A Review of Model-Driven Verification Techniques for Self-Adaptive Systems: A Feature-based Analysis

Nafiseh Kahani, Jeremy S. Bradbury and James R. Cordy

Kahani@cs.queensu.ca, jeremy.bradbury@uoit.ca, cordy@cs.queensu.ca

Technical Report 2018-633

School of Computing, Queen's University
Kingston, Ontario

April, 2018

Abstract

A self-adaptive software (SAS) system must be able to cope with complex requirements and changing environmental conditions to fulfill its desired behavior without violating its specified properties. Model-driven development (MDD) is a promising way to decrease the complexity of system design and implementation, and thereby facilitate formal verification of self-adaptive systems. In this paper, we review the state of the art in application of MDD approaches to the design and verification of SAS systems. To do so, we identify the key qualitative features of SAS, MDD, and formal verification systems, and evaluate the current MDD-based SAS verification approaches with respect to these features. We discuss the results and their implications, and identify the remaining challenges and research opportunities in the application of MDD techniques to SAS system design and verification.

Keywords:
Self-adaptive Software Systems, Model-driven Development, Formal Verification, Classification

1 Introduction

Application domains such as embedded, autonomic and ultra-large-scale systems require flexible, configurable, and self-optimizing software. In recent years, self-adaptive software (SAS) systems have gained popularity in these contexts. SAS systems can adapt their behavior by evolving their current configuration (e.g., by adding/removing components or modifying parameters) in response to a changing environment such that they continue to provide the required services and satisfy the intended properties. This capability is achieved by continually reconsidering environmental and system requirements, and applying adaptations to cope with changing conditions.

The uncertain and dynamic nature of SAS systems makes their design a complex task and raises many other challenges, such as safety and scalability issues. When the adaptation occurs at run-time, the updating of a system with new components requires that the *transfer of state* from old components to new components fully satisfies the expected functional and safety properties. While formal verification can be used to check that a system preserves certain properties, determining which aspects of the environment to monitor, analyzing potentially conflicting goals, accounting for incomplete or uncertain information, and adaptation complexity have all made formal verification of SAS systems a very challenging task. The number of adaptation configurations can grow exponentially with the number of components, and each configuration needs to be both specified and verified. In this context, model-driven development (MDD) techniques have the potential to simplify and strengthen SAS system design and verification.

MDD approaches use models which are an abstraction or reduced representation of a system built for specific purposes [1, 2, 3]. The models can be used to define the structure, behavior, and even the adaptation mechanisms of an SAS system. The defined model can be analyzed, simulated, and executed. It is also intended that the application of formal methods to such models should be automated. However, while the input for verification tools is often low-level mathematical representations of system, such as Kripke structures or finite transition systems, modeling concepts are based on high-level abstractions to independently specify relevant system aspects. Application of formal verification approaches to MDD-based SAS systems thus raises different issues than traditional approaches.

While a number of literature surveys on SAS systems have already been done [4, 5, 6, 7, 8, 9, 10, 11, 12, 13], to our knowledge no previous study has specifically focused on the application of MDD for designing SAS systems and their formal verification. In this paper, we try to fill this gap, in summary, our paper makes the following contributions:

1. We present a classification of the most important features of SAS systems, MDD approaches, and formal verification techniques. When applicable, we relate these features to an illustrative example.
2. We evaluate how and where current approaches to MDD-based SAS system verification are applied with respect to the specified features, and how verification is addressed in these approaches.
3. We discuss the patterns and challenges that can be inferred from current research efforts in the MDD-based SAS system verification field.

For researchers trying to address challenges in the design and verification of SAS systems using MDD approaches, this work provides a high-level literature review of existing contributions. Researchers can use the catalogued features to evaluate the suitability of their approaches; or perhaps more importantly to improve the capabilities of their approaches through addressing the required features. Overall, we aim to provide a better understanding of the current state of the art in the design and verification of SAS systems using MDD and formal verification approaches, and to identify new research opportunities in this field.

The remainder of this paper is organized as follows. Section 2 provides an overview of the research method used in our study. Section 3 presents the required background. Section 4 describes an adaptive failover system – an illustrative example used throughout our paper. Section 5 specifies a set of features that can affect the MDD-based SAS system verification, and evaluates some current approaches with respect to these features. Section 6 discusses research challenges in designing and verifying SAS systems using MDD and formal verification approaches. Section 7 examines related work, and we conclude the paper in Section 8.

2 Research Method

The main contribution of this paper is a classification of the system features that are key to MDD-based SAS system verification, and a presentation of example approaches to supporting these features. We begin by explaining the analysis we used to identify key features.

2.1 Approach Selection

Our feature catalogue is based on an analysis of the attributes of existing approaches to verification of SAS systems using MDD. In order to select the approaches for our study, we began by identifying the main conferences (i.e. Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE International Conference on Autonomic Computing (ICAC), Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE), Self-Adaptive and Self-Organizing Systems (SASO)), journals (i.e., ACM Transactions on Autonomous and Adaptive Systems (TAAS), Software Engineering for Self-Adaptive Systems (SESAS)), and workshops (i.e., Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA), Workshop on Self-Healing (WOSS)) in the field. Since our focus is on design of SAS systems and their verification in this context, so we excluded those approaches that apply to MDD but do not consider the verification of SAS systems, and vice versa. This step resulted in a list of 17 directly relevant papers.

In the next stage, we used a range of related search terms on websites, such as *Google Scholar*, to search for locateing more approaches. Our search terms are: "model-driven approaches", "model-driven engineering techniques" "self-adaptive systems" and "formal verification". This step identified an additional 12 papers. Our final list includes 29 published approaches.

2.2 Feature Selection

To identify relevant features, we first reviewed previously published surveys of the literature on SAS systems [5, 14, 15, 8, 16, 6, 11, 10, 12, 17], MDD [18, 2, 3, 19], and formal verification of SAS systems [4, 20, 21]. We then analyzed the features identified in these surveys to select those that can be relevant to MDD-based SAS system verification. We labeled each selected feature with a reference to the corresponding previous publication(s). In some cases, the attribute values of shared features are at a different level of granularity than in the original publication, to adapt to the information we were able to collect from the selected approaches. We classified the extracted features in three dimensions: self-adaptive, model-driven, and formal verification. To help validate that our list of features is complete and correct, we evaluated the 29 selected approaches with respect to each of these features.

3 Background

To understand how using MDD and formal verification has the potential to improve SAS systems, we present a brief background of these two research areas and their characteristics.

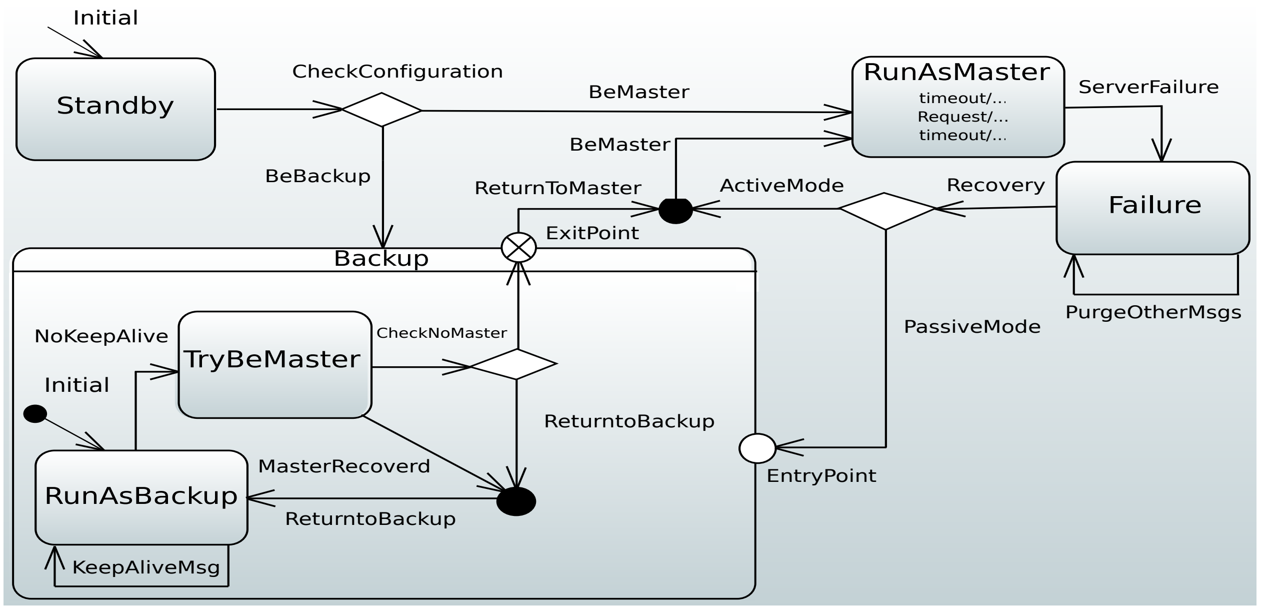

Figure 1: Adaptive Failover System State Machine Diagram.

3.1 Model-driven Development (MDD)

MDD uses models as the primary artifacts in the software development process [22, 3, 2]. Models of a system can be used at both design-time and run-time to provide abstract, precise and unambiguous representations of the system. Metamodels are used to define the appropriate and necessary structures, and the properties to which a model must conform. The structure, possible behaviors and environment of an SAS system can all be represented in models conforming to appropriate respective metamodels. Model transformation lies at the very core of MDD, supporting model-to-text transformations to code or reports, and model-to-model transformations between levels, kinds or versions of models [2]. In addition, the models can be analyzed before the actual functionality is implemented. This allows discovery of design flaws and inconsistencies in specifications early in the software development. Errors can thus be corrected more easily and at lower cost, which is important for safety-critical applications.

3.2 Formal Verification

Formal methods can be used to ensure that an SAS system avoids undesirable behaviors and satisfies desirable ones. The high complexity of SAS systems makes the verification process challenging and expensive. (Semi)-automatic verification techniques can be used to facilitate the assessment of required properties of the system to increase its reliability. Formal methods can be used to verify and validate the models against several different kinds of properties, namely global, local, and adaptation. Global properties specify requirements that must be satisfied by the adaptive system throughout any execution of the system. Local properties specify requirements on each individual configuration/domain satisfied by the specific adaptive system. To ensure the system properties, it is not sufficient to consider each model/configuration separately, but the adaptation process as a whole needs to be checked [23]. Furthermore, adaptation properties must be satisfied during the adaptation process itself [24].

4 A Motivating Example

To ground our discussion, we use an *adaptive failover system* as an illustrative example [25]. This system involves a set of server components to handle client requests. It relies on either passive or active replications [26], two common strategies for maximizing availability when building real-time distributed fault-tolerant systems. In passive replication, one server component works as a *master*, handling all the client requests while the *backup* servers are largely idle. Passive replication does not create run-time overhead, except for handshake operations, and for receiving state updates from the master [27]. In active replication, client requests are multicast and can be served by all service components. In case of server failure, the remaining servers can continue to provide the service to the clients, which leads to faster failure recovery.

Fig. 1 shows how the *adaptive failover system* can be modeled using UML-RT (additional details of this model can be found in [25]). The model consists of two main states, where the server can run as either a master or a backup server. A failure state simulates the failure of the master server and is reached after a randomly calculated time. When the server recovers from a failure, it may restart as either a master or a backup server, depending on the replication mode and other parameters. The tasks of a master server and a backup server are different. The master server is required to update its state by sending two kinds of messages: *IAmAlive* (sent to the backup servers), and *IAmMaster* (sent to the environment component). If it fails in sending these messages, its execution is considered to have failed and a new server must be ranked up. It is also responsible for receiving and processing client requests.

5 Classification of Features

Our classification scheme is grouped into self-adaptive, model-driven, and formal verification features. To specify these features, we analyzed existing surveys in each of the three fields [2, 4, 5, 14, 15, 16]. We then explored how features of each kind can be effective and useful in MDD-based SAS system verification. For each feature group we began with a top-level classification based on the existing surveys of the area. We then refined and further classified features from the point of view of leveraging the surveyed formal verification and MDD approaches in SAS systems.

5.1 Self-Adaptation Features

The self-adaptation dimension includes features of the SAS system itself that can be useful in MDD-based SAS system verification.

Application Domain. Self-adaptation can apply to a wide range of application domains, often involving an unpredictable, safety-critical, and dynamic environment. Examples include embedded systems (e.g., automotive and automation systems [28, 29, 23, 30, 31, 32, 33, 34]); parallel computing (e.g., cloud computing [35, 36]); service-oriented architectures (web-services [37, 38, 39, 40, 41]); multi-agent systems (e.g., mobile applications [42, 43, 44], robotic systems [45, 46, 47, 48, 49, 50]); and ultra-large-scale software (ULSS) systems (e.g., transit systems [51, 52]). Adaptation helps these systems react to changing environmental conditions or recover from system failures during execution, which otherwise could cause injury, financial loss, system thrashing problems, or environmental impact. Thus, the type of SAS system can affect the modeling and verification process. For example, verification of hard real-time systems is particularly challenging as it needs an understanding of the complete system and its detailed specifications. As shown in Fig. 2, the assessed approaches concentrate primarily on the embedded and multi-agent domains. One of the reasons is that the safety requirements of these domains are more demanding, which can motivate researchers to work on these areas more. Our illustrative example is a distributed embedded system.

Management Level [5, 14, 16]. Reconfiguration management of the SAS systems can be classified as *centralized* or *decentralized* [5]. In centralized approaches, such as Bucchiarone [51], Yang [44], Inglés-Romero [46], Klarl [50], Iftikhar [49] and Fleurey [47], adaptation control is performed by a specialized component, where the domain concerns are separated from the adaptation ones. While the most common approach, centralized control suffers from a single point of failure, and scalability problems in large-scale distributed systems. In a decentralized approach (e.g., Adler [23], Khakpour [31] and Hachicha [33]) a system consists of several components, where the control over the system is distributed across components. Decentralized control is crucial for quality properties, such as resilience, and flexibility. However, verification of decentralized adaptation is more difficult due to the collaboration of multiple components, which can impose consistency problems. As shown in Fig. 2, the majority of the assessed approaches are based on centralized control, indicating that there is a need for more focused research on decentralized approaches.

In our illustrative example, the failover system is performed in decentralized mode, that is, each server takes the backup or master role based on an initial configuration and environment conditions (e.g., is any master available?), and after that the backup servers send their request to the available master server.

Architecture [11]. This dimension refers to whether the architecture of the SAS system is *flat* or *hierarchical*. Unlike flat approaches, hierarchical components are arranged in multi-level hierarchies. The hierarchical structure can deal with complex systems in a scalable way, which makes them more suitable for SAS systems. Most of the assessed approaches have hierarchical structures, where components can

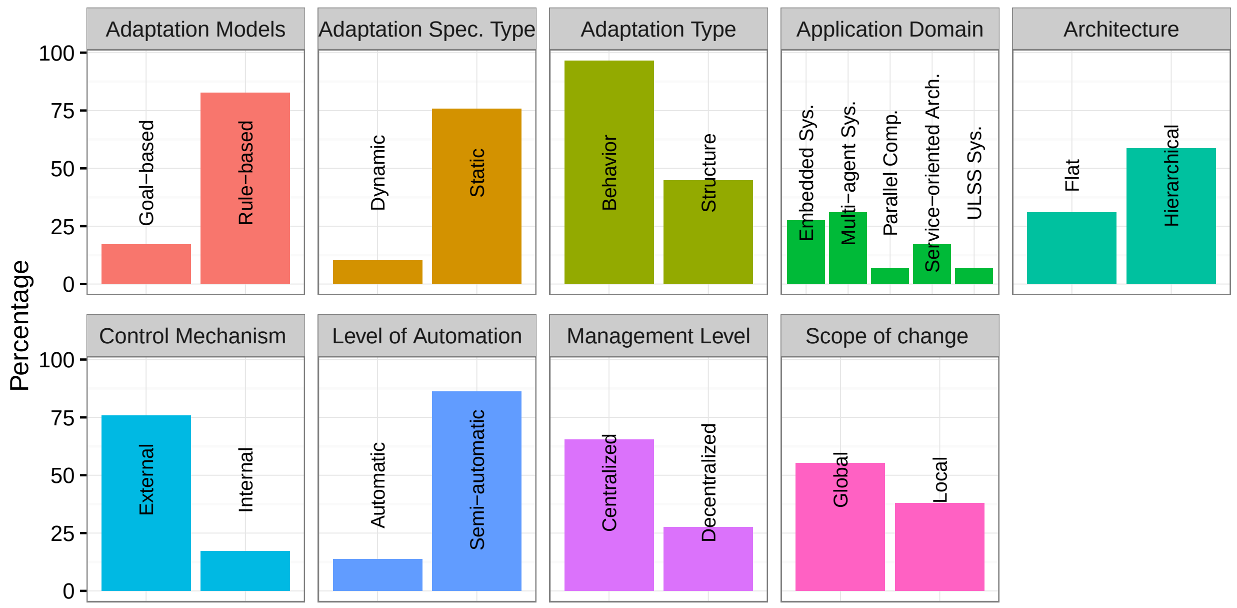

Figure 2: SAS feature summary of assessed approaches.

communicate with each other according to the system specification (Fig. 2). This allows integration of multiple interconnected components implementing a similar behavior under the control of the same composite to facilitate the verification process. Examples of hierarchical structure can be found in approaches such as [31, 45, 53, 51, 38, 48, 49]. Zhang [42] and Becker [54] are examples of flat architectures. Our illustrative example uses a flat architecture.

Adaptation Type [5, 14]. Adaptation of SAS systems can be structural, behavioral, or a combination of the two. Behavioral adaptation (e.g., supported by [28, 29, 32, 48, 23, 49, 50]) focuses on dynamic changes to the functionality or interaction of the computational entities. Transition systems can be used to represent system behavior in terms of states and transitions. Story patterns, which are an extension of UML Object diagrams, also support modeling the behavior of the system. The MechatronicUML approach [45] uses graph transformation rules represented by story patterns to formally describe the behavioral adaptations of the system. Zhang [42] uses formal models for the behavior of adaptive systems.

Structural adaptation is based on changing the system's architecture or environmental constraints. Structural SAS systems support fundamental reconfiguration operations, such as adding, deleting, refining, and updating new/existing components and their interconnections [5]. Eckardt [52], and Becker [45, 54, 53] propose systems in which structure is updated using these operations. Becker and Giese [53] use graph transformations for the specification of component reconfigurations. In addition to these fundamental operations, approaches for adaptable systems need to support composition of the operations using constructs, such as sequencing, choice, and iteration [5]. Performing reconfiguration operations in response to changes by installing, updating, integrating, and composing/decomposing components can be extended to run-time. Thus, it is required to check that these dynamic changes do not cause inconsistency problems, or unexpected changes in the behavior of the system. For example, a component can be safely removed from a system only if it does not have cyclic dependencies with other components. Approaches such as [31, 51, 45, 52, 55, 38, 37] allow for both structural and behavioral adaptation. As shown in Fig. 2, few of the assessed approaches address structural adaptation. Dynamic architectures and specifically dynamic components of the SAS systems can make the modeling and verification of structural adaptation challenging [5] . Our illustrative example supports both behavioral and structural adaptation (for details see [25]).

Control Mechanism [15, 8]: Adaptation mechanisms can be divided into two categories: *internal* and *external*. Internal control mechanisms are woven into system itself as a built-in component of the code. External control uses a separate adaptation engine running in parallel with the target system [38]. An advantage of internal mechanisms is a simpler architecture and run-time infrastructure for interaction and management of adaptations. They are also better suited to handling local adaptations, such as exception handling [15]. However, they are more specific to the application and hidden in the application code, and so can be difficult to build, generalize, model, reuse, analyze, and verify. By contrast, external approaches, such as Bucchiarone [51] and [49] allow specification and reuse of adaptation strategies across multiple systems and system properties. External control separates the concerns of target system functionality from the adaptation, which may assist in modeling and verification. Kŕikava [38] and Yang [44] propose an external approach based on feedback control loops. Examples using internal control can be found in Adler [23], Fleurey [47], and Zhang [42]. Our study shows that the majority of the assessed approaches use external control mechanisms (Fig. 2).

In our example system, the control mechanism is internal for behavioral adaptation, which is embedded in the server component's behavior. While for structural adaptation, the control mechanism is implemented by external approaches.

Scope of Change [14, 16]. This dimension identifies whether adaptation is localized, *local*, or affects the whole system, *global*. In local adaptation, verification overhead is reduced by verifying only a small part of a large system. The work by [31] is an example where adaptation is localized. If adaptation affects the entire system, then a more thorough verification is required to verify the adaptation process. Schaefer [28, 29] and Klarl [50] use global adaptation, where different components are involved. Reasons for Change can be *external*, such as dynamic features of the environment (e.g., communication infrastructure) independent of the system, or *internal* reasons, such as reconfiguration, or evolved requirements [14]. Features, such as the frequency (i.e., how often a specific change occurs), type (e.g., functional or non- functional) and where the change has occurred (e.g., application or infrastructure) play an important role to determine the adaptation and verification overhead in terms of cost, time, and performance. Fig. 2 shows how the assessed approaches address this feature.

In our illustrative example, adaptation occurs globally, and all components are required to perform adaptations (i.e., master and backup servers, and clients).

Adaptation Specification Type [16, 6]. This dimension indicates if the adaptation specification is *static* or *dynamic* based on when it is specified. Static refers to pre-defined or pre-constrained adaptations, which are specified at design-time but implemented at run-time. Whereas in dynamic, the adaptations are not pre-constrained and instead are both defined and implemented during run-time. Both static and dynamic adaptations can play a key role in SAS systems, however; dynamic adaptations are more complex to implement and verify. Furthermore, it is challenging to determine if unknown dynamic adaptations may violate uncertain and changing properties. Most of the approaches, such as [42, 54, 45, 52, 49, 34, 41, 46, 50, 44], focus on static self-adaptation of the software in response to changes in the environment. Fig. 2 shows that only a few assessed approaches support dynamic specification. It is evident that dynamic adaptation is one of the main challenges requiring more attention, most urgently in safety-critical systems. Adaptation in our illustrative example can be both static and dynamic (for details see [25]).

Adaptation Models [47]. Adaptation models can be classified into two types: *rule-based* and *search-based*. Rule-based approaches use a set of rules or policies to specify when and how adaptation should be performed, and under which conditions. Efficient run-time evaluation of adaptation rules allows predefinition of the specified target configurations, supporting design-time analysis of whether the system can be transformed to a model satisfying critical requirements. However, rule-based approaches have scalability issues with large sets of rules [56]. Several approaches [53, 33, 57, 54, 51, 52, 36, 38, 47, 37, 29, 57, 46, 50] use a set of rules to perform the adaptation. Papers such as [31, 39] use policies as a mechanism to adapt and control the system behavior. Search-based approaches provide another way to avoid explicit specification of the adaptation, by simply specifying the goals that the adapted system should achieve. Andersson [14] examines different dimensions of goals, such as evolution (static to dynamic), flexibility (rigid, constrained, unconstrained), and duration (temporary to persistent). Goal-based approaches are triggered by conditions or events, and guided by utility functions to find the best or a suitable target configuration fulfilling these goals [56]. However, search-based approaches suffer from costly run-time reasoning and planning processes, and provide less support for validation [56]. One can leverage both the efficiency of rule-based and the scalability of search-based approaches by combining them. Fleurey and Solberg [47] use rules to cope with the variability space explosion problem, combined with a search for suitable configurations. Ahmad [30] expresses non-functional requirements as goals. In Fig. 2 we can see that rule-based approaches currently outnumber goal-based ones.

In the context of our example, the adaptation model is rule-based, where the main rule checks the availability of the master server, and tries to replace the master when there is no master server available.

Level of Automation [14, 16]. This dimension captures the degree of automation during adaptation, ranging from *entirely automatic* to *semi-automatic* with the aid of other systems or human advice [14]. For example, in Bucchiarone [51] and Ghezzi[43], manual effort is needed to select the most suitable adaptation in the case that more than one adaptation are possible. Semi-automatic adaptation requiring human intervention is known as the *human-in-the-loop* problem. Using a human-in-the-loop approach in areas, such as machine learning [58] and search-based software engineering [59] benefits from both the efficiency of these techniques and the quality of human judgments and intelligence. However, keeping the users in the loop makes them a critical part of the adaptation process. SAS systems often consist of hundreds of configurations to cope with a large variety of possible environmental conditions, requiring the user to select the correct model from the huge solution space of a safety-critical system. Hence, the user needs to deeply understand how the system behaves at design-time/run-time to guarantee the quality of system properties. Keeping the users of SAS systems in the loop also requires trust and transfer policies (e.g. business policies) [15]. An example of an automated approach can be found in Khakpour [31] where there is no need for outside intervention during adaptation. As shown in Fig 2, few of the assessed approaches provide fully automatic adaptation. Full automation of adaptation therefore represents a potential research opportunity. In our illustrative example, adaptation is automated.

5.2 Model-Driven Development Features

In the following we examine the main features of MDD that are useful for verification of SAS systems.

Specification Approach [4, 6, 12]. Modeling languages can be useful in tackling the ever-increasing complexity of SAS development, and addressing the need for new requirements languages to handle the uncertainty present in SAS systems [10]. Different modeling languages can be used depending on the type of system and the particular verification problem that the system is being modeled for [60]. While there are many different modeling languages, we concentrate on those used in the literature for the verification of SAS systems. Fig. 3 shows that almost every kind of modeling language has been used to specify SAS systems, and there is no real consensus as to which is most appropriate. A detailed comparison of these languages would be required to provide more insight.

General-purpose Modeling Languages (GPMLs). GPMLs, such as the System Modeling Language (SysML) and the Unified Modeling Language (UML) can be used to express the structure and/or behavior of a wide range of systems. Hachicha [33] and Becker [54] use UML diagrams. In Zhang and Cheng [42], UML State diagrams are used to specify the behavior of the system. The ADAM approach [43] supports SAS systems modeled by Activity diagrams. An example using SysML for modeling SAS can be found in [30].

Domain-specific Modeling Languages (DSMLs). DSMLs, such as Simulink are dedicated to modeling in a particular domain or context. DSMLs support concepts and notations tailored to the particular problem domain, and thus can support higher-level abstractions than GPMLs. Several approaches [47, 38, 36, 40, 32, 28, 39, 34, 46] use DSMLs for the modeling of SAS systems. Trapp [32] has used Simulink in modeling automotive systems.

Behavioral Modeling Languages (BMLs). Modeling languages, such as finite-state automata (FSA), and Petri Nets (PNs) have been used to model the behavior of SAS systems. State machines are especially well-suited for discrete-time systems. Both Tan [48] and Radu [37] use an extension of FSA called Statecharts to model SAS. Statecharts are well-suited for modeling complex systems at different levels of abstraction by refining states using hierarchical sub-charts and their composition. MechatronicUML [45] uses real-time Statecharts, a combination of UML state machines and timed automata.

PNs are used to formally capture the dynamic semantics of concurrent and distributed systems. The work by Zhang [42] is an example of using PNs in SAS, to model different behavioral variants of a process.

Formal Modeling Languages (FMLs). FMLs, such as Z, force an analysis of the system requirements. Z notation is a model-oriented specification language used for describing and modeling abstract data. SAS systems with complex data operations, especially complex state transformation functions are modeled in Z. Zhang [42] provides an example using a Z model.

Mathematical Modeling Languages (MMLs). Markov models and graphs are examples of mathematical models. Markov models are a stochastic model consisting of a list of the possible states to model and analyze reliability and dependability of the systems. In Ghezzi [43] a Markov model of the SAS system is generated from UML interaction diagrams.

Graphs models can be used to model system states, and can represent SAS input and output models using variations of typed, attributed graphs. In Becker [54, 53], the SAS system's states are modeled by graphs, and a set of graph transformation rules is defined by story patterns. Bucchiarone [51] uses the Algebraic graph transformation (AGT) typed graphs and typed graph grammars to model static and behavioral parts of the system respectively.

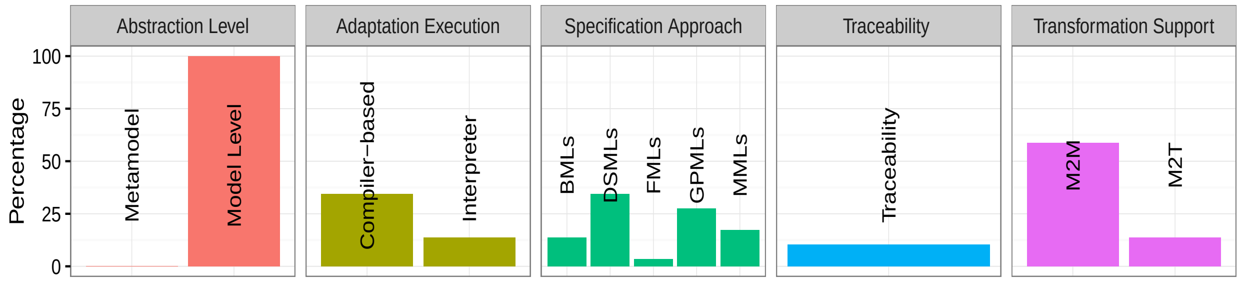

Figure 3: MDD feature summary of assessed approaches.

In the context of our example, we use the UML-RT language which is a DSML in the context of soft real-time system development [25]. UML-RT designed as a UML profile uses an extension of UML structure diagram (i.e., capsule diagrams) for structural specification of the system, and an extension of UML state machines for behavioral specification.

Transformation Support [19, 6, 2]. Model transformations can be classified into two main categories, model-to-model (M2M), and model-to-text (M2T). M2M transformations can be used to transform an input model into a formal model. The transformed models are then checked to verify that the desired properties hold. An M2T transformation can be used to implement adapting operations by generating the adapted code from the model(s). There are many model transformation languages and tools applicable to SAS systems [2, 61]. Examples of using M2T and M2M transformations to perform transformation operations for adapting SAS systems can be found in approaches, such as [38, 43, 30, 23, 53, 34, 47, 28, 29, 40, 32, 37, 48, 33, 41, 50, 44]. Křikava [38] uses M2T to translate the model elements into source code artifacts that implement the self-adaptive control layer. Both Schaefer [29] and Adler [23] use M2M transformations to reduce the complexity of verification in SAS systems. Fig. 3 shows that the majority of the assessed approaches use M2M transformations. However, this is not an indication that M2T research is not important in SAS. Rather, it is likely an artifact of the fact that many frameworks and tools already provide M2T features out-of-the-box [2].

In our example, tools such as IBM Rational Rose-RT can be used to support code generation from UML-RT models to different programming languages. It is also possible to apply M2M techniques and provide analysis and verification facilities.

Traceability Capabilities [19, 16, 2]. Traceability has become a key part of the development of safety-critical systems. Traceability links can be established between model elements at different levels of abstraction, between requirements and model elements, or between model elements and code [62]. In SAS, traceability can help to identify the origin of errors, to perform impact analysis for created/modified elements, or to re-check consistency of models after changes are applied. Because requirements can change rapidly at run-time in SAS systems, traceability monitoring is critical to verification, but as shown in Fig. 3, only a few of the assessed approaches, such as Křikav [40] leverage this feature.

In our example, supporting traceability would help monitor adherence to requirements at run-time, in order to keep the system model synchronized with the run-time environment.

Adaptation Execution [18, 2]. Practical modeling of SAS systems requires executable models. When the operational semantics of a model are completely defined, the model is complete enough to be executable. Strategies to make executable models execute include: compiler-based, interpreter-based, and hybrid of both. While interpretive approaches may be flexible but have problems with large models, compiler-based methods may have good performance but lack flexibility. Ghezzi [43], Iftikhar [49], and Bucchiarone [51] use interpreters to support the execution of adaptation engine operations. In Ghezzi [43], the model is executed by an interpreter that drives the execution of the system. There are several examples of compiler-based approaches [38, 42, 32, 29, 40, 48, 41, 50]. Zhang [42] for example proposes a method to construct adaptation models and automatically generate adaptive programs from the models. We can see in Fig. 3 that the majority of the assessed approaches are compiler-based. Overall, model execution is one of the features not well addressed by existing MDD approaches, not only for SAS systems, but also for other domains. In our example, we use a compiler-based approach since the MDD tools supporting UML-RT provide only code-based execution.

Abstraction Level [20]. This dimension addresses the level at which verification can be exercised:

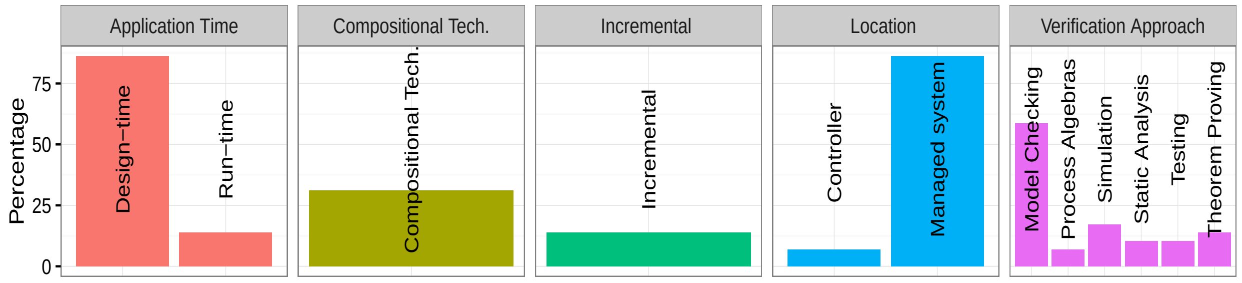

Figure 4: Formal verification feature summary of assessed approaches.

metamodel-level versus *model-level*. Metamodel-level verification uses metamodel information to verify properties of any well-formed model [20], using sophisticated formal verification techniques, such as theorem proving. By contrast, model-level verification works on specific input and output models, using model checking or testing techniques. Because model-level verification works with a lower level of abstraction, the range of properties it can deal with is broader [20]. Approaches, such as Adler and others [23, 47, 53, 52, 42, 37], address verification at the model-level, and require models to be represented at run-time. Surprisingly, none of the studied approaches uses a meta-model approach, which implies that no generic solution is presented. All of the assessed approaches specifically work at the instance-level. In our example, using UML-RT for modeling allows the use of many different model-based checking tools through model transformation to representations handled by the tools.

5.3 Formal Verification Features

We review here the features of formal verification methods to be considered when verifying SAS systems using MDD.

Location [17]. The two main subsystems of an SAS system are the *managed system*, which delivers the basic function or service, and the *adaptation controller*, which deals with the adaptations of the managed system for the appropriate adaptive behaviors [63]. The adaptation controller might itself be adaptive so that the required properties can be evaluated on the managed system or the controller. Properties, such as accuracy and safety can be evaluated for both the controller and the managed system. However, most properties can only be observed in the managed system [17], as recognized in [53], [52], [49], and [42]. Fig. 4 shows that despite the importance of adaptation verification at the controller level, the assessed approaches mostly focus on the managed system. In our example we need to verify both the managed system (the master and backup components) and the controller. The whole system should be verified, using an exhaustive analysis of fail-over scenarios.

Application Time [4]. This dimension addresses the time at which formal methods can be applied, *run-time* or *design-time*. Verification at run-time helps avoid verification difficulties in highly dynamic environments with limited computational resources. However, the uncertainty in SAS systems can make modeling the system's behavior before deployment time-consuming and costly. Approaches such as [48, 44, 34] support run-time verification. Becker [54] proposes a technique to verify structural safety properties for infinite-state systems with structural adaptation behavior at run-time. Examples of design-time verification can be found in approaches, such as [47, 28, 41, 32, 29, 49, 42, 57, 46, 50]. Verification of safety-critical applications, such as autonomous cars must be preformed at design-time, to ensure a priori that adaptation does not fail at run-time. However, the unpredictable environment of SAS systems may cause the incomplete and uncertain knowledge at the design-time to be invalidated at run-time. Thus for SAS, to guarantee the fulfillment of the system's properties, formal methods should be applied at both design-time and run-time. Our study shows that only a few approaches support run-time verification (Fig. 4), despite the fact that verification at run-time helps avoid problems such as state explosion. In the context of our example, verification should be applied at both run-time and design-time.

Incremental Verification [11]. Incremental verification [64] involves saving and updating previous verification analysis in response to change, in order to avoid repeating the entire verification process. Thus, only the validity of properties that may be affected by changed parts need to be re-established. While Bucchiarone [51] incrementally checks conflicts of newly added adaptation rules with respect to existing ones. Overall, the majority of the studied approaches do not support incremental verification (Fig. 4). In our example, incremental verification would significantly increase efficiency of the verification process.

Compositional Techniques [11]. Approaches should support design and verification of large and complex SAS systems. Time and memory consumption for modeling and verification can increase exponentially with system size and complexity. Compositional techniques, such as *decomposed verification*, *interface abstract interpretation*, *functional decomposition*, and *assume-guarantee reasoning* [29], can boost the scalability and efficiency of SAS verification using modular and incremental techniques, as for example in Becker [45]. Eckardt [52] uses compositional verification to enable scalable formal verification of large mechatronics systems.

Decomposed verification enables scalable formal verification as a sequence of steps using a range of verification techniques, such as theorem provers and model checkers [65]. Schaefer [29, 28] reduces verification complexity by decomposing the verification task for the global system model into a number of less complex verification tasks for parts of the model. Interface abstract interpretation (e.g., used in [23], [29]) is a static analysis method used to reduce the complexity of the verification process [66]. Tools for abstract interpretation can lead to false positives, which hampers the possibility of fully automating the process [67]. In functional decomposition (e.g., used in [23]), there is a separation between functionality and behavioral adaptation in a module using a set of predetermined functional configurations that it may adapt to [28]. Assume-guarantee reasoning (e.g., used in [28]) decomposes the system into several components, implementing verification of the system as the separate verification of each of the components [68]. Similarly to the incremental verification techniques, only a few of the assessed approaches apply compositional techniques. Compositional techniques could significantly increase the verification efficiency of the adaptation process in our example.

Verified Properties [16, 4, 17]. The SAS adaptation process may create a behavior that violates one or more critical properties of the system. The properties of interest may vary according to the structural and behavioral configuration of SAS systems, and the nature of the models. They can be classified as either functional properties, describing specific functions of the system, or non-functional properties, describing operational qualities of the system [63]. Both can be further classified according to whether they are observed in the target system, or in the adaptation process. Properties can also be categorized according to whether they can be verified at design-time, or at run-time. A detailed review of adaptation properties can be found in Villegas [17].

Becker [54] uses graph patterns to model safety properties (hazards, accidents) of large multi-agent systems in the mechatronics domain. Bucchiarone [51] defines a set of operational properties (corrective, enhancing, direct (normal) self-adaptation) to check system correctness using related rules. A graph transformation-based approach [53] uses simulation and invariant checking techniques to check the correctness of the modeled SAS systems. Invariant checking is used to verify that a given set of graph transformations will never reach a forbidden state. Another important thing is to assure that the adaptation does not lead the system into inconsistent, deadlock, or unstable states. Bucchiarone [51] uses the consistency constraints to check that all reachable system states are consistent. Zhang [42] checks consistency among the models. Schaefer [29] uses static analysis methods to check that models are consistent with respect to syntactical constraints. A number of approaches [36, 55, 23, 42, 37] check for deadlock-freedom. Adler [23] and [36] both propose frameworks to verify the stability of the adaptation process. Other properties to be guaranteed are: liveness (e.g., checked in [42, 45]); safety (e.g., checked in [45, 28, 32, 55]); fairness (e.g., checked in [36, 55]); reachability (e.g., checked in [38]); scalability (e.g., checked in [38]); response time (e.g., checked in [43]); robustness (e.g., checked in [42]), (e.g., checked in [43]); and termination (e.g., checked in [55]).

In our example, properties, such as safety, liveness, reachability, and correctness need to be preserved. For example, there is always a available master server to serve the backup ones (liveness); two servers must not be master at the same time (safety); and the master will return to the backup mode after failing (reachability).

Property Specification [4]. One of the most common ways to specify properties is the use of temporal logics [69]. On the one hand, Linear Temporal Logic (LTL) provides a mathematically notation to specify properties that hold during the adaptation process over a linear time. Zhang [42], Luckey [36], Křikav [40], Yang [44], and Adler [23] all use LTL. On the other hand, Computation Tree Logic (CTL) describes properties using a branching tree-like structure, where different operations may occur in each state. In CTL the model of time is a non-deterministic tree, leading to the various states. Adler [23], Iftikhar [49] and Schaefer [28] use CTL as a property specification to specify model properties.

Verification Approaches [20, 4]. Techniques for the formal verification of SAS systems can be classified as *semi-automatic* or *automatic*. Approaches can benefit from combining different formal techniques, such as simulation and model checking. Adler [23] uses theorem proving, model checking, and specialized verification methods to verify the system. In the context of the failover system, many different approaches can be applied.

Model Checking. Since the number of system states can grow exponentially with the number of components, model checking [70] is often too expensive to apply directly to SAS systems of more than moderate size. To handle larger models, model checking can be combined with abstraction or decomposition techniques. Schaefer [28] proposes a framework for modular model-based verification of adaptive systems using model checking. Symbolic model checking, such as Binary Decision Diagrams (BDDs), can also cope with large models by representing sets of states rather than enumerating individual states. Bounded Model Checking (BMC) is a complementary technique to BDD-based unbounded model checking. Bounded means only those states are explored which can be reachable within a limited number of steps. Many approaches [42, 47, 36, 23, 30, 52, 32, 40, 28, 34] use model checking to validate and verify SAS system models and properties.

Model checking tools, such as Alloy (e.g., used by Fleurey [47] to check functional properties and perform simulations), NuSMV (e.g., used by Adler [23], Schaefer [28]), UPPAAL (e.g., used by Becker [45], Iftikhar ActivForms, and Eckardt [52]), SPIN (e.g., used by Kŕikava [38] and Klarl [50]), and PRISM (e.g., used by Ghezzi [43] and Yang [44]) have all been used to verify properties of adaptive systems expressed in temporal or other logics. As shown in Fig. 4, model checking is the most popular method for formal verification of SAS systems.

Process Algebras. Process algebras [71] are commonly used to describe and verify properties of concurrent systems based on send and receive messages through channels. Process algebra techniques can be difficult to extend to other types of formalisms and the corresponding analysis. Example process algebras include the Calculus of Communicating Systems (CCS), Communicating Sequential Processes (CSP), Algebra of Communicating Processes (ACP), and the Π-calculus. Process algebra is employed in Mateescu's work [37], and Adler [23] uses Π-calculus to check stability.

Logical Inference/Theorem Proving. Unlike model checking, theorem proving [72] does not suffer from the state explosion problem, since many properties can be verified without enumerating all possible states. However, it can be error-prone, and requires substantial user interaction and advanced proof skills. There are many theorem proving systems, including Isabelle/HOL, Coq, Prototype Verification System (PVS), and A Computational Logic for Applicative Common Lisp (ACL2). The approaches of Adler [23], Hachicha [33], and Schaefer [29, 28] use theorem proving to evaluate the correctness and the validity of the transformations performed on SAS models.

Static Analysis. Static analysis techniques can be used to guarantee that models are consistent with syntactical constraints [73]. Although static analysis can find flaws in the structural design of the models at initial design stages, it cannot assure the overall correctness of the design. Bucchiarone [51] employes static analysis techniques supported by the algebraic graph tool AGG to analyze SAS system models. Static analysis is also used by Schaefer [29] to check models for structural consistency. Bucchiarone [74] uses static conditions to verify that consistency and operational properties are maintained across adaptations.

Testing. Testing as a general approach can be leveraged for different types of SAS systems. However, testing can only show the presence of errors, not their absence. Model-based Testing (MBT) [75] has recently gained popularity as a way to address this issue. MBT systematically generates test suites from a formal model, with the aim of thoroughly testing whether the implementation of the system behaves according to its specification. Testing cannot in general deal with all kinds properties, such as safety. A combination of testing with other verification methods, such as model checking is needed to fully reason about system correctness. Zhang [42] uses MBT to guarantee conformance between system models and their subsystems, and Tan [48] and Bartel [41] also use MBT for SAS systems.

Simulation. In this technique, a formal model of the system is established and simulated. Model-based simulation uses a small subset of model states, and like testing cannot provide correctness guarantees. The results of a simulation can vary widely based on the scenarios executed and similarly to testing can only uncover the faults, not prove their absence. Bucchiarone [51] uses simulation to find suitable adaptation sequences using adaptation rules in AGG. Zhang [42], Iftikhar [49], and Ghezzi [43] employ a simulation method to validate SAS models against specified properties. MechatronicUML [45] simulates the behavior of mechatronic systems to ensure correctness of their design.

6 Discussion

This section highlights the important features that are not well supported by current approaches, the major remaining challenges in modeling and verification of SAS systems, and threats to validity of our study.

6.1 Features

Our work shows that existing formal approaches for adaptation mainly focus on structural or behavioral changes in SAS systems. It is evident that a real need exists for work on both behavioral and structural verification of MDD-based SAS systems to improve their quality and reliability. In a same way, adaptation time of the majority of approaches is static, whereas, both static and dynamic adaptations are important in SAS systems, as changes in the system such as reconfiguation of the system can be happened during deployment, code generation, or run-time. Traceability also receives less attention in the assessed approaches. While, an existing challenge in SAS systems is a need to traceability from requirements to implementation [16], which can be supported through establishing the links between models and code.

In general run-time formal verification is not supported by the majority of studied approaches to SAS. While, run-time verification can help increase the evidence for assurances by using information from the running system and its environment. Therefore, applying formal methods at both design-time and run-time can boost assurance of the system. Incremental verification is another important feature not supported well by the assessed approaches. In systems that do not support incrementality, the complete verification process must be re-performed in dealing with changing models.

6.2 MDD in SAS Systems

As we have seen, MDD can aid in providing a comprehensive description of the system, using different models to describe different aspects [2], thus reducing the complexity of verification. On the other hand, dealing with multiple models, especially at run-time, can cause problems with consistency among models. To avoid that, tracebility, and incremental approaches can be applied to keep the models consistent. The level of abstraction is another major difficulty in modeling of SAS systems, which must be carefully selected to include only the required details. A high-level abstraction may not contain the required information, but a too detailed abstraction would make the system complex or less comprehensible [10].

Modeling behavioral adaptation can be complex because of interdependencies between the modules of a system. This can be partially addressed by separating adaptation-specific behavior from non-adaptive behavior. Another concern is the synchronization between run-time models and the evolving system, as system requirements may evolve and change at run-time. These changes make it difficult to define and verify a complete model before system deployment. Therefore, models need to be evolved and maintained at run-time to reflect theses changes. *Models@run.time* approaches are leveraged to provide the required abstractions of a running system, and its environment [76, 63]. In case of adaptation, changes are planned and analyzed using run-time models before they are propagated to the running system. Such adaptations require automatic reasoning and planning mechanisms that work online and on top of run-time models [77]. Taking time between propagating the changes between run-time models and running systems makes them unsuitable for hard real-time systems. In addition, MDD benefits can be leveraged to the run-time phases of SAS systems [56].

6.3 Verification of SAS Systems

It is necessary to show that when switching between models/ configurations (structural or behavioral), essential system properties (e.g. local, global, adaptation) are preserved. It is not sufficient to simply consider each model/configuration separately; rather, adaptation as an entire process along with the input and output models must be checked. The existing techniques focus primarily on verifying the adapted system and its subsystems; however, verification of the adaptation process itself has received less attention.

A major difficulty in verification of SAS systems is that the usability of formal approaches depends on their integration in MDD-based SAS systems. Formal verification cannot always be immediately applied in model-based development. On the one hand, we often have low-level mathematical concepts models as the input for verification tools, on the other hand, the modeling concepts are based on a high-level abstraction to specify the relevant system aspects. Therefore, there is a need of bridging the gap between high-level modeling concepts and low-level verification input, such as Schaefer [29]'s approach. Another challenge is related to selecting the most appropriate verification approach, as the usability of formal approaches depends on the specific domain and the kind of property one wishes to verify. Sometimes, it needs to use different verification techniques to perform the same verification task, as different properties are suited to different verification techniques. For example, model checking is well-suited to safety and security properties.

6.4 Threats To Validity

The main threat to our findings is missing information. Not all of the assessed approaches provide the detailed specifications of their design and implementation, so a risk of missing information for some of these approaches are possible. However, we believe it is low enough for our results to still provide the current state of application of MDD to the design and verification of SAS systems. A second threat comes from the list of the assessed approaches. We searched the websites, such as *Google Scholar* and examined the main venues in the filed to identify the existing approaches to verification of SAS systems using MDD. However, there is still a risk of missing some related approaches. To decrease this threat, we also checked the related work of the assessed approaches, along with previously published surveys of the filed to make sure that our list of the assessed approaches is complete.

7 Related Work

A number of authors have explored different aspects of SAS systems. Bradbury [5] surveyed 14 formal specification approaches for self-adaptation based on graphs, process algebras, and logic formalisms. The survey concluded that existing approaches need to be enhanced to cope with issues regarding expressiveness and scalability. Anderson [14] has classified the modeling dimensions of SAS systems into: goals (what is the system supposed to achieve), changes (causes for adaptation), mechanisms (system reactions to changes) and effects (the impact of adaptation upon the system). Weyns [4] has surveyed formal methods in SAS systems.

Tahvildari [15] proposed a taxonomy, based on concerns of adaptation, i.e., how (should adaptation be performed), what (elements should be changed), when (should adaptation actions be applied) and where (the adaptation should occur). Cheng [63] proposed a research roadmap to identify critical challenges for the systematic software engineering of SAS systems. There have been other surveys, such as Villegas's work [17] on quality evaluation of self-adaptive systems, and Macìas-Escrivá's work [8] reviewing recent progress on self-adaptivity from the standpoint of computer science and cybernetics. McKinley [78] focused on the techniques and technologies in SAS systems.

Unlike these works, our work focuses on the required properties and intended goals of application of MDD approaches to the design and verification of SAS systems.

8 Conclusion

We have reviewed the state of the art in leveraging MDD techniques to the design and verification of SAS systems, and have identified existing research challenges. Our work is potentially useful to many parties in the modeling and SAS communities, including practitioners who need to solve a specific SAS problem, and researchers trying to address existing challenges in using MDD in verification of SAS systems. The highlighted features and challenges that are not addressed by different approaches might inspire researchers to tackle these challenges.

References

[1] J. Rothenberg, L. E. Widman, K. A. Loparo, N. R. Nielsen, The nature of modeling, Rand 3027.

[2] N. Kahani, J. R. Cordy, Comparison and evaluation of model transformation tools, in: Technical Report 2015-627, 2015, pp. 1–42.

[3] N. Kahani, M. Bagherzadeh, J. Cordy, J. Dingel, D. Varró, Survey and classification of model transformation tools, Software and Systems Modeling (SoSyM) (2018) 1–47.

[4] D. Weyns, M. U. Iftikhar, D. G. D. L. Iglesia, T. Ahmad, Survey of formal methods in self-adaptive systems, in: Proc.of the Fifth Int. Conf. on Computer Science and Software Engineering, 2012, pp. 67–79.

[5] J. S. Bradbury, J. R. Cordy, J. Dingel, M. Wermelinger, A survey of self-management in dynamic software architecture specifications, in: Proc. of the 1st ACM SIGSOFT workshop on Self-managed systems, 2004, pp. 28–33.

[6] M. Becker, M. Luckey, S. Becker, Model-driven performance engineering of self-adaptive systems: a survey, in: Proc.of the 8th int. ACM SIGSOFT Conf. on Quality of Software Architectures, 2012, pp. 117–122.

[7] V. Bauer, M. Broy, M. Irlbeck, C. Leuxner, M. Spichkova, M. Dahlweid, T. Santen, Survey of modeling and engineering aspects of self-adapting & self-optimizing systems, in: Technical Report-TUM-I1324, 2011, pp. 1–44.

[8] F. D. Macìas-Escrivá, R. Haber, R. D. Toro, V. Hernandez, Self-adaptive systems: A survey of current approaches, research challenges and applications, Expert Systems with Applications 40 (18) (2013) 7267–7279.

[9] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive and Mobile Computing 17 (2015) 184–206.

[10] J. Bocanegra, J. Pavlich-Mariscal, A. Carrillo-Ramos, On the role of model-driven engineering in adaptive systems, in: Computing Conf. (CCC), 2016 IEEE 11th Colombian, 2016, pp. 1–8.

[11] R. de Lemos, D. G. abd C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, Software engineering for self-adaptive systems: Research challenges in the provision of assurances, in: Software Engineering for Self-Adaptive Systems III, 2017, pp. 1–29.

[12] Z. Yang, Z. Li, Z. Jin, A thematic study of requirements modeling and analysis for self-adaptive systems, 2017, pp. 1–12.

[13] T. Patikirikorala, A. Colman, J. Han, L. Wang, A systematic survey on the design of self-adaptive software systems using control engineering approaches, in: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2012, pp. 33–42.

[14] J. Andersson, R. Lemos, S. Malek, D. Weyns, Modeling dimensions of self-adaptive software systems, in: Soft. Eng. for Self-Adaptive Systems, 2009, pp. 27–47.

[15] M. Salehie, L. Tahvildari, Self-adaptive software: Landscape and research challenges, ACM Trans. on Autonomous and Adaptive Systems 4 (2) (2009) 1–40.

[16] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Muller, D. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle, Software engineering for self-adaptive systems: A research roadmap, (Eds.): Self-Adaptive Systems (2009) 1–26.

[17] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, R. Casallas, A framework for evaluating quality-driven self-adaptive software systems, in: Proc. of the 6th Int. symposium on Software engineering for adaptive and self-managing systems, 2011, pp. 80–89.

[18] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, 2012.

[19] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM Systems Journal 45 (3) (2006) 621–645.

[20] D. Calegari, N. Szasz, Verification of model transformations: A survey of the state-of-the-art, Theoretical Computer Science 292 (2013) 5–25.

[21] G. Tamura, N. Villegas, H. Müller, J. Sousa, B. Becker, M. Pezze, G. Karsai, S. Mankovskii, M. Pezzé, W. Schäafer, L. Tahvildari, K. Wong, Towards practical runtime verification and validation of self-adaptive software systems, 2013, pp. 108–132.

[22] R. France, B. Rumpe, Model-driven development of complex software: A research roadmap, 2007, pp. 37–54.

[23] R. Adler, I. Schaefer, T. Schuele, E. Vecchié, From model-based design to formal verification of adaptive embedded systems, in: Formal Methods and Software Engineering, 2007, pp. 76–95.

[24] M. Cordy, A. Classen, P. Heymans, A. Legay, P. Schobbens, Model checking adaptive software with featured transition systems, in: Assurances for Self-Adaptive Systems, 2013, pp. 1–29.

[25] N. Kahani, N. Hili, J. R. Cordy, J. Dingel, Evaluation of UML-RT and papyrus-RT for modelling self-adaptive systems, in: 9th Workshop on Modelling in Software Engineering (MiSE'2017), 2017, pp. 12–18.

[26] R. Guerraoui, A. Schiper, Software-based replication for fault tolerance, IEEE 30 (4) (1997) 68–74.

[27] J. Balasubramanian, S. Tambe, C. Lu, A. Gokhale, C. Gill, D. C. Schmidt, Adaptive failover for real-time middleware with passive replication, in: Real-Time and Embedded Technology and Applications Symposium, 2009, pp. 118–127.

[28] I. Schaefer, A. Poetzsch-Heffter, Compositional reasoning in model-based verification of adaptive embedded systems, in: IEEE Int. Conf. on Software Engineering and Formal Methods, 2008, pp. 95–104.

[29] I. Schaefer, Integrating formal verification into the model-based development of adaptive embedded systems, PhD Thesis (2008) 1–276.

[30] M. Ahmad, Modeling and verification of functional and non functional requirements of ambient, self adaptative systems, PhD Thesis (2013) 1–115.

[31] N. Khakpour, S. Jalili, C. Talcott, M. Sirjani, M. Mousavi, Formal modeling of evolving self-adaptive systems, Science of Computer Programming 78 (1) (2012) 3–26.

[32] M. Trapp, R. Adler, M. Förster, J. Junger, Runtime adaptation in safety-critical automotive systems, in: Software Engineering, 2007, pp. 1–8.

[33] M. Hachicha, R. B. Halima, A. H. Kacem, Design and timed verification of self-adaptive systems, in: IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), 2017, pp. 227–232.

[34] G. Callow, Extending relational model transformations to better support the verification of increasingly autonomous systems, PhD Thesis (2013) 1–381.

[35] J. Meyer, Modellgetriebene skalierbarkeitsanalyse von selbst-adaptiven komponentenbasierten softwaresystemen in der cloud, in: University of Paderborn, Masterarbeit, 2011, pp. 1–131.

[36] M. Luckey, G. Engels, High-quality specification of self-adaptive software systems, in: Proc. of the 8th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2013, pp. 143–152.

[37] R. Mateescu, P. Poizat, G. Salaun, Adaptation of service protocols using process algebra and on-the-fly reduction techniques, IEEE Transactions on Software Engineering 38 (4) (2012) 755–777.

[38] F. Kŕikava, Domain-specific modeling language for self-adaptive software system architectures, PhD Thesis (2013) 1–251.

[39] F. Alvares, E. Rutten, L. Seinturier, Behavioural model-based control for autonomic software components, in: IEEE International Conference on Autonomic Computing (ICAC), 2015, pp. 187–196.

[40] F. Kŕikava, P. Collet, R. B. France, Actress: Domain-specific modeling of self-adaptive software architectures, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, 2014, pp. 391–398.

[41] A. Bartel, B. Baudry, F. Munoz, J. Klein, T. Mouelhi, Y. L. Traon, Model driven mutation applied to adaptative systems testing, in: IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2011, pp. 408–413.

[42] J. Zhang, B. H. Cheng, Model-based development of dynamically adaptive software, in: Proc. of the 28th Int. Conf. on Software engineering, 2006, pp. 371–380.

[43] C. Ghezzi, L. S. Pinto, P. Spoletini, G. Tamburrelli, Managing non-functional uncertainty via model-driven adaptivity, in: Proc. of the 2013 Int. Conf. on Software Engineering, 2013, pp. 33–42.

[44] Z. Yang, Z. Jin, Z. Li, Achieving adaptation for adaptive systems via runtime verification: A model-driven approach, 2017, pp. 1–20.

[45] S. Becker, S. Dziwok, C. Gerking, C. Heinzemann, S. Thiele, W. Schäfer, M. Meyer, U. Pohlmann, C. Priesterjahn, M. Tichy, The MechatronicUML design method-process and language for platform-independent modeling, in: Technical Repeport tr-ri-14-337, 2014, pp. 1–368.

[46] J. F. J.F. Inglés-Romero, C. Vicente-Chicote, Towards a formal approach for prototyping and verifying self-adaptive systems, in: International Conference on Advanced Information Systems Engineering, 2013, pp. 432–446.

[47] F. Fleurey, A. Solberg, A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems, in: Model Driven Engineering Languages and Systems, 2009, pp. 606–621.

[48] L. Tan, Model-based self-adaptive embedded programs with temporal logic specifications, in: Int. Conf. on Quality Software (QSIC'06), 2006, pp. 151–158.

[49] M. U. Iftikhar, D. Weyns, Activforms: Active formal models for self-adaptation, in: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2014, pp. 125–134.

[50] A. Klarl, Engineering self-adaptive systems with the role-based architecture of Helena, in: IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2015, pp. 3–8.

[51] A. Bucchiarone, H. Ehrig, C. Ermel, P. Pelliccione, O. Runge, Modeling and analysis of self-adaptive systems based on graph transformation, in: Die Professoren der Fakultt IV, Elektrotechnik und Informatik, 2013.

[52] T. Eckardt, C. Heinzemann, S. Henkler, M. Hirsch, C. Priesterjahn, W. Schäfer, Modeling and verifying dynamic communication structures based on graph transformations, Computer Science-Research and Development 18 (1) (2013) 3–22.

[53] B. Becker, H. Giese, Modeling of correct self-adaptive systems: a graph transformation system based approach, in: Proc. of the 5th Int. Conf. on Soft computing as transdisciplinary science and technology, 2008, pp. 508–516.

[54] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling, Symbolic invariant verification for systems with dynamic structural adaptation, in: Proc.of the 28th Int. Conf. on Software engineering, 2006, pp. 72–81.

[55] M. Luckey, B. Nagel, S. Nair, S. Seshadri, C. Thanos, J. Rybka, Quality assurance for adaptation in business processes (2013) 1–63.

[56] T. Vogel, H. Giese, Language and framework requirements for adaptation models, 2011, pp. 1–12.

[57] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, J. Jézéquel, Modeling and validating dynamic adaptation, in: International Conference on Model Driven Engineering Languages and Systems, 2008, pp. 97–108.

[58] J. A. Fails, D. R. Olsen, Interactive machine learning, in: Proc. of the 8th Int. Conf. on Intelligent user interfaces, 2003, pp. 39–45.

[59] M. Harman, The relationship between search based software engineering and predictive modeling, in: Proc. of the 6th Int. Conf. on Predictive Models in Software Engineering, 2010, pp. 1–13.

[60] S. A. Seshia, N. Sharygina, S. Tripakis, Modeling for verification, 2014, pp. 1–29.

[61] N. Kahani, M. Bagherzadeh, J. Dingel, J. Cordy, The problems with eclipse modeling tools: a topic analysis of eclipse forums, in: Proc.of the ACM/IEEE 19th Int. Conf. on Model Driven Engineering Languages and Systems, 2016, pp. 227–237.

[62] S. Winkler, J. Pilgrim, A survey of traceability in requirements engineering and model-driven development, Software and Systems Modeling (SoSyM) 9 (4) (2010) 529–565.

[63] B. H. Cheng, K. I. Eder, M. Gogolla, L. Grunske, L. Litoiu, H. A. Muller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, Using models at runtime to address assurance for self-adaptive systems, in: Models run time, 2014, pp. 101–136.

[64] Y. Lakhnech, S. Bensalem, S. Berezin, S. Owre, Incremental verification by abstraction, in: Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, 2001, pp. 98–112.

[65] H. Giese, A formal calculus for the compositional pattern-based design of correct real-time systems, in: Technical Repeport tr-ri-03-240, 2003, pp. 1–36.

[66] E. M. Clarke, D. E. Long, K. L. McMillan, Compositional model checking, in: Logic in Computer Science, 1989, pp. 353–362.

[67] A. Ferrari, A. Fantechi, S. Gnesi, Lessons learnt from the adoption of formal model-based development, in: NASA Formal Methods, 2012, pp. 24–38.

[68] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: Logics and models of concurrent systems, 1985, pp. 123–144.

[69] P. Wolper, Temporal logic can be more expressive, Information and control 51 (1) (1983) 72–99.

[70] E. M. Clarke, O. Grumberg, D. Peled, Model checking, 1999.

[71] J. Baeten, A brief history of process algebra, Theoretical Computer Science 335 (2) (2005) 131–146.

[72] S. A. Cook, The complexity of theorem-proving procedures, in: Proc.of the third annual ACM symposium on Theory of computing, 1971, pp. 151–158.

[73] F. Nielson, H. R. Nielson, C. Hankin, Type and effect systems, in: Principles of Program Analysis, 1999, pp. 283–363.

[74] A. Bucchiarone, H. Ehrig, C. Ermel, P. Pelliccione, O. Runge, Rule-based modeling and static analysis of self-adaptive systems by graph transformation, in: Software, Services, and Systems, 2015, pp. 582–601.

[75] L. Apfelbaum, J. Doyle, Model based testing, 1997, pp. 296–300.

[76] G. Blair, N. Bencomo, R. B. France, Models@run.time, Computer 42 (10) (2009) 22–27.

[77] A. Bennaceur, R. France, G. Tamburrelli, T. Vogel, P. J. Mosterman, W. Cazzola, F. M. Costa, A. Pierantonio, M.Tichy, M. Akşit, P. Emmanuelson, Mechanisms for leveraging models at runtime in self-adaptive software, Models@run.time 8378 (2014) 19–46.

[78] P. K. McKinley, M. Sadjadi, E. P. Kasten, B. H. C. Cheng, Composing adaptive software, in: IEEE Compute, 2004, pp. 56–64.