A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems

Sara Mahdavi-Hezavehi^{a,*}, Vinicius H.S. Durelli^b, Danny Weyns^{c,d}, Paris Avgeriou^e

^a Nijenborgh 9, 9747 AG Groningen, room 576 Bernoulliborg, Postal code 9700 AK, The Netherlands

^b Avenida Trabalhador San-carlense, 400 - Centro, São Carlos, São Paulo, Postal code 13566-590, Brazil

^c Katholieke Universiteit Leuven, Department of Computer Science, Leuven BE-3001, Belgium

^d Linnaeus University, Sweden

^e Nijenborgh 9, 9747 AG Groningen, room 574 Bernoulliborg, Postal code 9700 AK, The Netherlands

ARTICLE INFO

Article history:

Received 5 January 2016

Received in revised form 17 February 2017

Accepted 31 March 2017

Available online xxx

ABSTRACT

Context

Handling multiple quality attributes (QAs) in the domain of self-adaptive systems is an understudied research area. One well-known approach to engineer adaptive software systems and fulfill QAs of the system is architecture-based self-adaptation. In order to develop models that capture the required knowledge of the QAs of interest, and to investigate how these models can be employed at runtime to handle multiple quality attributes, we need to first examine current architecture-based self-adaptive methods.

Objective

In this paper we review the state-of-the-art of architecture-based methods for handling multiple QAs in self-adaptive systems. We also provide a descriptive analysis of the collected data from the literature.

Method

We conducted a systematic literature review by performing an automatic search on 28 selected venues and books in the domain of self-adaptive systems. As a result, we selected 54 primary studies which we used for data extraction and analysis.

Results

Performance and cost are the most frequently addressed set of QAs. Current self-adaptive systems dealing with multiple QAs mostly belong to the domain of robotics and web-based systems paradigm. The most widely used mechanisms/models to measure and quantify QAs sets are QA data variables. After QA data variables, utility functions and Markov chain models are the most common models which are also used for decision making process and selection of the best solution in presence of many alternatives. The most widely used tools to deal with multiple QAs are PRISM and IBM's autonomic computing toolkit. KLAPER is the only language that has been specifically developed to deal with quality properties analysis.

Conclusions

Our results help researchers to understand the current state of research regarding architecture-based methods for handling multiple QAs in self-adaptive systems, and to identify areas for improvement in the future. To summarize, further research is required to improve existing methods performing tradeoff analysis and pre-emption, and in particular, new methods may be proposed to make use of models to handle multiple QAs and to enhance and facilitate the tradeoffs analysis and decision making mechanism at runtime.

© 2016 Published by Elsevier Ltd.

1. Introduction

Software systems operating in changing environments need human supervision to adapt to changes. This need for adaptation may stem from a variety of anticipated or unforeseen factors including, but not limited to, changes in the environments, changes in quality at-

tributes (QAs) priorities, and new stakeholders' requirements. As the complexity of these software systems increases, so does their (human) supervision overhead. Therefore, one of the main goals during the design and implementation of such systems is to decrease the amount of human involvement by enabling the system to adapt itself while it is executing. Thus, the software system becomes capable of autonomously adapting to changing conditions at runtime. This in turn decreases the cost and operation time required for adapting the system while fulfilling certain QAs at runtime. Although high complexity and heterogeneity of software systems hinder design and development of such software systems, many promising self-adaptation

* Corresponding author.

Email addresses: s.mahdavi.hezavehi@rug.nl (S. Mahdavi-Hezavehi); durelli@icmc.usp.br (V.H.S. Durelli); danny.weyns@kuleuven.be (D. Weyns); paris@cs.ru.nl (P. Avgeriou)

methods¹ have been developed and evaluated over the past fifteen years.

One well-known approach to engineer adaptive software systems is architecture-based self-adaptation. Central to architecture-based self-adaptive methods is the use of a feedback loop equipped with a runtime architectural model to monitor the software system behavior during execution, evaluate the model for requirements compliance, and if necessary, perform adaptations, either at system, module, or parameter level. A classic example is the Rainbow framework [15].

An additional challenge for self-adaptive systems comes when they need to deal with multiple QAs. This implies that the system should be able to correctly identify and locate a condition or event that requires adaptation (i.e., a situation which triggers the need for adaptation), prioritize the adaptation options, choose the optimal adaptation alternative (i.e., the alternative which meets the QAs at a satisfying level), adapt the system accordingly, and presumably handle the positive or negative chain of effects (i.e., implications) caused by the adaptation. When the number of adaptation concerns increases, so does the complexity of the decision making process. Although different techniques (e.g. utility functions, event-condition-action rules) combined with architecture-based self-adaptive methods have been used to handle adaptations in the presence of multiple objectives [9,28], finding systematic ways to recognize and handle time-critical adaptations remains an open challenge.

In order to develop models that capture the required knowledge of the QAs of interest, and to investigate how these models can be employed at runtime to make adaptation decisions, we need to examine current self-adaptive methods. Therefore, in this study we aim to identify and summarize current methods for handling multiple QAs in architecture-based self-adaptive systems. To that end, we conducted a systematic literature review, which is a well-defined method to identify and evaluate studies in a specific domain regarding a particular set of research questions [21]. By performing a systematic literature review, we obtain a fair and unbiased evaluation of selected topic using a rigorous and reliable method. The main contributions of our study include: (a) a list of the most important QAs sets handled (i.e., most frequently discussed) in self-adaptive systems, (b) specifications of existing methods for handling multiple QAs in architecture-based self-adaptation, (c) advantages and limitations of these methods, and (d) areas for future work.

1.1. Background

This section gives a brief introduction of the self-adaptive systems domain, the main requirements and different dimensions in the design and implementation of such systems. Furthermore, we list descriptions of QAs that will potentially be identified during the data extraction phase of our review.

1.1.1. Architecture-based self-adaptive systems

A self-adaptive system is capable of modifying its structure or behavior for various reasons that are difficult or even impossible to anticipate at design time. A change in the system's environment, system faults, changes in the priority of QAs, and the need for handling new requirements are among the well-known reasons for triggering adaptation. To that end, a self-adaptive system continuously monitors its

self, gathers data, analyzes them, decides if adaption is required, and if so applies the adaptation. The challenging aspect of designing and implementing a self-adaptive system is that not only should it apply the changes at runtime, but also should fulfill the system QAs up to a satisfying level, while overcoming uncertainty and its impacts on self-adaptation.

A common approach for handling self-adaptation in the domain of self-adaptive systems is the use of feedback loops based on the MAPE-K model (Fig. 1).

In this model, four components (i.e., Monitor, Analyze, Plan, and Execute) of a feedback loop communicate and collaborate with each other, exchanging data through a shared Knowledge repository to provide the required functionalities, hence MAPE-K. MAPE-K model was first introduced by IBM in their white paper (IBM, 2005), and since then it has been widely used in the design of self-adaptive systems.

1.1.2. QAs in self-adaptive systems

One of the main contributions of this systematic literature review is a list consisting of the most commonly addressed combinations of QAs in the domain of self-adaptive systems. However, to avoid ambiguity, we used a list of QAs with their definitions as a guideline for extracting correct and consistent QAs sets from the primary studies.

Salehi and Tahvildari discuss how a set of well-known QAs from ISO 9126-1 quality model can be linked to the main self-* properties [30]. The authors argue that self-configuring potentially impacts several quality factors, such as maintainability, functionality, portability, and usability. According to that study, self-optimizing has a strong relationship with efficiency, and self-protecting can be associated with reliability of the system. Finally, in self-healing, the main goal is to maximize the availability, survivability, maintainability, and reliability of the system [19].

Weyns and Ahmad concluded that efficiency/performance, reliability, and flexibility are the most addressed QAs in the domain of architecture-based self-adaptive systems, and these QAs are claimed to be improved by current self-adaptation methods. Moreover, the results of that paper indicate that security, accuracy, usability, and maintainability are relevant QAs to the domain of self-adaptive systems as well [32].

Based on the above, we present the following list of QAs; the definitions are based on ISO/IEC 25012 and IEEE9126 ("ISO9126 - Software Quality Characteristics," 2001), ("ISO/IEC 25012:2008 - Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model," 2008):

- **Performance:** efficiency of the software by using the appropriate amounts and types of resources under stated conditions and in a specific context of use.
- **Reliability:** refers to the ability of software to maintain a specified level of performance while running in specified conditions.
- **Flexibility:** capability of the software to provide quality in use in the widest range of contexts of use, including dealing with unanticipated change and uncertainty.
- **Maintainability:** refers to the capability of software to be modified. Modifications may consist corrections, improvements or adaptation of the software to changes in the environment, and in functional and non-functional specifications.
- **Availability:** is the ability of software to be in a state to perform required functions at a given point in time, under specified conditions of use. Therefore, it can be considered as a combination of maturity (i.e., ability to avoid failure), fault tolerance and recoverability (i.e., ability to re-establish a specified level of performance after failure).

¹ In this document "self-adaptation method" refers to any type of solution(s) proposed to implement adaptation actions in a system with multiple QAs in the domain of self-adaptive systems. This includes monitoring, analysis, plan, execution, and knowledge functionalities.

Fig. 1. MAPE-K reference model.

It is worth emphasizing that we did not limit the selection of the primary studies to the papers only addressing the above-mentioned QAs. All the studies addressing any set² of two or more QAs were included in our final set of primary studies and were used in the data analysis phase as they potentially contained relevant data to our domain of interest. Note that this includes studies addressing other types of quality attributes (e.g., business quality attributes such as cost [4] as well.

The main purpose of including these QAs and their definitions was to use this list as a basis for our work, and a means for clarity which helps to avoid ambiguity in data extraction regarding QAs. This was especially helpful to the researchers who performed the filtering of papers and data extraction to make sure the extracted list of QAs refers to the same definitions.

1.2. Problem statement and justification

The objective of this study is to explore current architecture-based methods handling multiple QAs in the domain of self-adaptive systems. As observed by Weyns and Ahmad handling QAs in the domain of self-adaptive systems has not received much attention [32]. The results of that paper indicate that less than half of the existing research in the domain of self-adaptive systems considers multiple QAs in the proposed methods, while little attention is given to the interplay between these QAs. In addition, most of the work addressing multiple QAs only discusses the positive effect of self-adaptation on a particular QA, while possible negative effects of self-adaptation on the rest of the QAs in the system are often ignored.

In order to support QA tradeoffs in architecture-based self-adaptive systems, it is essential to first identify and investigate existing approaches in the domain. Therefore, we conduct this systematic literature review to investigate the characteristics of existing self-adaptive methods dealing with multiple QAs. We use Goal-Question-Metric (GQM) paradigm [3] to define the goal of our systematic review.

Purpose: Study and analyze

Issue: Architecture-based methods handling multiple QAs

Object: In self-adaptive systems

Viewpoint: From the view point of researchers.

In this systematic review, we aim at identifying and assessing existing architecture-based methods handling multiple QAs in self-adaptive systems. Our research questions focus on investigating different aspects of existing methods and helping to identify open problems and areas for improvement. The target audience of this systematic review includes researchers who would like to obtain an overview of the topic, and practitioners who would like to get familiar with existing methods in order to apply them in industry.

In this document “handling” refers to (a) fulfilling multiple QAs through self-adaptation in order to achieve goals of the system (i.e. tradeoff analysis), or (b) fulfilling one or more QAs and investigating implications of self-adaptation on a QA(s) of the system. The results of this review provide an in-depth overview of current methods, help to identify areas for improvement, and serve as a basis for proposing better solutions in the future.

1.3. Related systematic literature reviews

Weyns et al. performed a systematic review on methods for architecture-based self-adaptive systems [32]. Their results indicate that self-adaptation is primarily used to improve performance, reliability, and flexibility QAs; the tradeoffs implied by self-adaptation have not

² In this study, “a set of QAs” refers to a set with two or more QAs elements. In this paper “set of QAs” and “multiple QAs” are used interchangeably.

received much attention. Yang et al. perform a systematic review of requirements modeling and analysis for self-adaptive systems [37]. In their review they investigate modeling methods, requirements engineering activities, QAs, and application domains of existing research. In their results, they present a list of identified modeling methods, requirements engineering activities, and QAs while adaptability is the most frequently addressed QA. Furthermore, they found that online applications and service-based systems are currently the mostly cited application domains. Other surveys on self-adaptive systems such as Weyns et al. [34] and Patikirikoral et al. [26] investigate how formal methods and control engineering can be used to design self-adaptive systems; to the best of our knowledge, there is no systematic review related to architecture-based self-adaptive systems domain with focus on multiple QAs and tradeoff analysis.

2. Research Method: systematic literature review

An important step of conducting a systematic literature review is creating a protocol [21]. By specifying all the steps performed in the systematic review, the protocol defines how the review is conducted. More specifically, the protocol contains the research questions, search strategy to identify and collect relevant primary studies, inclusion and exclusion criteria for filtering out irrelevant papers, and methods for extracting data and synthesizing them to answer research questions.³

2.1. Research questions

The main goal of this systematic literature review as stated in Section 1.2 is refined into the following five research questions.

1. What are the characteristics of the existing architectural methods for handling multiple QAs in self-adaptive software systems?
 - 1.1. Which sets of QAs are addressed by existing methods?
 - 1.2. How many feedback loops are used in current methods and how are they arranged to handle multiple QAs?
 - 1.3. What are the application domains of current methods dealing with multiple QAs?
 - 1.4. What are the development paradigms of current methods dealing with multiple QAs?
 - 1.5. What are the limitations and strengths of existing methods dealing with multiple QAs?
2. How do these methods handle multiple QAs?
 - 2.1. What models are used to represent and analyze multiple QAs characteristics?
 - 2.2. How do these methods address multiple QA tradeoffs?
3. How are the decision making and tradeoff analysis of multiple QAs performed at runtime?
 - 3.1. What runtime tactics/strategies⁴ are used to realize adaptation in systems dealing with multiple QAs?
 - 3.2. Is the decision making unit dealing with multiple QAs centralized or decentralized?
 - 3.3. What tools/languages are used/developed to support QAs tradeoff in existing methods?

- 3.4. Does the method provide any evidence for validating the result of decision making mechanism regarding satisfaction of QAs after adaptation (i.e., assurances)?

We pose research question one to study current methods for handling multiple QAs in architecture-based self-adaptation, and to get an overview of what QAs are addressed by the existing methods. Furthermore, we collect information about their feedback loop mechanisms and the corresponding application domains or development paradigms. Lastly, we study the limitations and strong points of current methods to identify areas for improvement for future work. Note that in this systematic review we are interested in studying architecture-based self-adaptive methods. By architecture-based methods we mean that the system should use an architectural model of itself to monitor and adapt its structure or runtime behavior. In addition, it should be possible to map the MAPE-K functionalities to the components of the adaptation module in the managing system. However, the focus of this research question is not exclusively on investigation of architectural characteristics of existing methods. This research question investigates different aspects (e.g., addressed QAs, application domains) of an existing architecture-based method and only some of them (e.g., feedback loop arrangement) are architectural by nature.

Research question two helps to scrutinize these methods with an emphasis on QAs. We are interested to investigate how QAs are affected by the application of self-adaptation methods. In other words, we are interested in determining whether QAs are intentionally affected by the self-adaption method to achieve system's goals or affected as a side effect of applying the self-adaptation method while addressing other QAs. Furthermore, we are interested in identifying the models that are used to measure and quantify QAs characteristics in order to update them at runtime. In addition, we want to figure out what mechanisms are used by these methods to balance the QAs in self-adaptation process. To be more specific, we are interested to figure out how the methods decide which QAs should be taken into account in order to choose the best-suited self-adaptation configuration in different circumstances. By answering this research question, we also try to investigate to what extent the existing methods address multiple QA tradeoffs, and find open areas for future work.

Finally, we pose research question three to investigate how the decision making and the tradeoff analysis of multiple QAs are performed at runtime. We aim at identifying the main activities taking place by the adaptation module (i.e., managing system) to apply changes on the system. In addition, we investigate if the decision making units of current methods are arranged in centralized or decentralized formats. We are also interested in identifying the tools/languages which are used/developed to deal with multiple QA tradeoffs, and to figure out if there is evidence that the result of decision making mechanism (i.e., adaptation option) guarantees fulfillment of systems' QAs after adaptation (i.e. assurances). The answer to this question may be useful for both researchers and practitioners in the domain of self-adaptive systems.

2.2. Search strategy

The search strategy is used to search for primary studies and is based on:

- a. Preliminary searches to identify existing systematic reviews and assessing potential relevant primary studies,
- b. Trial searches and piloting with different combinations of search terms derived from the research questions,
- c. Reviews of research results, and
- d. Consultation with experts in the field.

³ <http://www.cs.rug.nl/~paris/SLR-MultipleConcerns-TR.pdf>.

⁴ Similar to [9] tactics correspond to a set of factors: the conditions of applicability, a sequence of actions, and a set of intended effects after execution of adaptation on the system. Furthermore, a strategy refers to a flow of actions with intermediate decision points with the purpose of fixing a specific type of system problem.

Similar to determining a “quasi-gold” standard as proposed by Zhang and Babar, we manually searched a small number of venues to crosscheck the results we got from automatic search [38]. This helped to create valid search strings. To perform the manual search, we selected venues based on their significance for publishing research in the context of self-adaptive systems. For the publication time, we limited the manual search to the period of January first of 2000 and 20th of July of 2014. Leaving a few exceptions aside, the development of self-adaptive software hardly goes back beyond a decade ago; after the advent of autonomic computing [5]. Note that even though some major conferences on self-adaptive systems started to emerge after 2005 (e.g., SEAMS), we chose to start the search in the year 2000 to avoid missing any primary study published in other venues. Therefore, we manually searched the following venues:

- International Conference on Software Engineering
- Software Engineering for Adaptive and Self-Managing Systems
- Transactions on Autonomous and Adaptive Systems

In the manual search, we considered title, keywords, and abstract of each paper. After finishing the manual and automatic search for the aforementioned venues, we compared the results to get an estimation of the coverage of the automatic search. The results from the automatic search included all primary studies found for the “quasi-gold” standard (i.e., the “quasi-gold” standard should be a subset of the results returned by the automatic search).

2.2.1. Search method and terms for automatic search

We used automatic search to search through selected venues. By automatic search we mean search performed by executing search strings on search engines of electronic data sources. Although manual search is not feasible for databases where the number of published papers can be over several thousand [1], we still incorporated a manual search (i.e., “quasi-gold” standard) into the search process to make sure that the search string works properly. We included any type of primary study (empirical, theoretical, methodological, etc.) as long as it was relevant to the research domain.

One of the main challenges of performing an automatic search to find relevant primary studies in the domain of self-adaptive systems is a lack of standard, well-defined terminology in this domain. Due to this problem, and to avoid missing any relevant primary study in the automatic search, we preferred to use a more generic search string and include a wider number of papers in the primary results. Later, we filtered out the irrelevant papers to get the final set of primary studies for data extraction purpose. We used the research questions and a step-wise strategy to obtain the search terms; the strategy was as follows:

1. Derive major terms from the research questions and the topics being researched.
2. If applicable, identify and include alternative spellings, related terms and synonyms for major terms.
3. When the database allows, use the “advance” or “expert” search option to insert the complete search string.
 - a. Otherwise, use Boolean “or” to incorporate alternative spellings and synonyms, and use Boolean “and” to link the major terms.
4. Pilot different combinations of search terms in test executions.
5. Check pilot results with “quasi-gold” standard.
6. Discussions between authors to define a stable final search string.

As a result, the following terms are used to formulate the search string:

Self, Dynamic, Autonomic, Manage, Management, Configure, Configuration, Configuring, Adapt, Adaptive, Adaptation, Monitor, Monitoring, Heal, Healing, Architecture, Architectural

The search string consists of three parts, Self AND Adaptation AND Architecture. The alternate terms listed above are used to create the main search string. This is done by connecting these keywords through logical OR as follow:

(self OR dynamic OR autonomic) AND (manage OR management OR configure OR configuration OR configuring OR adapt OR adaptive OR adaptation OR monitor OR monitoring OR analyze OR analysis OR plan OR planning OR heal OR healing OR optimize OR optimizing OR optimization OR protect OR protecting) AND (architecture OR architectural)

Note that many different terms/combinations of terms (e.g., concerns, quality concerns, quality properties, QAs, requirements, non-functional requirements, etc.) are used to refer to QAs in the literature. Moreover, in many cases the specific names (e.g., performance, reliability, etc.) of quality properties are used in papers. Therefore, to avoid missing any relevant paper, we decided to keep the search string generic and omit any terms related to quality in the string.

The reference search string went through modifications to match the search features of electronic sources (e.g., different field codes, case sensitivity, syntax of search strings, and inclusion and exclusion criteria like language and domain of the paper) provided by each of the electronic sources.

2.2.2. Search scope and sources

The scope of the search is defined in two dimensions: publication period (time) and venues. In terms of publication period, we limited the search to papers published from the 1st of January 2000 to the 20th of July 2014 as mentioned in Section 2.2.

For each venue, we documented the number of papers that was returned. Also, we recorded the number of papers left for each venue after primary study selection on the basis of title and abstract. Moreover, the number of primary studies finally selected from each venue was recorded. This information is mainly recorded for documentation purposes, but may also be used later in the data analysis phase. The venues we searched are listed in Table 1.

To get the list of venues for automatic search, we have included the list of venues searched by Weyns et al. in [32]. In that systematic literature review the authors included a list of high quality primary studies in the domain of self-adaptive systems, software architectures, and software engineering. Furthermore, to broaden the search scope, we used Microsoft Academic Search⁵ to find more relevant venues in the domains of self-adaptive systems and software architecture, and included them in the study.

To perform the automatic search of the selected venues, we used the most relevant electronic sources to software engineering research listed in [21]:

1. IEEE Xplorer
2. ACM digital library
3. SpringerLink
4. ScienceDirect

These libraries are the most relevant to the field of study and they cover most of the selected venues. In addition, they provide fairly

⁵ <http://academic.research.microsoft.com/>.

Table 1

List of venues to be searched automatically.

Conference proceedings	International Conference on Software Engineering (ICSE)
and symposiums	IEEE Conference on Computer and Information Technology (IEEECIT) IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO) European Conference on Software Architecture (ECSA) International Conference on Autonomic Computing (ICAC) International Conference on Software Maintenance (CSM) International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE) Working IEEE/IFIP Conference on Software Architecture (WICSA) International Conference of Automated Software Engineering (ASE) International Symposium on Architecting Critical Systems (ISARCS) International Symposium on Software Testing and Analysis (ISSTA) International Symposium on Foundations of Software Engineering (FSE) International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)
Workshops	Workshop on Self-Healing Systems (WOSS) Workshop on Architecting Dependable Systems (WADS) Workshop on Design and Evolution of Autonomic Application Software (DEAS) Models at runtime (MRT)
Journals/Transactions	ACM Transactions on Autonomous and Adaptive Systems (TAAS) IEEE Transactions on Computers (TC) Journal of Systems and Software (JSS) Transactions on Software Engineering and Methodology (TOSEM) Transactions on Software Engineering (TSE) Information & Software Technology (INFOSOF) Software and Systems Modeling (SoSyM)
Book chapters/ Lecture notes	Software Engineering for Self-Adaptive Systems (SefSAS) Software Engineering for Self-Adaptive Systems II (SefSAS) Assurance for Self-Adaptive Systems (ASAS) ACM SIGSOFT Software Engineering Notes (SIGSOFT)

powerful and easy to use search engines, which are suitable for automatic search of databases.

We are aware of the fact that a few of the venues may not be accessible through any digital library. This is an exception, and we plan to manually search these particular venues in the searching phase.

2.2.3. Overview of search process

We adapted the search process of [24] and use a four-phased searching process to collect the primary studies (Fig. 2). In the first phase, we manually search the venues listed in Section 2.2 to establish the “quasi-gold” standard. To perform the manual search we look into papers’ titles, keywords, abstracts, introductions, and conclusions. In addition, we record the total number of papers we looked at (per venue), number of relevant papers returned, and number of papers imported to the reference manager tool⁶ (per venue). This phase results in a set of papers that later on must be a subset of the automatic search results. In the next phase, we perform an automatic search of the selected venues listed in Section 2.2.2. Depending on the search engines’ options, two different searching strategies can be selected. If the search engine allows searching the whole paper, we use the search string to search the full paper; otherwise, titles, ab-

stracts and keywords must be searched. In this phase, the search string used to perform the search, search settings, number of returned papers, and the number of imported papers to the reference manager tool should be recorded (all factors should be stored per source searched). Next, we enter the filtering phase in which, we filter the results based on titles, abstracts, keywords, introductions, and conclusions, and also remove the duplicate papers. This results in a set of potentially relevant papers, and should be compared to the “quasi-gold” standard. If the condition holds (i.e. “quasi-gold” papers are a subset of automatic results), we start filtering the papers based on inclusion and exclusion criteria to get the final set of primary studies. Please note that at this point the “quasi-gold” standard is certainly a subset of the automatic results (i.e., the condition holds), as we have already tested our search string in the pilot search. In this phase, the inclusion/exclusion criteria (per paper), number of remaining papers, and list of remaining papers are recorded. In the final phase we collect and read the primary studies, and extract the data for data analysis.

2.2.4. Refining the search results

Studies need to cover all the inclusion criteria to be reviewed for the SLR:

2.2.4.1. Inclusion criteria

1. The study should be in the domain of self-adaptive systems. That is, the system should have a model of itself and should be able to monitor and adapt itself in certain circumstances in order to fulfill the systems’ functional and non-functional requirements.
2. The method that deals with self-adaptation should be architecture-based. This implies that the study should provide architectural solutions (i.e., components and architectural models) to handle and reason about the structure and dynamic behavior of the system. To be more specific, the system should use an architectural model of itself to monitor and adapt its structure or runtime behavior. In other words, it should be possible to map the MAPE-K functionalities to components of the adaptation module (managing system).
3. The method presented in the study should handle multiple QAs. This means that the self-adaptation method deals with multiple QAs, or in other words, investigates how application of self-adaptation may affect multiple quality properties of the system. Regarding quality attributes, various “ilities” attributes (e.g., availability, reliability), and “nesses” attributes (e.g., openness, robustness), and other QAs (e.g., performance, efficiency) may be included.

Papers are excluded if they have an intersection with any of the exclusion criteria presented below:

2.2.4.2. Exclusion criteria

1. The study is an editorial, position paper, abstract, keynote, opinion, tutorial summary, panel discussion, or technical report.
2. The study is not written in English.

2.3. Quality assessment

In this SLR, we use a quality assessment method to assess the reporting quality (not to be confused with the quality of research or the proposed method) of all the selected primary studies. The results from this assessment are used for data synthesis and interpretation of results in later stages. All the primary studies that were included in the SLR go through this quality assessment. Similar to (Sarmad et al., 2010), we adopted the quality assessment mechanism used in [12].

⁶ We used Mendeley, which is a free reference manager and academic social network. For further information please visit <https://www.mendeley.com/>.

Fig. 2. Search process.

The quality assessment of [12] was originally designed to evaluate the reporting quality of empirical studies. Therefore, we adapted (i.e., removed questions regarding sample identification/selection, and data collection through forms/interviews) the quality assessment questions in a way that they could be applied to all studies. To assess the quality of the primary studies, each study is evaluated against a set of six questions. Answers to each of the questions can be either "yes", "to some extent" or "no", and then numerical values are assigned to the answers (1 = "yes", 0 = "no", and 0.5 = "to some extent"). The final quality score for each primary study is calculated by summing up the scores for all the questions. The questions are as follows:

- **Q1:** Is there a rationale for why the study was undertaken?
- **Q2:** Is there an adequate description of the context (e.g., industry, laboratory setting, products used, etc.) in which the research was carried out?
- **Q3:** Is there a justification and description for the research design?
- **Q4:** Is there a clear statement of findings and has sufficient data been presented to support them?

- **Q5:** Did the researchers critically examine their own role, potential bias and influence during the formulation of research questions and evaluation?
- **Q6:** Do the authors discuss the credibility and limitations of their findings explicitly?

In order to be able to provide a broad overview of the subject, we decided not to use the quality assessment for inclusion/exclusion, and include all the relevant papers, even those with a lower quality score. The scores are used as an indication of primary studies' validity, and may be beneficial for researchers and practitioners in the domain of self-adaptive systems.

2.4. Data extraction

The selected primary studies were read in detail to extract the data needed to answer the research questions. Two of the authors read the primary studies, partially or fully, and simultaneously extracted data. The set of included papers was divided between the two authors and

one author initially read each paper. If the author encountered any issues (difficulty to understand, extract data, etc.) the paper was set aside for further discussions. Then the two authors switched between them this set of problematic papers so that they both read all these paper and discuss potential issues. Disagreements were resolved by consensus or by consultation with other authors who are experts in the study domain. Data was extracted using the data extraction form shown in Table 2. Note that the column with additional comments helped the authors when extracting data from primary studies, and performing analysis on the data. Explanations/comments (if any) in

Table 2

Data extraction form.

#	Field	Concern / mapping to research question	Additional comments
F1	Author(s)	Documentation	N/A
F2	Year	Documentation	N/A
F3	Title	Documentation	N/A
F4	Source	Reliability of primary study and documentation	N/A
F5	Keywords	Documentation	N/A
F6	Abstract	Documentation	N/A
F7	Citation count (Google scholar)	Documentation	N/A
F8	Method proposed	1	A short summary of the method.
F9	Managing module components	1	Specific components inside MAPE-K loop. Components may be parts of monitoring analyzing, planning, and effector modules. Human interference may also be included in certain scenarios.
F10	Managed system	1	Component(s) being managed by control modules. This may include domain specific software, and execution environment.
F11	Tools/languages	3.3	
F12	Feedback loop organization	1.2	Single, multiple, and application style ^a of the loop in the system (e.g., in layered style).
F13	Centralized or decentralized	3.2	N/A
F14	Domain/development paradigm	1.3, 1.4	If the domain is unclear, it should be stated as "unclear". Or, if a toy example, or industrial evaluation of the proposed method is presented, the domain of the example should be stated.
F15	QA/goal or side effect/relevant subsystem	1.1, 2	List of QAs. Specify if the QAs are objectives of adaptation or side effects of adaptation.
F16	Models	2.2	Refers to any type of model at design or runtime that represent a QA.
F17	QAs prioritization mechanisms	2.3	n/a
F18	Adaptation strategies/tactics	3.1	List of activities.
F19	Limitations / Weaknesses/ Strengths	1.5	A brief description of methods' limitations.
F20	Assurances	3.4	If explicit evidence is provided for satisfaction of QAs after adaptation, we try to map the approaches to those listed in [33].

^a Note that in this document "application style" has a broader definition than "architecture style"; it refers to any type of information regarding organization and application of feedback loops in a system. For example, application style may only refer to use of multiple or single feedback loops, or multiple feedback loops organized in layered format in a system.

this column were used to avoid misconceptions and resolve any disagreements among authors.

Data stored in [F1] to [F3], and [F5] to [F7] are only for documentation purposes. In [F4] we stored the searched sources (i.e., where the primary study was published) that indicate the reliability of the primary study. The rest of the data fields were used for data analysis and are briefly explained below:

- **Method proposed [F8]:** A summary of the proposed method should be recorded.
- **Managing module [F9]:** refers to the system part(s) responsible for monitoring the system and performing adaptation if needed.
- **Managed module [F10]:** refers to part(s) of the self-adaptive system that need to be monitored and adapted. It may include a variety of different items from parameters and methods to components, architecture style, and system resources [30]. Parameters or system components are not necessarily required to be affected by adaptation actions in order to be considered as managed modules. All the parameters or system components that are monitored are taken into account as managed modules.
- **Tools/language [F11]:** refers to the existing tools/languages that are used by the proposed method or new tools/languages that are designed and implemented in the primary study.
- **Type of loop [F12]:** refers to the characteristics of the MAPE-K model used in the system design, as well as the format (i.e., single or multiple) of used feedback loop.
- **Centralized or decentralized [F13]:** it addresses the issue of having a single (i.e., centralized) or multiple (i.e., decentralized) control components (i.e., analysis and plan from MAPE-K) for decision making. In a centralized method, one control component realizes the need for adaptation of a software system/component, but in a decentralized setting the system's runtime behavior emanates from the localized decisions made by multiple control components and their interactions [22,36].
- **Domain/development paradigm [F14]:** application domain or development paradigm of the proposed method.
- **QA [F15]:** refers to any QA handled by the self-adaptation method. It can be one of the goals of the self-adaptation system, or it can be due to the application of self-adaptation method (i.e., side effect of the method). For instance, if a system needs to be scalable (i.e., objective of the system) the proposed method should handle scalability. But, if the performance is affected as a side effect of supporting scalability, it should be recorded as an impact of the proposed method.
- **Models [F16]:** refers to models used to represent QAs of the system, their characteristics, and measurements both at runtime and design time. For instance, if the intended QA is reliability, the characteristics may be fault tolerance, and recoverability, and the measurement indicates the threshold for adaptation. In other words, these models may be used to define when and how the QAs should be updated, and what their dimensions (i.e., characteristics) are.
- **QAs prioritization mechanisms [F17]:** refers to mechanisms used to give preference to certain QA over the rest for the purpose of decision making and achieving system's goals in different circumstances (e.g., utility functions).
- **Adaptation strategies/tactics [F18]:** adopted from [9], refers to a flow of actions over a sizable time frame and scope, with intermediate decision points, to fix a type of system problem and meet quality related issues. Actions may refer to any step of decision making process, such as selecting the appropriate data to be collected, as well as analysis and planning methods. All these actions should be identified and recorded as elements of adaptation strategies.

- **Limitations / Strengths [F19]:** addresses the methods' technical limitations or strong points as discussed in the primary study. Strengths may be any of the following options:
 - (1) Rigorous evaluation methods (e.g. industrial example, case study).
 - (2) The proposed method is compared with similar existing methods.
 - (3) The method is not domain specific.
 - (4) The method is applicable in both centralized and decentralized settings.
- **Assurance [F20]:** refers to method's ability to deliver evidence that the system satisfies its stated functional and non-functional requirements during its operation in the presence of self-adaptation [7]. If the primary study provides any mechanism to check whether or not the system complies with its functional and non-functional requirements; we use Table 3 to categorize the identified mechanism.

Approaches listed in Table 3 are adopted from [33]. Listed approaches are representatives of the assurances approaches that have been developed through the years. Depending on the nature of the approach (i.e., manual, automatic, and combination of both) approaches are categorized into three different groups (i.e., human-driven, system-driven, and hybrid approaches).

3. Results

In this section, we analyze the data collected from the primary studies and answer the research questions. Our main goal in this section is to give an overview of different aspects that current approaches address (e.g., type of QAs, common models etc.). In the next section we further synthesize the results and present a summary/classification of dimensions (i.e., Table 21) that are typically addressed in architecture-based self-adaptive methods dealing with multiple QAs.

Table 3
Approaches for assurances.

Group	Approaches
Human-driven approaches	- Formal proof - Simulation
System-driven approaches	- Runtime verification - Sanity checks - Contracts
Hybrid approaches	- Model checking - Testing

Fig. 3. Number of published primary studies per year.

3.1. Results overview and demographics

We automatically searched 28 venues, and after merging the results from different sources and removing duplicates we ended up with 7453 papers. After applying the inclusion and exclusion criteria and filtering out the irrelevant papers we were left with 288 papers. At this stage it was not possible to filter out more papers based on titles, abstracts, and conclusions anymore. Therefore, to filter these 288 papers, we started reading the whole papers and applying inclusion and exclusion criteria in order to get the final set of papers. Finally, we ended up with 54 primary studies on which we performed the data extraction. Fig. 3 shows the number of primary studies published per year between 1st of January 2000 and 20th of July 2014. This figure shows that the first primary studies started to appear in the year 2002; however, the number of published papers is low (i.e., only ten papers) before the year 2009. The majority of primary studies (i.e., 46 papers) have been published after 2008 with the peak (i.e., 14 papers) in the year 2013. This indicates that before the year 2008 handling of multiple QAs through architecture-based approaches has been under-studied, and since the year 2009 more researchers started focusing on this topic in the domain of self-adaptive systems.

Table 4 shows that 34 primary studies out of 54 were found in conferences, 11 primary studies were published in books, six were found in journals, and three primary studies belonged to workshops.

In Fig. 4, we categorized the primary studies based on the venues in which they were published. Our results indicate that the Software Engineering for Adaptive and Self-managing Systems (SEAMS) symposium with 17 publications is the most prominent venue. The Journal of Systems and Software and the International Conference on Software Engineering have the lowest number of publications with only four primary studies each. SEAMS which was established in the year 2006, is a venue focusing on engineering self-managing and adaptive systems and currently is the most relevant venue to our domain of interest.

For each primary study, we answered the six quality assessment questions (see Section 2.3) and summed up the scores of all questions to calculate the final quality assessment score. Fig. 5 gives an overview of the status of the quality assessment scores for the primary studies. As indicated in Fig. 5, the majority of primary studies (i.e., 33 papers) gained scores between 4 and 6; this suggests that primary studies included in our review are of an acceptable quality (i.e., quality of reporting) in general.

Table 4

List of primary studies per publication type.

Publication types	Number of papers	Paper identifications
Conferences	34	S4, S5, S6, S9, S10, S11, S12, S14, S15, S16, S17, S19, S20, S22, S23, S24, S25, S26, S27, S28, S30, S31, S32, S39, S41, S42, S43, S46, S47, S48, S49, S50, S52, S54
Books	11	S1, S7, S13, S18, S21, S29, S34, S35, S38, S51, S53
Journals	6	S3, S8, S36, S40, S44, S45,
Workshops	3	S2, S33, S37

Fig. 4. Number of primary studies published per venues.

To further analyze the quality of the primary studies, in Fig. 6 we provide a breakdown of the primary studies' scores based on the quality assessment questions (QAQ).

Regarding QAQ1 (i.e., is there a rationale for why the study was undertaken?), it is clear that most of the primary studies (i.e. 89%) acquired the highest score, meaning that primary studies present adequate rationale for why the study was undertaken; only 11% of the primary studies scored 0, 5 and there is no primary study with score zero.

Results of QAQ2 (Is there an adequate description of the context in which the research was carried out?) indicates that most of the pri-

mary studies (i.e., 85%) provide sufficient description for the research context, 9% of the primary studies have a mediocre context description, and 6% of the primary studies do not provide description for their research context.

Results for QAQ3 (Is there a justification and description for the research design?), that 69% of the primary studies comprise decent justification and description for their research designs, 22% of the primary studies to some extent provide justification and description, and 9% of the primary studies do not provide justification and description for their research designs.

Regarding QAQ4 (Is there a clear statement of findings and has sufficient data been presented to support them?), our results indicate that the majority of the primary studies contain comprehensible statement of findings, and provide adequate data to support their conclusions, 26% of the primary studies offer an average amount of data to support their finding and their results are not presented in a highly straightforward format. Finally, in 4% of the primary studies the findings are presented ambiguously and the primary studies lack any data to rigorously support their findings.

Our results for QAQ5 (Did the researchers critically examine their own role, potential bias and influence during the formulation of research questions and evaluation?) suggests that more than half (i.e., 55%) of the primary studies do not address potential bias and influence of the authors on the formulation of the research questions and evaluation. However, there is no evidence to show whether this absence of examining author's bias is due to difficulty of performing such inspection, or it is simply considered insignificant and is neglected by the researchers. Regarding the rest of the results (i.e., 45%), 43% of the primary studies discuss potential influence of the researchers to some extent, and only in 2% of the results authors meticulously discuss their own role and bias in the formulation of research questions.

Lastly, our results for QAQ6 (Do the authors discuss the credibility and limitations of their findings explicitly?) indicate that almost a third of the primary studies (i.e., 30%) lack discussion of the credibility of the results and do not explicitly address the limitations of the main findings. In 33% of the primary studies, authors address the limitations of their findings to some extent, and in 37% of the pri-

Fig. 5. Quality assessment scores of primary studies.

Fig. 6. Quality assessment scores breakdown.

many studies authors provide a thorough discussion on the limitations and credibility of their findings.

Most primary studies included in our review are of an acceptable quality. However, answers to the quality assessment questions indicate that current research lacks a thorough investigation of credibility and limitations of the methods, as well as the potential bias and influences that authors may have had on the formulation of the research questions and evaluation.

3.2. RQ1: What are the characteristics of existing architectural methods for handling multiple QAs in self-adaptive software systems?

We used data collected in [F8], [F9], [F10], [F12], [F14], and [F15] to answer the first research question. The answer to this research question gives an overview of current methods handling multiple QAs in architecture-based self-adaptive systems.

3.2.1. RQ1.1: Which sets of QAs are addressed by these methods?

This research question investigates which QAs sets are the most commonly addressed QAs in this domain. We used the data from field [F16] to answer this research question. We answer this research question from two different perspectives: (1) We list the most commonly addressed QAs and indicate whether they are the main reason for adaptation (i.e., goal), or they are affected as a consequence of adaptation (i.e., side effect) in the system; (2) We list the most commonly addressed sets of QAs, and QAs in general, by the current methods and indicate which QAs are goal/side effect.

Fig. 7 indicates the most commonly addressed QAs identified in the primary studies (for a detailed list, see Table 5). The length of the bars represent the number of primary studies, the blue section shows the number of primary studies in which the QA is the goal of the system, and the red section shows number of primary studies in which the QA is affected due to adaptation. Performance is the most promi-

Fig. 7. Most commonly addressed QAs in primary studies.

Table 5

List of primary studies with most common QAs.

QA	Number of papers	Paper identifiers
Performance	36	S1, S6, S7, S9, S11, S12, S13, S14, S15, S16, S19, S20, S21, S22, S23, S24, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S37, S41, S42, S44, S45, S46, S47, S48, S49, S51
Reliability	15	S1, S2, S6, S7, S8, S17, S18, S31, S33, S38, S39, S42, S46, S53, S54
Cost	13	S9, S14, S19, S21, S22, S23, S24, S27, S29, S40, S42, S45, S46
Availability	11	S3, S7, S12, S13, S15, S17, S19, S39, S40, S51, S53
Scalability	7	S2, S13, S26, S28, S47, S49, S51

nent QA, and is addressed by 37 primary studies from which seven of the primary studies specifically investigate response time. In 33 of the primary studies, performance is considered as one of the goals of the system, and the adaptation is applied to improve a system's over-

all performance. In four primary studies, performance is explored to determine how the systems adaptation may have affected this QA. Scalability is the least addressed QA (i.e., seven papers), and is mainly investigated as the goal of adaptation (i.e., six papers).

It is worth noting that some primary studies do not clearly discuss if certain QAs are the goal of the adaptation or are only affected as a consequence of adaptation. For instance, in paper S40 it is not clear if cost and availability are the goals or side effects of system adaptation. On the other hand, in some primary studies, QAs are considered as both goals and side effects of adaptation (e.g., S18 and S22).

Note that to deal with cost we adopted the concept of business quality attributes from [4]. In their book, Bass et al. discuss that in addition to the qualities that define how the system should function, there are a number of business quality goals such as time to market, cost and benefit, etc. that center on cost and shape the system's architecture. These business qualities are treated the same way (e.g., need to be specified by scenarios and to be made testable) that the other system quality attributes are treated. Following the same viewpoint, in this systematic review we treated cost (i.e., expenditure of developing and maintain the system) as a quality attribute where applicable. In Table 6 we listed all the primary studies addressing cost, and categorized them based on the definitions they provided for cost in their work.

Note that in two primary studies (i.e., S40 and S42) cost is explicitly treated as a quality attribute, however, we could not extract any specific definition from these two papers.

Fig. 8 represents Venn diagram of QAs, both as goals and side effects, and number of papers addressing the QAs. The intersections of sets show the number of papers addressing those sets of QAs. As indicated, performance and cost have the highest (i.e., 11 papers), and performance, cost, availability, and reliability; and cost, availability, and reliability have the lowest number of papers, which is zero.

Table 7 presents a list of QA sets, which are handled by the primary studies. Performance and cost appear as the most common set of QAs (i.e., in seven papers). Contrary to the fact that scalability was the least addressed QA in Table 5, performance and scalability is one of the second most common sets of QAs and is explored in three primary studies. Finally, performance and dependability are the least addressed set of QAs.

In Section 1.1.2 we listed the QAs (i.e., performance, reliability, flexibility, maintainability, and availability) which we initially in-

Fig. 8. Venn diagram of QAs and number of papers addressing them.

Table 7

Most commonly occurred sets of QAs.

QAs sets	Paper identifiers	Number of papers
Performance and cost	S9, S14, S21, S23, S24, S29, S45	7
Performance and scalability	S26, S28, S49	3
Performance and security	S11, S20, S44	3
Performance and reliability	S1, S31, S33	3
Performance, reliability and cost	S42, S46	2
Performance and availability	S12, S15	2
Performance and dependability	S35, S41	2

tended to investigate in this systematic review. However, by conducting the review a full list of QAs was extracted that includes all of the quality properties which are addressed in the existing papers. Unlike QAs presented in Tables 5 and 6, these QAs were not addressed frequently, however, we provide the full list of these QAs in Appendix C.

3.2.2. RQ1.2: How many feedback loops are used in current methods and how are they arranged to handle multiple QAs?

In Table 8, we list identified feedback loop application styles and primary studies using them. Our results indicate that most (i.e., 27 papers) of the current architecture-based self-adaptive methods dealing with multiple QAs are in accordance with classical MAPE-K reference model and use one feedback loop to execute the adaptation actions. In five primary studies (i.e., two feedback loops in S6, S7, S25; and three feedback loops in S21 and S30) a combination of multiple feedback loops is used to perform adaptation actions. Furthermore, we identified five primary studies, in which rather unconventional application styles of feedback loops are used. In S1 and S2 feedbacks are used in layered style, S32 uses multiple interacting feedback loops, S26 uses multiple feedback loops, and S5 uses multiple interacting feedback loops in layered style.

Since the multiple interacting feedback loops, and layered feedback loops are not used frequently, we briefly explain these styles. The method proposed in S5 requires multiple feedback loops to deal with multiple concerns (e.g., failures, performance), meaning, each loop corresponds to a different concern. Each concern needs its spe-

Table 6

Categories of costs identified in the primary studies.

	Paper identifiers	Definition of cost
Operational cost	S9	Minimizing the operational cost of for server pool.
	S14	Keeping the cost of server pool within its operating budget.
	S19	Cost of running a system in the cloud should be minimized.
	S21	Guaranteeing low operational cost.
	S23	Operational cost for active servers.
Development cost	S29	Keeping the operating cost of the server pool within its operating budget.
	S24	Minimizing cost of development.
	S46	Monetary cost of adding new servers.
	S22	Minimizing operational and development costs.
Both	S27	Operation of the rack server system at minimum cost.
	S45	Keeping the cost of providing the service low.
Unspecified	S40	Quality attribute; no additional definition is provided.
	S42	Cost is considered a non-functional quality of service requirement.

Table 8
Feedback loop(s)'s arrangements in investigated methods.

Feedback loop application style	Paper identifiers	Number of papers	Number of papers per QAs sets			Not mentioned
			Set of two QAs	Set of three QAs	Set of >3 QAs	
One feedback loop	S3, S4, S8, S9, S10, S12, S13, S14, S15, S16, S17, S18, S19, S22, S24, S27, S28, S29 S31, S34, S36, S39, S45, S46, S49, S52, S54	27	17	3	5	2
Two feedback loops	S6, S7, S25	3	N/A	1	1	1
Layered	S1, S2	2	1	1	N/A	N/A
Three feedback loops	S21, S30	2	2	N/A	N/A	N/A
Multiple interacting feedback loops in layered	S5	1	N/A	N/A	N/A	1
Multiple feedback loops	S26	1	1	N/A	N/A	N/A
Multiple interacting feedback loops	S32	1	1	N/A	N/A	N/A
Unknown	S11, S20, S23, S33, S35, S37, S38, S40, S41, S42, S43, S44, S47, S48, S50, S51, S53	17	–	–		

cific models and model operations and they are realized through different feedback loops. The feedback loops have to interact and coordinate their adaptation actions to avoid any conflict. Regarding layered application style, in S1 for instance, the self-management functionalities are scattered in a three-layer model, in which, different types of changes are handled by different levels of adaptation. In S2, the proposed infrastructure consists of multiple layers, and each of them fulfills different aspects of feedback loop functionalities.

We also categorized and listed the papers based on the number of QAs they address, to identify any potential correlation between the styles of feedback loops and the number of QAs they aim to handle. From Table 9, we can observe that most of the feedback loops styles

Table 9
Application domains of methods.

Domain	Paper identifiers	Number of papers
Robotics	<i>S1, S15, S26, S31, S50, S52</i>	6
Mobile applications	<i>S18, S33, S48</i>	3
Health care	<i>S4, S51</i>	2
Not mentioned.	<i>S5, S16</i>	2
Transportations	<i>S32, S39</i>	2
Monitoring and safety systems	<i>S17</i>	1
Other	<i>S2, S3, S6, S8, S9, S10, S11, S12, S13, S14, S24, S25, S28, S34, S35, S36, S40, S41, S43, S44, S45, S46, S47, S49</i>	22

are aimed at handling sets of two QAs. Interestingly, two independent feedback loops seems to be the only style that is particularly used to handle more than two QAs. Note that the last column of the table (i.e., marked as “not mentioned”) includes number of papers in which the number of addressed QAs were not explicitly specified.

In Fig. 9, we present a clustered chart to indicate how the feedback loop application styles are used to deal with the most frequently identified QAs.

As indicated in the figure, the conventional style of feedback loop (i.e., one feedback loop) is extensively used to deal with all five common QAs in domain of architecture based self-adaptive systems. Two feedback loops and layered feedback loops styles are used to deal with performance, reliability, and availability QAs, and three feedback loops style is used to handle performance and cost. From this figure we can see that the most uncommon styles (i.e., different types of multiple feedback loops) do not address an extensive number of QAs. This might be due to novelty of these methods, and therefore, they currently focus on a fewer number of QAs.

3.2.3. RQ1.3: What are the application domains of current methods dealing with multiple QAs?

Purpose of this research question is to find the application domains of current methods. Most of the proposed methods (i.e., seven papers) are applied in the domain of web-based systems, followed by SOA-based systems and robotics. Some of the domains in Table 9 may overlap, but this categorization maps each method to the closest domain. As indicated in Table 9, 22 of the primary studies are listed under “other” category. The methods presented in these primary studies are used in a variety of information technology domains, and it was not possible to classify them to any distinct categories.

Performance, in combination with a variety of other QAs (e.g., reliability, availability, and scalability), is the most frequently (i.e., nine papers: S1, S5, S26, S31, S32, S33, S48, S51, and S16) addressed QA among listed domains of Table 9. Regarding the QAs sets, performance and reliability is the most common (i.e., addressed in S1, S6, S31, S33, and S38) QAs set which is handled in these domains. It is worth mentioning that there were no particular patterns in the sense of sets of QAs addressed in particular domains.

3.2.4. RQ1.4: What are the development paradigms of current methods dealing with multiple QAs?

Since the actual application domains were not always mentioned explicitly, we used data collected for research question 1.3 to categorize these particular set of papers based on their commonly used technological/development paradigms (Table 10).

Similar to domains, performance is the most commonly (i.e., seven papers) handled QA among develop paradigms of Table 10. With regard to QAs sets, performance and cost are the most common (i.e., S21, S23, and S29, S45) combination of QAs in these development paradigms. In Tables 9 and 10, papers listed in italic fonts represent papers dealing with performance-along with various QAs.

3.2.5. RQ1.5: What are the limitations and strengths of existing methods dealing with multiple QAs?

We use data collected in [F19] to answer this research question. As discussed in Section 2.4, for the sake of simplicity we used a set of pre-specified answers for data extraction regarding the strengths of methods. For limitations, we investigated the technical limitations and unresolved issues explored in selected papers. For a full list of all the identified limitations please see Table 26 of Appendix D. Note that in some cases authors discuss the limitations of their approaches in depth, and it is not feasible to summarize the technical discussions

Fig. 9. Feedback loop styles and common QAS.

Table 10

Identified development paradigms.

Development Paradigm	Paper identifiers	Number of papers
Web based systems	S20, S22, S23, S29, S30, S35, S45	7
SOA based systems	S7, S21, S38, S42, S53, S54	6
Cloud computing	S19, S27, S37	3

in a comprehensible list. For those details we refer the reader to the corresponding primary studies.

Although the identified limitations were mostly specific to the methods, we could categorize and present a number of reoccurring limitations in Table 11. Lack of sufficient knowledge about the self-adaptive system, simplifying the problem, omitting some details to reduce the complexity of the problem, and state space explosion are some of the recurring limitations in the primary studies. The common element in all these limitations stems from the complexity of the problem, and the fact that over-simplification of the circumstances results in unrealistic problem statements that consequently create even more uncertainty in the system. Consequently, proper use of available knowledge accompanied by efficient solutions may help to mitigate some limitations of the current methods.

As discussed in Section 2.4, we used a set of pre-specified options to extract strengths of the methods. In Table 12 we list all the identified strengths of investigated methods in the primary studies. Use of rigorous evaluation methods has the highest frequency (i.e., 15 papers) among the listed strengths; on the other hand, comparison of method with similar existing methods, as well as, rigorous evaluation methods and applicability of the method in both centralized and decentralized settings both have the lowest frequency, and have been used in one primary study.

3.2.6. Summary of selected primary studies

Since the main objective of posing RQ1 is to present an overview of current research in the domain, in this section we briefly discuss primary studies. However, rather than describing all the architecture-based self-adaptive methods discussed in the primary studies, we decided to give a summary of the most representative methods.⁷ We speculate that the primary studies with most citations can be deemed

Table 11

List of identified reoccurring limitations.

Limitation theme	Paper identifiers
Lack of sufficient knowledge	S8, S39
Simplifying the problem, and omitting some details to reduce the complexity of the problem	S1, S17, S20, S26, S53
State space explosion	S35, S41

Table 12

Identified strengths of methods.

Strength	Paper identifiers	Number of papers
Rigorous evaluation methods	S4, S6, S9, S11, S17, S20, S22, S23, S24, S27, S28, S29, S39, S40, S52, S15, S30	15
Rigorous evaluation methods and comparison of method with similar existing methods	S15, S30	2
Applicable in both centralized and decentralized settings	S32, S45	2
Comparison of method with similar existing methods	S26	1
Rigorous evaluation methods, and applicable in both centralized and decentralized settings	S31	1

representative. Thus, we decided to give an overview of the most cited self-adaptive methods at each intersection of the Venn diagram shown in Fig. 8. By doing so, we are also describing how these methods cope with the challenge of finding the best balance between these widely investigated QAs. The methods are listed according to the QAs sets they handle:

- **Availability and cost.** According to the Venn diagram in Fig. 8, only one primary study deals with availability and cost (i.e., S40). In this primary study, Perez-Palacin et al. propose an approach for analyzing tradeoffs between the system adaptability and its quality of service. According to Perez-Palacin et al., their approach can help architects when making decisions concerning what adaptations should be made to fulfill system QAs. The proposed approach relies on a set of metrics that allow architects to evaluate the system adaptability. More specifically, Markov models are used to compute the QAs and stochastic Petri nets are used to represent the sys-

⁷ A short summary for each method presented in the selected primary studies is recorded and available in our extracted data file in <http://www.cs.rug.nl/~paris/SLR-MultipleConcerns-TR.pdf>.

tem. Perez-Palacin et al. describe an example in which availability and cost are taken into account during system adaptation.

- **Performance, cost, and availability.** S19 is the only primary study that addresses performance, cost, and availability. In such study, Zoghi et al. propose a method for eliciting, evaluating, and ranking elements that can affect the quality (i.e., control points) of web applications deployed in cloud environments. The proposed method encompasses several phases that take high-level adaptation goals and render them into lower level MAPE-K loop control points. The MAPE-K loop monitors metrics related to goals and evaluates them. These loops are then activated at runtime using search-based algorithms. The search-based algorithm looks for the most effective control points to achieve the adaptation goals. In this context, the goal of adaptations is to optimize an objective function that is defined over certain QAs. More specifically, the adaptation algorithms try to reduce the gap between the collected and desired goal's metrics by iteratively changing control points. Control points are changed one at a time until no further enhancements can be found.
- **Availability, performance, and reliability.** This set of QAs is considered by Tichy and Giese in S7. Tichy and Giese propose an enhancement of the Jini architecture, which is a self-healing, service-based architecture. Tichy and Giese identified a set of architectural principles that can be used to improve some dependability attributes, i.e., availability and reliability. These architectural principles were then realized in an architecture that extends Jini with several infrastructure services that systematically control and configure availability and reliability. In addition to Jini's lookup service, four different service types are used: nodes, a service description storage, monitors, and monitor responsibility storage. This solution comprises two control loops; the monitor and the related service instance form a closed control loop. In order to control which monitor is supervising which service, monitors need to acquire a responsibility for a service. Therefore, an additional overlaying control loop is used to monitor the monitors themselves.
- **Availability and reliability.** As shown in the Venn diagram in Fig. 8, three studies appear at the intersection of availability and reliability. In S39, Weyns et al. give an overview of the main capabilities required for engineering a decentralized self-adaptive system. Two case studies are used to describe key attributes that set decentralized self-adaptive system from the more commonly found centralized self-adaptive systems. Among the quality of service related concerns dealt with in the case studies, the authors mention availability and reliability. The lessons learned throughout the development of the systems are presented in the two case studies and are generalized in the form of a high-level reference model. Drawing from their experience and the resulting reference model, Weyns et al. identify key future research challenges in this area.
- **Reliability and performance.** Five studies deal with reliability and performance (i.e., S1, S6, S31, S33, and S38). S33 is the most cited of the five. In S33, Rouvoy et al. report on an approach tailored to self-adaptive component-based architectures. More specifically, they describe an extension of the MUSIC component-based planning framework, which is aimed at optimizing the overall utility of applications when they need to dynamically adapt themselves to varying execution contexts. In particular, their approach emphasizes changes in the service provider landscape in order to plug in interchangeable components and services. In this approach, utility functions are used to represent the utility of components and adaptations are performed automatically with the purpose of optimizing the application utility in different contexts.

- **Availability and performance.** This set of QAs is addressed in four primary studies: S12, S13, S15, and S51. Considering this particular subset, the most cited primary study is S15, which present an architectural approach to building self-aware and self-adaptive robotic systems. This approach, which is centered around the concept of meta-level components, allows roboticists to realize adaptive layered robotic systems. Edwards et al. propose specialized meta-level components for the three main activities of robotics systems: sensing, computation, and control. In addition, the meta-level components in each layer are monitored, managed, and adapted by other higher layer meta-level components. According to Edwards et al., this meta-level layering helps to ensure that QAs are satisfied in a flexible and adaptable fashion.
- **Cost, performance, and reliability.** This set of QAs is investigated in two primary studies (i.e., S42 and S46) of which S42 is the most cited. In S42, Cardellini et al. put forward a self-adaptive method tailored to service-oriented systems and whose main purpose is fulfilling non-functional quality of service requirements (i.e., cost, performance, and reliability). To determine the most effective adaptation action, the method relies on the formulation and solution of a linear programming problem, which is derived from a behavioral model of the system that is updated at runtime through monitoring activities.
- **Performance and cost.** According to our results, performance and cost are by far the most investigated set of QAs. As shown in Fig. 8, nine primary studies propose self-adaptive methods that accommodate variability in cost and performance. Among these primary studies, S45 is the most cited one. In S45, Garlan et al. describe the Rainbow Framework, which is a quintessential example of a self-adaptive method that relies on abstract architectural models to monitor and adapt systems at runtime. As stated by Garlan et al., architectural models give both an overview of the system, showing all the components and their connections as well as indicating key properties about these components. In this context, architectural elements are annotated with several properties as, for instance, expected throughputs and latencies. Essentially, these properties dictate when adaptations should be triggered. As a result, when some property is violated, the framework carries out global- and module-level adaptations on the running system.

Summary of answer to RQ1: Performance and cost are the most frequently addressed QAs; however, performance and reliability are the most commonly addressed individual QAs in primary studies. Most of the existing methods consist of one control component that fulfills the adaptation tasks through one feedback loop. Current self-adaptive systems dealing with multiple QAs mostly belong to the domain of robotics and web-based systems paradigm. Our results indicate that, the listed limitations in the primary studies are technical and domain specific. Almost half of the primary studies discuss the technical limitations of their methods. On the other hand, strengths of the methods can be mapped into more generic categories and rigorous evaluation of methods is the most common category of strength.

3.3. RQ2: How do these methods handle multiple QAs?

We used data collected in [F12], [F15], [F16] and [F17] to answer this research question. This research question strives to acquire insights into the details of the adaptation methods. In particular, the answer to this research question clarifies what models are used to represent and analyze multiple QAs characteristics, and what mechanisms are used to deal with tradeoffs between multiple QAs that need to be handled.

3.3.1. RQ2.1: What models are used to represent and analyze multiple QAs characteristics?

In Table 13, we list the most commonly used models, papers identifiers, number of papers using the models, and number of papers addressing common QAs. As described in Section 2.4, in the context of this study a *model* refers to any mechanism used to represent non-functional requirements and their characteristics, and measurements. Our results indicate that models are used in two different ways in order to handle multiple QAs in self-adaptive systems. The first is to use a certain model to express an individual QA, and its different characteristics. For instance, QA data variables and thresholds can be used to define different characteristics of performance (i.e., throughput and latency), or a utility function can be used both to express the QA characteristics, and also to support preferences of one character-

Table 13

List of common models identified in the primary studies.

Models	Paper identifiers	Number of papers	Number of papers per QAs
QA data variables	S6, S8, S17, S19, S21, S22,	16	Performance (12)
	S24, S25, S28, S30, S34, S39, S41, S45, S46, S49		Cost (6)
Utility functions	S1, S6, S12, S14, S18, S25, S29, S38, S46, S48, S50, S52, S54	12	Reliability (6)
			Availability (3)
Markov Chains	S18, S23, S35, S40	4	Scalability (2)
			Performance (8)
Service level agreement (SLA)/ thresholds/ criteria	S42, S44, S45, S46	4	Cost (3)
			Reliability (6)
Goal models	S26, S52	2	Availability (1)
			Scalability (none)
Other models	S3, S7, S8, S18, S23, S33, S35, S39, S40, S53	12	Performance (2)
			Cost (2)
None	S2, S4, S5, S9, S10, S11, S13, S15, S16, S20, S26, S27, S31, S32, S37, S51	16	Reliability (1)
			Availability (1)

istic over the other by using weights. The second is to use the same models to represent the tradeoffs between multiple QAs in the entire system or certain system component. For instance, utility functions may be used to get the utility of each component in the system, and then the overall utility of the whole system, or discrete-time Markov chains may be used to represent system's behavior, which are system's states and related properties of interest and the probabilities of transitioning from one system state to another. In this way, the model is used to indicate the system's status and how it may transition from one state to another depending on circumstances and applied adaptation actions. In general, QA data variables are mostly used for expressing the characteristics, utility functions for preferences, SLA/threshold/criteria for priority and identification of violations, and Markov chains for expressing quality properties of systems with stochastic behavior. While extracting data to answer this research question, we observed that these models are mainly produced to be used at runtime by the machine to support self-adapting properties.

From Table 13, we can see that QA data variables are the most widely used mechanisms (i.e., 16 papers) to represent QAs and their characteristics. QA data variables are the most basic mechanism for QAs representation and address different aspects of QAs at an abstract level. Utility functions, including weighed utility functions, are the second most commonly used mechanism. Note that both utility functions, and QA data variables should be combined with other mechanisms to assist the decision making process. On the other hand, Markov model, which is a particular type of representation that allows specifying behavior sequences of a stochastic system, is only used in four primary studies. When Markov models are used additional mechanisms are required to analyze the state space of such a model to support decision-making (e.g. an analysis component that uses a runtime model checker). This implies that despite having a variety of proved mathematical instruments that can be used to model stochastic behavior of a self-adaptive system, these instruments are not widely used in the existing research.

As for the QAs, we can see from Table 13 that in most cases the models are used to deal with performance, cost, and reliability quality attributes. Regarding the commonly addressed sets of QAs, two different QAs sets were discernible among certain models. We observed that the QAs data variables as well as the utility functions are frequently used to handle performance and cost (i.e., three and two times respectively) qualities. In addition, service level agreements are often used to deal with performance, cost, and reliability (i.e., two times) quality attributes.

We also identified a number of less common models (i.e., expressed as "other models" in Table 13) which we list in Table 14. As indicated in the table, generalized stochastic Petri nets are used in two primary studies (i.e., S7 and S40) and the rest of the models are used in only one primary study.

Table 14

List of other models identified in the primary studies.

Other models	Paper identifiers
Generalized stochastic Petri nets	S7, S40
Properties and property predictor functions	S3
Combination of metrics and runtime models (that describe the adaptation rules)	S8
Feature models	S18
Scenario models	S23
Activity diagrams and automata	S33
Bridge models	S35
Deployment architecture models	S39
Requirement models	S39
Service workflow specifications	S53

3.3.2. RQ2.3: How do these methods address multiple QA tradeoffs?

In Table 15 we list all the prioritization mechanisms and the relevant sets of QAs identified in the primary studies. Our results indicate that 13 primary studies use explicit prioritization mechanisms to analyze QAs and choose the best self-adaptation configuration to deal with these non-functional requirements. The most common mechanisms are utility functions and weighted utility functions that are used in nine (i.e., S12, S14, S25, S46, S48, S18, S29, S38, and S54) primary studies. As briefly discussed before, utility functions may be used to get the utility of each component in the system, and then the overall utility of the whole system regarding a set of QAs. Weighted utility functions, on the other hand, use different factors to reinforce the priority of certain QAs over others when calculating utility.

Overall, our results indicate that existing approaches rarely consider priority of QAs. Moreover, this finding implicitly suggests that many of the current approaches do not support preemption (i.e., preempting a running strategy and schedule a new one to address a new event with a higher priority and/or utility), while applying adaptation and ignore the occurrence of new conditions that call for urgent adaptation. Overall, explicit discussions of tradeoffs between multiple QAs were seldom present in primary studies.

From another perspective, in Table 16 we present a list of QA sets, which are handled by the primary studies, and a breakdown of the roles of identified QAs. This table indicates which QAs were the main goal of adaptation, and which QAs were addressed due to effects of adaptations. Performance and cost appear as the most com-

Table 15

List of common prioritization mechanisms and the relevant QAs.

QAs prioritization mechanisms	QAs sets	Paper identifiers	Number of papers
Utility functions	Performance, availability, cost, reliability, accuracy, modifiability, usability	S12, S14, S18, S25, S29, S38, S46, S48, S54	9
Ordering of adaptation steps in structural adaptation	Extendibility, reusability	S10	1
Rank control points	Performance, cost, availability	S19	1
Polling scheduler	Performance, cost	S24	1
Coordination mechanisms	Reliability, availability	S39	1

Table 16

Roles of identified QAs as goal or side effect of adaptation.

QAs sets	Paper identifiers		
	All QAs as Goals	QAs as both goal and side effect (s)	QAs as side effects
Performance and dependability	S41		S35
Performance and availability	S12, S15		
Performance and reliability	S1, S33	S31: performance as side effect	
Performance, reliability and cost	S42, S46		
Performance and security	S11, S20, S44		
Performance and scalability	S26, S28, S49	S28: Scalability as side effect	
Performance and cost	S14, S21, S23, S24, S45	S9: Cost as side effect S29: Cost as side effect	

mon set and in most of the cases both of the QAs are goals of adaption. In two primary studies (S9 and S29) performance is the goal of adaptation and cost is the side effect. Performance and dependability are the least addressed set of QAs and they are both considered as goals of adaptation in the identified primary study (i.e., S41).

In Table 25 of Appendix C we present the sets of QAs which were identified in addition to our initially specified list of qualities in Section 1.1.2; QAs affected as side effects of adaptation are shown in *italic*.

We further analyzed our results by comparing answers to this question with answers to RQ1 to explore the specifications of existing methods regarding multiple QA tradeoffs as follows:

Use of models in multiple QAs tradeoff analysis. Our results (presented in Table 13) indicate that one of the objectives of using models can be the representation of the tradeoffs between multiple QAs in the entire system, or certain system component. For instance, discrete-time Markov chains may be used to model system's behavior; in this approach the model is used to indicate the system's status and how it may transition from one status to another depending on circumstances. In addition, depending on the types of QAs of interests, a variety of models may be used to reason about different QAs and then trade-off analysis among them can be performed. An elaborate discussion of models provided in Section 3.4.1.

Feedback loops arrangements and multiple QAs. We compared results of RQ1.1 (i.e., addressed QAs) and RQ1.2 (i.e., feedback loop arrangements) to find out whether or not there is any correlation between arrangements of feedback loops and common QAs sets. Our analysis suggests that there is a correlation between methods using a single loop and performance and cost QAs set; and five papers fall in this category (i.e., S9, S14, S24, S29, and S45). This indicates that although some effort have been made to use multiple feedback loops, conventional feedback loop (i.e., a single loop) is still the most commonly used format, and performance and cost seems to be the most important QAs set which these feedback loops deal with.

Domain/development paradigm and multiple QAs. A comparison between results of RQ1.3, RQ1.4 (i.e., domain/development paradigm) and RQ1.1 (i.e., addressed QAs) shows that among the methods in domain of web-based systems, which has the highest frequency in included primary studies; performance (addressed in S20, S22, S23, S29, S30, S35, and S45) and cost (addressed in S22, S23, S29, and S45) are the most commonly addressed QAs.

Summary of answer to RQ2: The most widely used mechanisms/models to measure and quantify QAs sets are QA data variables. After QA data variables, utility functions and Markov chain models are the most common models which are also used for decision making process and selection of the best solution in presence of many alternatives. These models are frequently used to deal with performance and cost QAs in the system. Priority mechanisms are almost non-existent among current methods; utility functions and weighted utility functions are used to prioritize QAs in a few cases.

3.4. RQ3: How the decision making and tradeoff analysis of multiple QAs are performed at runtime?

We used data collected in [F11], [F13], [F18], [F19], and [F20] to answer this research question and get a better understanding of existing decision making and tradeoff analysis mechanisms. We are specifically interested in investigating runtime strategies for adaptation, arrangement formats of decision making units, used/developed tools and languages, the existence of evidence for quality assurances, and limitations and strengths of current methods.

3.4.1. RQ3.1: What runtime tactics/strategies are used to realize adaptation in systems dealing with multiple QAs?

Adopted from [9] adaptation strategies/tactics refer to a flow of actions over a sizable time-frame and scope, with intermediate decision points, to fix a type of system problem and meet quality related issues. Therefore, to answer this research question, we collected data about all the strategies/tactics performed in components of MAPE-K loop to find potential commonalities in different methods. Our results confirm that although all the existing adaptation methods consist of a sequence of steps which may be considered as representation of adaptations strategies, we observe that planning/execution in self-adaptation is done in an ad hoc manner, and there is a lack of standard specification of sequence of actions required for achieving system's quality goals.

By analyzing the collected strategies/tactics we observed that self-adaptive systems treat multiple QAs in two different types of approaches: proactive and reactive. In "proactive" self-adaptive systems, fulfillment of QAs, or QAs sets, are considered as a goal of adaptation, on the other hand, in "reactive" self-adaptive systems fulfilling certain QA is not necessarily the reason for triggering the adaptation. In "proactive" approaches, the managing part of the system continuously probes the managed system to identify QAs violations and then initiates the adaptation. However, in "reactive" self-adaptive systems, the system primarily aims at discovering and handling significant changes and failures, and then may consider adverse effects of adaptation on other QAs. Note that in most cases, the pro-activeness or reactiveness of the approaches are not explicitly discussed and it can only be inferred from the adaptation strategies.

3.4.2. RQ3.2: Is the decision making unit dealing with multiple QAs centralized or decentralized?

In Table 17, we categorized the primary studies based on the arrangement settings of the control modules, i.e., a single (i.e., centralized) or multiple (i.e., decentralized) control components for decision making. In a centralized method, one control component realizes the need for adaptation of a software system, makes adaptation decisions, and applied adaptation. But in a decentralized setting the system's runtime behavior emanates from the localized decisions made

Table 17

List of centralized and decentralized adaptation methods in primary studies.

Control module(s) arrangements setting	Paper identifiers	Number of papers	Number of common QAs sets
Centralized	S1, S3, S6, S7, S8, S10, S11, S21, S24, S25, S29, S33, S34, S36, S38, S41, S42, S46, S54	19	Performance, reliability (2) Performance, cost (1) Performance, cost, reliability (2)
Decentralized	S4, S12, S15, S19, S20, S23, S28, S37, S39, S51	10	Performance, availability (2)
Can be used for centralized and decentralized	S13, S14, S22, S45, S47	5	Performance, cost (2)
Not mentioned	S2, S5, S9, S16, S17, S18, S26, S27, S30, S31, S35, S40, S43, S44, S46 ^a , S48, S49, S50, S52, S53	20	–

^a Since S46 investigates two different frameworks (i.e., Rainbow and Zanshin), it is listed in both centralized and not mentioned categories; therefore, the sum of number of papers in this table exceeds the number of primary studies (i.e., 54 papers).

by multiple control components and their interactions. The arrangement style of the control modules varies based on the applicability of the style in that particular software system (e.g., cloud-based systems, service-oriented systems etc.). However, in some primary studies (e.g., S13) decentralization is the main objective of the presented framework which should be achieved to help to scale up the system. As listed in Table 17, in five primary studies the presented method is applicable in both centralized and decentralized settings, if required. In 10 primary studies, the presented methods are only applicable in decentralized format; and in 19 primary studies the presented methods only fit in centralized format. We could not derive the control modules arrangements settings for 20 primary studies as it was not clear from the description provided in the primary studies.

Regarding QAs, in general performance is the most commonly addressed QA regardless of control module arrangement setting. As for QAs sets, performance and reliability are more often handled in systems with centralized control modules, and performance and cost in systems supporting both centralized and decentralized control module arrangements. In systems with decentralized control modules performance and availability seems to be the most important quality attributes. This is sensible as in a system with distributed control modules, availability of all the distributed modules, which are responsible for supporting the self-adaptivity, potentially should have a higher priority over cost.

3.4.3. RQ3.3: What tools/languages are used/developed to support QAs tradeoff in existing methods?

Research question 3.3 aims at identifying two different categories of tools and languages. The first category refers to the "used" tools and languages; this category includes those technologies that already existed and were simply reused by these methods. The second category refers to the "developed" tools/languages that were created as part of the research done in the paper. Most of our identified tools and languages fall under the first category, as they were not specifically developed/created to deal with the "multiple QAs" aspect of the method; however, they have indirectly played a part in the presented method that deals with multiple QAs.

From 54 primary studies, 15 primary studies used or developed tools to support the presented method. PRISM, which is a probabilistic model checker tool, is the most commonly used (i.e., used in S15, S23, S25, and S39) tool among the primary studies. For a full list of identified tools, their short descriptions, type of tools, and related paper identifiers please see Appendix B. Note that for S18, it is not clear if the probabilistic model checker tool is in fact PRISM.

We classified all the identified tools to indicate how these tools support multiple QA tradeoffs (Table 18). The first category includes those tools that directly support multiple QAs tradeoff analysis. Generally speaking, tools listed in this category facilitate model checking, investigation of system configurations regarding quality properties of interest, and identification of quality violations. The second category of tools mainly includes the tools which are useful to obtain and analyze data variables/measurements of certain quality property in the system. The last category consists of general purpose tools, which have been used for different objectives and do not necessarily deal with multiple QA tradeoffs.

We compared results of this research question with RQ1.1 to identify correlations between commonly used tools (i.e., probabilistic model checkers) and addressed QAs sets. Our analysis indicates that there is no remarkable correlation between the type of tools and the addressed QAs sets. Performance and availability (S15), performance and cost (S23), and reliability and availability (S39) are the QAs sets addressed by the methods using probabilistic model checker tools.

Table 18
List of identified tools.

Category #	Tool categories	Name of tool	Paper identifiers
1	Directly support multiple qualities tradeoff analysis	- PRISM	S15, S18, S23,
		- XTEAM	S25, S39, 43, 51
2	Support obtaining the required quality measurements	- Groove tool	S22, S28, S19
		- Factory	
3	Tools not specifically dealing with quality attribute	- IBM's autonomic computing toolkit	S2, S3, S25, S26, S34, S39
		- Optimization, performance Evaluation and resource Allocator (OPERA) tool	
		- CMU's tailor	
		- MOFScript	
		- Configuration and deployment tool	
		- WEKA	
		- gocc	
		- Acme and AcmeStudio toolset	
		- DeSi	

In Table 19, we list all the identified languages, a short description for each language, and papers identifiers.

As indicated, the Stich language which is a language for architecture-based self-adaptation is the most commonly used language among the primary studies. It is worth mentioning that in Table 19, KLAPER is the only language that has been specifically developed to deal with quality properties analysis. The rest of the identified languages are general-purpose languages; thus they were not intended to directly deal with QAs tradeoff analysis (i.e., non-quality attribute). However, since all these languages were utilized in the selected primary studies, and to keep the integrity and consistency of the presented data, we decided to list all of the identified languages in Table 19 as well.

Table 19
List of identified languages.

Name	Short description	Paper identifiers
Stich	A language for representing repair strategies within the context of an architecture-based self-adaptation framework [10].	S14, S24, S29
Contextual query language	A formal language for representing queries to information retrieval systems such as search engines, and bibliographic catalogs.	S3
JGraLab API	A free Java class library that provides mathematical graph-theory objects and algorithms.	S8
Answer Set Programming (ASP)	A language for knowledge representation and declarative problem-solving, which has been developed in the field of non-monotonic reasoning and logic programming.	S31
KLAPER/D-KLAPER	An intermediate “kernel” language to support performance and reliability analysis of component-based systems based on model transformations.	S35
Adapt cases and the system and context modeling language	A UML-based domain specific modeling language.	S43
Common variability language	A domain independent language for specifying and resolving variability. It facilitates the specification and resolution of variability over any instance of any language defined using a MOF-based meta-model.	S54

3.4.4. RQ3.4: Does the method provide any evidence for validating the result of decision making mechanism regarding satisfaction of QAs after adaptation (i.e., assurances)?

From 54 primary studies, 24 primary studies deliver some sort of evidence to indicate that the system is capable of satisfying its stated functional and non-functional requirements during its operation in the presence of self-adaptation. We categorized the identified assurances approaches, the group type of the approaches, and paper identifiers in Table 20.

Summary of answer to RQ3: All the existing adaptation methods follow a step-wise format, which resembles the concept of strategies, to initiate and fulfill adaptation tasks. However, there are not many commonalities among these methods and the “strategies” are quite ad-hoc and problem specific. A variety of tools/languages are used or developed by the existing methods. The most widely used tools are PRISM and IBM's autonomic computing toolkit, and the most commonly used language in these methods is Stich. Less than half of the primary studies provide explicit assurance for their proposed approaches; hybrid approaches, and especially testing, are the most common type of approaches used in current methods for handling multiple QAs in self-adaptive systems.

4. Discussion

In this section, we provide a summary of the main findings, limitations to the review, and threats to validity.

4.1. Main findings

A wide range of QAs sets are addressed, but certain QAs sets are more emphasized. Although we were initially interested in assessing methods addressing certain QAs (namely performance, reliability, flexibility, maintainability, and availability), our results derived from analysis of data collected for research question 1.1 (i.e., Which QAs sets are addressed by these methods?) indicated that a wide range of QAs sets are discussed by existing methods. To be more specific, we identified 23 QAs in the primary studies; however, current methods seem to focus on a particular set of QAs (i.e., performance, reliability, cost, availability and scalability). Nonetheless, there is much space for improvement as many significant aspects (e.g., prioritization, modeling, and preemption) for quality management are neglected in current methods. To tackle these shortcomings, one can focus on a limited number of QAs (e.g., two QAs) to minimize the solution space, and tackle the problem from different angles in depth.

Multifaceted use of “models” to deal with multiple QAs. Our results derived from analyzing the data collected to answer RQ2.1 (i.e., What models are used to represent and analyze multiple QAs

Table 20
List of identified approaches for assurances.

Group	Approaches	Papers identifiers
Human-driven approaches	- Formal proof	S53
	- Simulation	S6, S7, S19, S22, S23, S26, S29, S30, S37
System-driven approaches	- Runtime verification	S3, S35, S52
	- Sanity checks	S17, S53
Hybrid approaches	- Model checking	S7, S18, S23, S27, S43, S3, S4, S6, S25, S40, S42, S44, S49, S50
	- Testing	

characteristics?) indicate that current self-adaptive methods use “models” in two different formats in order to deal with multiple QAs. The first one focuses on investigating effects of adaptation on characteristics of a certain QA through modeling data variables of the quality and detecting threshold violations. The second format explores the QAs sets in a bigger picture. In these modeling formats, the focus is on multiple QAs’ changes at different states of the system at runtime and serving the decision making process to find the best possible solutions. One interesting approach may be to combine the two formats of modeling. This would result in a thorough analysis of the system, and would not only help to explore quality data variables, but also inspect changes of a variety of QAs, and QAs sets, at different states of the system at runtime.

“Proactive” versus “Reactive” treatment of multiple QAs. Analysis of data collected to answer research question 3.1 (i.e., what tactics/strategies are used to realize adaptation in the system at runtime?) indicate that self-adaptive systems are designed to treat multiple QAs through two different types of approaches. In “proactive” approaches, the managing part of the self-adaptive system continuously probes the managed system in search of QAs violations, or utility declines and if any sign is identified, the system initiates the adaptation actions. In “proactive” self-adaptive systems, QAs are considered as goals of adaptation and are viewed as first class citizens. On the other hand, in “reactive” self-adaptive systems the managing part of the system monitors the managed system to identify significant changes, failures, and faults, which trigger the need for adaptation. In “reactive” self-adaptive systems, fulfilling certain set of QAs is not necessarily the reason prompting the self-adaptation. When discovered, the system primarily aims at fixing the problem, which may be a violation of a functional requirement, and then may consider adverse effects of self-adaptation on QA sets and take action. In these systems, fulfilling the QAs, or QAs sets, is not of primary concern, and changes on QA sets are considered as side effects of application of adaptation.

Preemption is hardly used. The first step to support preemption in a self-adaptive system is to monitor affected QAs sets continuously. During the application of adaptation, the system should explore new events, investigate their effects on all the QAs of interest, and adjust the decision making process to handle the most important QA first. However, our results, and in particular the data collected to answer the research question 2.2 (i.e., How do these methods address multiple QA tradeoffs?), indicate that mechanisms for investigating the priority of QAs, among multiple QAs of the system, and consequently selecting the order in which they should be treated are not widely used and therefore an essential prerequisite for supporting preemption is absent.

Most styles of feedback loops are aimed at dealing with sets of two QAs. Analysis of data collected to answer research question 1.2 (i.e., How many feedback loops are used in current methods and how are they arranged to handle multiple QAs?) indicates that the conventional application style of feedback loop (i.e., one feedback loop) is extensively used to handle the common QAs in the domain. Two feedback loops and layered feedback loops styles are used to deal with performance, reliability, and availability. The style with three feedback loops is used to handle performance and cost. Our results also indicate that, probably due to their novelty; the less common styles (e.g., application of different types of multiple feedback loops) do not address an extensive number of QAs.

Lack of standardized adaptation strategies/tactics schemes for MAPE-K functionalities. As proposed by Cheng et al. in [9], adaptation strategies/tactics refers to a flow of actions over a sizable time frame and scope, with intermediate decision points, to fix a type of

system problem and meet quality related issues. Data analysis for research question 3.1 (i.e., What tactics/strategies are used to realize adaptation in the system at runtime?) indicated that despite the fact that the concept of adaptation strategies/tactics is widely accepted, not many methods comply with use of clear adaptation strategies/tactics schemes when it comes to application of adaptation in self-adaptive systems handling multiple QAs. Although, every adaptation method actually consists of multiple steps, which may be mapped into strategies, the strategies are not tangible and are often ad hoc, and this lack of a systematic framework for performing adaptation actions towards a specific goal can be observed in roughly all of the MAPE-K functionalities.

4.2. Dimensions of realizing multiple QAs in self-adaptive systems

Based on the results of this systematic review, we propose a list of dimensions for realizing multiple QAs in self-adaptive systems. This list consists of a variety of dimensions that may be addressed when dealing with multiple QAs in self-adaptive systems. The presented dimensions refer to the most significant aspects of dealing with multiple QAs. We derived these dimensions from different phases of the studied approaches, from problem specification and diagnoses, to applying adaptation solutions. Therefore, by explicitly listing the dimensions, we aim at emphasizing the importance of these dimensions, and the fact that addressing them helps to deal with multiple QAs more effectively. Our identified dimensions are comparable to those of Andersson et al. [2] as following: QAs specification can be mapped into Goals; QAs monitoring and treatment as well as the handling can be mapped into Mechanisms, and QAs guarantees corresponds to Effects. Our dimensions however specifically focus on QAs during and after adaptation.

This classification of dimensions is certainly not exhaustive, but could be used as a basis for researchers who are interested in improving and optimizing current self-adaptation methods, or those who are willing to propose new efficient solutions in a more structured and comprehensible manner.

We mapped the identified dimensions into four different categories: QA specification, QA monitoring and treatment, QAs handling mechanism, and QAs guarantees (see Table 21).

Table 21
Dimensions of realization of multiple QAs in self-adaptive systems.

Dimension	Description
QAs specification	
Goal/side effect	Fulfilling a QA may be the main goal of the adaptation, or the QA may be affected due to application of adaptation.
QAs monitoring and treatment	
Proactive or reactive	A method may handle a QA in proactive (i.e., QA are considered as goals of adaptation) or reactive (i.e., fulfilling certain QA is not necessarily the reason prompting the self-adaptation) way.
Prioritization	Determines the order in which the QAs are dealt with.
Preference	While dealing with one QA, the system should recognize other violations, and take actions.
QAs handling mechanism	
Modeling type	Models may be used to present different characteristics of a QA, or may be used to indicate how QAs are affected in different states of system life time.
Exclusivity of models and/or feedback loops to concerns	A model or a feedback loop is designated to one specific QA or a particular set of QAs.
QAs guarantees	
Human driven, system-driven, hybrid	The ability to deliver evidence that the system satisfies its stated non-functional requirements during its operation in the presence of self-adaptation.

The purpose of the first category (i.e., QA specification) is to specify the level of importance of certain QAs in a self-adaptive system. Often in systems with multiple QAs, changes in one or more of the QAs (i.e., goal) trigger the need for adaptation; however, the adaptation itself affects several other QAs (i.e., side effect), and those side effects are rarely handled. For instance, if fulfillment of a particular QA is the main goal of the adaptation, then it should be treated differently (e.g., assurances should be provided) compared to the rest of the QAs in the system. In many cases researchers present a long list of QAs which should be handled, and the proposed method only deals with a few of them (i.e., goal QAs), which may decrease the credibility of the method. Therefore, while dealing with multiple QAs, it is useful to clarify the role of each QA in adaptation scenarios to help to increase the validity of a method.

The purpose of the second category (i.e., QAs monitoring and treatment) is to represent different factors that should be present when building a strategy for monitoring and treating multiple QAs. By taking these important aspects into account, researchers can build more structured and rigorous adaptation strategies. The first factor (i.e., proactive or reactive) determines how the self-adaptive system monitors and treats the QAs sets, and whether or not the QAs are considered as first class citizens in the system. The second factor (i.e., prioritization) indicates how the self-adaptive system treats QAs in case of multiple quality violations, and prescribes that the order in which the QAs are dealt with should be specified. Finally, the last factor refers to parallel monitoring and treatment of multiple QAs. Meaning, while the system is dealing with one QA, it should simultaneously identify other violations, and if required, take actions immediately.

This category (i.e., QAs handling mechanisms) focuses on the mechanism that deals with the QAs. To be more specific, this category indicates that a self-adaptation mechanism may include different formats for modeling QAs sets, and/or may prescribe certain formats for using the model, as well as the feedback loops. Clearly, researchers may select different approaches regarding modeling and use of feedback loops when creating the adaptation mechanisms; but the rationale for the decisions should be expressed. Explicitly mentioning the decisions and the rationale behind them will help other researchers and practitioners to better understand their proposed method, and apply them in their other circumstances.

Finally, the last category refers to the ability of the system to indicate it is capable of satisfying its set of QAs as expected during and after application of adaptation. Depending on the nature of the approaches, they are categorized in three different types. Listed approaches to provide guarantee is a representative of assurances approaches that have been developed through the years is adopted from [33].

4.3. Limitations to the review and threats to validity

Domain of the study. Creating a search string for the automatic search phase was a challenging task due to lack of a well-defined terminology in the domain of architecture-based self-adaptive systems. To overcome this issue and assure the credibility of the search string, we followed a few steps. First, we created five possible variations for the search string. Then, we performed a pilot search for each of the created search strings on a set of selected venues. Based on the number of returned results, we excluded strings returning extremely high or low number of primary studies. Next, we altered the remaining search strings and performed the previous step to find the string with the most optimal result. Finally, we established a “quasi-gold” standard and crosschecked the automatic search results of a selected num-

ber of venues with the manual search results of the same set of venues to make sure the string works properly.

Modification of data extraction form. After performing data extraction for a specific number of primary studies, we noticed that the data extraction form need to be improved. Therefore, we modified our primary data extraction form to make the form efficient. First, we added structured answers for a few of the data fields (i.e., limitations and strengths, and assurances) to mitigate the difficulty of data extraction, and data analysis for these particular data fields. Second, we added a new data field (i.e., languages) to extract additional data from the primary studies. Since we only altered the format of extracted data for certain data fields (i.e., limitations and strengths, and assurances), and extracted supplementary data to investigate a new aspect of the studied methods, these modifications do not impose any threats to the validity of the systematic review.

Modification of research questions. After finishing data extraction, we realized a major part of the data collected for one of the sub research questions of RQ1 (i.e., “What artifacts are produced by these methods?”) is highly heterogeneous and difficult to analyze. Besides, the collected data mostly did not provide any additional value regarding handling of multiple QAs in architecture-based self-adaptive systems. We noticed the only useful part of extracted data concerns the use of models to deal with QAs. Therefore, we decided to remove the sub research question, and in order to still be able to make use of the collected data, merge the useful part of data we collected for this sub research question to the results of the most relevant research question (i.e., “What models are used to represent and analyze multiple QAs characteristics?”) and perform then analysis.

5. Conclusion and future work

We performed a systematic literature review to study existing architecture-based methods for handling multiple QAs in self-adaptive systems. We automatically searched 28 venues to collect our primary studies. Results of our study suggest that performance, reliability and cost are considered as the most important QAs in self-adaptive systems, and are frequently addressed by current methods. Current architecture-based self-adaptive methods mainly employ a single feedback loop based on MAPE-K to handle the managed system; however, there have been some efforts to combine loops and use separate feedback loops to exclusively handle certain concerns. Our results suggest that QA data variables and utility functions are widely used to model QA's characteristics, and to represent the tradeoffs between multiple QAs in the entire system or certain system component. Our results also show that testing is the most common approach for providing assurances in current methods.

Our quality assessment results for the primary studies confirm that the selected studies are of a high quality of reporting. However, most of the primary studies lack investigation of credibility and limitations of the methods.

Based on our main findings listed in Section 4.1, we point a few interesting opportunities for future work: (a) our results indicate that although current approaches address several different QAs, most of them do not elaborate on how they are handled and if the QAs are guaranteed to be improved after adaptation; (b) the use of models in architecture-based self-adaptive systems may be enhanced to include two different aspects of multiple QAs at the same time: (i) QA data variables, and also (ii) a reflection on QAs changes in different states of the system. Combination of both aspects in models helps to perform the tradeoff analysis more efficiently; (c) further research is required to improve tradeoff analysis. This improvement may concern monitoring and treatment of QAs (i.e., proactive, and reactive) and

Table 22

List of included primary studies.

Paper#	Author(s)	Year	Title
S1	Heaven William, Sykes Daniel, Magee Jeff, Kramer Jeff	2009	A case study in goal-driven architectural adaptation
S2	Valetto, Giuseppe, Kaiser, Gail	2002	A case study in software adaptation
S3	S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo and A. Mamelli, and G.A. Papadopoulos	2012	A development framework and methodology for self-adapting applications in ubiquitous computing environments
S4	Mohamed Zouari, Maria-Teresa Segarra, Françoise André	2010	A framework for distributed management of dynamic self-adaptation in heterogeneous environments
S5	T. Vogel, H. Giese	2012	A language for feedback loops in self-adaptive systems: executable runtime megamodels
S6	Naeem Esfahani, Ahmed Elkhodary, and Sam Malek	2013	A learning-based framework for engineering feature-oriented self-adaptive software systems
S7	Matthias Tichy, Holger Giese	2004	A self-optimizing run-time architecture for configurable dependability of services
S8	Mehdi Amouia, Mahdi Derakhshanmaneshb, Jurgen Eberth, Ladan Tahvildaria	2012	Achieving dynamic adaptation via management and interpretation of runtime models
S9	Luckey Markus, Nagel Benjamin, Gerth Christian, Engels Gregor	2011	Adapt cases: extending use cases for adaptive systems
S10	Thomas Vogel, Holger Giese	2010	Adaptation and abstract runtime models
S11	hang-Wen Cheng, An-Cheng Huang, David Garlan, Bradley Schmerl, Peter Steenkiste	2004	An architecture for coordinating multiple self-management systems
S12	Faniyi Funmilade, Lewis Peter R, Bahsoon Rami, Yao Xin	2014	Architecting self-aware software systems
S13	Paulo Casanova, Bradley Schmerl, David Garlan, and Rui Abreu	2011	Architecture-based run-time fault diagnosis
S14	Shang-Wen Cheng, David Garlan, Bradley Schmerl	2006	Architecture-based self-adaptation in the presence of multiple objectives
S15	George Edwards, Joshua Garcia, Hossein Tajalli, Daniel Popescu, Nenad Medvidovic, Gaurav Sukhatme, Brad Petrus	2009	Architecture-driven self-adaptation and self-management in robotics systems
S16	Anthony Richard J, Pelc Mariusz, Byrski Witold	2010	Context-aware reconfiguration of autonomic managers in real-time control applications
S17	Dorn Christoph, Taylor Richard N	2013	Coupling software architecture and human architecture for collaboration-aware system adaptation
S18	Carlo Ghezzi, Amir Molzam Sharifloo	2013	Dealing with non-functional requirements for adaptive systems via dynamic software product-lines
S19	Zoghi Parisa, Shtern Mark, Litoiu Marin	2014	Designing search based adaptive systems: a quantitative approach
S20	Paulo Casanova, David Garlan, Bradley Schmerl, Rui Abreu	2013	Diagnosing architectural run-time failures
S21	Norha Villegas, Gabriel Tamura, Hausi Müller, Laurence Duchien, Rubby Casallas	2013	DYNAMICO: a reference model for governing control objectives and context relevance in self-adaptive software systems

Table 22 (Continued)

Paper#	Author(s)	Year	Title
S22	Shang-Wen Cheng, David Garlan, Bradley Schmerl	2009	Evaluating the effectiveness of the rainbow self-adaptive system
S23	Javier Cámara, Rogério de Lemos	2012	Evaluation of resilience in self-adaptive systems using probabilistic model-checking
S24	Javier Camara, Pedro Correia, Rogério de Lemos, David Garlan, Pedro Gomes, Bradley Schmerl and Rafael Ventura	2013	Evolving an adaptive industrial software system to use architecture-based self-adaptation
S25	Ahmed Elkhodary, Naeem Esfahani, and Sam Malek	2010	FUSION: a framework for engineering self-tuning self-adaptive software systems
S26	Hiroyuki Nakagawa, Akihiko Ohsuga, Shinichi Honiden	2011	gocc: a configuration compiler for self-adaptive systems using goal-oriented requirements description
S27	Luckey Markus, Engels Gregor	2013	High-quality specification of self-adaptive software systems
S28	Yan Liu, Ian Gorton	2007	Implementing adaptive performance management in server applications
S29	Shang-Wen Cheng, Vahe V Poladian, David Garlan, Bradley Schmerl	2008	Improving architecture-based self-adaptation through resource prediction
S30	Gabriel Tamura, Norha M. Villegas, Hausi A. Muller, Laurence Duchien, Lionel Seinturier	2013	Improving context-awareness in self-adaptation using the DYNAMICO reference model
S31	Daniel Sykes, Domenico Corapi, Jeff Magee, Katsumi Inoue, Jeff Kramer, Alessandra Russo	2013	Learning revised models for planning in adaptive systems
S32	Regina Hebig, Holger Giese and Basil Becker	2010	Making control loops explicit when architecting self-adaptive systems
S33	Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli	2013	Managing non-functional uncertainty via model-driven adaptivity
S34	David Garlan, Bradley Schmerl	2002	Model-based adaptation for self-healing systems
S35	Vincenzo Grassi, Raffaella Mirandola, Enrico Randazzo	2009	Model-driven assessment of QoS-aware self-adaptation
S36	Thomas Vogel, Holger Giese	2014	Model-driven engineering of self-adaptive software with EUREMA
S37	Franck Chauvel, Nicolas Ferry, Brice Morin, Alessandro Rossini, and Amor Solberg	2013	Models@runtime to support the iterative and continuous design of autonomic reasoners
S38	Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz	2009	MUSIC: middleware support for self-adaptation in ubiquitous and service-oriented environments
S39	Danny Weyns, Sam Malek, and Jesper Andersson	2010	On decentralized self-adaptation: lessons from the trenches and challenges for the future
S40	Diego Perez-Palacin, Raffaella Mirandola, and José Merseguer	2014	On the relationships between QoS and software adaptability at the architectural level
S41	Kaustubh R. Joshi, Matti Hiltunen, Richard Schlichting, William H. Sanders, and Adnan Agbaria	2004	Online model-based adaptation for optimizing performance and dependability

Table 22 (Continued)

Paper#	Author(s)	Year	Title
S42	Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola	2009	QoS-driven runtime adaptation of service oriented architectures
S43	Markus Luckey, Christian Gerth, Christian Soltenborn, Gregor Engels	2011	QUAASY – Quality Assurance of Adaptive SYstems
S44	Jie Yang, Gang Huang, Wenhui Zhu, Xiaofeng Cui, and Hong Mei	2009	Quality attribute tradeoff through adaptive architectures at runtime
S45	David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste	2004	Rainbow: architecture-based self-adaptation with reusable infrastructure
S46	Konstantinos Angelopoulos, Vitor E. Silva Souza, and João Pimentel	2013	Requirements and architectural approaches to adaptive software systems: a comparative study
S47	Habtmu Abie, Reijo M. Savola, and Illesh Dattani	2009	Robust, secure, self-adaptive and resilient messaging middleware for business critical systems
S48	Svein Hallsteinsen, Erlend Stav, and Jacqueline Floch	2004	Self-adaptation for everyday systems
S49	R. Asadollahi, M. Salehie, L. Tahvildari	2009	StarMX: a framework for developing self-managing Java-based systems
S50	Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek	2011	Taming uncertainty in self-adaptive software
S51	Mohamed Zouari, Ismael Bouassida Rodriguez	2013	Towards automated deployment of distributed adaptation systems
S52	Erik M. Fredericks, Byron DeVries, Betty H. C. Cheng	2014	Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty
S53	Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, Raffaela Mirandola	2009	Towards self-adaptation for dependable service-oriented systems
S54	Amanda S. Nascimento, Cecília M.F. Rubira, and Fernando Castor	2013	Using CVL to support self-adaptation of fault-tolerant service compositions

incorporating preemption in tradeoff analysis in self-adaptive systems with multiple QAs; (d) further work is required to (i) identify and investigate ad hoc adaptation strategies, (ii) propose consistent and standardized strategies, as well as, a systematic framework for performing adaptation actions for each of MAPE-K functionalities.

Uncited references

[8, 11, 13, 14, 16–18, 20, 23, 35].

Acknowledgments

Authors would like to thank Laurence de Jong for his contribution to this study.

Appendix A

Appendix B

Table 23

Full list of identified tools.

Name of tool	Short description	Type of tool	Paper identifiers
PRISM	A tool for formal modeling and verification of systems that exhibit stochastic behavior.	Probabilistic model checker tools	S15, S18, S23, S25, S39
IBM's autonomic computing toolkit.	A collection of self-managing autonomic technologies, tools, scenarios, and documentation that is designed for users wanting to learn, adapt, and develop autonomic behavior in their products and systems ^a .	Autonomic computing toolkit	S22, S28
CMU's Tailor	Helps the decision making process by reacting to the differences between the running and the nominal architecture found by gauges.	Architecture transformation tool	S2
Eclipse EMF, Papyrus UML tool, UML2JavaTransformation tool, OWL2JavaTransformation	EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor ^b . Code generating tools transform the model into source code.	Modeling frameworks and code generating tools	S3
MOFScript	MOFScript is a tool for model to text transformation, e.g., to support generation of implementation code or documentation from models ^c .	Model to text transformation tools	S3
N/A	The tool makes a set of choices with regard to the variation points contained in the framework (number and placement of context managers and adaptation managers, connections among them, optional components, policies) and the adaptation system deployment.	A configuration and deployment tool	S4

Table 23 (Continued)

Name of tool	Short description	Type of tool	Paper identifiers
Optimization, Performance Evaluation and Resource Allocator (OPERA) tool	The OPERA is a layered queueing model used to evaluate the performance of web applications deployed on arbitrary infrastructures.	Performance tool	S19
Jmeter testing tool	It is a Java application designed to load test functional behavior and measure performance ^d .	Testing tools	S22
WEKA	Contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes ^e .	Data mining tool	S25
eXtensible Tool-chain for Evaluation of Architectural Models (XTEAM)	Implements a model-driven engineering (MDE) approach to software architecture that combines extensible modeling languages based on architectural constructs with a model interpreter framework that enables rapid implementation of customized dynamic analyses at the architectural level ^f .	Architectural models checking tool	S25
gocc: A Configuration Compiler for Self-adaptive Systems Using Goal-oriented Requirements Description	An architectural compiler for self-adaptive systems, which generates architectural configurations from the goal-oriented requirements descriptions.	Architectural compiler for self-adaptive systems	S26
Acme and AcmeStudio toolset	A generic software architecture description language (ADL) that can be used as a common interchange format for architecture design tools and/or as a foundation for developing new architectural design and analysis tools ^g .	Architecture description language (ADL) and architectural analysis tool	S34

Table 23 (Continued)

Name of tool	Short description	Type of tool	Paper identifiers
DeSi	An environment that supports flexible and tailorable specification, manipulation, visualization, and (re)estimation of deployment architectures for large-scale, highly distributed systems [25].	System deployment representing tool	S39
Groove tool	It is a tool for model checking of object-oriented systems based on graph transformations [29].	State space generation tool	S43
Factory	A tool that performs the deployment tasks. The factory processes the appropriate system configuration using a graph grammar-based approach and then sets up the system [39].	System adaptation customizing tool	S51

^a <http://www.ibm.com/developerworks/autonomic/r3/overview.html>.

^b <https://eclipse.org/modeling/emf/>.

^c <http://www.eclipse.org/gmt/mofscript/>.

^d <http://jmeter.apache.org/>.

^e <http://www.cs.waikato.ac.nz/ml/weka/>.

^f <http://softarch.usc.edu/gedwards/xtteam.html>.

^g <http://www.cs.cmu.edu/acme/>.

Table 24

List of identified QAs in the primary studies.

QA	Paper identifiers	QA	Paper identifiers
Accuracy	S32, S38	Manageability	S2
Accountability	S6	Modifiability	S48
Content fidelity	S22	Robustness	S37
Dependability	S35	Reusability	S51
Efficiency	S51	Safety	S52
Extensibility	S10	Security	S11
Flexibility	S16	Stability	S16
Interoperability	S47	Usability	S32, S48

Appendix C

Appendix D

Table 25

QAs sets for the identified QAs in the primary studies.

QAs sets	Paper identifiers	QAs sets	Paper identifiers
Response time, reliability, quote quality, and accountability	S6	<i>Performance and dependability</i>	S35
Scalability, manageability, and reliability	S2	<i>Robustness, efficiency and performance</i>	S37
Reusability and extensibility	S10	Accuracy, reliability, cost, and power consumption	S38
Performance and security	S11	Scalability, performance, security, and interoperability	S47
Performance, stability, flexibility, and responsiveness	S16	Modifiability, usability, and performance	S48
Performance, cost, and content fidelity	S22	Efficiency, robustness, scalability, reusability, data availability, and response time	S51
<i>Performance, accuracy, usability</i>	S31	Efficiency and safety	S52

Table 26

Identified technical limitations methods.

Brief description of technical limitation	Paper identifiers	Brief description of technical limitation	Paper identifiers
The utility functions only consider the utility of components regardless of their relations and interactions.	S1	ADAM supports parallelism only in the implementation methods and not at the Activity Diagram level.	S33
Ad hoc feedback loop which depend on the manually constructed gauge rules.	S2	Externalized repair seem to introduce some significant latency.	S34
The framework has a steep learning curve. Moreover, the current implementation of the middleware offers only basic support for security.	S3	State-space explosion when generating an analysis model from the intermediate model.	S35
Developing adaptation rules is not easy and requires knowledge about the adaptable software.	S8	EUREMA does not support the concurrent execution of interdependent feedback loops.	S36
For elaborate description of limitations see [15].	S11	SLA is not currently supported.	S38
For elaborate description of limitations see [6].	S13	Availability of partial knowledge, uncertainty, coordination and communication overhead.	S39
For elaborate description of limitations see [9].	S14	For elaborate description of limitations see [27].	S40
It is mentioned that the adaptation of human architecture/structure is not discussed.	S17	State-space explosion.	S41
Evaluation experiment is based on simulation; it is not realistic.	S19	The approach is not always the best solution for all QAs.	S44
No support for hierarchical structures.	S20	Rainbow is centralized; however, it can be applied in a distributed setting. Nevertheless, the coordination and other distributed computing issues present a challenge for future research.	S45
To realize multiple control loops, it is assumed that components can be executed concurrently / The approach is based on the goal-oriented requirements description. The sub-goals should be manually refined to derive all features from the requirements.	S26	Runtime overhead imposed by the adaptation middleware.	S48
For elaborate description of limitations see [31].	S30	It is not possible to manage the framework and its properties dynamically.	S49
For the sake of simplicity, some technical limitations are enforced.	S31	A high number of variables (representing abstract services and concrete services) is a potential hurdle to the optimization problem.	S53

References

- [1] M.S. Ali, M. Ali Babar, L. Chen, K.-J. Stol, A systematic review of comparative evidence of aspect-oriented programming, *Inf. Software Technol.* 52 (9) (2010) 871–887. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0950584910000819>.
- [2] J. Andersson, R. Lemos, S. Malek, D. Weyns, Modeling dimensions of self-adaptive software systems, in: B.H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science, 5525, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 27–47. http://dx.doi.org/10.1007/978-3-642-02161-9_2.
- [3] V.R. Basili, G. Caldiera, H.D. Rombach, Goal question metric paradigm, *Encyclopedia of Software Engineering*, Wiley & Sons, 1994:528–532.
- [4] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, third ed., Addison-Wesley Professional, 2012.
- [5] R. Calinescu, Emerging techniques for the engineering of self-adaptive high-integrity software, *Assurances for Self-Adaptive Systems*, 2013:297–310, Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-36249-1_11.
- [6] P. Casanova, B. Schmerl, D. Garlan, R. Abreu, Architecture-Based Run-Time Fault Diagnosis, in: I. Crnkovic, V. Gruhn, M. Book (Eds.), *Software Architecture*, Springer, Berlin Heidelberg, 2011, pp. 261–277. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-23798-0_29.
- [7] B.H.C. Cheng, K.I. Eder, M. Gogolla, L. Grunske, M. Litoiu, Using Models at Runtime to Address Assurance for Self-Adaptive Systems, 2014:101–136.
- [8] .
- [9] S.-W. Cheng, D. Garlan, B. Schmerl, Architecture-based self-adaptation in the presence of multiple objectives, In: *Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems - SEAMS'06*, 2, 2006: <http://dx.doi.org/10.1145/1137677.1137679>.
- [10] S.-W. Cheng, D. Garlan, A language for architecture-based self-adaptation, *J. Syst. Software* 85 (12) (2012) 2860–2875. <http://dx.doi.org/10.1016/j.jss.2012.02.060>.
- [11] Computing, A., Paper, W., & Edition, T. (2005). *An architectural blueprint for autonomic computing*. June.
- [12] T. Dingsoyr, T. Dyba, Empirical studies of agile software development: a systematic review, *Inf. Software Technol.* 50 (2008) 833–859, <http://dx.doi.org/10.1016/j.infsof.2008.01.006>.
- [13] .
- [14] .
- [15] D. Garlan, B. Schmerl, P. Steenkiste, Rainbow: architecture-based self-adaptation with reusable infrastructure, In: *International Conference on Autonomic Computing*, 2004. *Proceedings*, IEEE, 2004, pp. 276–277, <http://dx.doi.org/10.1109/ICAC.2004.1301377>.
- [16] .
- [17] .
- [18] ISO/IEC 25012:2008 Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model. (n.d.). Retrieved May 29, 2014, from http://www.iso.org/iso/catalogue_detail.htm?csnumber=35736.
- [19] ISO/IEC 9126-1 Software Eng. - Product quality - Part 1: quality model, Int. Standard Organization, 2001.
- [20] .
- [21] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Retrieved from <http://www.citeulike.org/group/14013/article/7874938>, 2007.
- [22] R. De Lemos, H. Giese, H. Müller, Software engineering for self-adaptive systems: a second research roadmap, *Software Engineering for Self-Adaptive Systems*, 2013:1–32, Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-35813-5_1.
- [23] .
- [24] S. Mahdavi-Hezavehi, M. Galster, P. Avgeriou, Variability in quality attributes of service-based software systems: a systematic literature review, *Inf. Software Technol.* 55 (2) (2013) 320–343. <http://dx.doi.org/10.1016/j.infsof.2012.08.010>.
- [25] Mikic-rakic, M., Malek, S., Beckman, N., & Medvidovic, N. (2004). A tailorable environment for assessing the quality of deployment architectures in highly distributed settings.
- [26] T. Patikirikoral, A. Colman, J. Han, L. Wang, A systematic survey on the design of self-adaptive software systems using control engineering approaches, In: *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012, pp. 33–42, <http://dx.doi.org/10.1109/SEAMS.2012.6224389>.
- [27] D. Perez-Palacin, R. Mirandola, J. Merseguer, On the relationships between QoS and software adaptability at the architectural level, *J. Syst. Software* 87 (2014) 1–17, <http://dx.doi.org/10.1016/j.jss.2013.07.053>.
- [28] Raheja, R., Cheng, S., Garlan, D., & Schmerl, B. (2010). *Improving architecture-based self-adaptation using preemption*. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-14412-7_2.
- [29] Rensink, A. (2004). The GROOVE simulator: a tool for state space generation, 479–485.
- [30] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, *ACM Trans. Auton. Adapt. Syst.* 4 (2) (2009) 14:1–14:42, <http://dx.doi.org/10.1145/1516533.1516538>.
- [31] Tamura, G., Villegas, N.M., Müller, H.A., Duchien, L., & Seinturier, L. (2013). Improving context-awareness in self-adaptation using the DYNAMICO reference model, 153–162. Retrieved from <http://dl.acm.org/citation.cfm?id=2487336.2487361>.
- [32] D. Weyns, T. Ahmad, Claims and evidence for architecture-based self-adaptation: a systematic literature review, *Software Architecture*, 2013, Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-39031-9_22.
- [33] D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, G. Tamburrelli, Perpetual assurances in self-adaptive systems, In: *Assurances for Self-Adaptive Systems*, Dagstuhl Seminar 13511, 2014.
- [34] D. Weyns, M.U. Iftikhar, D. Gil, D. Iglesia, T. Ahmad, A survey of formal methods in self-adaptive systems categories and subject descriptors, In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, 2012, pp. 67–79. 978-1-4503-1084-0/0/12/06.
- [35] .
- [36] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, K.M. Göschka, On patterns for decentralized control in self-adaptive systems, in: R. de Lemos, H. Giese, H.A. Müller, M. Shaw (Eds.), *Software Engineering for Self-Adaptive Systems II*, Springer, Berlin, Heidelberg, 2013, pp. 76–107. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-35813-5_4.
- [37] Z. Yang, Z. Li, Z. Jin, Y. Chen, A systematic literature review of requirements modeling and analysis for self-adaptive systems, In: *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, LNCS, 8396, 2014, pp. 55–71, http://dx.doi.org/10.1007/978-3-319-05843-6_5.
- [38] H. Zhang, Ali Babar, On Searching Relevant Studies in Software Engineering, British Informatics Society Ltd, 2010.
- [39] M. Zouari, I. Bouassida, R. Towards, A. Deployment, Towards Automated Deployment of Distributed Adaptation Systems, 2013. Mohamed Zouari, Ismael Bouassida Rodriguez, to cite this version: Adaptation Systems.