Assurance Using Models at Runtime for Self-Adaptive Software Systems

Kerstin I. Eder[1], Norha M. Villegas[2,3], Frank Trollmann[4], Patrizio Pelliccione[5], Hausi A. Müller[2], Daniel Schneider[6], Lars Grunske[7], Bernhard Rumpe[8], Marin Litoiu[9], Anna Perini[10], Martin Gogolla[11], Nauman A. Qureshi[12], Betty H.C. Cheng[13]

[1] University of Bristol, UK
Kerstin.Eder@bristol.ac.uk
[2] University of Victoria, Canada
{nville,hausi}@cs.uvic.ca
[3] Icesi University, Colombia
[4] TU Berlin, Germany
Frank.Trollmann@dai-labor.de
[5] Università degli Studi dell'Aquila, Italy
patrizio.pelliccione@univaq.it
[6] Fraunhofer IESE - Kaiserslautern, Germany
daniel.schneider@iese.fraunhofer.de
[7] TU Kaiserslautern, Germany
grunske@informatik.uni-kl.de
[8] RWTH Aachen, Germany
rumpe@se-rwth.de
[9] York University, Canada
mlitoiu@yorku.ca
[10] CIT- FBK - Povo Trento, Italy
perini@fbk.eu
[11] Universität Bremen, Germany
gogolla@informatik.uni-bremen.de
[12] Fondazione Bruno Kessler - Trento, Italy
qureshi@fbk.eu
[13] Michigan State University, US
chengb@cse.msu.edu

Abstract. Self-adaptive software systems modify their behaviour at runtime in response to changes in the system or its environment. The fulfilment of the system requirements and reachability of the system goals needs to be guaranteed even in the presence of adaptations. Thus, a key challenge for self-adaptive software systems is assurance. Traditionally, confidence in the correctness of a system is gained during system development. Evidence to support assurance is collected based on a variety of activities and processes performed at development time. In the presence of self-adaptation, however, some of the assurance tasks need to be performed at runtime. This calls for continuous assurance throughout the software life cycle. One of the most promising avenues of research in this area is to use models at runtime as a foundation for developing runtime assurance techniques. This chapter focuses on investigating the use of models at runtime for assurance of self-adaptive software systems. It defines what we understand by a model at

runtime, specifically for the purpose of assurance, and puts this definition into the context of existing work. We then outline selected research challenges. The chapter concludes with an exploration of selected application areas where models at runtime could contribute a considerable value proposition compared to existing techniques.

1 Introduction

Self-adaptive software (SAS) systems modify their behaviour at runtime in response to changes in the system or in its environment.[1] A SAS system generally consists of the part that delivers the basic function or service, often referred to as the *target* or *managed system*, and the part that manages or controls that target system through an *adaptation process*, often referred to as the *controller* [MAB+02] or *autonomic manager* [KC03]. The target system can be viewed as a steady-state program [ZCG07]. It is not adaptive and applicable for one specific execution environment. The controller is the part of a SAS system that can, via the invocation of an adaptation process, transform this steady-state program to a different steady-state program—one that is suitable for a different set of environment constraints. As such, the steady-state program that delivers the basic function or service of a SAS system is the target of the adaptation process that is managed by the controller. For non-adaptive systems design-time assurance techniques are used to ensure requirements conformance. Design-time assurance comprises several disciplines including software quality engineering, assurance and control; software safety; software reliability; software security; as well as software verification and validation. In the presence of runtime adaptations in a SAS system the fulfilment of the system requirements and the reachability of the system goals need to be guaranteed at runtime. Thus, a key challenge for SAS systems is to develop runtime assurance techniques.

The IEEE Standard Glossary of Software Engineering Terminology defines *assurance* as "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements" [IEE90]. [2] Since a SAS system modifies itself at runtime due to changes in its execution environment or its functional and non-functional requirements, assurance tasks necessary to provide confidence that the software conforms to its goals must be performed not only at design time but also at runtime—either continuously or whenever the system adapts.

In practice, assurance tasks may comprise verification, validation, test, measurement, conformance to standards, certification and other tasks—all contributing towards gaining a high degree of confidence that both the processes employed and the end product satisfy established technical requirements, standards, and procedures. Given the increasing use of self-adaptive software systems in high-assurance applications (e.g.,

[1] This chapter uses the acronym SAS to refer to any software application that exposes self-* features.

[2] This chapter uses the term *software assurance* rather than the more specific term *software quality assurance* to not only include software quality concerns but also safety, reliability and security concerns.

power-grid management, transportation management systems, telecommunication systems and health-monitoring), it is of paramount importance to develop rigorous methods and techniques for the runtime assurance of these systems.

Assurance is required for both, functional properties (i.e., those describing specific functions of the system such as the result of a calculation) and non-functional properties (i.e., those describing the operational qualities of the system such as availability, efficiency, performance, reliability, robustness, security, stability, and usability). Guaranteeing these properties at runtime in SAS systems is particularly challenging due to the varying assurance needs posed by a changing system or execution environment, both fraught with uncertainty. Nevertheless, the properties specified in the system requirements need to hold *before*, *during*, and *after* adaptation.

Software assurance for traditional infomation systems is chiefly a software design or development concern. For highly dynamical or self-adaptive systems continuous assurance over the entire life cycle is essential. Thus, for software systems that change or evolve at runtime, software assurance becomes a critical runtime obligation. Continuous assurance throughout the entire software life cycle provides unprecedented opportunities for monitoring, analysing, guaranteeing, and predicting system properties throughout the operation of a software system. The fact that many variables that are free at development time are bound at runtime allows us to tame the state space explosion, thus enabling the investigation of states that could not have been considered at development time. This provides new opportunities for runtime verification and validation (V&V) leading to assurance of critical system properties at runtime [TVM+12]. One of the most promising avenues of research in this area is to use models at runtime as a foundation for developing runtime assurance techniques.

This chapter presents models at runtime as a foundation for the assurance of SAS systems and discusses related research challenges. Section 2 reviews assurance criteria, both functional and non-functional, whose fulfilment depends on or can be affected by self-adaptivity and therefore requires assurance at runtime. Section 3 classifies models at runtime and discusses the application of runtime models from two perspectives. The first one refers to the use of models at runtime at the three levels of dynamics that have been identified as important drivers in the engineering of SAS systems. The second perspective involves the autonomic or MAPE-K loop as a reference model for engineering adaptation mechanisms, and concerns the application of runtime models to support assurance tasks throughout the adaptation process. Section 4 identifies research challenges in the area of runtime models for SAS system assurance tasks. Section 5 characterizes existing methods used for assurance of SAS systems. Section 6 describes selected application areas that exhibit the type of assurance challenges that we consider can be addressed using models at runtime. Finally, Section 7 concludes the chapter.

2 Assurance Criteria for Self-Adaptive Software Systems

Assurance criteria for SAS systems include functional and non-functional requirements whose fulfilment depends on or can be affected by self-adaptation. It is important to distinguish between assurance criteria applicable to the *target system* (i.e., criteria that relate to properties of the current or a potential future state of that system), and assur-

ance criteria applicable to the *adaptation process* itself. Sections 2.1 and 2.2 discuss functional and non-functional requirements as fundamental assurance criteria for SAS systems.

2.1 Functional Requirements

A functional requirement specifies a function that a system or system component must be able to perform [IEE90]. Functional requirements are typically formulated as prescriptive statements to be satisfied by the system. While it is still common in practice to describe functional requirements using natural language, the potential for misinterpretation of such descriptions is considerable due to the inherent ambiguity of natural languages. Formal languages with well defined semantics provide a far more rigorous and reliable means for specifying functional requirements in the context of system design. We will therefore restrict our discussion to formal descriptions.

Functional requirements directly relate to the functions f of a system. They are typically defined in terms of relating the inputs I to the system with the outputs O of the system, with the expectation that $f : I \rightarrow O$ As such, functional requirements describe "what" the system has to provide to meet the expectations of its users, while non-functional requirements describe "how" such functional services should be provided.

The respective function may be a calculation, a dialogue flow in user interaction, data manipulation or other specific functions the system should execute. Taking the variety of systems into consideration, the input may be human commands as well as sensor input such as temperature or video streams. Similarly, the output may be pictures or continuous video as well as the correct execution of an actor, stopping a car upon activating its brake or opening of a valve. It should be noted that some systems produce continuous output based on continuous streams of inputs and typically never terminate. Functional requirements always deal with the system behaviour visible at the system boundaries (i.e., system interfaces). These boundaries can be humans, sensors, and actuators but also interacting systems.

Adaptivity of a system may become necessary to cope with changes in the environment, which are visible at the boundaries and influence the system's behaviour externally, or changing requirements, which leads to internal changes within the system, becoming observable at the system boundaries. While the former is a reaction to the system context and leads to retaining the functional behaviour in the presence of external change, the latter is a reaction to human (or system configuration) needs and leads to behavioural adaptations to accommodate the new requirements.

The specification of functional requirements can either concentrate on the functional behaviour of the system itself and disregard the context, thus assuming that the system is robust against any context variation. Alternatively, functional requirements can take into account the context of the system as well as explicit assumptions about its behaviour. The functional requirements are formalized in an "assume/guarantee" style—assuming a set of conditions or restrictions holds, then the application of the function guarantees that the results satisfy a set of properties. The definition of pre- and postcondition is an example of this style of functional requirements specification. The precondition takes

the input and the system state into account, but does not typically allow full definition of context behaviour.

While functional requirements are always system specific, it is possible to classify them into distinct sets of properties. The most commonly used classification was introduced by Alpern and Schneider who distinguish between *safety properties* that state that "nothing bad" will happen during the execution of a system and *liveness properties* that stipulate that "something good" will eventually happen during the execution of a system [AS87]. Safety and liveness properties have an important difference when it comes to demonstrating that they hold for a given system. If a system violates a safety property, then there exists a finite execution of the system to demonstrate this. This does not hold for liveness properties. A liveness property cannot be violated based on finite executions of a system; this is because, irrespective of the system behaviour captured in the finite execution, something good could still happen later. In general, the verification of liveness properties requires infinite executions. This is a fundamental difference between these two classes of properties with profound consequences on the methods suitable to verify them. In practice, liveness properties are often constrained to be within bounds to enable verification.

The kind of system and the level of detail required are key factors that determine how functional requirements are described. Behavioural specifications can rely on recent input and output, which is useful for stateless services, but can include the full history of inputs and outputs (e.g., history streams [BS01]) or an abstraction thereof, using states to capture the input history ((i.e., various kinds of state-based specifications). Behavioural specifications may abstract from time (event driven) or include various forms of time, such as equidistant time slices as in synchronous or clocked systems, continuous time as in mathematical calculus and even super dense time, where there may exist several events at the same time, but with internal order [Lee09].

Common formalisms used to express functional requirements are Linear-Time Temporal Logic (LTL) [Pnu81] and Computational Tree Logic (CTL) [BAMP81], both included in the powerful logic CTL* [CE82]. Several languages have been proposed to facilitate the specification of functional properties, examples range from basic assertion languages such as PSL [Acc04], used in electronic system design, to scenario-based visual languages, such as Message Sequence Charts [HT04] or Property Sequence Charts [AIP07]. These languages are often less expressive than pure temporal logic, but are designed to be intuitive and user friendly.

Beyond property-based specification, various algebraic specification and system modelling techniques have been developed including Statecharts [], UML and its various descendants, SDL [], VDM [], Z [], the B Method [], Event-B [ABH$^+$10], architectural description languages [], Mathlab/Simulink [] to name a few representative examples. Many of these techniques support automatic code generation from the system model as well as formal reasoning about the model at varying levels of abstraction. Traditionally, these techniques are used during system design to ensure the system

Until now we only considered the system boundaries. However, today many systems are decomposed into distributed subsystems and components. Sometimes they need physical distribution as in plants or airplanes to control the overall system, sometimes they need computational or storage distribution as in large data bases or the cloud.

Sometimes the distribution is for reliability reasons (e.g., allowing redundancy or separating control processors from controlled computations).

To be able to describe adaptive systems, it is important that the formalism allows certain forms of under-specification, but also allows to constrain this freedom of the implementation accordingly. First of all this is important to allow the modelling of bandwidths of acceptable behaviour. Only if the system violates this bandwidth, adaptation needs to take place. Second, if the system is itself distributed, non-determinism naturally occurs and thus must be captured. Third, not all inputs are important enough to be taken into account in a systems specification, but some slight change of the behaviour may occur dependent on invisible inputs.

As a remark, we can view the system context as another system comprising the rest of the world. Just the inputs and outputs are inverted. Having a specification mechanism that allows us to describe under-specification, we are able to describe the unconstrained behaviour of the context.

2.2 Non-Functional Requirements

Chung *et al.* [CPL09] discuss definitions and possible classifications of non-functional or extra-functional requirements. Their working definition states that while functional requirements directly relate to a function $f : I \rightarrow O$ of the software system, non-functional requirements are just about anything that addresses characteristics of f, I, O or relationships between I and O. Due to the fact that several of these characteristics end with the suffix "-ility" (e.g., usability, dependability, verifyability, availability, interoperability, and scalability), they are sometimes referred to as the *-ilities*.

Non-functional requirements such as performance, dependability, safety, security, and their corresponding quality attributes such as latency, throughput, capacity, confidentiality, and integrity can constitute assurance concerns from the perspective of both the target system and the adaptation mechanism. Avižienis *et al.* [ALRL04] and Barbacci *et al.* [BKLW95] provide two comprehensive taxonomies of software quality attributes useful for the identification of assurance criteria in SAS systems.

It is necessary to validate and continually monitor non-functional requirements on both the target system and the adaptation process, for example via probabilistic monitoring [GZ09,Gru11]. The desired properties of the target system can change due to changes in the target systems context-of-use (e.g., user, platform or environment context [SCF$^+$06]), as well as changes introduced by adaptations. In the latter case, it is partially possible to derive the impact of adaptations on properties of the target system by analysing adaptation properties such as stability, accuracy, settling time, small overshoot, and robustness. It is possible to take advantage of this relation to detect some of the consequences of adaptations performed by controllers (i.e., monitors, analysers, planners, and executors) or imposed by a changing environment (e.g., a failing component or a deficient Internet connection).

There are several non-functional assurance criteria which can be better guaranteed at runtime than at design time. For example, it is easier to assess latency when it is possible to measure and continually monitor delay times in the running system. Table 1 presents examples of non-functional assurance criteria with corresponding quality

attributes (cf. Columns 1 and 2). Adaptation properties (cf. Column 3), defined as assurance criteria that concern the adaptation process [VMT+11b], can be mapped to quality attributes measurable at runtime and at both the target system and the adaptation mechanism. Where to measure a property, either of the adaptation process or the target system, will depend on its definition and its assessment metric. For example, settling time, defined as the time required for the adaptation process to take the target system to a desirable state, must be measured on the target system since the need for the adaptation and the conditions for a desired state can be observed at this level only. Moreover, settling time can be measured through different quality attributes. It will depend on the non-functional property that must be satisfied. For example, if the concern is performance, settling time can be observed in terms of the time the system takes to perform a particular process. When the accepted time limit for this process is violated, the adaptation process will be invoked. Once the process time is back to the desired limits, the target system has reached its desired state. Settling time will then be the time elapsed between the moment at which the need for adaptation was detected and the moment at which the system reached the accepted processing time. Villegas *et al.* provide a more comprehensive catalogue of adaptation properties and corresponding quality attributes useful to drive the identification of assurance criteria applicable to the adaptation process [VMT+11b]. This study also surveys definitions for the assurance criteria presented in Table 1.

Table 1. Examples of non-functional assurance criteria that can be better guaranteed at runtime than at design time as well as their mapping to quality attributes and adaptation properties. A mapping between adaptation properties and quality attributes is suggested as a starting point toward the identification of metrics that supports the evaluation of the adaptation process [VMT+11b].

Assurance Criteria (non-functional)	Quality Attribute	Adaptation Properties
Latency	Performance	Stability, accuracy, settling time, overshoot, scalability
Throughput	Performance	Stability, accuracy, settling time, overshoot, scalability
Capacity	Performance	Stability, accuracy, settling time, overshoot, scalability
Safety	Dependability	Stability
Availability	Dependability	Robustness, settling time
Reliability	Dependability	Robustness
Confidentiality	Security	Security

Assuring these criteria at runtime requires effective monitoring mechanisms and models at runtime to analyse guarantee and predict the qualities of the target system and the adaptation process dynamically. To implement these mechanisms effectively

requires a solid understanding of the interdependencies between non-functional assurance criteria, quality attributes and adaptation properties as exemplified in Table 1. This mapping constitutes a valuable starting point to identify assurance criteria and adaptation properties. On the one hand, this mapping supports the identification of assurance criteria according to the target system's desired quality attributes. For example, latency, throughput and capacity are relevant assurance criteria when performance is the negotiated quality attribute. On the other hand, it is useful for the identification of adaptation properties that must be guaranteed in the adaptation mechanism according to quality attributes. For example, when performance is a key quality attribute for the target system, stability, accuracy, settling time, small overshoot and scalability constitute relevant properties to be guaranteed in the adaptation process. Of course these mappings also depend on the actual target system, its technical implementation, and the performed adaptations.

3 Models at Runtime (MART)

SAS systems require rethinking the notion of software life cycle for which the distinction between development time and execution time stages is not meaningful any more (e.g., PLASTIC,[14], SMScom[15]). Recent approaches recognize the need to produce, manage and maintain software models all along the software's life time to assist the realization and validation of system adaptations while the system is in execution [Inv07,BBF09,BG10,ACDR+11].

By building on this conclusion, our aim is to exploit models of different aspects of the application (e.g., requirements, specification, design, architecture, implementation, infrastructure, instrumentation, and context-of-use) and lifecycle phases (e.g., design time, development time, configuration time, load time, and runtime) to deal with the inherent dynamics of self-adaptation in software systems. These abstractions, combined with suitable instrumentation, could provide effective techniques for monitoring, analysing, guaranteeing, and predicting system properties throughout the operation of a SAS system.

The kind of models used at runtime can be classified by (1) their purpose—descriptive, prescriptive, constructive, or predictive, (2) their underlying model languages—for example, the 14 UML 2.2 structural and behavioural diagrams, Statecharts, Petri Nets, and logic based models, such as Temporal Logics, and (3) the aspects they describe—data structure, task or process state, I/O behaviour or interaction pattern.

One of the main ideas of using models at runtime (MART) for assurance is to exploit the causal connection [Mae87] between the model and its system under study at runtime. This connection determines synchronization between the model and the running system. On the one hand, runtime models can change to reflect changes in the running system correctly—we say that they are in descriptive causal connection. This enables assurance techniques to analyse abstract models instead of the actual implementation of the application to collect information for assurance. On the other hand, the model can

[14] FP6 IST EU PLASTIC project http://www.ist-plastic.org/.
[15] Carlo Ghezzi, ERC Advanced Investigator Grant N. 227977 [2008-2013]

be changed to cause an adaptation of the application (i.e., prescriptive causal connection). This can be used to implement adaptations of the running system that are required to assure system properties.

In the scope of assurance, models at runtime can be used as a basis for assuring functional as well as non-functional properties of the system as described in Section 2. From this perspective, models can have various roles. Depending on what the models describe, they can be used as a source of information about aspects of the running system. For instance, goal models can represent the requirements that need to be assured, the current state of the system, adaptations or the context-of-use. Models at runtime can have several purposes for runtime assurance. Among others, they can be used as information sources for monitoring aspects of the running system, to influence the system via model manipulation, and as a basis for analysis methods such as model-based verification and model-based simulation. For analysis methods, models are usually beneficial as they provide easy-to-access high level knowledge about the system.

Development time modelling approaches already exploit these advantages and enable the assertion of certain properties of a developed system. The use of models at runtime has the advantage that some of the analysis constraints are relaxed as the current runtime state is available for reasoning, reaction, and regulation. At development time full assurance is required to reason about all possible states. Several of these variables which are unknown at development time are bound at runtime and can allow for a more focused analysis of the current state and possibly several neighbouring ones. This is especially useful for factors which can only be estimated at development time (e.g., network delay). A running system can continually monitor these aspects and react to them.

The following subsection discusses the role that models at runtime can play to support the assurance of SAS systems at three levels of dynamics that have been identified as important drivers in the engineering of SAS systems. Subsection 3.2 introduces a way to integrate runtime models into feedback loops for continuing runtime assurance.

3.1 Models at Runtime and the Dynamics of Self-Adaptive Software

Researchers from the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) research community identified three subsystems that are key in the design of effective context-driven self-adaptation: the control objectives manager, the adaptation controller, and the context monitoring system [VTM+12]. These subsystems represent three levels of dynamics in self-adaptation that can be controlled through a feedback loop each, i.e. the *control objectives*, the *adaptation*, and the *monitoring* feedback loops respectively. Villegas *et al.* provide a comprehensive characterization of these three levels of dynamics in SAS systems[VTM+12].

Assurance criteria in general drive the control objectives, adaptation, and monitoring feedback loops, as well as their interactions, and thereby, govern the behaviour of both the target system and the adaptation process. For example, system administrators can provide the control objectives manager with the required specifications. Then, the control objectives manager sends these specifications, in the form of adaptation goals, to the adaptation controller, and, in the form of monitoring requirements, to the monitor-

ing system. Thus, these specifications govern the behaviour of the adaptation process, and the behaviour of the SAS system through the adaptation process.

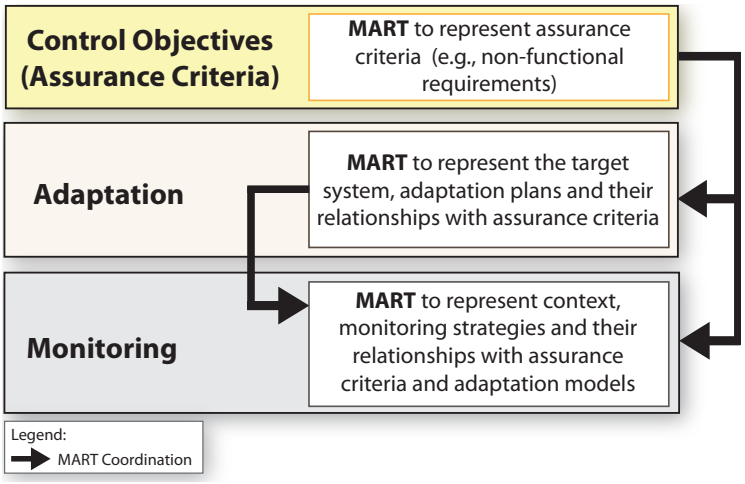

Fig. 1. The three levels of models at runtime (MART) for the assurance of SAS systems.

We argue that runtime models provide abstractions that are crucial to support the feedback loops that control the three levels of dynamics identified in SAS systems. From this perspective, models at runtime (cf. MART in Figure 1) could be developed specifically for each level of dynamics to support the control objectives manager, adaptation controller, and the monitoring system as depicted in Figure 1. The figure also shows the interactions between these models and between the respective subsystems in a SAS system.

At the *Control Objectives* level, models at runtime represent requirements specifications subject to assurance in the form of functional and non-functional requirements. At the *Adaptation* level, models at runtime represent states of the managed system, adaptation plans and their relationships with the assurance specifications. At the *Monitoring* level, models at runtime represent context entities, monitoring requirements, as well as monitoring strategies and their relationships with assurance criteria and adaptation models. Most importantly, runtime models at these levels must have efficient and effective methods of interaction between them. This is because changes in requirement specifications may trigger changes at both the adaptation and the monitoring levels, and the associated runtime models. Similarly, changes in adaptation models may imply changes in monitoring strategies or context entity models. In any case, runtime models at the adaptation and monitoring levels must maintain an explicit mapping to the models defined at the control objectives level which specify the requirements.

In summary, the architecture of SAS systems contains three interacting but functionally self-contained levels, one each dedicated to control objectives, adaptation and monitoring of the SAS system. Designing a SAS system *for assurance*, as opposed to

leaving assurance to after system design, requires the tight integration of assurance objectives into each level in the SAS architecture. We argue that this can most effectively be achieved by introducing dedicated runtime models that embody specific assurance criteria, focused either on the target system or the adaptation process, and thus support the assurance process at each of these levels.

3.2 Models at Runtime along the Adaptation Process

As a starting point for a research methodology we analyzed the MAPE-K loop. Kephart and Chess proposed this autonomic manager as a foundational component of IBM's autonomic computing initiative [KC03]. It constitutes a reference model for designing and implementing adaptation mechanisms in SAS systems. The MAPE-K loop is an abstraction of a feedback loop where the dynamic behaviour of a managed system is controlled using an autonomic manager. The MAPE-K comprises four phases—Monitor (M), Analyzer (A), Planner (P) and Executor (E)—that operate over a knowledge base (K).

1. Monitors to gather and pre-process relevant context information from entities in the execution environment that can affect the desired properties and from the target system;
2. Analysers to support decision making on the necessity of self-adaptation;
3. Planners to generate suitable actions to affect the target system according to the supported adaptation mechanisms and the results of the analysis phase;
4. Executors to implement actions with the goal of adapting the target system; and
5. A knowledge base to enable data sharing, data persistence, decision making, and communication among the components of the feedback loop, as well as arrangements of multiple feedback loops (e.g., the Autonomic Computing Reference Architecture (ACRA) [IBM06]).

To illustrate the role of models at runtime as enablers of assurance mechanisms for self-adaptation, Figure 2 presents an extension of the MAPE-K loop, where assurance tasks complement each stage of the loop [TVM$^+$12], and the knowledge base is replaced by models at runtime (MART). We aptly name the feedback loop depicted in this figure *MAPE-MART loop*.

MAPE elements interact with models at runtime along the adaptation process to either obtain or update information about system states, the environment, and assurance criteria. *Monitors* keep track of relevant context information according to monitoring conditions in the system itself (*assurance monitors*) and its adaptations (*adaptation monitors*). Monitors interact with models at runtime for example to make monitored data available throughout the adaptation process, or to monitor the states of models or changes in assurance criteria. *Analysers* will then use monitored context to identify whether desired conditions are being or could potentially be violated. Analysers can also update models with identified symptoms. Again, we can distinguish between *assurance analysers* analysing the system itself and *adaptation analysers* analysing the adaptation. *Adaptation planners* use the symptoms provided by analysers to define a new adaptation plan. Adaptation plans can be defined in the form of models that are

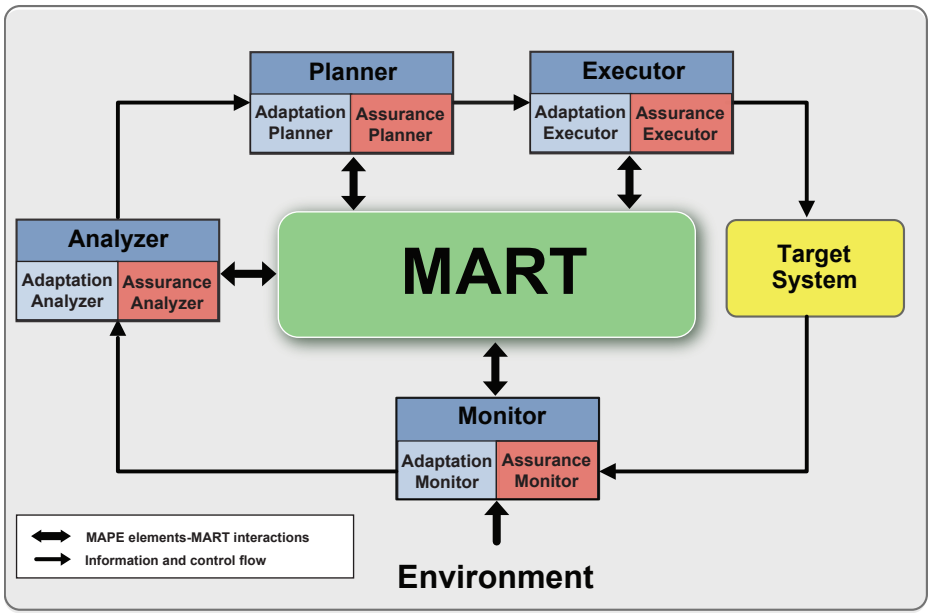

Fig. 2. MAPE-MART loop: The MAPE-K loop from autonomic computing extended with models at runtime, and assurance instrumentation as foundational elements for the assessment of SAS systems.

processable by executors to adapt the target system. Then, *assurance planners* check whether the plan is correct with respect to the assurance criteria. Finally, adaptation executors perform the plan and after that, *assurance executors* check whether both the system remains in a safe state and the desired properties are accomplished. These verifications can be optimized using runtime models.

The assurance of the target system is performed by the adaptation mechanism itself. This is what most contributions on SAS systems have done so far: implementing an adaptation mechanism to control the dynamic behaviour of a target system to satisfy requirements [VMT+11b]. Nevertheless, as depicted in Figure 2, assurance must be implemented throughout the adaptation mechanism to guarantee desired properties even during adaptation. As an example of assurance intended to guarantee the adaptation process, suppose *settling time*, the time required for the adaptation mechanism to take the target system to the desired state, has been defined as the assurance concern for a particular adaptive system. Thus, assurance mechanisms must keep track of the time the adaptation mechanism is taking to complete the adaptation process—generally goals must be reached within a suitable time interval. An extremely long adaptation process could be useless or even detrimental to the system's safety. The desired thresholds, monitoring conditions, and entities to be monitored can be specified using models at runtime. For instance, using goal-based models [WSB+09] or contextual RDF graphs [VMT11a]. Executor mechanisms for dealing with this problem may include

switching to a more effective adaptation algorithm, or adapting models used throughout the process.

The following section analyses short- and long-term challenges related to the application of models at runtime in the assurance of SAS systems. That is, the use of models at runtime as enablers for assurance tasks in the adaptation process as depicted in Figure 2.

4 Research Challenges for Assurance at Runtime

This section presents selected research avenues and research challenges for the assurance of SAS systems using runtime models.

4.1 Research Avenues

Software assurance is a large field with many subfields (e.g., software quality, V&V, safety, and several 'ilities') that spans the realms of software engineering, systems engineering, control engineering, and many other engineering disciplines. From a software engineerg perspective assurance at *runtime* appears to be a rather recent research topic. In contrast runtime assurance in control engineering traces its roots to the industrial revolution, to devices such as the centrifugal governor. This device used a flyball mechanism to sense the rotational speed of a steam turbine and adjust the flow of steam into the machine. By regulating the turbines speed, it provided the safe, reliable, consistent operation that enabled the proliferation of steam-powered factories [MAB+02].

In an instrumented, interconnected and intelligent world where not just everyone is conversing, but also every thing is talking to every other thing constantly, control and runtime assurance are core components in dynamical systems, providing high performance, high confidence, and reconfigurable operation in the presence of uncertainties. The continuous integration of sensors, networks, cloud computing, and control presents significant opportunities for engineering in general and software engineering in particular. A key goal is to provide certifiable trust in resulting systems which is a truly formidable challenge for runtime software assurance.

Over the past 20 years, several research venues (i.e., journals, conferences, and workshops) have emerged in the broad software engineering research community to discuss the design and evolution as well as assurance of self-* systems including self-adaptive systems, self-managing systems, self-healing, autonomic systems, self-organizing systems as well as real-time systems.

Mining the rich histories, theories and experiences of fields such as biology, control engineering, and software engineering are worthwhile starting points for assurance at runtime research. In particular, we need survey papers that investigate models used for design time and runtime assurance techniques in these fields including discussions on the synergy between them. Moreover, it is useful to relate canonical practical applications to these findings. In a most stimulating 2002 control survey paper Murray *et al.* posit that in modern control feedback is a central tool for uncertainty management. By measuring the operation of a system, comparing it to a *reference*, and adjusting available control variables, the controller can assure proper operation even if its dynamic

behaviour is not fully known or external disturbances its behaviour. In software this reference can be realized with runtime models and evidence for assurance is gathered by checking conformance to the reference model. Murray *et al.* argue that the challenge is to go from the traditional view of control systems as a single process with a single controller, to recognizing control systems as a heterogeneous collection of physical and information systems, with intricate interconnections and interactions [MAB+02]. One manifestation of this is the three levels of runtime control models discussed in Section 3.

The field of self-adaptive and self-managing systems has produced a spectrum of runtime models and patterns with control-centric models at one end and architecture-centric models at the end other. These models come with different attributes and properties that can be exploited for runtime assurance. There is plenty of room for research to comparopen esearch probe and evaluate the benefits and synergy of these different runtime model strategies [MKS09].

4.2 Selected Research Challenges

This section outlines selected open research problems and challenges encountered along the research avenues presented in the previous section.

- *Characterization of assurance criteria according to the separation of concerns.* Section 2.2 introduces a partial characterization of assurance criteria in the scope of non-functional requirements. These should be completed. What are the properties to be ensured with respect to the target system? What are the properties to be ensured with respect to the adaptation process? How do they relate? Which models is their assurance based on?
- *Analysis of methods and techniques according to assurance criteria.* Functional and/or non-functional requirements constitute the goals for self-adaptation. The effective application of models at runtime, as well as assurance methods and techniques depends on whether the aim is to guarantee functional or non-functional requirements. What are suitable methods and techniques to ensure functional requirements using models at runtime? What are suitable techniques to ensure non-functional requirements? How can the user be integrated into the assurance loop for both functional and non-functional requirements?
- *Analysis of assurance criteria, methods and techniques according to the SAS system's life cycle.* It is clear that not all assurance tasks can be shifted to run time. Factors like performance or the available run time memory can severely limit the power of an assurance technique. Thus, it should be explored which assurance tasks can be done in which phase of the SAS system's life cycle. What properties can be guaranteed at development, configuration, or load time as opposed to runtime? How do dynamic models apply to properties that must be guaranteed at runtime? What lightweight techniques can be applied to runtime assurance of SAS systems? How can models at runtime leverage the application of such lightweight techniques? How can we partition assurance concerns optimally throughout the system's life cycle? What are the opportunities to exploit techniques that become applicable at runtime but not at development time? What development-time assurance methods and techniques and models readily extend to runtime?

- *Analysis of models at runtime techniques for the assurance of the adaptation process.* As presented in Fig. 2, models at runtime play an important role as the abstraction mechanisms required to support every stage of the adaptation process. What models at runtime techniques are useful for supporting the relevance of runtime monitoring with respect to the assurance criteria? Moreover, to deal with the dynamic nature of functional and non-functional requirements, and the execution environment, every component of the adaptation process can be an adaptive component as well: how can models at runtime support changes in monitors, planners and executors according to changes in functional and non-functional requirements? What are suitable models at runtime techniques to ensure desired properties taking into account the different levels of dynamics in SAS systems (i.e., changes in requirements, relevant context, adaptation mechanisms, and the target system itself)?

- *Analysis of assurance criteria, models and techniques from the perspective of particular application domains.* In order to provide the developer of a SAS system with an indication of potential assurance criteria, models and techniques their classification with regards to the application domains they can be applied in is very useful. What assurance techniques are application-independent and what are domain-dependent? What properties are relevant for which application domains? What are assurance methods and models at runtime techniques applicable or not applicable in which application domains? (e.g., mission-critical systems, embedded systems, real-time systems).

- *Assurance of models at runtime.* If runtime models form the foundation of assurance tasks the quality of these tasks directly depends on the quality of the models. How do we evaluate models at runtime? What are the properties that must define the quality of a model at runtime (e.g., accuracy, performance, or safety)? How do these properties differ depending on the application domain?

- *Monitoring infrastructure.* One major concern of runtime assurance is the extraction of the required information from evolving models. How can we efficiently monitor models at runtime in order to gather knowledge as a basis for assurance? What kinds of models are suited for which assurance criteria?

- *Compelling reasons for models at runtime.* In order to convince researchers and practitioners to work on this subject we need compelling reasons for assurance using runtime models. What are the compelling advantages of using assurance based on models at runtime in comparison with assurance based on development time models? What are killer applications to demonstrate the challenges and opportunities for assurance using models at runtime?

- *Laws of probability theory and monitoring of extreme probabilities.* Non-functional requirements often need to be statistically quantified. This can lead to additional complications. What are the boundaries of monitoring capabilities for non-functional attributes that need to be statistically quantified? How can we effectively reason about probabilistic quality attributes where the desired probabilities are close to zero or one and thus extreme statistical methods are required?

4.3 Longer-term research challenges

- *Transitioning development-time assurance methods to runtime.* Assurance at runtime, compared to development time, is really a different problem. The state space at runtime is more constrained than at development time in that many free variables are bound at runtime. The challenge is to take advantage of the additional information available at runtime to improve the quality and the efficiency of the assurance assessment. What development-time assurance methods and techniques (i.e., the entire spectrum from light-weight to heavy-weight processing approaches) and models (i.e., descriptive, prescriptive, constructive and predictive) readily extend to runtime? How do traditional assurance models and methods from domains such as performance, safety, and reliability extend to runtime? Since a complete transition will probably not be feasible in most instances, how can the methods be partitioned between development time and runtime?
- *Extending reference models for the assurance of SAS systems.* Reference models have already proven themselves useful in the developement of complex SAS systems. For good integration into existing research runtime assureance and models should be integrated into these reference models. How can we extend reference models for SAS systems to include models and assurance at runtime (e.g., MAPE-K loop)? Can we leverage runtime assurance techniques from other disciplines (e.g., control theory)? Assuming models at runtime what reference architectures are appropriate for assurance at runtime? What are appropriate assurance reasoning techniques for different phases of the software life cycle (i.e., development, installation, load, and runtime) and how can assurance results from different lifecycle phase be combined to assure systems at runtime (e.g., incremental and compositional assurance)? What are ideal applications to demonstrate the challenges and opportunities for assurance using models at runtime?
- *Partitioning runtime assurance.* There is a variety of techniques that can be potentially used for runtime assurance. In order to structure the landscape of possible approaches and provide an overview on which approach can be used in which situation these approaches need to be partitioned. How can we partition runtime assurance according the different types of runtime changes (e.g., dynamic context, changing requirements, or evolving models)? How can we partition runtime assurance with respect to the properties that are assured. How can we partition runtime assurance with respect to the runtime models that are used for assurance. Are there other approaches to partitioning runtime assurance and what do we gain from such delineations?
- *Qualification.* In order to be applicable and trusted at runtime we require reliable tools and techniques. Can we qualify tools/techniques for assurance/adaptation so that they can be trusted to work on the system at runtime? What role do models at runtime play in this context? Would specific models at runtime make it easier to qualify a tool/technique?
- *Developing incremental and compositional runtime assurance methods.* Modern software systems should be more and more designed with adaptation and runtime evolution in mind. But even with good reactions to changes, the triggered adaptation should be performed preserving some properties. This calls for incremental

and compositional assurance for SAS systems. An enabling step, in this direction, is to split functional and non-functional requirements in sub-requirements associated with single services and components of the system. The idea is to decompose the requirement specification into properties associated to the behaviour of small parts of the system; in this way it is possible to check these properties locally, and to deduce from the local checks whether the system satisfies the overall specification. By decomposing the assurance task in such a way, it is not necessary to build a complete model of the system and thus the state explosion problem can be avoided. The main challenge of this approach is that local properties are typically not preserved at the global level essentially because of dependencies among the considered subparts of the system. However, this decomposition promotes lightweight V&V approaches because V&V activities can concentrate exclusively on subparts of the system. Even though this can imply in some cases reducing the V&V power, on the other side this will make V&V practical and efficient.

- *Exploiting the synergy between runtime assurance and runtime V&V.* Once we have mechanisms to assess assurance at runtime there is opportunity assess assurance continuously. In other words, the boundary between assurance and V&V becomes fuzzy. The models for runtime assurance methods and techniques should include properties such as frequency assessments, measures of accuracy, or rates of change. The instrumentation of the SAS system should not only satisfy the needs of the control process and V&V but also assurance needs.
- *Software engineering education for runtime.* Over the past 40 years software engineering education has largely concentrated on design and implementation methods and tools. There are many courses on software maintenance and evolution that deal with issues way past system deployment. However, it is rare that maintenance and evolution is discussed in the context of dynamic adaptation. Software engineering courses that deal with pure runtime issues such as self-adaptation are few and far between. However, to further the adoption of models at runtime techniques such courses build an important foundation. Two key challenges for this entire field is to figure out ways to integrate models, methods, techniques and tools for SAS systems into our existing software engineering courses and develop effective software engineering training methods for traditional software engineers to deal with the challenges of runtime issues.

5 Characterizing Assurance Methods

Researchers from communities related to the engineering of SAS systems have contributed valuable approaches to the state-of-the-art assessment of adaptive software. Rather than producing a comprehensive and systematic literature review of the state of the art, the goal of this section is to provide an overview of selected contributions. This initial characterization of assessment approaches provides a starting point for researchers to build on top of these contributions, or propose new ones, motivated by the challenges posed by the assurance of SAS systems, at runtime, supported by model-based techniques.

5.1 Classifying Assurance Methods According to Techniques

This section presents and classifies selected approaches applicable to the assurance of SAS systems according to techniques and methods used for their realization.

Goal-oriented approaches. The first step towards assuring software systems is the identification of assurance criteria. This task can result more complex for functional requirements because it requires a deep understanding of the application domain. Nguyen *et al.* argue for the effectiveness of goal-oriented techniques for deriving assurance criteria from functional requirements specifications [NPT+09]. At development time (or negotiation time), goal models can be used to specify stakeholder expectations for autonomous systems, and the criteria for deciding about system acceptable behaviour can be derived from these models. Moreover, goals, and especially high-level goals, have been recognized as less volatile than specific system requirements [vLDL98]. Thus, high-level goals seem to offer suitable candidate assurance criteria in highly dynamic systems. On this assumption relies the work on continuous requirements engineering by Qureshi *et al.* [QJP11,QLP11,QP10]. In their work, high-level goals representing system functional behaviour, also called hard goals, are decomposed into hard sub-goals. Alternative decompositions are qualified by quality criteria and user preferences that contribute positively or negatively to their ranking. From this perspective, in the work by Qureshi *et al.* requirements engineering at runtime focuses on capturing changes in relevant context and user preferences. Upon these variations, the system must select the most appropriate goal decomposition path to ensure the expected system behaviour.

The effectiveness of the assurance of SAS systems at runtime is highly dependent on changing conditions of the execution environment that can affect not only the target system, but also the adaptation mechanism and monitoring infrastructure. Ramirez and Cheng proposed an effective approach to manage changes in monitoring conditions (assessment criteria) according to environmental situations at runtime [RC11]. In their approach, software requirements are modelled using goal models based on the RELAX language [WSB+09]. Goal-based models are excellent candidates to be exploited as models at runtime to keep track of changes in SAS system requirements dynamically.

Direct-testing based methods. Multi-agent based software systems expose high levels of runtime dynamism. Therefore, testing techniques applicable to these systems can be good candidates to be borrowed to the assessment of SAS systems using models at runtime [NPB+09]. An important challenge in the validation of SAS systems at runtime using direct-testing techniques is the generation of test cases that are relevant to the system's current situation and goals. Nguyen *et al.* exploit evolutionary testing techniques to generate test cases automatically, based on quality functions, in the evaluation of system performance [NPT+09]. Quality functions are associated to stakeholder expectations of the behaviour of an autonomous system. They propose a systematic way for deriving quality functions and thresholds from a goal-oriented requirements model for the assessment of agents (e.g., the quality function associated to the goal of a cleaner agent to maintain its battery can be a minimum battery level to be satisfied). The evolutionary testing approach by Nguyen *et al.* allows the automatic generation of test cases with increasing difficulty levels, guided by a fitness function associated to the quality

of interest (e.g., a function inversely proportional to the total power consumption of the system throughout its lifetime).

Model checking. Model checking [CGP01,PPS09] was proposed in the 1980s independently by Clarke and Emerson [CE82] and by Quielle and Sifakis [QS82]. It assumes an available mathematical model of a system and a given formal specification, in some logical formalism, such as Linear Temporal Logic (LTL) [Pnu81] and Computational Tree Logic (CTL) [BAMP81]. The goal of model checking is to check in an algorithmic way the consistency between the given model and the given specification. Model checking has been largely used to verify hardware systems [BLPV95], which are by nature finite state systems, but it has been also used in the software development industry [CGP02]. Model checking has been used in several application domains to assure desired system properties and can be used to verify the SAS system based on its runtime models. The work presented in [WIdlIA12] surveys the use of formal methods in self-adaptive systems. This study shows that no standard tools have been emerged for formal modelling and verification of self-adaptive systems. However, authors found that 40.0% of the surveyed studies use tools for formal modelling or verification and that 30.0% of the studies that use tools employ it for model checking.

The work presented in [BHTV06] used model checking to check whether an architecture is a refinement of another one. This is obtained by defining refinement relationships between abstract and concrete styles. The defined refinement criteria guarantee both semantic correctness and platform consistency. The paper in [AZ12] proposes an approach to model check goal-oriented requirements for self-adaptive systems. The approach presented in [CdL12] makes use of probabilistic model checking to verify resilience properties of self-adaptive systems; in other words the authors aims at verifying whether the self-adaptive system is able to maintain trustworthy service delivery in spite of changes in its environment. In architecture-based domains, Pelliccione *et al.* exploit model checking at the software architecture level to verify properties of the system, its components, and the interactions among components [PIM09,PTBP08]. To deal with unplanned adaptations, Inverardi *et al.* proposed a theoretical assume-guarantee framework to efficiently define under which conditions to perform adaptation by still preserving the desired invariants [IPT09]. Model checking has also been applied in the domain of agent-based systems, for instance to assure adaptability to unforeseen conditions, behavioural properties, and performance [Gor01]. Finally, other approach use Petri Nets to enable the analysis of properties like the reachability of a certain state or deadlock-freeness [Mur89]. Some of these analysis capability have been extended to enhanced versions of Petri Nets like Colored Petri Nets [Jen03] and applied to check several aspects like performance [Wel02] or safety properties [CHC96].

Rule-based analysis and verification. Several approaches based on formal methods, specially graph-based formalisms, have been proposed to leverage rule-based analysis and verification of software properties. In particular, Becker and Giese proposed a graph-transformation based approach to model correct SAS systems at a high-level of abstraction. Their approach considers different level of abstractions according to the three-layer reference architecture proposed by Kramer and Magee for SAS systems in [KM07]. In their approach, Becker and Giese check the correctness of the modelled

SAS system using simulation and invariant checking techniques. Invariant checking is mainly used to verify that a given set of graph transformations will never reach a forbidden state. This verification process exposes a linear complexity on the number of rules and properties to be checked [BBG+06]. In another approach, Giese *et al.* use triple graph grammars as a formal semantics for specifying models, their relation and transformations. These models can be used as a basis for analysing the fulfilment of desired properties [GHL10]. In the self-healing domain, Bucchiarone *et al.* proposed an approach to model and verify self-repairing system architectures [BPVR09]. In their approach, dynamic software architectures are formalized as typed (hyper) graph grammars. This formalization enables verification of correctness and completeness of self-repairing systems. This approach was extended later by Ehrig *et al.* by proposing an approach to model self-healing systems using algebraic graph transformations and graph grammars enriched with graph constraints [EER+10]. This allows formal modelling of consistency and operational properties. In the quality-driven component-based software engineering domain, Tamura *et al.* formalized models for component-based structures and reconfiguration rules suing typed and attributed graph transformation systems to preserve QoS contracts [TCCD12,Tam12]. Based on this formalization, they provide means for formal analysis and verification of self-adaptation properties, both at development time and runtime by integrating the AGG tool in their system.

Synthesis. Synthesis techniques provide another mechanism applicable to the assurance of SAS systems. The goal of synthesis techniques is to generate the "correct" assembly code for the (pre-selected and pre-acquired) components that constitute the specified system, in such a way that it is possible to guarantee that the system exhibits the specified interactions only. [IST11] provides an instance of synthesis-based approaches applicable to networking. This approach considers application-layer connectors by referring to two conceptually distinct notions of connector: coordinator and mediator. The former is used when the networked systems to be connected are already able to communicate but they need to be specifically coordinated to reach their goal(s). The latter goes a step forward by representing a solution for both achieving correct coordination and enabling communication between highly heterogeneous networked systems.

Semantic web. Models at runtime are also required to support self-adaptation of context management infrastructures (i.e., the third level of dynamics in SAS systems). To manage context dynamically, the explicit mapping between assurance concerns and relevant context must be complemented with an explicit mapping between relevant context and infrastructural elements of the monitoring infrastructure. In this way, whenever changes in assurance criteria or relevant context occur, the dynamic adaptation of a representation of the monitoring strategy will trigger the adaptation of context sensors, context providers and context monitors accordingly. Resource description framework (RDF) graphs from semantic web are also promising candidates to be used as effective models at runtime in the assessment of SAS systems. Models at runtime in the form of RDF graphs can be exploited to represent relevant context, monitoring strategies, system requirements including assurance criteria, as well as to support changes in context management strategies at runtime. Ontologies and semantic-web based rules, defined according to the application domain, provide the means required to infer changes in the

monitoring infrastructure according to changes in requirements, assurance criteria or context [VMT11a,VMM+11].

5.2 Classifying Assurance Methods According to Non-Functional Criteria

This subsection classifies several runtime assurance approaches according to selected non-functional requirements.

Safety. For systems that are self-adaptive or even self-organizing the application of traditional safety assurance approaches is currently infeasible the application of traditional safety assurance approaches. This is mostly because these approaches heavily rely on a complete understanding of the system and its environment, which is difficult to attain for adaptive systems and impossible for open systems. A general solution approach is to shift parts of the safety assurance measures into runtime when all required information about the current state of the application can be obtained. Rushby was one of the first software engineering researchers to consider adaptive systems in which methods of analysis traditionally used to support certification at development time are instead used at runtime, and certification is performed just-in-time [Rus07]. Based on this work he later coined the notion of runtime certification [Rus08], using runtime verification techniques to partially perform certification at runtime. Following the same core idea of shifting parts of the assurance measures into runtime, Schneider *et al.* introduced the concept of conditional safety certificates (ConSerts) [ST11]. ConSerts are predefined modular safety certificates that have a runtime representation to enable dynamic evaluations in the context of open adaptive systems. Some initial ideas concerning the extension of ConSerts regarding other certifiable non-functional properties such as security have been published also [SBT11]. Priesterjahn and Tichy proposed a different approach based on the application of hazard analysis techniques during runtime [PT09]. This approach is closely related to previous work of its authors where they introduced a development-time hazard analysis approach for analysing all configurations a self-adaptive system can reach during runtime [GT06]. A corresponding extension also considers the time between the detection of a failure and the reconfiguration into another configuration [PSWTH11].

Performance. For assurance of performance properties which mostly focus on response time, throughput or utilization, common runtime models include regression models and queuing network models (QNM). For example, Hellerstein *et al.* [HDPT04] as well as Lu *et al.* [LAL+03] described dynamic regression models in the context of autonomic computing and self-optimization. Menascé and Bennani used QNM as predictive models for avoiding bottleneck saturation and for on line capacity sizing [N.B03]. Ghanbari *et al.* have used dynamically tuned layered queuing models, which are software specific versions of QNMs, for online performance problem determination and mitigation in cloud computing [GSLI11]. More recently, Barna *et al.* reported performance load and stress testing methods on on-line tuned runtime performance models [BLG11].

Reliability and availability. Runtime assurance for reliability and availability properties employ discrete time Markov chains, which are synchronized with the system and its

usage profile. An example for a specific approach is QoSMOS [CGK+11], which uses the KAMI approach [EGMT09] to keep the model including its parameters and the system consistent. QoSMOS employs probabilistic model checking at runtime to evaluate whether the system satisfies the current reliability requirements, where recently novel approaches [FGT11,MG10] have been developed to deal with the time complexity.

Usability. In applications with adaptive user interfaces it is often impossible to test each adaptation state with real users. Therefore, automated usability evaluation of such user interfaces often relies on models of the user or user interactions to automatically evaluate states of user interfaces [IH01]. Quade *et al.* introduced an approach that evaluates the usability of the current state of the user interface using models at runtime [QBL+11]. The evaluation is based on a simulation of user interactions based on the model of the user interface and a model of the user. Having these techniques available at runtime enables a more detailed modelling of the user as the model can be checked against data from the actual user interaction.

6 Compelling Applications for Models at Runtime

This section introduces application exemplars for which runtime models can play a major role in the assurance of desired properties and system goals. The goal of this section is to provide researchers with a catalogue of "killer applications" useful to conduct case studies on the assurance of SAS systems using models at runtime.

Kaleidoscope. Kaleidoscope is a multi-channel multimedia video streaming and video on demand system. Imagine an Olympics game or a football match where millions of users are simultaneously streaming, watching and querying videos about the event. The Kaleidoscope application aims to provide/share best quality video for its users. For this, Kaleidoscope must act as a proxy server that is used to store and forward multimedia contents to user devices. A device can be a notebook, a smartphone or a personal digital assistant (PDA). Kaleidoscope must detect both the video source and the user target device. Kaleidoscope is required to adjust or switch itself at runtime from one configuration variant to another, by providing best quality video to users concurrently and reliably. The broadcast is fetched from a video source via TV cable (TV broadcast) or either wired or wireless (Webcast) Internet connection. Users can configure preferences using the client application.

Autonomous Vehicle Service. Lets assume that the legal issues on autonomous cars will be solved. Several steps in the direction of autonomic driving are carried out by many car manufacturers currently, but the boldest steps are undertaken by Google that even has managed to get a license for an autonomic car in Nevada early 2012—only five years after the DARPA Great and Urban Challenges on autonomic cars. In a few more years, cars without drivers could come from and go to parking lots, or deliver things. In a carpooling scenario, autonomous vehicles booked by users could serve the user at a specific time and destination. Best routes will be planned intelligently based on current context information such as traffic conditions and weather. Ordering, booking, and

payment will be performed via smartphone applications. Elderly people will become mobile again, as they will be completely safe in an autonomic car. Moreover, cars will have intelligent driver assistance for anticipating potential hazards early and avoiding collisions.

Intelligent, yet safe autonomous driving software systems require effective methods to ensure their required qualities. Even though the functions of these vehicles are perceived as "intelligent", rather than on artificial intelligence they rely on standard algorithms from sensor fusion, context management, and control theory. In particular, these systems require special attention to context management infrastructures to guarantee the reliability of sensors and monitors. Autonomous vehicle software use models at several levels, specially for understanding relevant context situations: models are required to represent entities that affect the behaviour of the car, to specify quality of sensors, and to model context uncertainty. Given the dynamic nature of context information, these models must be available and manipulable at runtime. Another kind of important models are those that specify vehicle typical behaviour and are used to understand unusual behaviour patterns.

Models in autonomous vehicle software are typically developed implicitly and coded manually into the running system. However, for a solid theory of autonomous driving and intelligent driver assistance, these models need to be managed explicitly throughout the software lifecycle, which includes runtime.

Autonomous Agricultural Operations. *Precision agriculture* aims at the implementation of a comprehensive farming management concept. One of the main issues that is addressed by precision agriculture is the optimization of the productivity and efficiency when operating on the field, by tailoring soil and crop management to match the conditions at every location. This can be achieved through the utilization of different information sources such as GPS, satellite imagery and IT systems. More recently, there are efforts to further improve productivity and efficiency by increasing the amount of automation on the field to the point of autonomous operation. Examples are harvesting fleets with several harvesters where only one is operated by humans, autonomous tractors to pick up the crop from the harvesters, and tractor implement automation (TIA) where tractors are controlled by implements to execute implement-specific tasks. These application scenarios have in common that different vehicles or machines are combined on the field in order to fulfil (partially) autonomous tasks. The assurance and certification of important properties such as safety and security is obviously very important in this context, whereas traditional assurance techniques are not applicable without further ado. A general solution to this problem is to shift parts of the assurance measures into runtime. This can be achieved by means of suitable runtime models, and corresponding management facilities integrated into these systems.

Ambient Assisted Living. The number and capabilities of devices available at home are growing steadily. *Ambient Assisted Living (AAL)* aims use these technologies to assist users with disabilities in their daily live tasks, for example to monitor health conditions and detect emergency situations. Software applications in this domain are highly dynamic. Every home is different and can contain different devices that could be

exploited by AAL services. New generations of devices are produced on a regular basis requiring AAL services to evolve continuously to keep up to date with new technical developments. Moreover, similar devices produced by different vendors may considerably differ in their capabilities and interfaces. Nevertheless AAL systems must be able to use these devices as soon as they become available at the user's home.

In addition to variation in devices, users of AAL systems are subject to considerable variation. An AAL service must deal with an arbitrary number of people living at the same home, their disabilities and capabilities, and their current situations. Therefore, the system is required to adapt itself according to current users and their environment. Moreover, these systems must be open to future extensions such as the integration of new sensors or actuators for new applications.

To deal with this complex dynamics, AAL software requires models at runtime to reason about users and their context and thus deliver services accordingly. AAL systems involve human lives, therefore assurance is a major concern to guarantee safety and control despite the adaptive behaviour of these systems.

The Guardian Angels Project. In the context of AAL, the "Guardian Angels for a Smarter Planet" project is a key example that has potential to benefit from models at runtime. The following details are sourced from the webpage of the Publications Office of the European Union [Pub12]:

The Guardian Angels Flagship Initiative[16] aims to provide information and communication technologies to assist people in all stages of life. Guardian Angels are envisioned to be like personal assistants. They are intelligent (thinking), autonomous systems (or even systems-of-systems) featuring sensing, computation, and communication, and delivering features and characteristics that go well beyond human capabilities. It is intended that these will provide assistance from infancy right through to old age. A key feature of these Guardian Angels will be their zero power requirements as they will scavenge for energy. Foreseen are individual health support tools, local monitoring of ambient conditions for dangers, and emotional applications. Research will address scientific challenges such as energy-efficient computing and communication; low-power sensing, bio-inspired energy scavenging, and zero-power human-machine interfaces.

These devices by their very nature will need to be adaptive in terms of functional and non-functional properties. In addition, they will be used in critical situations that require high levels of dependability and hence the highest levels of safety assurance. The development of models at runtime can support runtime decision making and certification for this important and innovative application area.

7 Conclusion

The main objectives of this publication are to identify a set of short-term and long-term research questions to define a research agenda for the assurance of SAS systems using models at runtime. We also aim to provide the research communities, including Models@runtime, runtime V&V, Requirements@runtime, SEAMS (self-adaptive and

[16] http://www.ga-project.eu/

self-managing software systems), with starting points for conducting research where models at runtime constitute cornerstones for advancing the state of the art on the assurance of dynamic SAS systems.

This paper covers a fragment of the existing state of the art for this area of research. The reader should note, that we only included selected approaches. Other approaches also include approaches that have been applied in Model Driven Engineering for design time V&V, static analysis or model checking and could be leveraged to run time. Several of the research questions state the requirement for structuring the existing techniques, modelling approaches and of the area of models@run.time. This will serve as a basis for getting a better idea of potential techniques, the purposes they can be used for and their properties in a run time environment.

As reaction to one of the research questions this paper also states some compelling applications for assurance based on models@run.time. Their commonality is in a highly complex and adaptive run time application and environment. Such examples require means to deal with this complexity at run time and are a good motivation for having abstractions of complex processes and structures in the form of models@run.time as well as assurance of their properties during their run time execution and evolution.

References

[ABH+10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[Acc04] Accelera. *Property Specification Language Reference Manual (Version 1.1)*, June 2004.

[ACDR+11] Marco Autili, Vittorio Cortellessa, Davide Di Ruscio, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Eagle: Engineering software in the ubiquitous globe by leveraging uncertainty. In *Proceedings 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011)*, pages 488–491. ACM, 2011.

[AIP07] Marco Autili, Paola Inverardi, and Patrizio Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engineering (ASE)*, 14:293–340, September 2007.

[ALRL04] Avizienis Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(1):11–33, 2004.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[AZ12] Dhaminda B. Abeywickrama and Franco Zambonelli. Model checking goal-oriented requirements for self-adaptive systems. In *Engineering of Computer Based Systems (ECBS 2012)*, pages 33 –42, april 2012.

[BAMP81] Mordechai Ben-Ari, Zohar Manna, and Amir Pnueli. The temporal logic of branching time. In *Proceedings 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1981)*, pages 164–176. ACM, 1981.

[BBF09] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@run.time. *Computer*, 42:22–27, 2009.

[BBG+06] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In

ACM/IEEE International Conference on Software Engineering (ICSE 2006). ACM, 2006.

[BG10] Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings Workshop on Future of Software Engineering Research (FoSER 2010)*, pages 17–22. ACM, 2010.

[BHTV06] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Style-based modeling and refinement of service-oriented architectures. *Software and System Modeling*, 5(2):187–207, 2006.

[BKLW95] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, and Charles B. Weinstock. Quality attributes. Technical Report CMU/SEI-95-TR-021, CMU/SEI, 1995.

[BLG11] Cornel Barna, Marin Litoiu, and Hamoun Ghanbari. Autonomic load-testing framework. In *Proceedings 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011)*, pages 91–100. ACM, 2011.

[BLPV95] Jörg Bormann, Jörg Lohse, Michael Payer, and Gerd Venzl. Model checking in industrial hardware design. In *Proceedings 32nd ACM/IEEE conference on Design automation (DAC 1995)*, pages 298–303. ACM, 1995.

[BPVR09] Antonio Bucchiarone, Patrizio Pelliccione, Charlie Vattani, and Olga Runge. Self-repairing systems modeling and verification using AGG. In *Proceedings Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture (WICSA/ECSA 2009)*, pages 181–190. IEEE Computer Society, 2009.

[BS01] Manfred Broy and Ketil Stoelen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Monographs in Computer Science. Springer, 2001.

[CdL12] Javier Cámara and Rogério de Lemos. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012), 2012 ICSE Workshop on*, pages 53 –62, june 2012.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, chapter 5, pages 52–71. Springer Berlin / Heidelberg, Berlin/Heidelberg, 1982.

[CGK+11] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Transactions on Software Engineering (TSE)*, 37(3):387–409, 2011.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, 2001.

[CGP02] Satisha Chandra, Patrice Godefroid, and Christopher Palm. Software model checking in practice: an industrial case study. In *Proceedings 24th International Conference on Software Engineering (ICSE 2002)*, pages 431–441. ACM, 2002.

[CHC96] Seung Mo Cho, Hyoung Seok Hong, and Sung Deok Cha. Safety analysis using coloured petri nets. In *APSEC*, pages 176–193. IEEE Computer Society, 1996.

[CPL09] Lawrence Chung and Julio Cesar Prado Leite. On non-functional requirements in software engineering. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. Yu, editors, *Conceptual Modeling: Foundations and Applications*, pages 363–379. Springer, 2009.

[EER+10] Hartmut Ehrig, Claudia Ermel, Olga Runge, Antonio Bucchiarone, and Patrizio Pelliccione. Formal analysis and verification of self-healing systems. In *Proceedings 13th International Conference on Fundamental Approaches to Software Engineering (FASE 2010)*, pages 139–153. Springer, 2010.

[EGMT09] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In Stephen Fickas, Joanne Atlee, and Paola Inverardi, editors, *Proceedings 31st ACM/IEEE International Conference on Software Engineering (ICSE 2009)*, volume 4839, pages 111–121. IEEE Computer Society, 2009.

[FGT11] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings 33rd ACM/IEEE International Conference on Software Engineering (ICSE 2011)*, pages 341–350. ACM, 2011.

[GHL10] Holger Giese, Stephan Hildebrandt, and Leen Lambers. Toward bridging the gap between formal semantics and implementation of triple graph grammars. In *Proceedings 2010 Workshop on Model-Driven Engineering, Verification, and Validation (MODEVVA 2010)*, pages 19–24, Washington, DC, USA, 2010. IEEE Computer Society.

[Gor01] Diana F. Gordon. APT agents: Agents that are adaptive, predictable, and timely. In *Proceedings First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers (FAABS 2000)*, pages 278–293. Springer, 2001.

[Gru11] Lars Grunske. An effective sequential statistical test for probabilistic monitoring. *Information & Software Technology*, 53(3):190–199, 2011.

[GSLI11] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. Exploring alternative approaches to implement an elasticity policy. In *IEEE International Conference on Cloud Computing (CLOUD 2011)*, pages 716–723. IEEE Computer Society, July 2011.

[GT06] Holger Giese and Matthias Tichy. Component-based hazard analysis: Optimal designs, product lines, and online-reconfiguration. In Janusz Górski, editor, *25th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2006)*, volume 4166 of *LNCS*, pages 156–169. Springer, 2006.

[GZ09] Lars Grunske and Pengcheng Zhang. Monitoring probabilistic properties. In Hans van Vliet and Valérie Issarny, editors, *7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/SIGSOFT FSE 09)*, pages 183–192. ACM, 2009.

[HDPT04] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[HT04] David Harel and P.S. Thiagarajan. Message sequence charts. *UML for Real*, pages 77–105, 2004.

[IBM06] IBM Corporation. An architectural blueprint for autonomic computing. Technical report, IBM Corporation, 2006.

[IEE90] IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, December 1990.

[IH01] Melody Y. Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Computer Survey*, 33(4):470–516, 2001.

[Inv07] Paola Inverardi. Software of the future is the future of software? In *Proceedings 2nd international conference on Trustworthy global computing (TGC 2006)*, pages 69–85. Springer, 2007.

[IPT09] Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Towards an assume-guarantee theory for adaptable systems. In *Proceedings 2009 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009)*, pages 106–115. IEEE Computer Society, 2009.

[IST11] Paola Inverardi, Romina Spalazzese, and Massimo Tivoli. Application-layer connector synthesis. In *Formal Methods for Eternal Networked Software Systems (SFM 2011)*, volume 6659 of *LNCS*, pages 148–190, 2011.

[Jen03] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Volume 1)*, volume 1 of *EATCS Series*. Springer Verlag, April 2003.

[KC03] Jeff O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[KM07] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering (FOSE 2007)*, pages 259–268. IEEE Computer Society, 2007.

[LAL+03] Ying Lu, Tarek Abdelzaher, Chenyang Lu, Lui Sha, and Xue Liu. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *Proceedings 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003)*, page 208. IEEE Computer Society, 2003.

[Lee09] Edward A. Lee. Computing needs time. *Communications of the ACM (CACM)*, 52(5):70–79, 2009.

[MAB+02] R. Murray, Karl J. Aström, Stephen P. Boyd, Roger W. Brockett, John A. Burns, and Munzer A. Dahleh. Control in an information rich world, 2002.

[Mae87] Pattie Maes. Concepts and experiments in computational reflection. *SIGPLAN Notices*, 22(12):147–155, 1987.

[MG10] Indika Meedeniya and Lars Grunske. An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In *Proceeedings 21st IEEE International Symposium on Software Reliability Engineering (ISSRE 2010)*, pages 229–238. IEEE Computer Society, 2010.

[MKS09] Hausi A. Müller, Holger M. Kienle, and Ulrike Stege. *Autonomic computing: now you see it, now you dontdesign and evolution of autonomic software systems*, volume 5413 of *LNCS*, pages 32–54. Springer, 2009.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings IEEE*, 77(4):541–580, April 1989.

[N.B03] Daniel A. Menascéand Mohamed N.Bennani. On the use of performance models to design self-managing computer systems. In *Proceedings 29th International Computer Measurement Group Conference (CMG 2003)*, pages 7–12, 2003.

[NPB+09] Cu Nguyen, Anna Perini, Carole Bernon, Juan Pavón, and John Thangarajah. Testing in multi-agent systems. In *10th International Workshop on Agent Oriented Software Engineering (AOSE 2009)*, pages 180–190, 2009.

[NPT+09] Cu D. Nguyen, Anna Perini, Paolo Tonella, Simon Miles, Mark Harman, and Michael Luck. Evolutionary testing of autonomous software agents. In *Proceedings 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS 2009)*, pages 521–528. IFAAMAS, 2009.

[PIM09] Patrizio Pelliccione, Paola Inverardi, and Henry Muccini. Charmy: A framework for designing and verifying architectural specifications. *IEEE Transactions on Software Engineering*, 35:325–346, 2009.

[Pnu81] Amir Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PPS09] Doron Peled, Patrizio Pelliccione, and Paola Spoletini. *Wiley Encyclopedia of Computer Science and Engineering 6th Edition, 5-Volume Set.*, volume 3, chapter Model Checking, pages 1904–1920. Published by John Wiley & Sons in Hoboken, NJ, 2009.

[PSWTH11] Claudia Priesterjahn, Christoph Sondermann-Wölke, Matthias Tichy, and Christian Hölscher. Component-based hazard analysis for mechatronic systems. In *Proceedings IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2011)*, pages 80–87. IEEE Computer Society, March 2011.

[PT09] Claudia Priesterjahn and Matthias Tichy. Modeling safe reconfiguration with the fujaba real-time tool suite. In *Proceedings 7th International Fujaba Days*, pages 20–14, November 2009.

[PTBP08] Patrizio Pelliccione, Massimo Tivoli, Antonio Bucchiarone, and Andrea Polini. An architectural approach to the correct and automatic assembly of evolving component-based systems. *Journal of Systems Software*, 81:2237–2251, December 2008.

[Pub12] Publications Office of the European Union. FET Flagship Pilots. http://cordis.europa.eu/fp7/ict/programme/fet/flagship/6pilots_en.html, 2012. last access date: 07.10.2012.

[QBL+11] Michael Quade, Marco Blumendorf, Grzegorz Lehmann, Dirk Roscher, and Sahin Albayrak. Evaluating user interface adaptations at runtime by simulating user interaction. In *25th BCS Conference on Human Computer Interaction (HCI 2011)*, pages 497–502, 2011.

[QJP11] Nauman A. Qureshi, Ivan Jureta, and Anna Perini. Requirements engineering for self-adaptive systems: Core ontology and problem statement. In *23rd International Conference on Advanced Information Systems Engineering (CAiSE 2011)*, volume 6741 of *LNCS*, pages 33–47. Springer, 2011.

[QLP11] Nauman A. Qureshi, Sotirios Liaskos, and Anna Perini. Reasoning about adaptive requirements for self-adaptive systems at runtime. In *Proceedings 2nd International Workshop on Requirements@run.time (RE@run.time 2011)*, pages 16–22, August 2011.

[QP10] Nauman A. Qureshi and Anna Perini. Requirements engineering for adaptive service based applications. In *Proceedings 18th IEEE International Requirements Engineering Conference (RE 2010)*, pages 108–111, 2010.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings 5th Colloquium on International Symposium on Programming*, pages 337–351. Springer-Verlag, 1982.

[RC11] Andres J. Ramirez and Betty H. C. Cheng. Automatic derivation of utility functions for monitoring software requirements. In *Proceedings 14th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2011)*, pages 501–516. Springer, 2011.

[Rus07] John Rushby. Just-in-time certification. In *Proceedings 12th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2007)*, pages 15–24. IEEE Computer Society, 2007.

[Rus08] John Rushby. Runtime certification. In *Proceedings Eighth Workshop on Runtime Verification (RV 2008)*, pages 31–35. Springer, 2008.

[SBT11] Daniel Schneider, Martin Becker, and Mario Trapp. Approaching runtime trust assurance in open adaptive systems. In *Proceedings 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*, pages 196–201. ACM, 2011.

[SCF+06] Jean-Sébastien Sottet, Gaëlle Calvary, Jean-Marie Favre, Joëlle Coutaz, Alexandre Demeure, and Lionel Balme. Towards model driven engineering of plastic user interfaces. In *Proceedings 2005 international conference on Satellite Events at the MoDELS (MoDELS 2005)*, pages 191–200, Berlin, Heidelberg, 2006. Springer.

[ST11] Daniel Schneider and Mario Trapp. A safety engineering framework for open adaptive systems. In *Proceedings Fifth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2011)*. IEEE Computer Society, 2011.

[Tam12] Gabriel Tamura. *QoS-CARE: A Reliable System for Preserving QoS Contracts through Dynamic Reconfiguration*. Phd thesis, University of Lille 1 - Science and Technology and Universidad de Los Andes, May 2012.

[TCCD12] Gabriel Tamura, Rubby Casallas, Anthody Cleve, and Laurence Duchien. QoS Contract-Aware Reconfiguration of Component Architectures Using E-Graphs. In *Formal Aspects of Component Software*, volume 6921 of *LNCS*, pages 34–52. Springer, 2012.

[TVM+12] Gabriel Tamura, Norha M. Villegas, Hausi A. Müller, João P. Sousa, Basil Becker, Mauro Pezzè, Gabor Karsai, Serge Mankovskii, Wilhelm Schäfer, Ladan Tahvildari, and Kenny Wong. *Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems*, volume 5 of *LNCS*, pages 108–132. Springer, 2012.

[vLDL98] Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions Software Engineering (TSE)*, 24(11):908–926, 1998.

[VMM+11] Norha M. Villegas, Hausi A. Müller, Juan C. Munoz, Alex Lau, Joanna Ng, and Chris Brealey. A dynamic context management infrastructure for supporting user-driven web integration in the personal web. In *Proceedings Conference of the Center for Advanced Studies on Collaborative Research, Canada (CASCON 2011)*, pages 200–214. ACM, 2011.

[VMT11a] Norha M. Villegas, Hausi A. Müller, and Gabriel Tamura. Optimizing Run-Time SOA Governance through Context-Driven SLAs and Dynamic Monitoring. In *Proceedings IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011)*, pages 1–10. IEEE Computer Society, 2011.

[VMT+11b] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings 6th International Symposium on Software engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*, pages 80–89. ACM, 2011.

[VTM+12] Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, Laurence Duchien, and Rubby Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, volume 7475 of *LNCS*, pages 265–293. Springer, 2012.

[Wel02] Lisa Wells. Performance analysis using coloured petri nets. In *Proceedings 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2002)*, pages 217–, Washington, DC, USA, 2002. IEEE Computer Society.

[WIdlIA12] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 67–79, New York, NY, USA, 2012. ACM.

[WSB+09] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Bruel. RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In *Proceedings 17th International Requirements Engineering Conference (RE 2009)*, pages 79–88. IEEE Computer Society, September 2009.

[ZCG07] Ji Zhang, Betty H. C. Cheng, and Heather Goldsby. Amoeba-rt: Run-time verification of adaptive software. In Holger Giese, editor, *Models in Software Engineering, Workshops and Symposia at MoDELS 2007, Nashville, TN, USA, September 30 - October 5, 2007, Reports and Revised Selected Papers*, volume 5002 of *LNCS*, pages 212–224. Springer, 2007.