **HARDWARE SELECTION(18)** [58], [72], [81], [100], [118], [137], [145], [148], [160], [161], [177], [178], [181], [215]–[217], [220], [221], **MAINTENANCE SCHEDULES(2)** [100], [180], **NOT PRESENTED(1)** [128], **OTHER PROBLEM SPECIFIC(11)** [15], [38], [39], [42], [44], [49], [64], [74], [130], [151], [237], **PARTITIONING(1)** [218], **SCHEDULING(9)** [32], [47], [67], [68], [72], [123], [127], [142], [220], **SERVICE COMPOSITION(8)** [40], [77], [82], [124], [226], [231], [236], [241], **SERVICE SELECTION(19)** [12]–[15], [38], [39], [41]–[44], [74], [77], [82], [124], [130], [151], [168], [200], [241], **SOFTWARE PARAMETERS(2)** [143], [180], **SOFTWARE REPLICATION(12)** [58], [81], [102], [118], [137], [148], [177], [209], [215]–[217], [221], **SOFTWARE SELECTION(10)** [58], [81], [118], [137], [148], [181], [215]–[217], [221]
Reliability(71)	**ADL(4)** [61], [160], [161], [165], **ANY(2)** [133], [153], **CUSTOMAM(2)** [102], [164], **OPTIMPL(1)** [178], **OPTSTRUC(36)** [3], [7], [18], [30], [31], [51]–[57], [59], [60], [75], [76], [96], [97], [118], [128], [136], [137], [146]–[148], [177], [182], [183], [187], [204], [215], [216], [226], [234], [238], [244], **QUALMM(21)** [39], [66], [67], [73], [83], [88], [93], [121]–[124], [131], [149], [151], [155], [175], [176], [186], [192], [214], [236], **QUALSTD(2)** [199], [232], **UML(1)** [241], **WFL(2)** [40], [130]	**GENERAL(4)** [3], [53], [147], [182], **MB(23)** [7], [61], [66], [67], [96], [97], [102], [121], [123], [133], [149], [160], [161], [164], [165], [176], [177], [187], [199], [214], [232], [238], [244], **NMF(27)** [18], [31], [51], [52], [54]–[57], [59], [60], [73], [75], [76], [83], [88], [93], [118], [122], [128], [136], [137], [146], [148], [175], [186], [204], [234], **NOT PRESENTED(1)** [178], **SAF(17)** [30], [39], [40], [124], [130], [131], [147], [151], [153], [155], [183], [192], [215], [216], [226], [236], [241]	**ALLOCATION(20)** [61], [83], [88], [121]–[123], [133], [153], [155], [160], [161], [164], [175]–[178], [186], [187], [192], [214], **CLUSTERING(3)** [67], [122], [214], **COMPONENT SELECTION(12)** [7], [18], [96], [97], [147], [149], [160], [161], [177], [178], [182], [232], **HARDWARE PARAMETERS(2)** [160], [161], **HARDWARE REPLICATION(29)** [31], [51]–[57], [59], [60], [66], [75], [76], [102], [118], [136], [137], [146]–[149], [165], [177], [199], [204], [215], [216], [234], [244], **HARDWARE SELECTION(26)** [51]–[55], [57], [59], [60], [66], [118], [133], [136], [137], [146], [148], [160], [161], [176]–[178], [183], [199], [204], [215], [216], [234], **NOT PRESENTED(2)** [30], [128], **OTHER PROBLEM SPECIFIC(3)** [39], [130], [151], **SCHEDULING(6)** [67], [73], [83], [93], [123], [192], **SERVICE COMPOSITION(6)** [40], [124], [131], [226], [236], [241], **SERVICE SELECTION(5)** [39], [124], [130], [151], [241], **SOFTWARE REPLICATION(29)** [3], [7], [31], [51]–[57], [59], [60], [75], [76], [93], [102], [118], [136], [137], [146], [148], [165], [177], [204], [215], [216], [234], [238], [244], **SOFTWARE SELECTION(19)** [51]–[55], [57], [59], [60], [118], [133], [136], [137], [146], [148], [183], [204], [215], [216], [234]
Availability(25)	**ADL(2)** [61], [170], **OPTSTRUC(4)** [81], [157], [217], [226], **QUALMM(6)** [39], [77], [124], [151], [173], [236], **QUALSTD(1)** [100], **UML(1)** [241], **WFL(11)** [12], [14], [15], [29], [40], [42], [44], [74], [82], [106], [130]	**MB(4)** [61], [100], [157], [173], **NMF(2)** [81], [217], **SAF(19)** [12], [14], [15], [29], [39], [40], [42], [44], [74], [77], [82], [106], [124], [130], [151], [170], [226], [236], [241]	**ALLOCATION(2)** [61], [173], **COMPONENT SELECTION(1)** [157], **HARDWARE REPLICATION(4)** [81], [100], [157], [217], **HARDWARE SELECTION(3)** [81], [100], [217], **MAINTENANCE SCHEDULES(1)** [100], **OTHER PROBLEM SPECIFIC(9)** [15], [29], [39], [42], [44], [74], [130], [151], [170], **SERVICE COMPOSITION(8)** [40], [77], [82], [106], [124], [226], [236], [241], **SERVICE SELECTION(14)** [12], [14], [15], [39], [42], [44], [74], [77], [82], [124], [130], [151], [170], [241], **SOFTWARE REPLICATION(2)** [81], [217], **SOFTWARE SELECTION(2)** [81], [217]
General(22)	**ADL(2)** [5], [145], **ANY(4)** [79], [153], [205], [224], **OPTSTRUC(6)** [86], [113], [138], [139], [219], [242], **QUALMM(5)** [21], [33], [95], [150], [202], **UML(3)** [36], [91], [162], **WFL(3)** [71], [162], [243]	**GENERAL(2)** [138], [139], **MB(5)** [5], [33], [95], [145], [224], **NMF(1)** [79], **NOT PRESENTED(4)** [36], [91], [242], [243], **SAF(10)** [21], [71], [86], [113], [150], [153], [162], [202], [205], [219]	**ALLOCATION(7)** [5], [21], [33], [95], [145], [150], [153], **COMPONENT SELECTION(1)** [36], **GENERAL(5)** [79], [138], [139], [205], [224], **HARDWARE REPLICATION(1)** [95], **HARDWARE SELECTION(3)** [33], [145], [150], **NOT PRESENTED(1)** [91], **OTHER PROBLEM SPECIFIC(2)** [71], [113], **SCHEDULING(1)** [33], **SERVICE COMPOSITION(2)** [202], [219], **SERVICE SELECTION(6)** [113], [162], [202], [219], [242], [243], **SOFTWARE PARAMETERS(1)** [86]
Energy(18)	**ADL(1)** [165], **ANY(1)** [153], **OPTSTRUC(2)** [237], [240], **QUALMM(11)** [17], [28], [67], [68], [72], [84], [116], [155], [188], [206], [211], **QUALSTD(3)** [193], [194], [212]	**MB(9)** [28], [67], [68], [72], [165], [193], [194], [206], [212], **NMF(3)** [17], [116], [240], **SAF(6)** [84], [153], [155], [188], [211], [237]	**ALLOCATION(7)** [72], [84], [153], [155], [188], [206], [240], **CLUSTERING(2)** [67], [68], **COMPONENT SELECTION(1)** [206], **HARDWARE PARAMETERS(1)** [116], **HARDWARE REPLICATION(1)** [165], **HARDWARE SELECTION(2)** [72], [116], **OTHER PROBLEM SPECIFIC(1)** [237], **SCHEDULING(5)** [67], [68], [72], [206], [211], **SOFTWARE PARAMETERS(6)** [17], [28], [193], [194], [211], [212], **SOFTWARE REPLICATION(1)** [165]
Weight (5)	**OPTSTRUC(5)** [137], [147], [215]–[217]	**GENERAL(1)** [147], **NMF(2)** [137], [217], **SAF(3)** [147], [215], [216]	**COMPONENT SELECTION(1)** [147], **HARDWARE REPLICATION(5)** [137], [147], [215]–[217], **HARDWARE SELECTION(4)** [137], [215]–[217], **SOFTWARE REPLICATION(4)** [137], [215]–[217], **SOFTWARE SELECTION(4)** [137], [215]–[217]
Safety (4)	**CUSTOMAM(1)** [184], **OPTIMPL(2)** [178], [180], **OPTSTRUC(1)** [223]	**MB(3)** [180], [184], [223], **NOT PRESENTED(1)** [178], **SAF(1)** [223]	**ALLOCATION(1)** [178], **COMPONENT SELECTION(2)** [178], [184], **HARDWARE REPLICATION(1)** [223], **HARDWARE SELECTION(1)** [178], **MAINTENANCE SCHEDULES(1)** [180], **OTHER PROBLEM SPECIFIC(1)** [184], **SOFTWARE PARAMETERS(1)** [180], **SOFTWARE REPLICATION(1)** [223]

TABLE 5: Solution category - optimization specific categorization of the reviewed approaches.

QA	Optimization Strategy Type	Constraint Handling
Performance(84)	**EXACT PROBLEM-SPECIFIC(3)** [1], [112], [126], **EXACT STANDARD(14)** [12]–[15], [42]–[44], [69], [117], [169], [174], [218], [236], [241], **GENERAL(1)** [153], **METAHEURISTIC(45)** [2], [3], [23], [29], [32], [35], [38]–[40], [45], [46], [66], [73], [77], [83], [84], [88], [114], [122]–[124], [130], [133], [141], [143], [145], [160], [161], [170], [175]–[177], [188], [189], [196]–[198], [200], [220], [222], [226], [231], [233], [235], [237], **NOT PRESENTED(3)** [74], [90], [152], **PROBLEM-SPECIFIC HEURISTIC(23)** [1], [10], [26], [29], [67], [68], [80], [82], [87], [93], [109], [112], [121], [144], [151], [168], [179], [210], [213], [236], [239], [241], [245], **WITH GUARANTEE(2)** [112], [218]	**GENERAL(1)** [175], **NOT PRESENTED(30)** [13], [29], [38], [46], [66], [74], [77], [82], [87], [88], [90], [109], [112], [124], [126], [133], [143], [151], [152], [160], [161], [170], [196]–[198], [200], [210], [231], [241], [245], **PENALTY(11)** [32], [35], [39], [40], [80], [83], [189], [222], [226], [235], [237], **PROHIBIT(36)** [1]–[3], [10], [12], [14], [15], [23], [42]–[45], [69], [73], [93], [114], [117], [122], [123], [141], [144], [145], [153], [168], [169], [174], [176], [177], [179], [213], [218], [220], [233], [235], [236], [239], **REPAIR(8)** [26], [32], [67], [68], [84], [121], [130], [188]
Cost(74)	**EXACT PROBLEM-SPECIFIC(2)** [9], [229], **EXACT STANDARD(13)** [12]–[15], [42]–[44], [62]–[64], [218], [236], [241], **GENERAL(1)** [180], **METAHEURISTIC(45)** [18], [32], [38]–[41], [58], [72], [77], [81], [84], [100], [102], [108], [118], [123], [124], [128], [130], [137], [143], [145], [147]–[149], [157], [160], [161], [177], [178], [181], [188], [200], [209], [215]–[217], [220], [221], [226]–[229], [231], [237], **NOT PRESENTED(3)** [74], [152], [190], **PROBLEM-SPECIFIC HEURISTIC(13)** [47], [49], [67], [68], [82], [127], [142], [144], [151], [168], [207], [236], [241], **WITH GUARANTEE(1)** [218]	**NOT PRESENTED(25)** [9], [13], [38], [41], [74], [77], [82], [100], [124], [128], [137], [143], [151], [152], [157], [160], [161], [178], [200], [207], [215]–[217], [231], [241], **PENALTY(12)** [32], [39], [40], [72], [81], [108], [118], [148], [181], [209], [226], [237], **PROHIBIT(30)** [12], [14], [15], [18], [42]–[44], [47], [49], [58], [62]–[64], [102], [123], [127], [144], [145], [147], [149], [168], [177], [180], [190], [218], [220], [221], [227], [229], [236], **REPAIR(7)** [32], [67], [68], [84], [130], [142], [188]
Reliability(71)	**EXACT PROBLEM-SPECIFIC(2)** [155], [186], **EXACT STANDARD(7)** [7], [51], [52], [54], [56], [236], [241], **GENERAL(2)** [53], [153], **METAHEURISTIC(49)** [3], [18], [30], [39], [40], [57], [59], [60], [66], [73], [75], [76], [83], [88], [96], [97], [102], [118], [122]–[124], [128], [130], [133], [136], [137], [146]–[149], [160], [161], [164], [165], [175]–[178], [182], [183], [199], [204], [215], [216], [226], [232], [234], [238], [244], **NOT PRESENTED(2)** [61], [131], **PROBLEM-SPECIFIC HEURISTIC(11)** [55], [67], [93], [121], [151], [155], [187], [192], [214], [236], [241], **WITH GUARANTEE(1)** [31]	**GENERAL(1)** [175], **NOT PRESENTED(18)** [53], [61], [66], [88], [124], [128], [131], [133], [137], [151], [160], [161], [178], [186], [187], [215], [216], [241], **PENALTY(19)** [30], [39], [40], [56], [60], [75], [76], [83], [96], [97], [118], [136], [146], [148], [199], [204], [226], [232], [234], **PROHIBIT(30)** [3], [7], [18], [31], [51], [52], [54], [55], [57], [59], [73], [93], [102], [122], [123], [147], [149], [153], [155], [164], [165], [176], [177], [182], [183], [192], [214], [236], [238], [244], **REPAIR(3)** [67], [121], [130]
Availability(25)	**EXACT STANDARD(7)** [12], [14], [15], [42], [44], [236], [241], **METAHEURISTIC(12)** [29], [39], [40], [77], [81], [100], [124], [130], [157], [170], [217], [226], **NOT PRESENTED(2)** [61], [74], **PROBLEM-SPECIFIC HEURISTIC(7)** [29], [82], [106], [151], [173], [236], [241]	**NOT PRESENTED(12)** [29], [61], [74], [77], [82], [100], [124], [151], [157], [170], [217], [241], **PENALTY(4)** [39], [40], [81], [226], **PROHIBIT(8)** [12], [14], [15], [42], [44], [106], [173], [236], **REPAIR(1)** [130]
General (22)	**EXACT PROBLEM-SPECIFIC(2)** [79], [150], **EXACT STANDARD(1)** [162], **GENERAL(3)** [153], [205], [224], **METAHEURISTIC(10)** [5], [33], [71], [95], [138], [139], [145], [202], [219], [242], **NOT PRESENTED(3)** [21], [36], [91], **PROBLEM-SPECIFIC HEURISTIC(4)** [86], [113], [150], [243], **WITH GUARANTEE(1)** [79]	**NOT PRESENTED(10)** [21], [36], [91], [150], [162], [202], [219], [224], [242], [243], **PENALTY(5)** [33], [71], [113], [138], [139], **PROHIBIT(7)** [5], [79], [86], [95], [145], [153], [205]
Energy (18)	**EXACT PROBLEM-SPECIFIC(2)** [17], [155], **EXACT STANDARD(4)** [193], [194], [211], [212], **GENERAL(1)** [153], **METAHEURISTIC(6)** [72], [84], [165], [188], [237], [240], **NOT PRESENTED(1)** [206], **PROBLEM-SPECIFIC HEURISTIC(7)** [17], [28], [67], [68], [116], [155], [193]	**NOT PRESENTED(1)** [206], **PENALTY(2)** [72], [237], **PROHIBIT(11)** [17], [28], [116], [153], [155], [165], [193], [194], [211], [212], [240], **REPAIR(4)** [67], [68], [84], [188]
Weight(5)	**METAHEURISTIC(5)** [137], [147], [215]–[217]	**NOT PRESENTED(4)** [137], [215]–[217], **PROHIBIT(1)** [147]
Safety(2)	**GENERAL(1)** [180], **METAHEURISTIC(3)** [178], [184], [223]	**NOT PRESENTED(3)** [178], [184], [223], **PROHIBIT(1)** [180]

polynomial-time hard (NP-hard) problem. Evolutionary Algorithms (EAs) [18], [33], [37], [38], [41], [57]–[60], [71], [73], [76], [88], [92], [102], [108], [118], [128], [148], [149], [157]–[159], [161], [175], [196], [216], [221], [223], [227]–[229], [231], [237] are some of the most commonly used approximate methods in architecture optimization. EAs are seen as robust algorithms that exhibit approximately similar performance over a wide range of problems [98], hence their popularity in the software engineering domain.

A considerable number of papers (20% of overall papers) use exact methods, most of which are standard optimization techniques such as Linear Programming [174], [193], [208], [211], [212], while some propose problem-specific exact methods, based on knowledge or assumptions on the problem [8], [29], [55], [116], [171], [245]. Due to the ever-increasing complexity of software systems and the growing number of design options, exact approaches usually are not suitable as optimization techniques, hence the lower number of papers that employ these techniques.

Finally, general methods do not prescribe the optimization strategy but let the user select among several options. Percentages of each main optimization class are shown in Table 3, while Table 7 also shows the percentages for subcategories in relation to quality attributes.

Constraint Handling: Constraint handling techniques generally are problem specific and need a separate effort for their design. This may be one of the reason why a large percentage of papers (32%) do not introduce a constraint handling method. In fact, many papers that mentioned constraints do not describe the constraint handling technique used.

Among the used constraint handling approaches, constraint prohibition is the most studied one (44% in total). Penalty function is another widely used method with 19% of the papers, whereas repair mechanisms are less preferred, used by only 5% of the papers.

4.3 Validation

A summary of the validation-specific aspects that are extracted from the set of papers included in the literature review is given in Table 6. In the following, we summarize the main results for each validation subcategory.

An analysis of the survey results for the validation category presented in Table 6 reveals that most of the papers contain at least one form of validation for the overall approach. Only 10% provide no indication about the quality of the produced architecture specifications. However, a significant number of approaches use a simple form of validation such as simple examples (27%) or academic case studies (16%). Only a few approaches are compared with known results from benchmark problems (4%) or use industrial case studies (16%). However, none

of the investigated approaches that use industrial case studies provide detailed evidence that the quality of the implemented systems has been improved by optimizing the architecture specification.

Analyzing the results regarding the validation of the optimization strategy reveals that only a minority of the approaches (32%) provide detailed results on the appropriateness of the optimization algorithm. A closer look into the optimization validation reveals that especially approaches that employ or present an enhanced heuristic optimization algorithm use at least one base line heuristic algorithm (e.g. an evolutionary algorithm) for comparison.

TABLE 6: Validation category - quantitative summary of the results.

Validation of the overall approach	**Number of Papers**	**Percentages of occurrences**
Experiments	57	30%
Simple example	51	27%
Academic case study	31	16%
Industrial case study	30	16%
Not presented	19	10%
Benchmark problems	8	4%
Literature comparison	2	1%
Mathematical proof	1	<1%
Validation of the optimization strategy	**Number of Papers**	**Percentages of occurrences**
Not presented	127	68%
Comparison with baseline heuristic algorithm	35	19%
Internal comparisson	17	9%
Comparison with exact algorithm	7	4%
Comparison with random search	2	1%
Mathematical proof	1	<1%
Comparison with baseline algorithm	1	<1%

4.4 Cross Analysis

In this section we are extending the analysis of the survey data across the different taxonomy categories. Based on the observations from the reviewing process and the taxonomy construction, the following cross analysis questions (CAQs) are worth a deeper analysis:

- **CAQ1** What optimization strategies have been used with different quality attributes?
- **CAQ2** Is there a relationship between the quality attributes and the quality evaluation method?
- **CAQ3** How do quality attributes relate to degrees of freedom?
- **CAQ4** What is the relationship between the quality attributes and the domain?
- **CAQ5** Is there a preference of specific degrees of freedom in the different domains?
- **CAQ6** Are different validation approaches used in the different domains?
- **CAQ7** Is there a relationship between the domain and the optimization phase?
- **CAQ8** Is there a relationship between the dimensionality and the optimization phase?
- **CAQ9** Are different quality evaluation methods used in run-time and design-time approaches?
- **CAQ10** Is there a relationship between the constraints used in the problem formulation and the constraint handling strategies used in the optimization procedure?
- **CAQ11** What is the relationship between the optimization strategy used and the optimization validation?
- **CAQ12** What is the relationship between the degrees of freedom and the optimization strategy?
- **CAQ13** What types of validation are conducted for the different types of quality evaluation methods?

CAQ1 Optimization strategy and quality attribute: Due to the high complexity of optimization problems that arise in software engineering, metaheuristics are the most common optimization strategies used by the state-of-the-art approaches (Table 7). Most of the papers that use metaheuristics optimize reliability (49 papers, 69% of papers that address reliability), cost (45 papers, 61%), availability (12 papers, 55%), and performance (45 papers, 54%). Problem-specific heuristics are also very common when optimizing quality attributes such as performance (23 papers, 27%), cost (13 papers, 18%), and reliability (11 papers, 15%).

Exact algorithms, which are divided into problem-specific exact algorithms and standard exact algorithms, have also been tackled by the current research. Standard exact algorithms are in general more frequently used than problem-specific exact algorithms. Some of the quality attributes, such as safety, maintainability, and security have not been optimized with exact algorithms.

CAQ2 Quality attribute and quality evaluation method: The quality attributes also exhibit a relation with the evaluation strategies (cf. Table 7). For instance, model-based evaluations are widely used for quality attributes such as safety (75% of papers that address safety) and energy consumption (50%). However, model-based techniques have a lower proportion of the papers that address reliability (32%), performance (32%), and cost (26%). Reliability is usually evaluated with nonlinear mathematical functions (38%), whereas performance and cost are mostly evaluated with simple aggregation functions (52% and 59%, respectively).

CAQ3 Quality attribute and degree of freedom: Cross analysis table 7 depicts certain patterns with respect to the architecture degrees of freedom that are used in the optimization approaches versus the quality attributes. For instance, the reliability optimization approaches are mostly focused on hardware replication (41%), software replication (41%), hardware selection (37%), allocation (28%) and software selection (27%). On the other hand,

TABLE 7: Quality attributes versus other aspects.

Quality Attribute	Total	Optimization Strategy							Quality Evaluation				
		Exact		Approximative			General	Not presented	Simple aggregation functions	Model based	Non-linear mathematical functions	General	Not presented
		Standard	Problem specific	Meta-heuristic	Problem specific heuristic	with guarantee							
Performance	84	14 (17%)	3 (4%)	45 (54%)	23 (27%)	2 (2%)	1 (1%)	3 (4%)	44 (52%)	27 (32%)	11 (13%)	1 (1%)	1 (1%)
Cost	74	13 (18%)	2 (3%)	45 (61%)	13 (18%)	1 (1%)	1 (1%)	3 (4%)	44 (59%)	19 (26%)	10 (14%)	1 (1%)	1 (1%)
Reliability	71	7 (10%)	2 (3%)	49 (69%)	11 (15%)	1 (1%)	2 (3%)	2 (3%)	17 (24%)	23 (32%)	27 (38%)	4 (6%)	1 (1%)
General	25	1 (4%)	2 (8%)	10 (40%)	4 (16%)	1 (4%)	3 (12%)	3 (12%)	10 (40%)	5 (20%)	1 (4%)	2 (8%)	4 (16%)
Availability	22	7 (32%)	-	12 (55%)	7 (32%)	-	-	2 (9%)	19 (86%)	4 (18%)	2 (9%)	-	-
Energy	18	4 (22%)	2 (11%)	6 (33%)	7 (39%)	-	1 (6%)	1 (6%)	6 (33%)	9 (50%)	3 (17%)	-	-
Weight	5	-	-	5 (100%)	-	-	-	-	3 (60%)	-	2 (40%)	1 (20%)	-
Safety	4	-	-	3 (75%)	-	-	1 (25%)	-	1 (25%)	3 (75%)	-	-	1 (25%)
Reputation	4	2 (50%)	-	1 (25%)	2 (50%)	-	-	-	4 (100%)	-	-	-	-
Modifiability	3	-	-	3 (100%)	-	-	-	-	3 (100%)	-	-	-	-
Area	3	1 (33%)	-	2 (67%)	-	1 (33%)	-	-	1 (33%)	1 (33%)	1 (33%)	-	-
Security	1	-	-	1 (100%)	-	-	-	-	1 (100%)	-	-	-	-

Quality Attribute	Total	Transformation Operators											
		Allocation	Hardware replication	Hardware selection	Software replication	Scheduling	Component selection	Service selection	Software selection	Other problem specific	Software parameters	Service composition	Maintenance schedules
Performance	84	37 (44%)	3 (4%)	8 (10%)	3 (4%)	25 (30%)	6 (7%)	20 (24%)	1 (1%)	12 (14%)	2 (2%)	8 (10%)	-
Cost	74	17 (23%)	18 (24%)	18 (24%)	12 (16%)	9 (12%)	19 (26%)	19 (26%)	10 (14%)	11 (15%)	2 (3%)	8 (11%)	2 (3%)
Reliability	71	20 (28%)	29 (41%)	26 (37%)	29 (41%)	6 (8%)	12 (17%)	5 (7%)	19 (27%)	3 (4%)	-	6 (8%)	-
General	25	7 (28%)	1 (4%)	3 (12%)	-	1 (4%)	1 (4%)	6 (24%)	-	2 (8%)	1 (4%)	2 (8%)	-
Availability	22	2 (9%)	4 (18%)	3 (14%)	2 (9%)	-	1 (5%)	14 (64%)	2 (9%)	9 (41%)	-	8 (36%)	1 (5%)
Energy	18	7 (39%)	1 (6%)	2 (11%)	1 (6%)	5 (28%)	1 (6%)	-	-	1 (6%)	6 (33%)	-	-
Weight	5	-	5 (100%)	4 (80%)	4 (80%)	-	1 (20%)	-	4 (80%)	-	-	-	-
Safety	4	1 (25%)	1 (25%)	1 (25%)	1 (25%)	-	2 (50%)	-	-	1 (25%)	1 (25%)	-	1 (25%)
Reputation	4	-	-	-	-	-	-	4 (100%)	-	1 (25%)	-	3 (75%)	-
Modifiability	3	-	-	-	-	-	-	-	-	-	-	-	-
Area	3	2 (67%)	-	1 (33%)	-	2 (67%)	1 (33%)	-	-	-	-	-	-
Security	1	-	-	-	-	-	-	1 (100%)	-	1 (100%)	-	-	-

the most common degrees of freedom for performance, which is the most frequent quality attribute, are allocation (44%), scheduling (30%) and service selection (24%).

As can be observed from the gaps in Table 7, some degrees of freedom are not considered to optimize certain quality attributes. For instance, there are no papers optimizing availability, safety, or security by varying the scheduling. Furthermore, software selection is only used to optimize performance, cost, reliability, availability, and weight.

CAQ4 Quality attribute and domain: Results depicted in Table 8 indicate that there is a relationship between certain quality attributes and the domain. For instance, quality attributes such as energy, weight safety, and area are optimized only in the context of embedded systems, whereas security is considered only with information systems. Modifiability is only presented in a general setting, without specifying the domain. These observations confirm that certain quality attributes are important or can be measured only in a specific domain. For example, safety is an important quality attribute in embedded systems, especially in life-critical embedded systems, whereas information systems typically do not involve life- and safety-critical functionalities. Still, the most common quality attributes performance, costs, and reliability, are used in both domains.

CAQ5 Degree of freedom and domain: The cross analysis between the degrees of freedom and the domain is presented in Table 8. Some degrees of freedom are often considered in embedded systems, e.g. allocation (68%), hardware replication (80%), hardware selection (82%), and scheduling (61%). Some degrees of freedom can only be found in embedded systems, e.g. software replication, clustering, and maintenance schedules. On the other hand service selection and service composition are only present in information systems.

CAQ6 Validation approach and domain: Table 8 presents the results of the cross analysis between the

TABLE 8: Domain versus other aspects.

Quality Attribute	Total	Domain		
		ES	IS	GENERAL
Performance	84	32 (38%)	31 (37%)	21 (25%)
Cost	74	35 (47%)	26 (35%)	13 (18%)
Reliability	71	40 (56%)	13 (18%)	18 (25%)
Availability	25	5 (20%)	17 (68%)	3 (12%)
General	22	12 (55%)	6 (27%)	4 (18%)
Energy	18	17 (94%)	-	1 (6%)
Weight	5	4 (80%)	-	1 (20%)
Safety	4	4 (100%)	-	-
Reputation	4	-	3 (75%)	1 (25%)
Modifiability	3	-	-	3 (100%)
Area	3	3 (100%)	-	-
Security	1	-	1 (100%)	-
Degrees of Freedom	**Total**	**ES**	**IS**	**GENERAL**
Allocation	59	40 (68%)	5 (8%)	14 (24%)
Hardware replication	40	32 (80%)	2 (5%)	6 (15%)
Hardware selection	38	31 (82%)	4 (11%)	3 (8%)
Software replication	35	28 (80%)	-	7 (20%)
Scheduling	33	20 (61%)	1 (3%)	12 (36%)
Component selection	30	12 (40%)	2 (7%)	16 (53%)
Service selection	28	-	26 (93%)	2 (7%)
Software selection	24	21 (88%)	1 (4%)	2 (8%)
Other problem specific	18	5 (28%)	10 (56%)	3 (17%)
Service composition	12	-	12 (100%)	-
Software parameters	10	8 (80%)	1 (10%)	1 (10%)
Clustering	5	5 (100%)	-	-
General	5	4 (80%)	-	1 (20%)
Hardware parameters	4	1 (25%)	3 (75%)	-
Architectural pattern	3	-	-	3 (100%)
Not presented	3	2 (67%)	-	1 (33%)
Partitioning	2	1 (50%)	1 (50%)	-
Maintenance schedules	2	2 (100%)	-	-
Approach Validation	**Total**	**ES**	**IS**	**GENERAL**
Experiments	57	23 (40%)	23 (40%)	11 (19%)
Simple example	51	38 (75%)	-	13 (25%)
Industrial case study	31	21 (68%)	1 (3%)	8 (26%)
Academic case study	30	11 (37%)	14 (47%)	6 (20%)
Not presented	19	5 (26%)	3 (16%)	11 (58%)
Benchmark problems	8	6 (75%)	-	2 (25%)
Literature comparison	2	2 (100%)	-	-
Mathematical proof	1	1 (100%)	-	-

domain and the validation approach. The validation approaches taken in embedded systems vary more than in information systems, with examples, benchmark problems, literature comparison, and mathematical proof being used only in ES. In addition, it can be observed that the proportion of papers that use experiments and academic case studies as validation techniques is higher in information systems compared to embedded systems. In essence, "examples" was the most commonly used validation technique in embedded systems, with 38 papers (75%), whereas "experiments" were usually preferred in information systems.

TABLE 9: Phase versus other aspects.

Domain	Total	Phase		
		DT	RT	GENERAL
ES	100	85 (85%)	15 (15%)	1 (1%)
IS	41	5 (12%)	35 (85%)	1 (2%)
General	49	38 (78%)	10 (20%)	1 (2%)
Dimensionality	**Total**	**DT**	**RT**	**GENERAL**
SOO	75	57 (76%)	17 (23%)	1 (1%)
MOO	58	54 (93%)	3 (5%)	2 (3%)
MTS	51	15 (29%)	36 (71%)	-
General	7	3 (43%)	4 (57%)	-
Quality Evaluation	**Total**	**DT**	**RT**	**GENERAL**
SAF	80	40 (50%)	40 (50%)	-
MB	60	46 (77%)	12 (20%)	3 (5%)
NMF	40	36 (90%)	4 (10%)	-
Not presented	6	2 (33%)	4 (67%)	-
General	6	6 (100%)	-	-

CAQ7 Domain and optimization phase: The cross analysis of the domain and optimization phase is depicted in Table 9. It can be observed that the optimization techniques designed for embedded systems are usually performed at design time (85% of the papers are at design time), whereas in information systems, the optimization is mostly done at run time, with 85% of papers in information systems performing optimization at run time. In the other direction an analogous relation from phase to domain can also be observed. While the popularity of design-time optimization for embedded systems is understandable due to the difficulty of run-time adaptation of embedded systems, the low number of design-time approaches for information systems may be surprising.

CAQ8 Dimensionality and optimization phase: In a multi-objective optimization problem, the output of the optimization process in a set of (near) Pareto optimal solutions. As a result, a subsequent selection process is needed to choose among the near optimal architectures, which is usually not practical at run time of a software system. This explains the low percentage of approaches that use multi-objective optimization at run-time (only 5%), depicted in Table 9, and the high percentage of the approaches at run time that convert a multi-objective problem into a single-objective problem (MTS, 71%).

CAQ9 Quality evaluation and optimization phase: The analysis of the quality evaluation methods used at design and run time is presented in Table 9. Interestingly, there is a high number of papers at run-time that use simple additive functions (SAF). Model-based approaches, on the other hand, are not as often used at run-time (only 20%), when compared to design time. As model-based quality evaluation models are computa-

tionally more expensive than simple additive functions, their applicability may be limited at run time. Similarly, there is a higher percentage of papers that employ non-linear mathematical function at design-time (90%).

CAQ10 Constraint and constraint handling technique: Table 10 shows a cross analysis of the different constraints and the constraint handling techniques used during the optimization. The majority of the papers handle constraints with prohibition techniques, such 73% of papers with performance constraints, 53% of papers with cost constraints, and 62% of papers with physical constraints. Penalty functions are the second most commonly used constraint handling technique. A considerable number of papers used penalty techniques with cost constraint (41%), weight (45%), physical (38%) and timing (30%). Nevertheless, the proportion of papers that use penalty function as a constraint handling technique is lower than prohibition techniques for all constraints.

From the constraint handling perspective, prohibition techniques are commonly used with almost all constraints. On the other hand, constraint handling techniques defined in general are not very frequent. Repair techniques are very rarely addressed. One reason for this could be that they increase the complexity of the optimization process since they require extra knowledge about the problem to construct feasible results.

CAQ11 Optimization strategy and validation: Table 11 shows a cross analysis of the optimization strategy and the optimization validation. Note that not all optimization validation types are applicable to or meaningful for all optimization strategies. Not applicable or not meaningful combinations are marked N/A in the table. For example, there is no need to validate exact standard algorithms as their ability to find optimal solutions is already well studied in optimization literature.

For exact problem-specific approaches, only half of the papers present some form of validation, mostly a comparison with a baseline heuristic algorithm that is commonly used for the addressed optimization problem.

Looking at approximate techniques, we observe two main favorite optimization strategies: Evolutionary Algorithms as the most commonly used metaheuristic and constructive heuristics as the most common problem-specific heuristic. Interestingly, Evolutionary Algorithms are less frequently validated than many other metaheuristics, although it is known that an evolutionary algorithm's performance and quality of results can significantly vary for different optimization parameters and problem formulations [19], [20], [225].

CAQ12 Degree of freedom and optimization strategy: In optimization, the time and computational complexities are the aspects that are of interest. If a problem is solvable in polynomial time, i.e. it is not an *NP-optimization problem* as defined by Crescenzi et. al [65], then an exact algorithm might be the best solutions. However, the majority of the problems in architecture optimization cannot be solved in polynomial time. The degrees of freedom used with a specific problem is one of the components that defines the computational complexity of an optimization problem, among others such as the complexity of the quality evaluation function/procedure. As it can be observed from the results in Table 12, the majority of the degrees of freedom in architecture optimization, especially degrees of freedom that involve hardware, such as hardware replication (80%), hardware selection (82%), and hardware parameters (75%), are used in conjunction with approximate optimization algorithms. Similarly, many degrees of freedom that involve a change in the software part of the system are also used with metaheuristics, e.g. software replication (74%) and software selection (79%). On the other hand, clustering is used mostly with problem-specific heuristics (60%). Standard exact algorithms are not very frequently used in conjunction with most degrees of freedom, apart from service selection (36% of the papers).

CAQ13 Quality evaluation method and approach validation: Depending on the quality evaluation method that the reviewed approaches use, certain approach val-

TABLE 10: Constraints versus constraint handling techniques.

Constraint	Total	Constraint Handling				
		Prohibit	Penalty	Repair	General	Not presented
Cost	32	17 (53%)	13 (41%)	-	-	2 (6%)
Performance	26	19 (73%)	3 (12%)	4 (15%)	-	-
General	25	11 (44%)	10 (40%)	2 (8%)	-	3 (12%)
Weight	20	11 (55%)	9 (45%)	-	-	-
Physical	13	8 (62%)	5 (38%)	-	-	-
Timing	10	7 (70%)	3 (30%)	-	-	-
QoS values	10	7 (70%)	-	1 (10%)	-	2 (20%)
Precedence	9	7 (78%)	1 (11%)	1 (11%)	-	-
Memory	9	7 (78%)	1 (11%)	-	1 (11%)	-
Mapping	8	7 (88%)	-	-	1 (13%)	-
Reliability	7	4 (57%)	1 (14%)	-	-	2 (29%)
Requirements	6	4 (67%)	-	-	-	1 (17%)
Volume	6	3 (50%)	3 (50%)	-	-	-
Structural	5	2 (40%)	-	2 (40%)	-	1 (20%)
Area	3	3 (100%)	1 (33%)	-	-	-
Redundancy level	3	2 (67%)	-	1 (33%)	-	-
Delivery time	3	3 (100%)	-	-	-	-
Availability	2	1 (50%)	1 (50%)	-	-	-
Throughput	2	2 (100%)	-	-	-	-
Processing power	1	-	1 (100%)	-	-	-
Stability	1	1 (100%)	-	-	-	-
Path loss	1	-	1 (100%)	-	-	-
Functional correctness	1	1 (100%)	-	-	-	-
Design	1	1 (100%)	-	-	-	-
Dependability	1	1 (100%)	-	-	-	-

TABLE 11: Optimization strategy versus optimization validation.

Optimization Approach			Total	Comparison with baseline heuristic algorithm	Internal comparison	Comparison with exact algorithm	Comparison with random search	Mathematical proof	Not presented
Exact	Standard	Linear Programming	9 (4%)	-	-	-	N/A	N/A	9 (100%)
		Mixed-Integer Linear Programming (Milp)	5 (2%)	-	-	-	N/A	N/A	5 (100%)
		Integer Programming Algorithm	7 (3%)	-	-	-	N/A	N/A	7 (100%)
		Integer Linear Programming	4 (2%)	-	-	-	N/A	N/A	4 (100%)
		Exhaustive Search	1 (0%)	-	-	-	N/A	N/A	1 (100%)
		Sequential Quadratic Programming	2 (1%)	1 (50%)	-	-	N/A	N/A	1 (50%)
		Total Exact Standard	27 (13%)	1 (4%)	-	-	N/A	N/A	26 (96%)
	Problem Specific	Graph Partitioning	1 (0%)	1 (100%)	-	-	N/A	-	-
		Branch And Bound	3 (1%)	1 (33%)	2 (67%)	-	N/A	-	-
		Other Exact Problem Specific	6 (3%)	2 (33%)	1 (17%)	1 (17%)	N/A	-	2 (33%)
		Total Exact Problem Specific	10 (5%)	4 (40%)	3 (30%)	1 (10%)	N/A	-	2 (20%)
	Total Exact		38 (18%)	5 (13%)	3 (8%)	1 (3%)	N/A	-	29 (76%)
Approximative	Metaheuristic	Evolutionary Algorithm	71 (34%)	10 (14%)	6 (8%)	3 (4%)	1 (1%)	N/A	51 (72%)
		Greedy	4 (2%)	1 (25%)	2 (50%)	-	-	N/A	1 (25%)
		Simulated Annealing	13 (6%)	5 (38%)	3 (23%)	1 (8%)	-	N/A	4 (31%)
		Variable Neighbourhood Search	5 (2%)	2 (40%)	-	-	-	N/A	3 (60%)
		Ant Colony Optimization	4 (2%)	2 (50%)	-	-	1 (25%)	N/A	1 (25%)
		Hill Climbing	4 (2%)	2 (50%)	-	1 (25%)	-	N/A	1 (25%)
		Tabu Search	9 (4%)	6 (67%)	1 (11%)	-	-	N/A	2 (22%)
		Particle Swarm	1 (0%)	1 (100%)	-	-	-	N/A	-
		Other Metaheuristic	5 (2%)	2 (40%)	-	-	-	N/A	3 (60%)
		Total Metaheuristic	103 (49%)	23 (22%)	10 (10%)	5 (5%)	2 (2%)	N/A	63 (61%)
	Problem Specific	Constructive Heuristics	13 (6%)	3 (23%)	2 (15%)	1 (8%)	-	-	7 (54%)
		Other Problem Specific	15 (7%)	2 (13%)	1 (7%)	-	-	-	12 (80%)
		Greedy	7 (3%)	4 (57%)	1 (14%)	-	-	-	2 (29%)
		Branch And Bound Based	1 (0%)	-	-	-	-	1 (100%)	-
		Graph Partitioning	2 (1%)	1 (50%)	-	-	-	1 (50%)	-
		Dynamic Programming	2 (1%)	1 (50%)	1 (50%)	-	-	-	-
		Restricted Enumeration Of All Possible Solutions	2 (1%)	-	-	-	-	-	2 (100%)
		Total Approximative Problem Specific	42 (20%)	10 (24%)	6 (14%)	1 (2%)	-	1 (2%)	24 (57%)
	With Guarantee		4 (2%)	1 (25%)	1 (25%)	-	-	-	2 (50%)
	Total Approximative		0 (0%)	-	-	-	-	-	-

idations have been selected as shown in Table 13. For instance, when using a simple aggregation function, experiments are the most frequent validation technique, comprising 40% of the overall papers that use this kind of quality evaluation method. Model-based approaches instead have been most frequently validated with industrial case studies (32%). It can also be observed that these approaches have the highest proportion of papers that use industrial case studies as an approach validation technique among all other quality evaluation methods, which may indicate a possible relation among these two entities.

Another interesting result relates to the validation technique used for non-linear mathematical functions. The majority of the approaches that use non-linear mathematical functions use validation by examples (53%). A considerable fraction of papers in this category uses experiments (20%), and only a few papers use industrial case studies (8%).

In general, very few papers use benchmarks problems; more specifically, only 4% of the papers that use simple aggregation functions, 3% of all model-based approaches and 8% of papers that consider non-linear mathematical functions. This can be due to a lack of benchmark problems in the software engineering domain, which may be a research area that requires more attention. Mathematical proofs and literature comparison are even less frequently used as validation approaches. The only papers we found with mathematical proof as a validation technique use either simple aggregation functions, or non-linear mathematical functions as quality evaluation methods.

TABLE 12: Degree of freedom versus optimization strategy.

Degrees of Freedom	Total	Optimization Strategy						
		Approximative			Exact		General	Not presented
		Metaheuristic	Problem-specific heuristic	With guarantee	Exact standard	Exact problem-specific		
Allocation	59	32 (54%)	19 (32%)	1 (2%)	1 (2%)	5 (8%)	1 (2%)	4 (7%)
Hardware replication	40	32 (80%)	2 (5%)	1 (3%)	4 (10%)	-	1 (3%)	-
Hardware selection	38	31 (82%)	3 (8%)	-	3 (8%)	1 (3%)	1 (3%)	-
Software replication	35	26 (74%)	2 (6%)	1 (3%)	5 (14%)	-	1 (3%)	-
Scheduling	33	14 (42%)	14 (42%)	-	4 (12%)	1 (3%)	-	1 (3%)
Component selection	30	18 (60%)	3 (10%)	-	4 (13%)	2 (7%)	-	4 (13%)
Service selection	28	11 (39%)	7 (25%)	-	10 (36%)	-	-	1 (4%)
Software selection	24	19 (79%)	1 (4%)	-	3 (13%)	-	1 (4%)	-
Other problem specific	18	10 (56%)	4 (22%)	-	4 (22%)	-	-	1 (6%)
Service composition	12	7 (58%)	4 (33%)	-	2 (17%)	-	-	1 (8%)
Software parameters	10	1 (10%)	5 (50%)	-	4 (40%)	1 (10%)	1 (10%)	-
Clustering	5	1 (20%)	3 (60%)	-	1 (20%)	-	-	-
General	5	2 (40%)	-	1 (20%)	-	1 (20%)	2 (40%)	-
Hardware parameters	4	3 (75%)	1 (25%)	-	-	-	-	-
Architectural pattern	3	3 (100%)	-	-	-	-	-	-
Not presented	3	2 (67%)	-	-	-	-	-	1 (33%)
Partitioning	2	1 (50%)	-	1 (50%)	1 (50%)	-	-	-
Maintenance schedules	2	1 (50%)	-	-	-	-	1 (50%)	-

TABLE 13: Quality evaluation versus approach validation.

Quality Evaluation	Total	Approach Validation							
		Experiments	Example	Industrial case study	Academic case study	Benchmark problems	Mathematical proof	Literature comparison	Not presented
Simple aggregation functions	80	32 (40%)	14 (18%)	8 (10%)	17 (21%)	3 (4%)	1 (1%)	-	10 (13%)
Model Based	60	15 (25%)	10 (17%)	19 (32%)	12 (20%)	2 (3%)	-	2 (3%)	4 (7%)
Non-linear mathematical functions	40	8 (20%)	21 (53%)	3 (8%)	2 (5%)	3 (8%)	-	-	3 (8%)
General	6	-	5 (83%)	-	1 (17%)	1 (17%)	-	-	-
Not presented	6	2 (33%)	2 (33%)	-	-	-	-	-	2 (33%)

5 Recommendations for Future Research

Based on the results of the literature review presented in the previous section, it is evident that the research area of architecture optimization has received a lot of attention over the last decades and significant progress has been made. However, the results also reveal a number of observations that can help to direct future research efforts in the community. In the following, to address the third research question (**RQ3**), we list important goals that should be achieved by the community in order to advance the research area.

Evidence on the quality of the resulting architectures and economic benefit. To further increase the penetration of architecture optimization approaches in industrial practice, it would be required to provide detailed success stories that indicate an economical benefit of applying the specific architecture optimization approaches. Inline with the idea of evidence-based software engineering [78], this requires a systematic analysis of systems that have been developed with and without the use of architecture optimization approaches, with respect to the achieved system quality and the spent effort.

Systematic exploration of effective degrees of freedom for different quality attributes. The more recent approaches reviewed in this paper focus on exploiting specific architecture degrees of freedom to achieve a certain quality goal. Further research effort is required for analyzing the individual approaches in order to understand the relationship between the degrees of freedom and quality attributes. These studies should identify the effect of each degree of freedom on different quality attributes. Furthermore, an investigation of a joint consideration of different degrees of freedom is an interesting starting point for future studies. Note that joint consideration of any set of degrees of freedom and any quality attributes requires the use of an architecture model as an input (cf. taxonomy category "architecture representation" in Section 3.2), because quality evaluation models are restricted to certain quality attributes.

Systematic validation of the optimization strategy. Based on the results presented in Table 6, it is evident that a majority of approaches do not validate the optimization strategy. A common theme is that a certain

optimization algorithm is picked and applied without comparing it to the portfolio of existing optimization approaches. This is valid for some papers that aim to introduce a new quality evaluation model; however, to further advance our knowledge on the performance and effectiveness of the optimization algorithms, we recommend to compare the optimization algorithms with the current state of the art approaches. This will allow for better algorithm selection in future. For the comparison, the community should identify a set of benchmark architecture optimization problems, similar to the ones already established in the field of reliability optimization for redundancy allocation [140]. For metaheuristics, a comparison with random search and well established metaheuristics is recommended. Since most algorithms are probabilistic, experiments with a sufficient number of runs should be used and analyzed with statistical tests. For setting up the experiments and analyzing them, the recently published guidelines by Arcuri and Briand [11] can be recommended.

Unified tool support. Tools that can be used to model software architecture optimization problems and that offer different optimization strategies could greatly support the above-mentioned research directions. A general optimization framework for software architectures could be devised, which could make use of (1) plug-ins that interpret different architecture models (from architecture description languages to component models) and provide degree of freedom definitions and (2) plug-ins to evaluate quality attributes for a given architecture model. Such frameworks have already been started with the Archeopteryx [5], PerOpteryx [133], [161], and AQOSA [145] approaches for metaheuristic optimization and a fixed software architecture model. Future research could extend them to be more generically applicable, and thus foster better collaboration among researchers, e.g. by the definition of benchmark problems.

Systematic guidelines for selecting the optimization approach based on the given problem. In the area of software architecture optimization, systematic guidelines for optimization-approach selection are currently lacking. There is a wide range of optimization algorithms available, which can be grouped into two main classes: exact and approximate algorithms. Depending on the available resources and time, on whether the goal is to find the optimal or near-optimal solutions, and on the size and complexity of the problem, the appropriate algorithm needs to be selected for the given problem.

Assuming problems of non-trivial size, the complexity of the problem is the most important factor that needs to be taken into account. For optimization, the time and computational complexities are the aspects that one is interested in. If a problem is solvable in polynomial time, i.e. it is not an *NP-optimization problem* as defined by Crescenzi et. al [65], then an exact algorithm might be the best solutions. However, the majority of the problems in architecture optimization cannot be solved in polynomial time. The degrees of freedom considered with a specific problem is one of the components that defines the computational complexity of an optimization problem. As can be observed from the results in Table 12, the majority of the degrees of freedom in architecture optimization are used in conjunction with approximate optimization algorithms.

All these aspects need more investigation. The taxonomy proposed in this paper is an initial step in this direction, since it provides a categorization of software architecture optimization problems. The investigation of the above aspects can lead to systematic guidelines for selecting the optimization approach based on the given problem.

Support for practitioners. To apply a software architecture optimization approach to a given system architecture, practitioners need to (1) model the software architecture in the formalism used by the approach and (2) identify the applicable degrees of freedom. Here, modeling the existing architecture is often the most difficult step, as it includes collecting information about the quality properties of the architecture. For example, the resource demands and other performance properties need to be determined for performance, e.g. by measurements [167]. For reliability, the values usually are estimated or based on historical data [48], [89]. Creating an accurate model requires a considerable effort, and seems to hinder the acceptance of architecture modeling in practice. Thus, future research should provide support for practitioners and partial automation to create such models.

Furthermore, most reviewed approaches use a specific formalism to describe the software architecture (cf. "architecture representation" category in sections 3.2 and 4.2). Thus, even if a practitioner has a formalized model for a software architecture available, the optimization approaches are not readily applicable. Here, software architecture optimization researchers should relate their required input models of the software architecture to UML or other widespread modeling languages, e.g. by providing tools that transform an UML model to the required formalism.

Reporting guidelines for software architecture optimization. The description of the solved optimization problem and the used optimization approach varies greatly among different papers in the surveyed domain. Not all values of our taxonomy were explicitly presented and could be quickly identified. Some values were only implicitly indicated, making it hard to extract them from the description of the work. Thus, comparing and relating different works is difficult.

Our taxonomy can serve as a reporting guideline for future work to improve the reporting standards in the area of software architecture optimization. Optimization papers should state explicitly how they relate to the taxonomy by prominently providing information for all taxonomy categories. Ideally, the same terms for the val-

ues of the taxonomy (e.g. different degrees of freedom) could be used, although we have to weigh common software architecture optimization terms (e.g. "allocation") against common terms in different sub-communities (e.g. "binding" in chip design for embedded systems).

6 Conclusions

In this article, we have presented the results of a systematic literature review on architecture optimization which included 188 different approaches. Based on this review, we derived a taxonomy that aims to help researchers to classify existing and future approaches in this research area. Using this taxonomy, we have analyzed the current approaches and presented the results in a way that helps researchers to relate their work to the existing body of knowledge and identify future research directions.

During the review process, we acquired knowledge of different research sub-areas, and presented the implications of their cross analysis via recommendations for future research. We structured the results to a number of tables, which are aimed to facilitate knowledge transfer among various research communities working in the architecture-optimization research area. We learned that although there are some communities that are already well connected (through cross-citation of their works), e.g. the community of reliability and performance architecture optimization (due to the similarities in their models), there still remain a number of communities that are isolated from others, e.g. the scheduling community or the community focusing primarily on the optimization strategies (irrespective of the optimized qualities). The information presented in this survey aims to bridge the gap among the communities and allow for easier knowledge transfer.

In summary, we believe that the results of our systematic review will help to advance the architecture-optimization research area, and since we expect this research area to grow in the future, we hope that also the taxonomy itself will become useful in developing and judging new approaches.

Acknowledgment

We are grateful to Kai Breiner, Franz Brosch, Heiko Koziolek and Adrien Mouaffo for their valuable feedback on this survey.

References

[1] T. F. Abdelzaher and K. G. Shin, "Optimal combined task and message scheduling in distributed real-time systems," in *IEEE Real-Time Systems Symposium*, 1995, pp. 162–171.

[2] A. Abraham, H. Liu, and M. Zhao, "Particle swarm scheduling for work-flow applications in distributed computing environments," in *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, ser. Studies in Computational Intelligence. Springer, 2008, vol. 128, pp. 327–342.

[3] M. Agarwal, S. Aggarwal, and V. K. Sharma, "Optimal redundancy allocation in complex systems," *Journal of Quality in Maintenance Engineering*, vol. 16, pp. 413–424, 2010.

[4] J. T. Alander, "An indexed bibliography of genetic algorithms in testing," Univ. of Vaasa, Finland, Tech. Rep. 94-1-TEST, 2008.

[5] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, "Archeopterix: An extendable tool for architecture optimization of AADL models," in *ICSE 2009 Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2009*. IEEE Computer Society, 2009, pp. 61–71.

[6] A. Aleti, B. Bühnová, L. Grunske, A. Koziolek, and I. Meedeniya. Optimization survey wiki page. [Online]. Available: https://sdqweb.ipd.kit.edu/wiki/OptimizationSurvey

[7] S. Amari and G. Dill, "Redundancy optimization problem with warm-standby redundancy," in *Reliability and Maintainability Symposium (RAMS), 2010 Proceedings*, 2010, pp. 1 –6.

[8] J. D. Andrews and L. M. Bartlett, "A branching search approach to safety system design optimisation," *Reliability Engineering & System Safety*, vol. 87, no. 1, pp. 23–30, 2005.

[9] Y. P. Aneja, R. Chandrasekaran, and K. P. K. Nair, "Minimal-cost system reliability with discrete-choice sets for components," *IEEE Transactions on Reliability*, vol. 53, no. 1, pp. 71–76, 2004.

[10] B. R. Arafeh, K. Day, and A. Touzene, "A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems," *Journal of Systems Architecture - Embedded Systems Design*, vol. 54, no. 5, pp. 530–548, 2008.

[11] A. Arcuri and L. C. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011*, R. N. Taylor, H. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 1–10.

[12] D. Ardagna, G. Giunta, N. Ingraffia, R. Mirandola, and B. Pernici, "QoS-driven web services selection in autonomic grid environments," in *On the Move to Meaningful Internet Systems 2006*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 4276. Springer, 2006, pp. 1273–1289.

[13] D. Ardagna and R. Mirandola, "Per-flow optimal service selection for Web services based processes," *The Journal of Systems and Software*, vol. 83, no. 8, pp. 1512–1523, Aug. 2010.

[14] D. Ardagna and B. Pernici, "Global and local QoS constraints guarantee in web service selection," in *Proc. of the IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 805–806.

[15] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, 2007.

[16] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Sec. Comput*, vol. 1, no. 1, pp. 11–33, 2004.

[17] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. G. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2001, pp. 95–105.

[18] A. Azaron, C. Perkgoz, H. Katagiri, K. Kato, and M. Sakawa, "Multi-objective reliability optimization for dissimilar-unit cold-standby systems using a genetic algorithm," *Computers & OR*, vol. 36, no. 5, pp. 1562–1571, 2009.

[19] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York: Oxford University Press, 1996.

[20] T. Bäck, A. E. Eiben, and N. A. L. van der Vaart, "An empirical study on gas without parameters," in *Parallel Problem Solving from Nature – PPSN VI (6th PPSN'2000)*, ser. Lecture Notes in Computer Science (LNCS). Springer-Verlag (New York), 2000, vol. 1917, pp. 315–324.

[21] J. Balasubramanian, A. S. Gokhale, A. Dubey, F. Wolf, C. Lu, C. D. Gill, and D. C. Schmidt, "Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems," in *IEEE Real-Time & Embedded Technology and Applications Symposium*. IEEE Comp. Society, 2010, pp. 69–78.

[22] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-Based Performance Prediction in Software Development: A Survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[23] S. Banerjee and N. Dutt, "Efficient search space exploration for hw-sw partitioning," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2004, pp. 122–127.

[24] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. AddisonWesley, 2003.

[25] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[26] M. Benazouz, O. Marchetti, A. Munier-Kordon, and P. Urard, "A New Method for Minimizing Buffer Sizes for Cyclo-Static Dataflow Graphs." in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, 2010, pp. 11–20.

[27] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. VLSI Syst*, vol. 8, no. 3, pp. 299–316, 2000.

[28] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, A. Chandrakasan and S. Kiaei, Eds. ACM, 1998, pp. 173–178.

[29] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for qoS-aware web service composition," in *Proccedings of the IEEE International Conference on Web Services (ICWS 2006)*. IEEE Computer Society, 2006, pp. 72–82.

[30] A. K. Bhunia, L. Sahoo, and D. Roy, "Reliability stochastic optimization for a series system with interval component reliability via genetic algorithm," *Applied Mathematics and Computation*, vol. 216, no. 3, pp. 929–939, 2010.

[31] A. Billionnet, "Redundancy allocation for series-parallel systems using integer linear programming," *IEEE Transactions on Reliability*, vol. 57, no. 3, pp. 507–516, 2008.

[32] T. Blickle, "Theory of evolutionary algorithms and application to system synthesis," Ph.D. dissertation, Swiss Federal Institute of Technology, Zurich, 1996.

[33] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Tech. Rep. TIK Report-Nr. 16, 1996.

[34] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[35] B. Boonea, S. Van Hoeckea, G. Van Seghbroecka, N. Jonckheereb, V. Jonckersb, F. D. Turcka, C. Develdera, and B. Dhoedta, "SALSA: QoS-aware load balancing for autonomous service brokering," *Journal of Systems and Software*, vol. 83, no. 3, pp. 446–456, Mar. 2010.

[36] S. Burmester, H. Giese, E. Münch, O. Oberschelp, F. Klein, and P. Scheideler, "Tool support for the design of self-optimizing mechatronic multi-agent systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, no. 3, pp. 207–222, June 2008.

[37] P. G. Busacca, M. Marseguerra, and E. Zio, "Multiobjective optimization by genetic algorithms: application to safety systems," *Reliability Engineering & System Safety*, vol. 72, no. 1, pp. 59–74, Apr. 2001.

[38] G. Canfora, M. Di Penta, R. Esposito, F. Perfetto, and M. L. Villani, "Service composition (re)binding driven by application-specific qoS," in *Proceedings of the 4th International Conference on Service-Oriented Computing - ICSOC 2006*, ser. Lecture Notes in Computer Science, A. Dan and W. Lamersdorf, Eds., vol. 4294. Springer, 2006, pp. 141–152.

[39] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for "qos"-aware service composition based on genetic algorithms," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 1069–1075.

[40] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for qoS-aware binding and re-binding of composite web services," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

[41] L. Cao, J. Cao, and M. Li, "Genetic algorithm utilized in cost-reduction driven web service selection," in *Proceedings of the International Conference on Computational Intelligence and Security, CIS 2005*, ser. Lecture Notes in Computer Science, Y. Hao, J. Liu, Y. Wang, Y. ming Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, Eds., vol. 3802. Springer, 2005, pp. 679–686.

[42] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-driven runtime adaptation of service oriented architectures," in *Proceedings of the 7th joint meeting ESEC/FSE '09*. ACM, 2009, pp. 131–140.

[43] V. Cardellini, E. Casalicchio, V. Grassi, and R. Mirandola, "A framework for optimal service selection in broker-based architectures with multiple QoS classes," in *Services computing workshops, SCW 2006*. IEEE computer society, 2006, pp. 105–112.

[44] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Flow-based service selection forweb service composition supporting multiple qos classes," in *2007 IEEE Int.l Conference on Web Services (ICWS 2007)*. IEEE Computer Society, 2007, pp. 743–750.

[45] M. Ceriani, F. Ferrandi, P. L. Lanzi, D. Sciuto, and A. Tumeo, "Multiprocessor systems-on-chip synthesis using multi-objective evolutionary computation," in *Genetic and Evolutionary Computation Conference, GECCO 2010,*, M. Pelikan and J. Branke, Eds. ACM, 2010, pp. 1267–1274.

[46] R.-S. Chang, J.-S. Chang, and P.-S. Lin, "An ant algorithm for balanced job scheduling in Grids," *Future Generation Computer Systems*, vol. 25, no. 1, pp. 20–27, Jan. 2009.

[47] Y.-S. Chen, C.-S. Shih, and T.-W. Kuo, "Processing element allocation and dynamic scheduling codesign for multi-function socs," *Real-Time Systems*, vol. 44, no. 1-3, pp. 72–104, 2010.

[48] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. ACM, 2008, pp. 111–120.

[49] P. H. Chou, R. B. Ortega, and G. Borriello, "Interface co-synthesis techniques for embedded systems," in *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, 1995*. IEEE Computer Society, 1995, pp. 280–287.

[50] C. A. C. Coello, "A survey of constraint handling techniques used with evolutionary algorithms," *Cornputer methods in applied mechanics and engineerlng*, vol. 20, no. 191, pp. 1245–1287, 2002.

[51] D. W. Coit, "Cold-standby redundancy optimization for nonrepairable systems," *IIE Transactions*, vol. 33, pp. 471–478, 2001.

[52] D. W. Coit, "Maximization of system reliability with a choice of redundancy strategies," *IIE Transactions*, vol. 35, no. 6, pp. 535–543, 2003.

[53] D. W. Coit and T. Jin, "Multi-Criteria Optimization: Maximization of a System Reliability Estimate & Minimization of the Estimate Variance," in *Proceedings of the 2001 European Safety & Reliability International Conference (ESREL)*, 2001, pp. 1–8.

[54] D. W. Coit, T. Jin, and N. Wattanapongsakorn, "System optimization with component reliability estimation uncertainty: a multi-criteria approach," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 369–380, 2004.

[55] D. W. Coit and A. Konak, "Multiple weighted objectives heuristic for the redundancy allocation problem," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 551–558, 2006.

[56] D. W. Coit and J. Liu, "System reliability optimization with k-out-of-n subsystems," *Int Journal of Reliabilty Quality and Safety Engeneering*, vol. 7, no. 2, pp. 129–142, 2000.

[57] D. W. Coit and A. E. Smith, "Reliability optimization of series-parallel systems using a genetic algorithm," *Reliability, IEEE Transactions on*, vol. 45, no. 2, pp. 254 – 260, 266, June 1996.

[58] D. W. Coit and A. E. Smith, "Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach," *Computers & OR*, vol. 23, no. 6, pp. 515–526, 1996.

[59] D. W. Coit and A. E. Smith, "Redundancy allocation to maximize a lower percentile of the system time-to-failure distribution," *IEEE Trans. on Reliability*, vol. 47, no. 1, pp. 79 –87, Mar. 1998.

[60] D. W. Coit and A. E. Smith, "Genetic algorithm to maximize a lower-bound for system time-to-failure with uncertain component weibull parameters," *Computers & Industrial Engineering*, vol. 41, no. 4, pp. 423 – 440, 2002.

[61] D. Cooray, S. Malek, R. Roshandel, and D. Kilgore, "RESISTing reliability degradation through proactive reconfiguration," in *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2010, pp. 83–92.

[62] V. Cortellessa, I. Crnkovic, F. Marinelli, and P. Potena, "Experimenting the automated selection of COTS components based on cost and system requirements," *J. UCS*, vol. 14, no. 8, pp. 1228–1255, 2008.

[63] V. Cortellessa, F. Marinelli, and P. Potena, "Automated selection of software components based on cost/reliability tradeoff," in *Software Architecture, Third European Workshop, EWSA 2006*, ser. Lecture Notes in Computer Science, V. Gruhn and F. Oquendo, Eds., vol. 4344. Springer, 2006, pp. 66–81.

[64] V. Cortellessa and P. Potena, "How can optimization models support the maintenance of component-based software?" in *Proceedings of the 2009 1st International Symposium on Search Based*

Software Engineering. Washington, DC, USA: IEEE Computer Society, 2009, pp. 97–100.

[65] P. Crescenzi, V. Kann, M. Halldórsson, M. Karpinski, and G. Woeginger, "A compendium of NP optimization problems," 2000.

[66] Y.-S. Dai and G. Levitin, "Optimal resource allocation for maximizing performance and reliability in tree-structured grid services," *IEEE Trans. on Reliability*, vol. 56, no. 3, pp. 444–453, 2007.

[67] B. P. Dave and N. K. Jha, "COHRA: Hardware-software cosynthesis of hierarchical distributed embedded system architectures," in *VLSI Design*. IEEE Comp. Society, 1998, pp. 347–354.

[68] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. VLSI Syst*, vol. 7, no. 1, pp. 92–104, 1999.

[69] P. de Oliveira Castro, S. Louise, and D. Barthou, "Reducing memory requirements of stream programs by graph transformations," in *Proceedings of the 2010 International Conference on High Performance Computing & Simulation, HPCS 2010*, W. W. Smari and J. P. McIntire, Eds. IEEE, 2010, pp. 171–180.

[70] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester, UK: John Wiley & Sons, 2001.

[71] M. Di Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. Di Nitto, "Ws binder: a framework to enable dynamic binding of composite web services," in *Proceedings of the 2006 international workshop on Service-oriented software engineering*. New York, NY, USA: ACM, 2006, pp. 74–80.

[72] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 1998.

[73] A. Dogan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *The Computer Journal*, vol. 48, no. 3, pp. 300–314, 2005.

[74] W.-L. Dong and H. Yu, "Optimizing web service composition based on qos negotiation," in *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006)*. IEEE Computer Society, 2006, p. 46.

[75] L. dos Santos Coelho, "An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications," *Reliability Engineering & System Safety*, vol. 94, no. 4, pp. 830 – 837, 2009.

[76] L. dos Santos Coelho, "Reliability-redundancy optimization by means of a chaotic differential evolution approach," *Chaos, Solitons & Fractals*, vol. 41, no. 2, pp. 594 – 602, 2009.

[77] V. K. Dubey and D. A. Menascé, "Utility-based optimal service selection for business processes in service oriented architectures," in *IEEE International Conference on Web Services, ICWS 2010*, 2010, pp. 542–550.

[78] T. Dybå, B. A. Kitchenham, and M. Jørgensen, "Evidence-based software engineering for practitioners," *IEEE Software*, vol. 22, no. 1, pp. 58–65, 2005.

[79] B. Eames, S. Neema, and R. Saraswat, "Desertfd: a finite-domain constraint based tool fordesign space exploration," *Design Automation for Embedded Systems*, vol. 14, pp. 43–74, 2010.

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

[81] C. Elegbede and K. Adjallah, "Availability allocation to repairable systems with genetic algorithms: a multi-objective formulation," *Reliability Engineering & System Safety*, vol. 82, no. 3, pp. 319 – 330, 2003.

[82] J. ElHaddad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and qoS-aware selection algorithm for automatic web service composition," *IEEE T. Services Computing*, vol. 3, no. 1, pp. 73–85, 2010.

[83] P. Emberson, "Searching for flexible solutions to task allocation problems," Ph.D. dissertation, University of York, UK, 2009.

[84] C. Erbas, S. Cerac-Erbas, and A. D. Pimentel, "Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 358–374, June 2006.

[85] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Des. Test*, vol. 10, pp. 64–75, October 1993.

[86] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13)*. ACM, 2011, pp. 234–244.

[87] K. Etminani and M. Naghibzadeh, "A Min-Min Max-Min selective algorihtm for grid task scheduling," in *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*. IEEE, 2007, pp. 1–7.

[88] I. D. Falco, A. D. Cioppa, U. Scafuri, and E. Tarantino, "Multiobjective differential evolution for mapping in a grid environment," in *High Performance Computing and Communications (3rd HPCC'07)*, ser. Lecture Notes in Computer Science (LNCS). Houston, TX, USA: Springer-Verlag (New York), Sept. 2007, vol. 4782, pp. 322–333.

[89] L. Fiondella and S. S. Gokhale, "Software reliability with architectural uncertainties," in *IEEE International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2008, pp. 1–5.

[90] N. FitzRoy-Dale and I. Kuz, "Towards automatic performance optimisation of componentised systems," in *Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems*. New York, NY, USA: ACM, 2009, pp. 31–36.

[91] H. Giese, S. Burmester, F. Klein, D. Schilling, and M. Tichy, "Multi-agent system design for safety-critical self-optimizing mechatronic systems with UML," in *2nd Workshop on Agent-Oriented Methodologies*, 2003.

[92] L. D. Giovanni and F. Pezzella, "An improved genetic algorithm for the distributed and flexible job-shop scheduling problem," *European Journal of Operational Research*, vol. 200, no. 2, pp. 395 – 408, 2010.

[93] A. Girault, E. Saule, and D. Trystram, "Reliability versus performance for critical applications," *J. Parallel Distrib. Comput*, vol. 69, no. 3, pp. 326–336, 2009.

[94] B. Glaser and A. Strauss, *Grounded Theory: The Discovery of Grounded Theory*. New York: de Gruyter, 1967.

[95] M. Glaß, M. Lukasiewycz, C. Haubelt, and J. Teich, "Lifetime Reliability Optimization for Embedded Systems: A System-Level Approach," in *Proceedings of RASDAT '10*, 2010, pp. 17–22.

[96] S. S. Gokhale, "Cost constrained reliability maximization of software systems," in *Reliability and Maintainability, 2004 Annual Symposium - RAMS*, 2004, pp. 195 – 200.

[97] S. S. Gokhale, "Software application design based on architecture, reliability and cost," in *Symposium on Computers and Communications (ISCC 2006)*. IEEE Computer Society, 2004, pp. 1098–1103.

[98] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[99] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2-3, pp. 179–204, 2001.

[100] D. Greiner, B. Galván, and G. Winter, "Safety Systems Optimum Design by Multicriteria Evolutionary Algorithms," in *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*. Springer. Lecture Notes in Computer Science. Volume 2632, Apr. 2003, pp. 722–736.

[101] L. Grunske, "Formalizing architectural refactorings as graph transformation systems," in *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD05)*. IEEE Computer Society, 2005, pp. 324–329.

[102] L. Grunske, "Identifying "good" architectural design alternatives with multi-objective optimization strategies," in *28th International Conference on Software Engineering (ICSE 2006)*. ACM, 2006, pp. 849–852.

[103] L. Grunske, "Early quality prediction of component-based systems - a generic framework," *Journal of Systems and Software*, vol. 80, no. 5, pp. 678–686, 2007.

[104] 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)*. IEEE Computer Society, 2008, pp. 283–292.

[105] 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 *Architecting Dependable Systems*, ser. Lecture Notes in Computer Science, R. de Lemos, C. Gacek, and A. B. Romanovsky, Eds., vol. 4615. Springer, 2006, pp. 188–209.

[106] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du, "ANGEL: Optimal Configuration for High Available Service Composition," in *IEEE International Conference on Web Services (ICWS 2007)*. IEEE Computer Society, 2007, pp. 280–287.

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

[108] A. B. Hadj-Alouane, J. C. Bean, and K. G. Murty, "A hybrid genetic/optimization algorithm for a task allocation problem," *Journal of Scheduling*, vol. 2, no. 4, 1999.

[109] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskander, "Component selection strategies based on system requirements' dependencies on component attributes," in *Systems Conference, 2008 2nd Annual IEEE*, 2008, pp. 1 –5.

[110] M. Harman, "The current state and future of search based software engineering," in *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007*, L. C. Briand and A. L. Wolf, Eds., 2007, pp. 342–357.

[111] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Department of Computer Science, King's College London, Tech. Rep. TR-09-03, April 2009.

[112] M. Hashemi and S. Ghiasi, "Throughput-driven synthesis of embedded software for pipelined execution on multicore architectures," *ACM Trans. Embedded Comput. Syst*, vol. 8, no. 2, 2009.

[113] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *5th International Semantic Web Conference, ISWC 2006*, ser. Lecture Notes in Computer Science, vol. 4273. Springer, 2006, pp. 130–143.

[114] X. He, Z. Gu, and Y. Zhu, "Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming," *Comput. J*, vol. 53, no. 7, pp. 1071–1091, 2010.

[115] J. Henkel, R. Ernst, U. Holtmann, and T. Benner, "Adaptation of partitioning and high-level synthesis in hardware/software co-synthesis," in *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design, 1994*. IEEE Computer Society, 1994, pp. 96–100.

[116] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1702–1714, 1999.

[117] C.-J. Hou and K. G. Shin, "Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems," in *Proceedings of the Real-Time Systems Symposium - 1992*. IEEE Computer Society Press, 1992, pp. 146–156.

[118] H.-Z. Huang, J. Qu, and M. J. Zou, "Genetic-algorithm-based optimal apportionment of reliability and redundancy under multiple objectives," *IIE Transactions*, vol. 41, no. 4, pp. 287–298, Apr. 2009.

[119] International-Standard-Organization, "ISO/IEC Standard for Software engineering Product quality," *ISO/IEC 9126-1 First edition 2001*, 2001.

[120] 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*, pp. c1 –24, 6 2007.

[121] S. Islam, R. Lindstrom, and N. Suri, "Dependability driven integration of mixed criticality SW components," in *ISORC 2006*. IEEE Computer Society, 2006, pp. 485–495.

[122] S. Islam and N. Suri, "A multi variable optimization approach for the design of integrated dependable real-time embedded systems," in *Embedded and Ubiquitous Computing, International Conference, EUC 2007*, ser. Lecture Notes in Computer Science, vol. 4808. Springer, 2007, pp. 517–530.

[123] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *DATE 2005*. IEEE Computer Society, 2005, pp. 864–869.

[124] N. Jafarpour and M. R. Khayyambashi, "Qos-aware selection of web service composition based on harmony search algorithm," in *Proceedings of the 12th international conference on Advanced communication technology*, ser. ICACT'10. IEEE Press, 2010, pp. 1345–1350.

[125] H. Jiang, C. Chang, D. Zhu, and S. Cheng, "A foundational study on the applicability of genetic algorithm to software engineering problems," in *Proc. 2007 IEEE Congress on Evolutionary Computation (CEC'07)*. CPS/IEEE Computer Society, 2007, pp. 2210–2219.

[126] K. Kaya and B. Uçar, "Exact algorithms for a task assignment problem," *Parallel Processing Letters*, vol. 19, no. 3, pp. 451–465, 2009.

[127] Y.-J. Kim and T. Kim, "A HW/SW partitioner for multi-mode multi-task embedded applications," *VLSI Signal Processing*, vol. 44, no. 3, pp. 269–283, 2006.

[128] A. Kishor, S. P. Yadav, and S. Kumar, "Application of a multi-objective genetic algorithm to solve reliability optimization problem," in *Proceedings of the Inter. Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 458–462.

[129] B. Kitchenham, "Procedures for performing systematic reviews," Keele University, Department of Computer Science, Keele University, UK, Technical Report TR/SE-0401, 2004.

[130] J. M. Ko, C. O. Kim, and I.-H. Kwon, "Quality-of-service oriented web service composition algorithm and planning architecture," *Journal of Systems & Software*, vol. 81, no. 11, pp. 2079–2090, 2008.

[131] N. Kokash and V. D'Andrea, "Evaluating quality of web services: A risk-driven approach," in *Business Information Systems, 10th International Conference, BIS 2007*, ser. Lecture Notes in Computer Science, vol. 4439. Springer, 2007, pp. 180–194.

[132] A. Koziolek, "Automated improvement of software architecture models for performance and other quality attributes," Ph.D. dissertation, Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, Karlsruhe, Germany, July 2011.

[133] A. Koziolek and R. Reussner, "Towards a generic quality optimisation framework for component-based system models," in *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, ser. CBSE '11. New York, NY, USA: ACM, New York, NY, USA, June 2011, pp. 103–108.

[134] H. Koziolek, "Performance Evaluation of Component-based Software Systems: A Survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, August 2010.

[135] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage Publications, Inc, 2004.

[136] S. Kulturel-Konak and A. E. S. D. W. Coit, "Efficiently solving the redundancy allocation problem using tabu search," *IIE Transactions*, vol. 35, no. 6, pp. 515–526, 2003.

[137] S. Kulturel-Konak, D. W. Coit, and F. Baheranwala, "Pruned pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives," *Journal of Heuristics*, vol. 14, no. 4, pp. 335–357, Aug. 2008.

[138] S. Künzli, "Efficient design space exploration for embedded systems," Ph.D. dissertation, Swiss Federal Institute of Technology, Zürich, Switzerland, Apr. 2006.

[139] S. Künzli, L. Thiele, and E. Zitzler, "Modular design space exploration framework for embedded systems," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 183–192, 2005.

[140] W. Kuo and R. Wan, "Recent Advances in Optimal Reliability Allocation," in *Computational Intelligence in Reliability Engineering. Evolutionary Techniques in Reliability Analysis and Optimization*, G. Levitin, Ed. Heidelberg: Springer, 2007, pp. 1–36.

[141] Y. Laalaoui, H. Drias, A. Bouridah, and R. Ahmed, "Ant colony system with stagnation avoidance for the scheduling of real-time tasks," in *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, 2009, pp. 1 –6.

[142] C. Lee, S. Kim, and S. Ha, "A systematic design space exploration of MPSoC based on synchronous data flow specification," *Signal Processing Systems*, vol. 58, no. 2, pp. 193–213, 2010.

[143] 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*, 2010, pp. 117–128.

[144] J. Z. Li, J. W. Chinneck, C. M. Woodside, and M. Litoiu, "Fast scalable optimization to configure service systems having cost and quality of service constraints," in *Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009, June 15-19, 2009, Barcelona, Spain*, S. A. Dobson, J. Strassner, M. Parashar, and O. Shehory, Eds. ACM, 2009, pp. 159–168.

[145] R. Li, R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron, "An evolutionary multiobjective optimization approach to component-based software architecture design," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011*, 2011, pp. 432–439.

[146] Y.-C. Liang and Y.-C. Chen, "Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm," *Reliability Engineering & System Safety*, vol. 92, no. 3, pp. 323 – 331, 2007.

[147] Y.-C. Liang and M.-H. Lo, "Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm," *J. Heuristics*, vol. 16, no. 3, pp. 511–535, 2010.

[148] Y.-C. Liang, M.-H. Lo, and Y.-C. Chen, "Variable neighbourhood search for redundancy allocation problems," *IMA Journal of Management Mathematics*, vol. 18, no. 2, pp. 135–155, April 2007.

[149] P. Limbourg and H.-D. Kochs, "Multi-objective optimization of generalized reliability design problems using feature models - A concept for early design stages," *Reliability Engineering & System Safety*, vol. 93, no. 6, pp. 815–828, June 2008.

[150] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Efficient symbolic multi-objective design space exploration," in *ASP-DAC 2008*. IEEE, 2008, pp. 691–696.

[151] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "QoS-aware service composition in dynamic service oriented environments," in *Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference 2009*, ser. Lecture Notes in Computer Science, vol. 5896. Springer, 2009, pp. 123–142.

[152] S. Malek, "A user-centric approach for improving a distributed software system's deployment architecture," Ph.D. dissertation, Graduate School of the University Of Southern California, 2007.

[153] S. Malek, N. Medvidovic, and M. Mikic-Rakic, "An extensible framework for improving a distributed software system's deployment architecture," *IEEE Trans. Software Eng*, vol. 38, no. 1, pp. 73–100, 2012.

[154] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A style-aware architectural middleware for resource-constrained, distributed systems," *IEEE Trans. Software Eng*, vol. 31, no. 3, pp. 256–272, 2005.

[155] B. S. Manoj, A. Sekhar, and C. S. R. Murthy, "A state-space search approach for optimizing reliability and cost of execution in distributed sensor networks," *Journal of Parallel and Distributed Computing*, vol. 69, no. 1, pp. 12–19, Jan. 2009.

[156] T. Mantere and J. T. Alander, "Evolutionary software engineering, a review," *Applied Soft Computing*, vol. 5, no. 3, pp. 315–331, 2005.

[157] M. Marseguerra, E. Zato, and L. Podofillini, "Genetic Algorithms and Monte Carlo Simulation for the Optimization of System Design and Operation," in *Comp. Intelligence in Reliability Engineering. Evolutionary Techniques in Reliability Analysis & Optimization*, G. Levitin, Ed. Heidelberg: Springer, 2007, pp. 101–150.

[158] M. Marseguerra, E. Zio, and S. Martorell, "Basics of genetic algorithms optimization for rams applications," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 977 – 991, 2006.

[159] M. Marseguerra, E. Zio, and L. Podofillini, "A multiobjective genetic algorithm approach to the optimization of the technical specifications of a nuclear safety system," *Reliability Engineering & System Safety*, vol. 84, no. 1, pp. 87 – 99, 2004.

[160] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola, and R. Reussner, "A hybrid approach for multi-attribute qoS optimisation in component based software systems," in *6th International Conference on the Quality of Software Architectures, QoSA 2010*, ser. LNCS, vol. 6093. Springer, 2010, pp. 84–101.

[161] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering, 2010*. ACM, 2010, pp. 105–116.

[162] A. Marzia, A. F. Francesca, A. D. Ardagna, B. Luciano, B. Carlo, C. Cinzia, C. Marco, C. Marco, D. P. F. Maria, F. Chiara, G. Simone, L. P, M. Andrea, M. Stefano, P. Barbara, R. Claudia, and T. Francesco, "The mais approach to web service design," in *10th International Workshop on Exploring Modelling Methods in Systems Analysis and Design, June 13-17, Porto, Portugal*, 2005, pp. 387–398.

[163] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, p. 105156, 2004.

[164] I. Meedeniya, A. Aleti, and L. Grunske, "Architecture-driven reliability optimization with uncertain model parameters," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2340 – 2355, 2012.

[165] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske, "Architecture-driven reliability and energy optimization for complex embedded systems," in *6th International Conference on the Quality of Software Architectures, QoSA 2010*, ser. Lecture Notes in Computer Science, vol. 6093. Springer, 2010, pp. 52–67.

[166] N. R. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors," in *Proceedings of the 22nd international conference on Software engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 178–187.

[167] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design*. Prentice Hall, 2004.

[168] D. A. Menascé, E. Casalicchio, and V. Dubey, "A heuristic approach to optimal service selection in service oriented architectures," in *Proceedings of the 7th Workshop on Software and Performance, WOSP 2008*, A. Avritzer, E. J. Weyuker, and C. M. Woodside, Eds. ACM, 2008, pp. 13–24.

[169] 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*, 2007, pp. 422–430.

[170] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malek, and J. P. Sousa, "A framework for utility-based service oriented design in SASSY," in *Proceedings of the first joint WOSP/SIPEW Inter. Conference on Performance Engineering*. ACM, 2010, pp. 27–36.

[171] D. A. Menascé, H. Ruan, and H. Gomaa, "Qos management in service-oriented architectures," *Perform. Eval.*, vol. 64, no. 7-8, pp. 646–663, 2007.

[172] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," in *Evolutionary Programming*, 1995, pp. 135–155.

[173] M. Mikic-Rakic, S. Malek, and N. Medvidovic, "Improving availability in large, distributed component-based systems via redeployment," in *Component Deployment, Third International Working Conference, CD 2005*, ser. Lecture Notes in Computer Science, vol. 3798. Springer, 2005, pp. 83–98.

[174] O. Moreira, F. Valente, and M. Bekooij, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proceedings of the 7th ACM & IEEE Inter. Conference on Embedded software, EMSOFT 2007*. ACM, 2007, pp. 57–66.

[175] I. Moser and S. Mostaghim, "The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[176] M. Nicholson, "Selecting a topology for safety-critical real-time control systems," Ph.D. dissertation, Department of Computer Science, University of York, 1998.

[177] M. Nicholson, A. Burns, and Y. Dd, "Emergence of an architectural topology for safety-critical real-time systems," Department of Computer Science,University of York, Tech. Rep., 1997.

[178] M. Nicholson and D. Prasad, "Design synthesis using adaptive search techniques and multi-criteria decision analysis," in *ICECCS 1996*. IEEE Computer Society, 1996, pp. 522 – 529.

[179] H. Oh and S. Ha, "A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling," in *Proceedings of the Seventh International Workshop on Hardware/Software Codesign, CODES 1999*. ACM, 1999, pp. 183–187.

[180] F. Ortmeier and W. Reif, "Safety optimization: A combination of fault tree analysis and optimization techniques," in *International Conference on Dependable Systems and Networks (DSN 2004)*. IEEE Computer Society, 2004, pp. 651–658.

[181] M. Ouzineb, M. Nourelfath, and M. Gendreau, "Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems," *Reliability Engineering & System Safety*, vol. 93, no. 8, pp. 1257 – 1272, 2008.

[182] M. Ouzineb, M. Nourelfath, and M. Gendreau, "An efficient heuristic for reliability design optimization problems," *Computers & OR*, vol. 37, no. 2, pp. 223–235, 2010.

[183] L. Painton and J. Campbell, "Genetic algorithms in optimization of system reliability," *IEEE Transactions on Reliability*, vol. 44, no. 2, pp. 172 –178, 1995.

[184] Y. Papadopoulos and C. Grante, "Evolving car designs using model-based automated safety analysis and optimisation techniques," *The Journal of Systems and Software*, vol. 76, no. 1, pp. 77–89, Apr. 2005.

[185] R. L. Pattison and J. Andrews, "Genetic algorithms in optimal safety design," *Proceedings of the Institution of Mechanical Engineers, Part E : Journal of Process Mechanical Engineering*, vol. 213, no. 3, pp. 187–197, 1999.

[186] J. E. Pezoa, S. Dhakal, and M. M. Hayat, "Maximizing service reliability in distributed computing systems with random node failures: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst*, vol. 21, no. 10, pp. 1531–1544, 2010.

[187] J. E. Pezoa and M. M. Hayat. (2009) Task reallocation for maximal reliability in distributed computing systems with uncertain topologies and non-markovian delays.

[188] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[189] F. Pop, C. Dobre, and V. Cristea, "Genetic algorithm for dag scheduling in grid environments," in *Intelligent Computer Communication and Processing,ICCP 2009*, 2009, pp. 299 –305.

[190] P. Potena, "Composition and tradeoff of non-functional attributes in software systems: research directions," in *Proceedings of the 6th joint meeting ESEC/FSE 07*. ACM, 2007, pp. 583–586.

[191] 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*. IEEE Computer Society, 2007, pp. 55–71.

[192] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *J. Parallel Distrib. Comput*, vol. 65, no. 8, pp. 885–900, 2005.

[193] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time markov decision processes," in *DAC 1999*, 1999, pp. 555–561.

[194] Q. Qiu, Q. Wu, and M. Pedram, "Dynamic power management of complex systems using generalized stochastic petri nets," in *DAC 2000*, 2000, pp. 352–356.

[195] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203 – 249, 2010.

[196] O. Räihä, K. Koskimies, and E. Mäkinen, "Genetic synthesis of software architecture," Department of Computer Science, University of Tampere, Tech. Rep. D-2008-4, 2008.

[197] O. Räihä, K. Koskimies, and E. Mäkinen, "Scenario-based genetic synthesis of software architecture," in *The Fourth International Conference on Software Engineering Advances, ICSEA 2009*. IEEE Computer Society, 2009, pp. 437–445.

[198] O. Räihä, E. Mäkinen, and T. Poranen, "Using simulated annealing for producing software architectures," in *GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference*. ACM, 8-12 July 2009, pp. 2131–2136.

[199] Y. Ren and J. Bechta Dugan, "Design of reliable systems using static and dynamic fault trees," *IEEE Transactions on Reliability*, vol. 47, no. 3, pp. 234 –244, Sept. 1998.

[200] H. Rezaie, N. Nematbakhsh, and F. Mardukhi, "A multi-objective particle swarm optimization for web service composition," in *Networked Digital Technologies - Second International Conference, NDT 2010. Proceedings, Part II*, ser. Communications in Computer and Information Science, vol. 88. Springer, 2010, pp. 112–122.

[201] J. Riauke and L. M. Bartlett, "An offshore safety system optimization using an spea2-based approach," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 222, no. 3, pp. 271–282, 2008.

[202] F. Rosenberg, M. B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic optimization of large-scale QoS-aware service compositions," in *IEEE SCC 2010*. IEEE Computer Society, 2010, pp. 97–104.

[203] SAE, "Architecture Analysis and Design Language (AADL)," *SAE standards*, vol. AS5506, no. 1, 2004.

[204] D. Salazar, C. M. Rocco, and B. J. Galvan, "Optimization of constrained multiple-objective reliability problems using evolutionary algorithms," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 1057–1070, 2006.

[205] T. Saxena and G. Karsai, "MDE-based approach for generalizing design space exploration," in *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010*, ser. LNCS, vol. 6394. Springer, 2010, pp. 46–60.

[206] C. Seo, S. Malek, and N. Medvidovic, "An energy consumption framework for distributed java-based systems," in *22nd IEEE/ACM International Conference on Automated Software Engineering ASE 2007*. ACM, 2007, pp. 421–424.

[207] C. Serban, A. Vescan, and H. F. Pop, "A new component selection algorithm based on metrics and fuzzy clustering analysis," in *Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS 2009*, ser. Lecture Notes in Computer Science, vol. 5572. Springer, 2009, pp. 621–628.

[208] S. Shan and G. G. Wang, "Reliable design space and complete single-loop reliability-based design optimization," *Reliability Engineering & System Safety*, vol. 93, no. 8, pp. 1218 – 1230, 2008.

[209] V. S. Sharma and M. Agarwal, "Ant colony optimization approach to heterogeneous redundancy in multi-state systems with multi-state components," in *Reliability, Maintainability and Safety, ICRMS 2009*, 2009, pp. 116 –121.

[210] V. S. Sharma and P. Jalote, "Deploying software components for performance," in *Component-Based Software Engineering, 11th International Symposium, CBSE 2008*, ser. Lecture Notes in Computer Science, vol. 5282. Springer, 2008, pp. 32–47.

[211] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, 2000,*. IEEE, 2000, pp. 365–368.

[212] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli, "Dynamic power management for portable systems," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM-00)*. ACM Press, 2000, pp. 11–19.

[213] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *DAC 07*. IEEE, 2007, pp. 777–782.

[214] N. Suri, A. Jhumka, M. Hiller, A. Pataricza, S. Islam, and C. Sârbu, "A software integration approach for designing and assessing dependable embedded systems," *The Journal of Systems and Software*, vol. 83, no. 10, pp. 1780–1800, 2010.

[215] H. A. Taboada, F. Baheranwala, D. W. Coit, and N. Wattanapongsakorn, "Practical solutions for multi-objective optimization: An application to system reliability design problems," *Reliability Engineering & System Safety*, vol. 92, no. 3, pp. 314 – 322, 2007.

[216] H. A. Taboada and D. W. Coit, "Data clustering of solutions for multiple objective system reliability optimization problems," *Quality Technology & Quantitative Management*, pp. 35–54, 2007.

[217] H. A. Taboada, J. F. Espiritu, and D. W. Coit, "MOMS-GA: A multi-objective multi-state genetic algorithm for system reliability optimization design problems," *IEEE Transactions on Reliability*, vol. 57, no. 1, pp. 182–191, 2008.

[218] S.-A. Tahaee and A. H. Jahangir, "A polynomial algorithm for partitioning problems," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 4, pp. 34–44, Mar. 2010.

[219] M. Tang and L. Ai, "A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[220] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "Design space exploration of network processor architectures," in *1st Workshop on Network Processors at the 8th International Symposium on High-Performance Computer Architecture (HPCA8)*, Cambridge MA, USA, Feb. 2002.

[221] Z. Tian, G. Levitin, and M. J. Zuo, "A joint reliability-redundancy optimization approach for multi-state series-parallel systems," *Reliability Engineering & System Safety*, vol. 94, no. 10, pp. 1568 – 1576, 2009.

[222] K. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: An NP-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[223] A. C. Torres-Echeverria, S. Martorell, and H. A. Thompson, "Design optimization of a safety-instrumented system based on RAMS plus C addressing IEC 61508 requiriments and diverse redundancy," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 162–179, Feb. 2009.

[224] N. Trcka, M. Hendriks, T. Basten, M. Geilen, and L. J. Somers, "Integrated model-driven design-space exploration for embedded systems," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XI*. IEEE, 2011, pp. 339–346.

[225] J. I. van Hemert and T. Bäck, "Robust parameter settings for variation operators by measuring the resampling ratio: A study

on binary constraint satisfaction problems." *J. Heuristics*, vol. 10, no. 6, pp. 629–640, 2004.
[226] Y. Vanrompay, P. Rigole, and Y. Berbers, "Genetic algorithm-based optimization of service composition and deployment," in *Proceedings of the 3rd international workshop on Services integration in pervasive environments*, ser. SIPE '08. New York, NY, USA: ACM, 2008, pp. 13–18.
[227] A. Vescan, "A metrics-based evolutionary approach for the component selection problem," in *Proceedings of the UKSim 2009: 11th International Conference on Computer Modelling and Simulation*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 83–88.
[228] A. Vescan and C. Grosan, "A hybrid evolutionary multiobjective approach for the component selection problem," in *Hybrid Artificial Intelligence Systems, Third International Workshop, HAIS 2008*, ser. Lecture Notes in Computer Science, vol. 5271. Springer, 2008, pp. 164–171.
[229] A. F. Vescan, "Construction Approaches for Component-Based Systems," PhD, Babesi Bolyai University, 2008.
[230] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *SEAMS 2011*, 2011, pp. 80–89.
[231] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, "Multiobjective optimization of SLA-aware service composition," in *SERVICES 2008*, 2008, pp. 368–375.
[232] S. A. Wadekar and S. S. Gokhale, "Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm," in *Proceedings of the 10th International Symposium on Software Reliability Engineering*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 104–114.
[233] G. Wang, W. Gong, and R. Kastner, "A new approach for task level computational resource bi-partitioning," in *15th International Conference on Parallel and Distributed Computing and Systems, PDCS2003*, 2003.
[234] N. Wattanapongskorn and D. W. Coit, "Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty," *Reliability Engineering & System Safety*, vol. 92, no. 4, pp. 395 – 407, 2007.
[235] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Tabu search with intensification strategy for functional partitioning in hardware-software codesign," in *FCCM 2002*. IEEE Computer Society, 2002, pp. 297–298.
[236] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A stochastic programming approach for QoS-aware service composition," in *CCGRID 08*. IEEE Computer Society, 2008, pp. 226–233.
[237] E. Yang, A. T. Erdogan, T. Arslan, and N. Barton, "Multi-objective evolutionary optimizations of a space-based reconfigurable sensor network under hard constraints," in *BLISS 07*. IEEE Computer Society, 2007, pp. 72–75.
[238] W.-C. Yeh and T.-J. Hsieh, "Solving reliability redundancy allocation problems using an artificial bee colony algorithm," *Comp. & Operations Research*, vol. 38, no. 11, pp. 1465–1473, 2010.
[239] H. Youness, M. Hassan, K. Sakanushi, Y. Takeuchi, M. Imai, A. Salem, A.-M. Wahdan, and M. Moness, "Optimization method for scheduling length and the number of processors on multiprocessor systems," in *Computer Engineering Systems, 2009. ICCES 2009. International Conference on*, 2009, pp. 231 –236.
[240] M. F. Younis, K. Akkaya, and A. Kunjithapatham, "Optimization of task allocation in a cluster-based sensor network," in *ISCC 03*. IEEE Computer Society, 2003, pp. 329–334.
[241] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Software Eng*, vol. 30, no. 5, pp. 311–327, 2004.
[242] C. Zhang, S. Su, and J. Chen, "DiGA: Population diversity handling genetic algorithm for qoS-aware web services selection," *Computer Communications*, vol. 30, no. 5, pp. 1082–1090, 2007.
[243] W. Zhang, Y. Yang, S. Tang, and L. Fang, "QoS-driven service selection optimization model and algorithms for composite web services," in *COMPSAC 07*. IEEE Computer Society, 2007, pp. 425–431.
[244] R. Zhao and B. Liu, "Redundancy optimization problems with uncertainty of combining randomness and fuzziness," *European Journal of Operational Research*, vol. 157, no. 3, pp. 716–735, 2004.
[245] T. Zheng and C. M. Woodside, "Heuristic optimization of scheduling and allocation for distributed systems with soft deadlines," in *Computer Performance Evaluations, Modelling Techniques and Tools. 13th Int. Conference,*, ser. Lecture Notes in Computer Science, vol. 2794. Springer, 2003, pp. 169–181.

Aldeida Aleti is a lecturer at Monash University in Melbourne, Australia. She received her PhD degree in Artificial Intelligence and Software Engineering from Swinburne University of Technology, Melbourne, in 2012. She is currently a member of the Centre for Research in Intelligent Systems and the Optimization lab at Monash University. Her research interests include modelling and optimization of combinatorial systems.

Barbora Buhnova is an assistant professor at the Masaryk University, Czech Republic. She received her PhD degree in computer science from the same university, for application of formal methods in component-based software engineering. She continued with related topics as a postdoc researcher at the University of Karlsruhe, Germany, and the Swinburne University of Technology, Australia. Besides construction of formal models of software systems and their quantitative analysis, she is attracted to modeling of socio-economic systems, for which she recently received a master degree in economics from Mendel University, Czech Republic.

Lars Grunske is Jun. Professor at the University of Kaiserslautern, Germany. He received his PhD degree in computer science from the University of Potsdam, Germany, (Hasso-Plattner-Institute for Software Systems Engineering), in 2004. He was Boeing Postdoctoral Research Fellow at the University of Queensland, Australia, and Lecturer at the Swinburne University of Technology, Australia. He has active research interests in the areas of modeling and verification of systems and software. His main focus is on architecture optimization and model-based dependability evaluation of complex software intensive systems.

Anne Koziolek (Martens) is a postdoc researcher at the University of Zurich, Switzerland. She received her PhD degree from Karlsruhe Institute of Technology (KIT), Germany, in 2011. Before, she studied computer science at the University of Oldenburg, Germany, and the University of West Georgia, USA, and graduated with a diploma degree in 2007. Her research interests include the iterative handling of software architecture and quality requirements, quality requirements prioritization, software quality prediction, model-based automated improvement of software quality, and empirical studies on all kinds of software architecture and requirements engineering topics.

Indika Meedeniya is a research fellow at Swinburne University of Technology, Melbourne Australia, where he received his PhD in 2012. He graduated with a BSc in Electronic and Telecommunication Engineering from the University of Moratuwa, Sri Lanka in 2005, and he worked as a Tech-Lead in developing performance-and reliability-critical software at Millennium IT until 2008. His research interests include modeling and architecture-based evaluation of probabilistic quality attributes, software architecture optimisation and methods to deal with uncertainty in design-time estimates.