
Matthias Becker
Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn
Zukunftsmeele 1
33102 Paderborn, Germany
matthias.becker@uni-paderborn.de

Markus Luckey
Department of Computer Science, University of Paderborn
Zukunftsmeele 1
33102 Paderborn, Germany
markus.luckey@uni-paderborn.de

Steffen Becker
Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn
Zukunftsmeele 1
33102 Paderborn, Germany
steffen.becker@uni-paderborn.de

ABSTRACT
To meet quality-of-service requirements in changing environments, modern software systems adapt themselves. The structure, and correspondingly the behavior, of these systems undergoes continuous change. Model-driven performance engineering, however, assumes static system structures, behavior, and deployment. Hence, self-adaptive systems pose new challenges to model-driven performance engineering. There are a few surveys on self-adaptive systems, performance engineering, and the combination of both in the literature. In contrast to existing work, here we focus on model-driven performance analysis approaches. Based on a systematic literature review, we present a classification, identify open issues, and outline further research.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics—Performance measures; D.2.11 [Software Engineering]: Software Architectures—Languages

General Terms
Design, Performance

Keywords
software performance, model-driven performance engineering, self-adaptation, self.*

1. INTRODUCTION
Modern business information systems run in highly dynamic environments. Dynamics range from unpredictably changing numbers of concurrent users asking for service to virtualized infrastructure environment with unknown load caused in neighboring virtual machines or varying response times of required external services. Despite this dynamics, these systems are expected to fulfill their performance requirements. Designers achieved this in the past by overprovisioning of hardware, which is, however, neither cost-effective nor energy preserving. Self-adaptation is one primary means developed over the last years to cope with these challenges. The ideas is that systems always react to their dynamic environment by restructuring their components and connectors, exchanging components or services, or altering their hardware infrastructure. The most prominent use case for the latter is autonomous replication of the business logic on new (virtualized) hardware nodes in combination with self-adaptive load balancing as offered in Infrastructure-as-a-Service (IaaS) clouds [1].

To deal with performance requirements of classical, non-adaptive systems, researches have developed model-based and model-driven performance engineering approaches [10]. These approaches allow early design-time performance evaluations based on system models to validate performance requirements or answer hardware sizing questions. However, classical performance engineering approaches use static system models (e.g., UML2 models) which do not support an explicit adaptation viewpoint. However, as this viewpoint models the self-adaptive behavior of today’s systems, we cannot apply classical performance engineering approaches without extensions. This raises the question how and to which extent newly developed approaches for model-driven performance engineering of self-adaptive system take this novel viewpoint into account.

There are already some literature surveys on modeling self-
adaptive systems, classical performance engineering, and even a few on performance engineering in self-adaptive environments partially answering our question. However, we focus on model-driven approaches as we aim for full automation and design-time questions, e.g., will the system’s adaptation rules ensure stability.

To address the raised question, we performed a systematic literature review on model-driven performance engineering approaches for self-adaptive systems. We highlight the essential ideas of these approaches and classify them according to several dimensions identified during our study. Using the results of the survey, we identify missing aspects and highlight future research directions.

The contribution of this paper is a systematic literature review of the current state-of-the-art in performance engineering for self-adaptive systems. In our survey, we restrict our focus to model-driven approaches, i.e., approaches with full automation. This also implies that we restrict our survey to approaches which deal with explicitly modeled adaptation rules. As application domain, we restrict our survey to business information systems. We classify the identified approaches and point out future research to fill open gaps.

The paper is structured as follows. In the following section, we give a detailed motivating example from which we derive our research questions. To answer them, Section 3 briefly revisits background knowledge on self-adaptive system modeling and performance engineering. By elaborating on this background, we first derive a survey method for our literature survey (Section 4.1). Second, we present the classification in Section 4.2 which we then use to evaluate the papers in our study (Section 4.3). Finally, we discuss our findings in Section 4.4. In section 5, we discuss related literature surveys before we conclude in Section 6.

2. EXAMPLE SCENARIO

The purpose of our survey is to identify and understand model-driven performance engineering approaches for self-adaptive business information systems from the point of view of software engineers and researchers. In this section, we outline our motivation by providing a small example scenario. Using the presented scenario, we illustrate some challenges in model-driven performance engineering of self-adaptive systems and formulate research questions.

Consider a web-based application implemented as a three-tier architecture, as illustrated in Figure 2. The system runs on infrastructure owned by the service provider, i.e. own web server, application server, and database server.

When building our system, we have to determine the sizing of the infrastructure via capacity planning [33]. To achieve a reasonable performance of the system, various factors have to be considered. First, varying numbers of users can cause different load levels over time. Second, an increase of the total number of users over time has to be considered, which requires additional free capacity. Classical model-driven performance engineering can help to determine the best sizing. We model our system architecture with performance annotations, as shown in Figure 2, and derive an analysis model, e.g. a Layered Queuing Network (LQN), via automatic model transformations. Solving the analysis model gives us a prediction of how our system will perform. This helps us to decide on the sizing of our system.

However, we have to take peak work loads into account, i.e. analyze the worst case. Hence, our system still will be overdimensioned for the average work load. The average utilization of real world server system, similar to our example, is estimated to be about 5% to 20% [1]. The energy efficiency, however, of a server with only 20% load is at its lowest level [25]. This results in unnecessary costs.

Infrastructure-as-a-Service (IaaS) [39] cloud environments are advocated to solve this problem and to save costs. In an IaaS cloud, resources, i.e. CPU, HDD, and RAM, can be allocated and released on demand during run-time. Running our system in an IaaS cloud allows us to allocate only as many resources as needed to handle the occurring load. And the cloud provider accounts only the resources we allocated.

We can design and implement a self-adaptive system which smartly adapts to its load to benefit from the flexibility an IaaS cloud offers. The challenge is to design the right adaptation strategies and their parameters for our system, e.g. we need to decide which and how many resources have to be allocated or released. We also need to ensure that system adaptations are agile enough to cope with occurring peak loads. Adaptations also must not block the system, e.g. by oscillating between multiple adaptations. Last but not least, we want our adaptation strategies to be cost efficient, i.e. allocating as little resources as possible at any time.

Classical performance engineering cannot be applied for self-adaptive systems since static system structures are assumed. However, there exist some approaches which apply model-driven performance engineering on self-adaptive systems. We want to identify, understand, and examine these approaches according to their capabilities and limitations.

For this purpose, we formulate two research questions R1 and R2 which we want to address in this survey:

R1 What can be achieved with the state-of-the-art model-driven performance engineering approaches for self-adaptive systems? What are the limitations?

R2 What are the missing aspects raising a need for further research?
3. BACKGROUND

Model-driven performance engineering of self-adaptive systems is a combination of two research areas, as illustrated in Figure 3: Self-adaptive system analysis and performance engineering. In order to answer our research questions, we need to understand both research areas and their characteristics. Hence, we conducted three steps:

S1 We identified some prominent classes of modeling and analysis approaches for self-adaptive systems,
S2 we identified some prominent classes of model-driven performance engineering approaches, and
S3 we surveyed approaches which apply model-driven performance engineering to self-adaptive systems.

We shortly present the results of our first two steps in the following. The presented approaches for self-adaptive system modeling and analysis (S1) and performance engineering (S2) build the basis for our survey approaches, which we present in Section 4.

3.1 Modeling & Analysis of Self-Adaptive Systems

The modeling and analysis of self-adaptive systems is attracting increasing attention in the research community. However, existing and emerging approaches most often concentrate on the functional correctness of self-adaptive systems, e.g. stability, using model checking approaches. Performance analysis is hardly addressed, evidence for the performant application of adaptation rules is hardly given.

In this section, we describe some prominent representatives for model-based design approaches for the analysis of self-adaptive systems. Regarding the selection, we focused on business information systems, i.e. approaches for embedded systems are explicitly out of focus. Further, we aim at high-abstraction modeling languages that are or may be used in model-driven engineering approaches. That is, we are basically focusing on meta-model based languages. We do not claim to be complete, but aim to give an overview of the currently used techniques to infer research questions later on.

The majority of all approaches dealing with self-adaptation uses the pattern of feedback loops. One of the most used feedback loops is the MAPE-K [36], an architectural model from the autonomic computing domain that divides the process of adaptation into four phases: monitor (M), analyze (A), plan (P), and execute (E). Data that is collected and used during adaptation is stored in the so-called knowledge base (K). The object being adapted is named managed element. Another often used pattern used for modeling self-adaptive systems is the Three-Layer Architecture [28]. The bottom component control layer contains the interconnected components of the system with basic self-tuning capabilities. The change management layer effects changes to the underlying component architecture in response to new states (reported from below) or new objectives (reported from above). The uppermost goal management layer deals with change management plans in response to requests from the layer below or from new goals.

GOAL-ORIENTED REQUIREMENTS ENGINEERING. Goal-oriented requirements engineering approaches use goal trees to model requirements for self-adaptive systems [35, 40]. In [9], an approach using a controlled natural language (RELAX) combined with goal-oriented requirements engineering is presented. RELAX can be translated into temporal logic and thus be analyzed [44]. Sykes et al. [43] describe a formal approach for self-adaptive software architectures with tasks synthesized from high-level goals. For the planning of valid configurations, they use temporal logic and model-checking, thus being able to formally analyze their architectures.

UML-BASED LANGUAGES. There are several approaches that extend the UML to introduce the viewpoint of adaptation. Adapt Cases, for instance, are a use case based modeling language to explicitly describe adaptivity on a platform-independent modeling level [31]. Their level of detail is sufficiently high to perform first quality checks as shown in [30]. To that extent, the Adapt Case approach utilizes graph-transformation systems and model-checking. Basically, functional quality properties are checked, e.g. stability, deadlock-freedom, and loop-freedom. Strands [21] is one of several concepts that have been proposed as a UML profile by Hebig et al. The profile is meant to model control loops as first class entities when architecting self-adaptive systems with UML.

ADLs. Darwin is an architecture description language with components being first class entities. In [28, 29] the authors show how to design self-adaptive systems using Darwin and the Three-Layer Architecture described above. Again, the authors of this approach use a model checking approach to identify valid configurations.

DSLs. EMF/Ecore is a generic Eclipse technology to specify meta-models and their instances as well as to generate editors for meta-model based languages. As such, this technology is often used for the development of domain-specific languages (DSL). In [16] Fleurey et al. describe an approach to specify self-adaptive systems in terms of variants using an EMF-based meta-model that is instantiated using tabular-
like editors. Further, they use the constraint solver Alloy to check functional properties and perform simulations.

The commonality of all these approaches is their focus on functional analysis and accordingly their lack of modeled information for performance analyses.

### 3.2 Performance Engineering

In this section, we present different classes of model-driven performance engineering (PE) approaches and some prominent representatives for each class. We classify selected approaches based on our expertise in this area.

The approaches presented in this section can be used for performance engineering of business information systems. Self-adaptive system design is not taken into account. However, some of the approaches for self-adaptive systems in Section 4.3 are based on the here presented approaches.

We distinguish four major classes in model-driven performance engineering: model-based, UML-based, component-based, and bridge model-based approaches for performance engineering. We do not claim to provide a complete classification of all approaches here, but give a rough overview.

**Model-based PE.** There exist some model-based performance engineering approaches [2,10]. All of these approaches have in common that they derive analysis models from architectural models. The concrete modeling languages, however, are diverse.

The PRIMA-UML approach [11] by Cortellessa and Miranda is a prominent representative of model-based performance engineering approaches. In PRIMA-UML architectural models can be specified using annotated UML deployment diagrams, sequence diagrams, and use case diagrams. An analysis model, i.e., a Layered Queuing Network, can be derived from these annotated UML models.

**UML-based PE.** UML-Ψ [32] is a simulative performance engineering approach. The approach uses performance annotation languages to annotate UML diagrams. These annotated diagrams are transformed into simulations models, implemented as a C++ program. The simulation results are written back into the original UML model.

The most prominent representatives for UML performance annotation languages are UML SPT [37] and MARTE [38]. Although performance annotations for general purpose systems are possible with MARTE and SPT, the focus of these languages are real-time systems and embedded systems.

**Component-based PE.** Approaches in this class are similar to UML-based approaches but restrict themselves to component-based system aspects. Component-based performance engineering is intended to determine the performance of a software system which is assembled from components. Representatives for this class are CB-SPE [6] and Palladio [5].

The Component-based Software Performance Engineering (CB-SPE) approach uses UML SPT to annotate architectural models with resource demands. The approach integrates the modeling tool ArgoUML as well as the performance solver RAQS. Although the focus of UML SPT is embedded systems, the CB-SPE approach can be used to model general purpose systems.

Palladio is an Eclipse-based tool suite for designing and analyzing component-based software. The Palladio Component Model (PCM) uses a UML-like notation extended with performance annotations. The tool suite enables modeling of software systems and supports several performance analysis methods, i.e., analytical methods using layered queuing networks and simulation-based methods.

**Bridge models for PE.** Some approaches for model-driven performance engineering especially cope with the automatic transformation from architectural models to analysis models. The goal of approaches in this class is to alleviate the transformation by introducing a bridge model between architectural models and analysis models. This makes it necessary to implement two transformations, but enables a flexible reuse of once defined transformations.

The KLAPER approach [19] by Grassi et al. is a prominent representative of bridge model approaches. The approach is not meant to provide own architectural models, but transformations from the bridge model to analysis models. For each architectural model a transformation to the KLAPER intermediate language has to be provided. The KLAPER approach is used by the D-KLAPER approach, which supports self-adaptive system architectures, compare Section 4.3.

### 4. SURVEY

In this section, we first outline the review method (4.1) used to survey approaches for performance engineering of self-adaptive systems. Second, we detail our classification scheme (4.2) and the results of the survey (4.3). Finally, we discuss the results and give recommendations for further research (4.4).

#### 4.1 Review Method

In order to achieve objective and unbiased results, we conducted our review according to guidelines for systematic literature review [24] by Kitchenham and Charters.

We have presented the background and research questions for our survey in Section 2 and Section 3. In this section we will outline our review method and explain our criteria for including or excluding approaches.

Our data sources for the survey have been Google Scholar\(^1\), Microsoft Academic Search\(^2\), and DBLP\(^3\). We conducted our search during the period from October 2011 to February 2012. We included approaches found with combinations from both of the following keyword groups:

\(^1\)http://scholar.google.com/
\(^2\)http://academic.research.microsoft.com/
\(^3\)http://dblp.uni-trier.de/


- **Keyword group for self-adaptive systems**
  - self-adaptation
  - self-adaptive
  - self-

- **Keyword group for performance engineering**
  - performance engineering
  - QoS engineering
  - performance analysis
  - QoS analysis

Since we are only interested in model-driven performance engineering of self-adaptive business information systems, we explicitly excluded approaches in the area of embedded system engineering as well as all non model-driven approaches. Model-driven approaches in our sense only include approaches using higher abstraction-level models to derive lower abstraction-level models via model transformations as defined by Stahl et al. [42]. We also excluded approaches which only deal with modeling of self-adaptive system, but do not include quality analyses of these systems.

### 4.2 Classification

Figure 4 shows a feature diagram of our classification scheme, which we will detail in the following. First, we started with a coarse-grained classification (top-level white features) based on our expertise. Second, we derived further classification criteria (white features) from the characteristics of self-adaptive system design approaches and performance engineering approaches, we presented in Section 3. Finally, we refined the classification based on our findings (grey features).

We classify the approaches we have found according to four categories: adaptation, architecture, analysis, and applicability.

With the first classification criterion, adaptation, we classify the approaches according to which self-adaptation pattern and self-adaptation strategy are implemented. As described in Section 3 the most common self-adaptation patterns are feedback loops and layered approaches. We further distinguish between two different adaptation strategies: reactive and proactive. We call an adaptation strategy reactive, if the system triggers its self-adaptation when a goal is already violated. If the system predicts that it might miss a goal some time in the near future and hence adapts itself preventively, we call that proactive.

The second classification criterion is the system architecture. The approaches we have found were restricted to either component-based architectures or service-oriented architectures (SOA). We also examine which modeling languages are used to model the system architecture, e.g. the UML or custom modeling languages.

Performance analysis is the third classification criterion where we classify the approaches according to when the approach is applied (time), which analysis method and models are used, and if transformations are provided. We distinguish between two different points in time when performance engineering can be applied in self-adaptive system design: at design-time and at run-time. Self-adaptive systems can adapt themselves to meet predefined goals. The goals may be functional goals, e.g. correctness of responses, or non-functional goals, like response time lower than 5ms, reliability over 97%, etc.

A main challenge at design-time is to identify proper adaptation strategies, i.e. it has to be analysed whether the rules are sufficient to achieve the system’s QoS goals assuming a system context. At run-time performance engineering can be applied to measure the system’s context and to predict its performance trend. Furthermore, we classified the approaches according to which performance analysis method is used. There are three analysis methods: analytical, simulation, and prototyping [22]. The used analysis models are another classification criterion. Analysis models are formal models like layered queuing networks, Petri nets, or Markov models. In a model-driven approach, these analysis models are derived by architectural models via model transformations. Whether these transformations are provided is another classification criterion.

Finally, we examined the applicability of the approaches. On the one hand, the applicability depends on to which extend the approach provides tool support. That is whether only the performance analysis is tool-supported or a complete model-driven engineering from architecture modeling to performance analysis is tool supported. On the other hand, the applicability can be shown by a proof-of-concept implementation, or a case study.

### 4.3 Results

In this section, we present the results of our survey. The approaches we present have been selected using our review method outlined in Section 4.1. We examined and classified these approaches according to the classification scheme presented in the previous section. Our presentation is divided into two subcategories according to when the presented approaches are applied: at design-time or at run-time. Table 4.3.2 shows the results of our evaluation.

#### 4.3.1 Design-time

There already are a few research approaches which address the design-time performance analysis of self-adaptive systems. We will now briefly summarize these approaches.

**D-KLAPER.** The focus of the D-KLAPER approach [18, 20] by Grassi et al. is on a transformation chain from design models to analysis models. The approach uses a bridge model to alleviate the transformation from architectural models to analysis models.

The bridge model is used to describe an adaptive system as a set of resources (hardware and software) which offer and require services. System changes are modeled as a change in the binding between offered and required services.

Of special interest is the trade-off between costs and benefits of self-adaptations. Adaptations are modeled as a special kind of service call. Each adaptation service call can be annotated with quality attributes, e.g. failure rate. There is no modeling support for system design models. UML SPT [37]
is used instead. Furthermore, there is neither support for modeling adaptation rules nor were transformations from input models to intermediate models presented yet.

For the analysis, several analysis models are supported but may require additional transformations. In previous publications, transformations to Extended Queuing Networks, Discrete Time Markov Processes and Layered Queuing Networks have been presented. To model the dynamics of self-adaptive systems, (Semi-) Markov reward models are chosen as analysis models. For the analysis it is assumed that adaptations do not happen at the same frequency as system events. The systems are considered to be in a steady state when analyzed. Adaptation costs are analyzed separately using the results from system analysis.

**SimuLizar.** Based on proposed extensions [3] for Palladio, Meyer extends the PCM with reconfiguration strategies [34] in his Master’s thesis. These reconfiguration strategies are modeled using Story Diagrams [15]. An initial system configuration can be derived from a static Palladio Component Model.

Using the static architectural system model and the reconfiguration strategies, the self-adaptive system’s performance is evaluated by simulating a queuing network. The simulation uses a PCM model interpreter and updates the PCM model according to the reconfigurations during simulation. However, the simulator analysis is limited to the analysis of transient states yet. That is a complete set of adaptation strategies cannot be simulated.

Due to the reuse of the Palladio platform this approach provides reasonably good tool support for creating architectural models and performance analysis. However, the creation of reconfiguration strategies using Story Diagrams is not integrated into the Palladio suite yet.

**4.3.2 Run-time**

Predicting the performance of a self-adaptive system during run-time and triggering reconfiguration strategies is the main characteristic of the approaches within this category. The performance predictions use analysis run-time models which are derived from design-time architectural models. The run-time analysis models may be updated according to measured performance values during the operation. Within this category, the following approaches have been identified

**Descartes.** With Descartes [26], Kounev presents a vision for a proactive performance engineering approach in self-adaptive cloud systems. The approach is to integrate QoS models into the system’s components: in that way a component is self-aware of its own performance properties.

Furthermore, the approach is planned to provide automatic model extraction based on performance monitoring data. The model extraction is used to generate analysis models on-the-fly. These models are used to predict the performance running system.

The author also plans to predict the system’s workload and the effect of a self-adaptation a priori. Hence, the Descartes approach incorporates three control loops: A MAPE-K control loop for self-adaptation, a loop for refining and calibrating the online analysis models, and a control loop to forecast the workload.

Up to now, the Descartes project does neither provide any automatic model transformations nor tool support. However, both automatic transformations and tool support are planned.

**QoSMOS.** QoSMOS [14, 8] is an approach to model and implement self-adaptive systems whose self-adaptation is driven by QoS requirements. It uses the KAMI approach [17] to analyse the performance of the system during run-time.

The system designer has to manually derive an analysis model from the architectural model. However, automatic model transformations are planned. The manually derived analysis model serves as initial input for the online performance analysis and is updated during run-time, using the KAMI approach.

The QoSMOS approach integrates several tools into a complete tool suite to model the architecture and QoS requirements of a self-adaptive system.

**FUSION.** Elkodary et al. present an approach, named FUSION [13], which uses feature diagrams as the system model. The system’s architecture variants are represented as a feature diagram, where a feature configuration reflects the current architecture configuration of the system. Hence, self-adaptation is realized by switching between different system configurations.
The self-adaptation in FUSION is goal-driven, i.e. relying on predefined functional or non-functional goals. Each goal consists of a metric and an utility. A metric is something measurable, like response time. An utility is a feature which has influence on the metric, e.g. a load-balancing component. The influence factor of a utility on a metric is initially assumed by the system designer. During run-time the influence factor can be learned and corrected in the system’s knowledge base. An utility is triggered when FUSION detects that a goal is violated. The violation of a goal is detected via defined monitoring functions.

The authors have extended the XTEAM tool to support the modeling of goals and features. Since this approach is based on learning no analysis-models, like Queuing Networks, are required. Hence, automatic model-transformation is not required either.

SAFCA. The Self-Adaptive Framework for Concurrency Architectures (SAFCA) [45, 46] is a self-adaptation approach based on the MAPE-K control loop. The basic idea is that concurrent architectures for a system are compared regarding their performance during run-time and the architecture which promises better performance will be selected as new system architecture.

Performance values are continuously monitored and compared to historical performance values stored in a knowledge database. The knowledge database is the K component of the MAPE-K adaptation loop. Self-adaptation will be triggered when the current performance values indicate a bad performance. For this purpose, the performance is analyzed using queuing networks which correspond to the concurrent architectures.

The approach does neither provide automatic model transformations nor tool support for designing the system and evaluating its performance.

4.4 Discussion
Looking at the evaluation Table 4.3.2, we can make the following observations.

Half of the surveyed approaches use reactive self-adaptation strategies. A proactive self-adaptation strategy, however, is preferable since it enables the operation of a system without violating its goals.

Well-approved self-adaptation patterns, like feedback loops and the three-layer pattern, are not applied by all of the surveyed approaches.

The used architectural models vary in the used notation but most approaches assume component-based architectures. Queuing networks are used as analysis model in half of all approaches. D-KLAPER uses (Semi-) Markov Reward Models (SMRM) and FUSION is a learning-based approach. Model-transformations from architecture models to analysis models are rarely provided. D-KLAPER provides a bunch of transformations from its bridge models to analysis models. SimuLizar automatically derives simulation code from an architecture model.

Only the QoSMOS approach provides complete model-driven tool support. That is from modeling architectural models, automatic derivation of analysis models, and analysis, and evaluation of the analysis. However, we could not find any tool support for evaluating alternative adaptation strategies.

All approaches have shown their applicability by providing a proof-of-concept implementation. However, none of the approaches have conducted a case study to validate their applicability. Hence, the applicability in practice is still questionable.

In general, we have noticed a discrepancy between analytical approaches for self-adaptive system engineering and performance engineering. In the area of self-adaptive system engineering the analytical methods are mainly used to evaluate whether adaptation rules maintain the system’s functionality. Non-functional aspects such as performance are not analyzed at all or just informally.

In the area of performance engineering, dynamic contexts such as varying deployments of systems running in IaaS clouds are neglected. These dynamic contexts, however, are the original reason why a system has to adapt itself.

Furthermore, no approach does support engineers in deciding on adequate adaptation strategies. This is due to the lack of analyzing the adaptation rules themselves. Instead, only steady states of the system are analyzed. Reconfiguration times and costs are commonly neglected.

We recommend implementing complete model-driven tool support including modeling of self-adaptive systems with adaptation strategies. For convenience, tools should provide automatic transformations to analysis models as well as comprehensive evaluation of analyses. For design-time analysis, additional models should be introduced such as dynamic usage models which take varying contexts into account.

Finally, we recommend conducting case studies to prove the applicability of performance engineering of self-adaptive systems.

5. RELATED WORK
The research community has provided several surveys on both research areas we addressed in this survey: self-adaptive system modeling and performance engineering. Up to now, these research areas were isolated from each other but are converging recently. To the best of our knowledge, there does not exist a survey that focuses on the combination of both research areas. In the following paragraph, we will briefly describe the various surveys and their results.

Dobson et al. Dobson et al. survey the state of the art in autonomic communications which is closely related to self-adaptive systems [12]. They aim at identifying emerging trends and techniques such as decentralization new theories that can handle information on different abstraction levels, biological-inspired models, and trust and security engineering.
<table>
<thead>
<tr>
<th>Adaptation</th>
<th>Pro-/Reactive</th>
<th>Pattern</th>
<th>Architecture</th>
<th>Architectural Model</th>
<th>Performance Analysis</th>
<th>Method</th>
<th>Analysis Model</th>
<th>Transformation</th>
<th>Additional Models</th>
<th>Tool Support</th>
<th>Analysis</th>
<th>Complete MD</th>
<th>Validation</th>
<th>Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>proactive</td>
<td>reactive</td>
<td></td>
<td>UML-based PCM</td>
<td></td>
<td>analytical</td>
<td>Markov</td>
<td>(incomplete)</td>
<td>Bridge</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>D-KLAPER</td>
<td></td>
<td></td>
<td>SimuLizar</td>
<td>PCM components</td>
<td></td>
<td>simulative</td>
<td>QN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Descartes</td>
<td>(unspecified)PCM</td>
<td></td>
<td>analytical</td>
<td>learning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FUSION</td>
<td>components</td>
<td></td>
<td></td>
<td>(unspecified) learning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>QoSMOS</td>
<td>(unspecified)SOA</td>
<td></td>
<td></td>
<td>QN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SAFCA</td>
<td>(unspecified)SOA</td>
<td></td>
<td></td>
<td>QN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Surveyed approaches

**Kephart.** Kephart identifies some of the main scientific and engineering challenges of autonomic computing [23]. His focus is mainly on architectures, technologies, and (formal) tools, but also on human aspects, i.e. the human interaction with self-adaptive systems.

**Salehie et al.** Salehie et al. group existing approaches by the corresponding discipline [41]. Disciplines include software engineering, artificial intelligence, control theory, and distributed computing. The authors identify several challenges related to the engineering of self-adaptive systems, self-* properties, the adaptation process, and the human interaction with self-adaptive systems.

**Bradbury et al.** Bradbury et al. focus on studying formal specification approaches for self-managing systems [7]. The authors identify shortcomings in the expressiveness and scalability of the studied approaches.

**Koziolek.** Koziolek surveys approaches for performance prediction of component-based software systems [27]. The survey examines 13 approaches concerning general features, modeling formalisms, maturity, and applicability in industry. He concludes that performance engineering approaches matured over the last years but generic approaches for all kinds of component-based systems might not be practicable. Koziolek suggests domain-specific approaches instead. One technical domain, for example, could be self-adaptive systems as addressed in our survey.

**Becker et al.** Becker et al. review performance prediction approaches for component-based systems [4]. The focus of their survey is on practical applicability. The authors discuss strengths and weaknesses of performance engineering approaches from the viewpoint of engineers and give suggestions for further research. The survey of Becker et al. limits itself to component-based systems with static system architectures. In contrast to this, we focus on self-adaptive systems, which have dynamic system architectures. However, we have seen that some of our surveyed approaches are based on component-based architectures.

To summarize, none of these surveys investigates the combination of self-adaptive system modeling and performance engineering in particular. Therefore, we hope to provide the research community with a valuable new viewpoint on current research results and challenges.

6. CONCLUSION
In this paper, we identified and classified approaches for performance engineering of self-adaptive systems. We examined existing approaches according to their capabilities and limitations and revealed some discrepancies between self-adaptive system modeling and performance engineering. Our survey equally helps engineers and researchers. Engineers get an overview of existing approaches and researchers can tackle the limitations we have identified.

7. ACKNOWLEDGMENTS
This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre “On-The-Fly Computing” (CRC 901).

8. REFERENCES


