Using Models at Runtime to Address Assurance for Self-Adaptive Systems

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

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

Abstract. A self-adaptive software system modifies its behavior at runtime in response to changes within the system or in its execution environment. The fulfillment of the system requirements needs to be guaranteed even in the presence of adverse conditions and adaptations. Thus, a key challenge for self-adaptive software systems is assurance. Traditionally, confidence in the correctness of a system is gained through a variety of activities and processes performed at development time, such as design analysis and testing. In the presence of self-adaptation, however, some of the assurance tasks may need to be performed at runtime. This need calls for the development of techniques that enable continuous assurance throughout the software life cycle. Fundamental to the development of runtime assurance techniques is research into the use of models at runtime

N. Bencomo et al. (Eds.): Models@run.time, LNCS 8378, pp. 101–136, 2014.

(M@RT). This chapter explores the state of the art for using M@RT to address the assurance of self-adaptive software systems. It defines what information can be captured by M@RT, specifically for the purpose of assurance, and puts this definition into the context of existing work. We then outline key research challenges for assurance at runtime and characterize assurance methods. The chapter concludes with an exploration of selected application areas where M@RT could provide significant benefits beyond existing assurance techniques for adaptive systems.

1 Introduction

A self-adaptive system (SAS) modifies its behavior at runtime in response to changes in the system itself or in its environment.[1] An SAS generally comprises a component that delivers the basic function or service, often referred to as the *target* or *managed system*, and another component that controls or manages 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 [ZC06a, GCZ08]. It is not adaptive and is applicable to a specific execution environment. The SAS controller can, via the invocation of an adaptation process that implements *adaptive logic* [ZC06a], transform this steady-state program to a different steady-state program—one that is suitable for a different set of environmental conditions [ZC06a]. As such, the steady-state program that delivers the basic function or service of an SAS is the target of the adaptation process that is managed by the controller. During the adaptation process, it is important to provide assurance that the system does not become inconsistent (e.g., no data is lost and transactions are not interrupted) [KM90, ZCYM05, ZC06b].

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] For non-adaptive systems, assurance is typically performed at design and development time. In practice, assurance tasks comprise verification, validation, test, measurement, conformance to standards, and certification. Collectively, these tasks all contribute to gaining confidence that both the processes employed and the end product satisfy established technical requirements, standards, and procedures. In the presence of runtime adaptations in an SAS, the fulfillment of the system requirements need to be guaranteed at runtime, even during the adaptation process [ZC05, ZC06b, VMT+11b]. Thus, software assurance becomes a critical runtime concern, giving rise to the need for continuous assurance over the entire life cycle of a software system. Given the increasing use of SASs in safety-critical applications (e.g., power-grid management, transportation management systems, telecommunication systems, and health-monitoring), assurance for SASs is of paramount importance. The development of rigorous methods and techniques that extend

[1] This chapter uses the acronym SAS to refer to any software-based system 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.

assurance from development time to runtime is therefore a high priority on the research agenda for the SAS research community.

Assurance is required for both functional properties (i.e., those describing specific functions of the system such as the result of a computation) 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) [VMT+11b]. Guaranteeing these properties at runtime in SASs is particularly challenging due to the varying assurance needs posed by a changing system or execution environment, both fraught with uncertainty [RJC12, EM13]. Nevertheless, the properties specified in the system requirements need to hold *before*, *during*, and *after* adaptation [ZC06a, ZC06b, ZGC09].

Continuous assurance throughout the entire software life cycle provides unprecedented opportunities for monitoring, analyzing, 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 enables us to *tame* the state space explosion, thus enabling the exploration of states that could not have been considered at development time. This reduction in state space provides new opportunities for runtime verification and validation (V&V), leading to assurance of critical system properties at runtime [TVM+12]. Fundamental to the development of runtime assurance techniques is research into models that can be used at runtime.

This chapter presents models at runtime (M@RT) as a foundation for the assurance of SASs and discusses related research challenges. Section 2 reviews assurance criteria, both functional and non-functional, whose fulfillment depends on or can be affected by self-adaptation and therefore requires assurance at runtime. Section 3 classifies different types of models used for M@RT and discusses the application of M@RT to support a spectrum of assurance issues. Section 4 identifies research challenges in the area of M@RT for SAS assurance tasks. Section 5 characterizes existing methods used for assurance of SASs. Section 6 describes selected application areas that exhibit the type of assurance challenges that we consider amenable to the use of M@RT. Finally, Section 7 concludes the chapter.

2 Assurance Criteria for Self-Adaptive Software Systems

Assurance criteria for SASs include functional and non-functional requirements whose fulfillment 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 assurance criteria applicable to the *adaptation process* itself. Sections 2.1 and 2.2 respectively discuss functional and non-functional requirements as fundamental assurance criteria for SASs.

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 a common 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 [Ber08, CNdRW06]. Formal languages with well-defined semantics provide a more rigorous and reliable means for specifying functional requirements in the context of system design. The following discussion is limited to formal descriptions.

Functional requirements decribe the behavioral objectives of 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 $f : I \rightarrow O$. A function f may be some type of computation, data manipulation, or other specific functions that the system should execute. Accordingly, the input I may be data from a user, values from a sensor, such as a temperature value or a sequence of images. Similarly, the output O may be pictures, continuous video, a braking signal for a car, or the opening of a valve. It is important ot note that functional requirements describe the system behavior that is visible at the system boundaries (i.e., system interfaces) [ZJ97]. The boundaries can be at the human-computer interface, sensors, actuators, or even at the boundaries between interacting systems. As such, functional requirements describe "what" the system has to provide in terms of its functional behavior to meet the expectations of its users, leaving "how" this functionality will be achieved to the design and implementation of the system.

System adaptation may become necessary to handle changes in the requirements or in the environment that are visible at its boundaries and influence its behavior externally. These adaptations may lead to internal changes that manifest as changed behavior observable at the system boundary. While the former is a reaction to the system context and leads to retaining the functional behavior in the presence of external change, the latter is a reaction to changing user needs or system configuration needs and leads to behavioral adaptations to accommodate the new requirements.

Because an SAS tends to respond to changes in the environment, functional requirements should take into account the context of the system as well as explicit assumptions about its behavior. Adaptation provides a means to alter the way a system satisfies its functional requirements, including the use of machine learning techniques [KM07], agent-based techniques [SAS14], bio-inspired techniques [BSG+09, MV14], and selecting specific target configuration from a collection of different target configurations [GCH+04, ZC06a], each of which satisfies the functional requirements, but may be better suited for a specific context and/or set of environmental conditions. The functional requirements may be formalized in an "assume/guarantee" style [JT96]—assuming a set of conditions or restrictions holds, then the application of the function guarantees that the results satisfy a set of required properties. The definition of pre- and postconditions is an example of this style of functional requirements specification.

Common formalisms used to express functional requirements are Linear-Time Temporal Logic (LTL) [Pnu81] and Computational Tree Logic (CTL) [BAMP81], both of which are included in the 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 modeling techniques have been developed, including Statecharts [Har87]; set-theoretic approaches, such as VDM [BJ78] and Z [ASM80]; process or operational-oriented, including SDL [Uni99], the B Method [Abr88], Event-B[ABH+10]; object-oriented languages, such as UML and its numerous variants[3]; architectural description languages [Cle96]; and Matlab/Simulink[4] to name a few representative examples. Traditionally, these techniques are used during system design and development to achieve increased confidence in the functional correctness of the system. Several of the above listed techniques support automatic code generation from the system model as well as formal verification at varying levels of abstraction.

Several complementary approaches have been used to specify functional requirements of an SAS, where uncertainty of the execution environment is implicitly or explicitly acknowledged by allowing more flexibility in how requirements can be satisfied. The SAS determines at runtime how to realize the specified functionality when placed in its target environment. This flexibility can be achieved by describing functional requirements in terms of policies that encode high-level specifications of functional objectives together with a set of operational constraints. This implicit approach to acknowledging uncertainty in the execution environment can utilize utility functions and a rule-based approach in the context of a goal-oriented functional requirements specification. Another approach is to explicitly acknowledge specific system functionality affected by uncertainty and thus allow specific points of flexibility in satisfying the requirements, such as that provided by the RELAX [WSB+09, CSBW09, RFJB12, FDC14a] and FLAGS [BPS10, PS11] approaches. Section 5.1 provides further details on these approaches.

2.2 Non-functional Requirements

If we consider functional requirements of a software system to be a function f that directly maps input I to output O ($f : I \rightarrow O$), then non-functional requirements refer to properties about f, I, O or relationships between I and O [CPL09]. Non-functional requirements such as performance, dependability, safety, security, and their corresponding quality attributes such as latency, throughput, capacity, confidentiality, and integrity can include 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 SASs.

It is necessary to validate and continually monitor non-functional requirements on both the target system and the adaptation process using techniques such as probabilistic monitoring [GZ09, Gru11], requirements monitoring [FF95], [FFvLP98], or utility function monitoring [GCH+04, RC11]. At runtime, the desired properties of the target system may no longer hold due to changes in the target system's context of use (e.g., user, platform, or environment context [SCF+06]), or side effects introduced by adaptations. In the latter case, it is possible to derive the impact of adaptations on properties of

[3] `www.uml.org`

[4] `http://www.mathworks.com`

the target system by analyzing adaptation properties such as stability, accuracy, settling time, small overshoot, and robustness. Specifically, it may be possible to take advantage of this relation to detect consequences of adaptations performed by controllers [KC03] or consequences of a changing environment (e.g., a failing component or a deficient Internet connection).

Several non-functional assurance criteria may be more easily 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$^+$11b], can be mapped to quality attributes measurable at runtime for both the target system and the adaptation mechanism. Where to measure a given property, either in the adaptation process or in the target system, will depend on its definition and its assessment metric. For example, *settling time* defined as the time required for the adaptation process to take the target system to a desirable state, must be measured on the target system since the need for the adaptation and the conditions for a desired state can only be observed at this level. Moreover, settling time can be measured through different quality attributes, depending on the specific non-functional property that must be satisfied. For example, if the concern is performance, settling time can be observed in terms of the time the system takes to perform a particular process. When the accepted time limit for this process is exceeded, the adaptation process will be invoked. Once the process execution time is back within desired limits, the target system will have reached its desired state. As such, settling time is the time elapsed between the moment at which the need for adaptation was detected and the moment at which the system reaches the desired new state. Villegas *et al.* [VMT$^+$11b] provide a comprehensive catalogue of adaptation properties and the corresponding quality attributes needed to identify the assurance criteria applicable to the adaptation process. This study also surveys definitions for the assurance criteria presented in Table 1.

Table 1. Examples of non-functional assurance criteria that are better guaranteed at runtime than at design time (including their mapping to quality attributes and adaptation properties) [VMT$^+$11b]

Assurance Criteria	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 M@RT to analyze, guarantee, and predict the qualities of the target system and the adaptation process dynamically. Implementing these mechanisms effectively requires a thorough analysis of the interdependencies between non-functional assurance criteria, quality attributes, and adaptation properties as presented 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 to identify adaptation properties, relevant to quality attributes, that are applicable to the adaptation mechanism. (For example, when performance is a key quality attribute for the target system, then 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

SASs require rethinking the notion of the software life cycle for which the distinction between development time and execution time stages is no longer starkly apparent (e.g., PLASTIC,[5], SMScom[6]). 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 executes [Inv07, BBF09, BG10, ACR+11, BDM+11, VTM+12, MV14] [CVM14].

Continuing with this line of reasoning, our objective is to explore models of different aspects of the application (e.g., requirements, specification, design, architecture, implementation, infrastructure, instrumentation, and context-of-use) and life cycle 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, analyzing, guaranteeing, and predicting system properties throughout the operation of an SAS.

The kind of models used at runtime can be classified by (1) their purpose—predictive, prescriptive, constructive, or descriptive; (2) their underlying modeling languages—for example, the 14 UML 2.2 structural and behavioral diagrams, State-charts, Petri Nets, and logic based models (e.g., Temporal Logics); and (3) the aspects they describe—data structure, task or process state, I/O behavior, or interaction pattern.

One of the main principles of using M@RT for assurance is to exploit the causal connection [Mae87] between the model and the system under development at runtime. This connection determines synchronization between the model and the running system. For example, M@RT can be updated to reflect changes in the running system —we say that they are in *descriptive causal connection*. This type of modeling enables assurance techniques to analyze abstract models instead of the actual implementation of the application when collecting information for assurance. In contrast, the model can be changed to cause an adaptation of the application (i.e., *prescriptive causal connection*). This use of modeling can be used to implement adaptations of the running system that are required to assure system properties.

[5] FP6 IST EU PLASTIC project `http://www.ist-plastic.org`

[6] Carlo Ghezzi, Self-Managing Situated Computing Grant, ERC Advanced Investigator Grant N. 227977, European Union, 2008–2013

In the scope of assurance, M@RT can be used as a basis for assuring functional as well as non-functional properties of the system (cf. Section 2). From this perspective, models can play 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. M@RT can have several purposes for runtime assurance. Among others, they can be used as information sources for monitoring aspects of a 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 use high-level knowledge about the system.

Development-time modeling approaches already exploit these advantages and enable the assertion of certain properties of a developed system. The use of M@RT 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 that are unknown at development time are bound at runtime and can allow for a more focused analysis of the current state and possibly several neighboring ones. This variable instantiation is especially useful for factors that can only be estimated at development time (e.g., network delay). A running system can continually monitor these aspects and react to them. The remainder of this section describes the dynamics of adaptive systems and the use of models during the adaptation process.

3.1 M@RT and the Dynamics of Self-Adaptive Software

The Software Engineering for Adaptive and Self-Managing Systems (SEAMS) community has identified three key subsystems needed for 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, each of which can be controlled through a corresponding feedback loop. Villegas *et al.* [VTM+12] provide a comprehensive characterization of these three levels of dynamics in SASs.

In general, assurance criteria drive the control objectives, adaptation, and monitoring feedback loops, as well as their interactions. As such, assurance governs the behavior of both the target system and the adaptation process. For example, system administrators can provide the control objectives manager with the required specifications. More specifically, the control objectives manager then sends the adaptation goals to the adaptation controller and monitoring requirements to the monitoring system. Thus, these specifications govern the behavior of the adaptation process and the behavior of the SAS throughout the adaptation process.

We argue that M@RT provide abstractions that are essential to support the feedback loops that control the three levels of dynamics identified in SASs. From this perspective, M@RT (cf. Figure 1) could be developed specifically for each level of dynamics to support the control objectives manager, adaptation controller, and the monitoring system. The figure also shows the interactions between these models and the respective subsystems in an SAS.

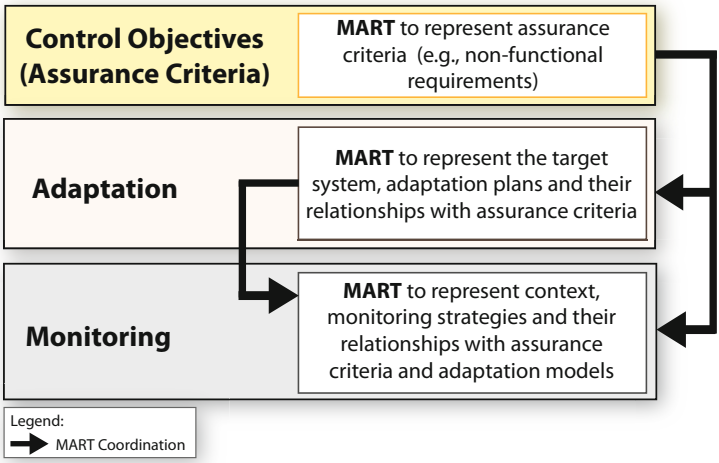

Fig. 1. The three levels of M@RT for the assurance of SASs

- At the *Control Objectives* level, M@RT represent requirements specifications subject to assurance in the form of functional and non-functional requirements.
- At the *Adaptation* level, M@RT represent states of the managed system, adaptation plans and their relationships with the assurance specifications.
- At the *Monitoring* level, M@RT represent context entities, monitoring requirements, as well as monitoring strategies and their relationships with assurance criteria and adaptation models.

Most importantly, M@RT at these levels must have efficient and effective methods of inter-level interaction since changes in requirement specifications may trigger changes at both the adaptation and the monitoring levels, as well as in the associated runtime models. Similarly, changes in adaptation models may imply changes in monitoring strategies or context entity models. In any case, M@RT at the adaptation and monitoring levels must maintain an explicit mapping to the models defined at the control objectives level that specify the requirements.

In summary, the architecture of SASs contains three interacting but functionally self-contained levels, each dedicated respectively to control objectives, adaptation, and monitoring of the SAS. Designing an SAS *for assurance*, as opposed to leaving assurance until after system design, requires the tight integration of assurance objectives into each level in the SAS architecture. We argue that this integration can most effectively be achieved by introducing dedicated M@RT that embody specific assurance criteria, focused either for the target system or the adaptation process.

3.2 Models at Runtime during the Adaptation Process

As a starting point for a research methodology we analyzed the MAPE-K loop in further detail. 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 SASs. The MAPE-K loop

is an abstraction of a feedback loop where the dynamic behavior 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). Each of these phases is briefly described next.

1. *Monitors* 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. *Analyzers* support decision making on the necessity of self-adaptation;
3. *Planners* generate suitable actions to affect the target system according to the supported adaptation mechanisms and the results of the Analyzer;
4. *Executors* implement actions with the goal of adapting the target system; and
5. A *Knowledge Base* enables 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]).

In order to illustrate the role of M@RT 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 M@RT. We aptly name the feedback loop depicted in this figure *MAPE-MART loop*.

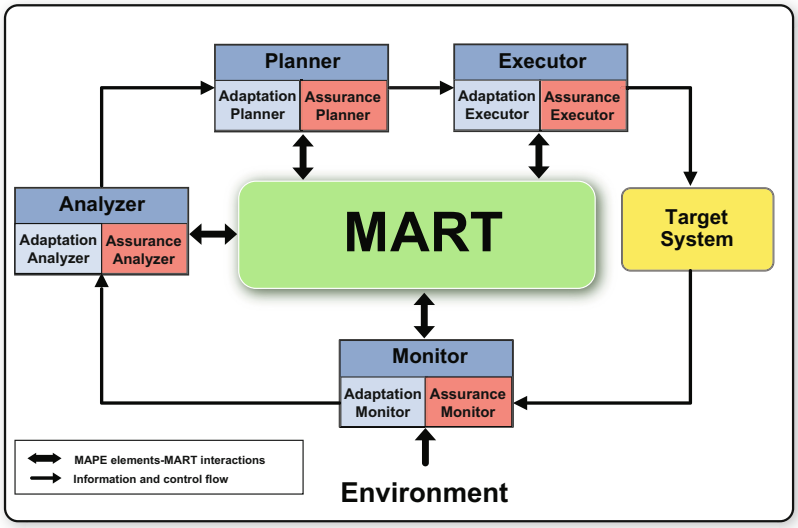

Fig. 2. MAPE-MART loop: The MAPE-K loop from autonomic computing extended with M@RT, and assurance instrumentation as foundational elements for the assessment of SASs

MAPE elements interact with M@RT 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*). For example, monitors interact with M@RT in order to make monitored data available throughout the adaptation process, or to monitor the states of models or changes in assurance criteria. *Analyzers* will then use monitored context to identify whether desired conditions are being or could potentially be violated. Analyzers can also update models with identified symptoms. Again, we can distinguish between *assurance analyzers* that analyze the system and *adaptation analyzers* that analyze the adaptation process. *Adaptation planners* use the symptoms provided by analyzers to define a new adaptation plan. Adaptation plans can be defined in the form of models that are 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, after which point, *assurance executors* check whether both the system remains in a safe state and the desired properties are achieved. These verification tasks can be optimized using M@RT.

4 Research Challenges for Assurance at Runtime

This section overviews selected research avenues and research challenges for the assurance of SASs using M@RT.

4.1 Research Avenues

Software assurance is a large field with many subfields (e.g., software quality, V&V, safety, trust, and several 'ilities') that spans the realms of software engineering, systems engineering, control engineering, and many other engineering disciplines. From a software engineering perspective, assurance at *runtime* for SASs appears to be an emerging area of research [GCZ08, FDB+08, IPT09, TVM+12, FGT11, SBT11, FRC13a, FDC14b]. In contrast, runtime assurance in control engineering traces its roots to the industrial revolution, applied to devices such as the centrifugal governor. This device used a flyball mechanism to sense the rotational speed of a steam turbine and to adjust the flow of steam into the machine. By regulating the turbine's speed, it provided the safe, reliable, and consistent operation that enabled the proliferation of steam-powered factories [MAB+02].

In an instrumented, interconnected, and intelligent world, control and runtime assurance are core components in SASs, 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 researchers in the field of 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 SASs.

Mining the rich histories, theories and experiences of fields such as biology, control engineering, and software engineering are worthwhile starting points for as-

surance at runtime research. In particular, we need survey papers that investigate models used for design time and runtime assurance techniques in these fields including research 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.* [MAB+02] posit that feedback is a central tool for uncertainty management in modern control. By measuring the operation of a system, comparing it to a *reference* at runtime, and adjusting available control variables, the controller can assure proper operation even in the presence of external disturbances or if its dynamic behavior is not fully known. In software, this reference can be realized with M@RT and evidence for assurance is gathered by checking conformance to the reference model. Murray *et al.* [MAB+02] 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 approach in software engineering is the three levels of runtime control models discussed in Section 3 [TVM+13].

The self-adaptive and self-managing systems community has produced a spectrum of runtime models [WMA10] [TVM+13] and patterns [RC10b, GH04] with control-centric models [KC03, HDPT04, IBM06, BSG+09] at one end and architecture-centric models [BCD97, OGT+, GCH+04, KM07] at the other end. These models come with different attributes and properties that can be exploited for runtime assurance. There is plenty of room for research to compare and evaluate the benefits and synergy of these different runtime model strategies [MKS09, TVM+13].

4.2 Selected Research Challenges

This section outlines selected open research problems and challenges aligned with the research avenues presented in the previous section. The focus is on the use of M@RT as a basis for developing runtime assurance techniques.

Runtime Assurance Criteria and Adaptation Properties. In Section 2.2 we related selected non-functional assurance criteria (e.g., latency) to adaptation properties (e.g., settling time) using quality attributes. One challenge is to extend this characterization of criteria and properties for the target system, controller, and adaptation process. While other approaches may be used to characterize and relate assurance criteria and adaptation properties, the properties are only meaningful if they can actually be measured. Monitoring infrastructure to measure properties is critical for runtime assurance methods. Over the past decade, the SAS community has published numerous papers on various aspects of monitoring. Many of these papers concentrate on the monitoring of raw measures in the managed system but only a limited number of approaches make the information amenable for runtime assurance assessment purposes, including functional requirements monitoring [FF95, FFvLP98, BWS+10, DDKM08, MPS08], assumptions monitoring [WSB11, RCBS12], and adaptive monitoring capabilities for changing environmental conditions [RC10a].

M@RT as a Foundation for Run-Time Assurance. While M@RT for SAS are increasingly being developed for complex SASs, including reference models [WMA10, VTM+12], few of these models are explicitly designed for runtime assurance. Thus, MART construction for runtime assurance is a key research challenge. The models introduced in Section 3 present good starting points for integrating assurance components into common SAS models. The central challenge for MART construction is to model uncertainty (e.g., environmental disturbances or evolving requirements). Understanding, managing, and leveraging uncertainty is important for delivering SASs with assurance guarantees such as reliability. Ramirez and Cheng [RJC12] have developed a taxonomy of uncertainty commonly faced by SAS, which could be used to facilitate uncertainty modeling and analysis efforts [EKM11, RCBS12]. Fields such as performance engineering and queuing theory have developed advanced models for many different applications. In particular, these fields have developed theories on how to transduce raw measures from a target system into meaningful measures for selected assurance criteria. However, performance constitutes just one dimension of the modeling and assurance problem. Many other quality criteria are applicable to SASs, such as trust, where quantification is rather difficult yet certifiable trust is one of the most important goals for an SAS [Dah10]. Moreover, models are needed to design trade-off analyses schemes for combinations of quality criteria. Models and quality criteria related to governance, compliance, and service-level agreements are of particular importance for service-oriented SASs [BHTV06, TVM+13]. Since M@RT form the foundation of many assurance tasks, the quality of these tasks directly depends on the quality of the models. Defining properties (e.g., accuracy, performance, or safety) for the evaluation of models at runtime is a significant research challenge [TVM+13].

To motivate researchers and practitioners to work on this subject we need compelling reasons for using M@RT for assurance [TVM+13]. A key goal for the SAS assurance research community is to develop exemplars that can be used to evaluate SAS runtime assurance techniques [TVM+13]. Most SAS conferences and workshops regularly call for exemplars but not usually explicitly targeted for SAS runtime assurance. An example of compelling motivation for work in this area is a 20-year science and technology research agenda and outlook for the US Air Force (USAF) [Dah10]. Approximately one third of this agenda is devoted to self-adaptive and autonomous systems with explicit calls for certifiable V&V techniques. V&V is also one of the most promising subfields of assurance where researchers can mine well-established design time models and transition them to runtime. The IBM autonomic computing initiative generated the highly acclaimed MAPE-K [KC03] and ACRA [IBM06] runtime models. The MAPE-K model separates four phases of the feedback loop and thus effectively decomposes the feedback loop assurance problem. The three-layer ACRA hierarchy facilitates integrated assurance reasoning from individually-managed resources at the lowest layer, to managing a collection of resources at the middle layer, to orchestrating an entire system by trading off resource managers at the top layer.

Run-Time Assurance Methods and Techniques. For SASs, the boundary between development time and runtime is rapidly disappearing [BG10]. As a result, we need to re-examine the distribution and effectiveness of assurance tasks over the entire life cycle

of an SAS. At the same time, we need to determine which models are most appropriate as a foundation for assurance tasks for the different stages of the software life cycle. In particular, we need to investigate whether models that are used for design-time assurance can be effectively used at runtime. In particular, what properties can be guaranteed at development, configuration, or load time as opposed to runtime. While not all assurance tasks can be transitioned to runtime, there is significant opportunity to conduct assurance tasks at runtime thereby making the system more resilient, reliable, responsive, secure, and cost-effective. Regardless of how dynamic a system really is, a substantial part of its assurance will always be done at development time. What (lightweight) design-time techniques can be readily transitioned to runtime? What development-time assurance methods, models, and techniques (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?

As illustrated in Figure 2, MART play an important role as the abstraction mechanisms required to support every stage of the SAS adaptation process. A key question is what MART 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, as well as the execution environment, every component of the adaptation process can also be an adaptive component. Thus, how can M@RT support changes in monitors, analyzers, planners and executors according to changes in functional and non-functional requirements? In the realm of control system engineering, changing the controller is referred to as *adaptive control* [AW94]. Another important avenue of research is how to characterize runtime assurance techniques according to the different levels of dynamics in SASs (i.e., changes in requirements, relevant context, adaptation mechanisms, and the target system itself).

Assurance obligations vary from one application domain to another. For example, the area of safety-critical systems has developed specialized assurance criteria and models—albeit mostly design-time techniques (e.g., ISO26262 for automotive subsystems,[7] and numerous safety standards set by the International Electrotechnical Commission).[8] The service-oriented architecture (SOA) community has developed SOA governance models—a combination of design time and runtime models—for assurance tasks for service-oriented systems on SOA platforms [SMB+09]. Thus, it is useful for researchers to classify runtime assurance criteria, models, and techniques according to their applicability to different domains and applications (e.g., application-independent, domain-dependent, mission-critical systems, embedded systems, real-time systems, etc.). Run-time assurance techniques can also be classified according to different types of runtime changes (e.g., dynamic context, changing requirements, or evolving models).

With the increasing use of computing-based systems for delivering critical societal services that demand long-running or even continuous operation (e.g., telecommunication, power grids, financial systems, etc.), even in the face of adversity, adaptation and runtime evolution [MV14] is a necessity, not a luxury. Even with meaningful reactions to changes, the triggered SAS adaptation should preserve selected core properties, thus posing a need for incremental and compositional assurance for SASs. An enabling

[7] http://www.iso.org/
[8] http://www.iec.ch/

step, in this direction, is to split functional and non-functional requirements into sub-requirements associated with single services and components of the system. The idea is to decompose the requirement specification into properties associated with the behavior of small parts of the system. Thus, it becomes possible to check these properties locally and to deduce from local checks whether the system satisfies the overall specification. By decomposing the assurance task in such a way, it may not be necessary to build a complete model of the system and thus the combinatorial state explosion problem is mitigated. The main challenge of this approach is that local properties are typically not preserved at the global level because of dependencies among the aggregate subparts of the system. Another approach to decomposing the assurance problem is to separate the verification of the functional properties from the verification of adaptation properties. Zhang *et al.* [ZGC09] developed AMOEBA, a modular verification approach for SASs where the functional properties are specified in terms of LTL and the adaptation properties are specified in terms of A-LTL [ZC06b]. With this separation of concerns, AMOEBA uses an assume/guarantee approach [JT96] to perform incremental model checking of both types of properties. AMOEBA-RT is an extension that monitors the adaptation properties at runtime based on state-based models of the adaptive logic [GCZ08].

As another example of assurance for the adaptation process, suppose *settling time* (i.e., the time required for the adaptation mechanism to take the target system to the desired state) has been defined as a performance-oriented assurance concern for a particular adaptive system. As such, the 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 render the system to be useless or even detrimental to the system's overall safety. The desired thresholds, monitoring conditions, and entities to be monitored can be specified using M@RT, such as goal-based models [WSB+09] or contextual RDF graphs [VMT11a, VMM+11].

5 Characterizing Assurance Methods

Researchers from communities related to the engineering of SASs have contributed a spectrum of approaches to the 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 how M@RT have been used as runtime assurance enablers in selected domains. This characterization of assurance approaches provides a starting point upon which researchers can build to address the research challenges posed by model-based runtime assurance of SASs.

5.1 Classifying Assurance Methods According to Techniques

This section presents and classifies selected existing approaches for runtime assurance of SASs according to the techniques and methods used for their realization.

Goal-Oriented Approaches. A first step towards assuring software systems is the articulation of assurance criteria. This task can be complex for functional requirements because it requires a deep understanding of the application domain. Nguyen *et al.* [NPT+09] argue that goal-oriented techniques are effective for deriving assurance criteria from functional requirements specifications. At development time (or requirements negotiation time), goal models can be used to specify stakeholder expectations for SASs, and the decision criteria for acceptable system behavior can be derived from these models. Moreover, goals, and especially high-level goals, have been recognized as more stable (i.e., less volatile) than specific system requirements [vLDL98]. Thus, high-level goals provide suitable candidate assurance criteria in highly dynamic systems. Qureshi *et al.* [QJP11, QLP11, QP10] rely on this assumption in their work on continuous requirements engineering. They represent functional behavior in terms of high-level goals (i.e., functional goals) that are decomposed into sub-goals. Alternative decompositions are qualified by quality criteria, user preferences, and context that contribute positively or negatively to their ranking. To ensure the expected behavior, the system must select the most appropriate goal decomposition path.

The effectiveness of the assurance of SASs at runtime is highly dependent on the 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 approach to manage changes in monitoring conditions according to environmental situations at runtime [RC11]. They specify requirements goal models using the RELAX language [WSB+09]. Recently, AutoRELAX has been developed to automatically add RELAX operators to goal models to handle uncertainty in the environment while minimizing the number of reconfiguration adaptations [FDC14a]. In a similar approach, Pasquale *et al.* [BPS10, PS11] developed FLAGS, a KAOS goal modeling framework that introduces the concept of a fuzzy goal whose satisfaction can be evaluated through fuzzy logic functions. Both goal-modeling approaches use fuzzy logic-based functions to add flexibility to the satisfaction criteria of goals in a goal-oriented model. In contrast to RELAX, however, FLAGS does not focus on identifying sources of uncertainty, but focuses rather on evaluating the degree to which a goal is satisfied. Goal-based models can be transitioned from design time to runtime to track changes in SAS requirements at runtime. Morandini *et al.* have investigated the life-cycle of goals at runtime [MPP09]. Souza *et al.* [SSLRM11] have developed a system, *Zanshin*, a requirements monitoring framework based on multiple feedback loops to monitor awareness requirements and progress towards adaptation objectives at runtime [ASaP13].

Automatic Test Case Generation- Based Methods. The complexity of system structure and behavior is growing exponentially, coupled with the comparable volume of possible scenarios and combinations of environmental conditions to be handled by an SAS. As such, successful strategies for automatic test case generation used for non-SAS application areas are being leveraged and explored for SAS testing. For example, given that multi-agent based software systems expose high levels of runtime dynamism, applicable testing techniques for these types of systems can be leveraged to assess SASs using M@RT [NPB+09]. An important challenge in the validation of SASs at runtime

using direct-testing techniques is the generation of test cases that are relevant to the system's current execution context and goals. As a means to evaluate system performance, Nguyen *et al.* [NPT+09] use evolutionary testing techniques to automatically generate test cases based on quality functions. Quality functions are associated with stakeholder expectations of the behavior of an autonomous system which are expressed as goal-oriented requirements. (e.g., the quality function associated with the goal of a cleaning agent to maintain its battery can be a minimum battery level to be satisfied). This approach 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). A complementary approach is taken by Fredericks *et al.* [FRC13b] where an SAS is exposed to a wide range of adverse environmental conditions that are used to generate SAS execution traces as the system adapts and reconfigures to handle the adverse conditions. These traces can then be analyzed for unexpected and/or unwanted behavior, both in the functional and in the adaptive logic. EvoSuite [FA11] is a framework that implements an evolutionary algorithm to generate test suites that consider a single coverage criterion, for instance the introduction of artificial defects into a program. Finally, a MAPE-T loop [FRC13a] has been proposed to provide a framework for monitoring the applicability and utility of test cases for an SAS as it undergoes environmental changes and reconfiguration. A set of research challenges were posed as part of the proposed framework, including explicit reference to the importance and use of M@RT. Veritas [FDC14b] is a recent realization of the MAPE-T loop that adapts test cases to ensure testing relevancy as an SAS reconfigures to handle changing environmental conditions.

Model Checking. Model checking [CGP01, PPS09] was proposed in the 1980s independently by Clarke and Emerson [CE82], and Quielle and Sifakis [QS82]. It assumes an available mathematical model of a system and a property to check against the model expressed in a formal logic, such as Linear Temporal Logic (LTL) [Pnu81] or Computational Tree Logic (CTL) [BAMP81]. The goal of model checking is to use an algorithmic approach to check the consistency between the given model and the property specification. Model checking has been used extensively to verify hardware [BLPV95] and software systems [CGP02] in many application domains to assure desired properties. Model checking at runtime is a key strategy to verify SASs based on runtime models. Weyns *et al.* surveyed formal methods in self-adaptive systems [WIdlIA12]. They showed that there are no standard tools for formal modeling and verification of self-adaptive systems. According to their survey, however, 40% of the surveyed studies use tools for formal modeling or verification, and 30% of those studies use model checking tools.

A number of model checking techniques have been used to analyze various properties of SASs. Baresi *et al.* used model checking to check whether an architecture is a refinement of another one [BHTV06]. Specifically, they defined refinement relationships between abstract and concrete styles. The defined refinement criteria guarantee both semantic correctness and platform consistency. In another approach, Abeywickrama and Zambonelli proposed to model check goal-oriented requirements for SASs [AZ12].

Cámara and de Lemos used probabilistic model checking to verify resilience properties of SASs, with the goal of verifying whether the self-adaptive system is able to maintain trustworthy service delivery in spite of changes in its environment [CdL12]. In architecture-based domains, Pelliccione *et al.* applied model checking at the software architecture level to verify properties of the system, its components, and the interactions among components [PIM09, PTBP08]. Filieri *et al.* have developed a runtime probabilistic model checking technique to detect harmful reconfigurations. To deal with unplanned adaptations, Inverardi *et al.* proposed a theoretical assume-guarantee framework to define under which conditions to perform adaptation by still preserving the desired invariants [IPT09]. Zhang and Cheng developed AMOEBA [ZGC09], a modular model checker to separately verify SAS functional properties in terms of LTL and the adaptive logic in terms of A-LTL (adapt-LTL). AMOEBA-RT [GCZ08] verifies runtime properties of SAS properties. Model checking has also been applied in the domain of agent-based systems, for instance to assure adaptability to unforeseen conditions, behavioral properties, and performance [Gor01]. Finally, Murata used Petri Nets to enable the analysis of properties, such as the reachability of a certain state or deadlock-freeness [Mur89]. Some of these analysis methods have been extended to enhanced versions of Petri Nets, such as Colored Petri Nets [Jen03] and applied to check properties such as performance [Wel02] or safety [CHC96].

Rule-Based Analysis and Verification. Several approaches based on formal methods, especially 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 SASs at a high-level of abstraction. Their approach considers different level of abstractions according to the three-layer SAS reference architecture proposed by Kramer and Magee [KM07]. In their approach, Becker and Giese check the correctness of the modeled SAS 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$^+$06]. In another approach, Giese *et al.* used triple graph grammars as a formal semantics for specifying models, their relation, and transformations. These models can be used as a basis for analyzing the fulfillment of desired properties [GHL10]. In the self-healing domain, Bucchiarone *et al.* proposed an approach to model and verify self-repairing system architectures [BPVR09]. In their approach, dynamic software architectures are formalized as typed hyper-graph grammars. This formalization enables verification of correctness and completeness of self-repairing systems. This approach was extended later by Ehrig *et al.* [EER$^+$10] to model self-healing systems using algebraic graph transformations and graph grammars enriched with graph constraints. This extension enables formal modeling of consistency and operational properties. In the quality-driven component-based software engineering domain, Tamura *et al.* [TCCD12, Tam12] formalized models for component-based structures and reconfiguration rules using typed and attributed graph transformation systems to preserve QoS contracts. Based on this formalization, they provide a means for formal analysis and

verification of self-adaptation properties, both at design time and runtime by integrating the Attributed Graph Grammar (AGG) system in their framework.

Synthesis. Another interesting avenue of research is to use synthesis techniques for assuring SASs. The goal of these 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. Inverardi *et al.* [IST11] proposed a synthesis-based approach for 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 further by representing a solution for both achieving correct coordination and enabling communication between highly heterogeneous networked systems. This work has been extended to also handle non-functional properties [DMIS13]. La Manna *et al.* [PGGB13] proposed an approach for reasoning about safeness of dynamic updates based on specification changes.

Semantic Web. A key challenge for establishing runtime assurance of SASs is the preservation of the relevance of runtime monitoring infrastructures with respect to assurance criteria and the system's execution environment. Specifically, monitoring strategies and infrastructures must adapt themselves dynamically. Models at runtime are also required to support self-adaptation of context management infrastructures (i.e., the third level of dynamics in SASs that was presented in Sect. 3.1). To manage context dynamically, the explicit mapping between assurance concerns and relevant context must be complemented with an explicit mapping between relevant context and infrastructure 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. Ramirez and Cheng [RCM10] used a goal-based approach to adapt the monitoring infrastructure to support the changing execution context for an SAS. Resource description framework (RDF) graphs, from semantic web, are good candidates to be used as effective M@RT in the assessment of SASs. 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, Vil13].

5.2 Classifying Assurance Methods According to Non-Functional Criteria

In this subsection, we classify surveyed runtime assurance approaches according to the non-functional requirements they address as assurance criteria.

Safety. For systems that are self-adaptive or even self-organizing, the application of traditional safety assurance approaches is currently infeasible. This obstacle is mostly due to the fact that these approaches rely heavily on a complete understanding of the system and its environment, which is difficult to attain for adaptive systems and as of yet impossible for open systems. Open systems, in contrast to self-adaptive systems that are generally closed systems, do not use measured outputs to determine control inputs required to adjust their behavior [HDPT04]. Therefore, open systems necessarily require a complete and accurate model of the system and its environment from which the control input must be derived. These models are generally impractical given that they must be robust to changes in the system and its environment and use no feedback mechanism to adjust themselves. A general solution 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 [Rus07] developed a strategy where development-time analysis techniques for certification are used at runtime, but the actual certification is performed as needed just-in-time. 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 portions of the assurance measures into runtime, Schneider *et al.* [ST13] introduced the concept of conditional safety certificates (ConSerts). 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 also been published [SBT11]. Priesterjahn and Tichy [PT09] proposed a different approach based on the application of hazard analysis techniques during runtime. This approach is closely related to their previous work where they introduced a development-time hazard analysis approach for analyzing all configurations that a self-adaptive system can reach during runtime [GT06]. A corresponding extension also considers the time between the detection of a failure and its reconfiguration [PSWTH11].

Performance. Regression models and queuing network models (QNM) are M@RT commonly used to reason about performance-based assurance properties relating to response time, throughput, or utilization. For example, Hellerstein *et al.* [HDPT04] and Lu *et al.* [LAL$^+$03] described dynamic regression models in the context of autonomic computing and self-optimization. Menascé and Bennani [MB03] used QNM as predictive models for avoiding bottleneck saturation and for online capacity sizing. Ghanbari *et al.* [GSLI11] used dynamically tuned layered queuing models, which are software specific versions of QNMs, for online performance problem determination and mitigation in cloud computing. More recently, Barna *et al.* [BLG11] reported performance load and stress testing methods on online tuned runtime performance models.

Reliability and Availability. Run-time assurance methods for reliability and availability properties use discrete time Markov chains that are synchronized with the system and its usage profile. For example, service-based systems built using the QoSMOS (QoS Management and Optimization of Service-based systems) framework [CGK$^+$11] translate high-level QoS requirements specified by their administrators into probabilistic

temporal logic formulae that are then formally and automatically analyzed to identify and enforce optimal system configurations. The QoSMOS self-adaptation mechanism can handle reliability and performance-related QoS requirements. QoSMOS [FGT11, MG10] uses the KAMI approach [EGMT09] to keep the model, including its parameters, and the system consistent; it uses probabilistic model checking at runtime to evaluate whether the system satisfies the current reliability requirements.

Security. Security considerations revolve around self-protection goals of an SAS, including confidentiality, integrity, authenticity, and authorization [BCdL11, KHW+01]. Run-time assurance of these goals is important in SASs since adaptation may produce emergent behavior that violates one or more other critical system properties. In particular, security assurance must be achieved without compromising system goals unrelated to security [RZN05, HMPB00]. For example, security considerations, such as confidentiality may conflict with availability goals. While the former, confidentiality, aims to protect the information in the system from unauthorized access, the latter, availability, is intended to ensure access to the system and the information a user is authorized to access. One way of counteracting an intrusion is by limiting access to the parts of the system that are affected by an attack. This approach clearly can have negative impact on availability. It is therefore important that, within an SAS, any remedial interventions invoked to preserve security goals also preserve the system properties not related to security. Achieving this balance requires decisions to be made at runtime based on evidence regarding the satisfaction of security goals obtained from analyzing the system and its environment, including user behavior.

Run-time security of an SAS involves not only protecting the target system, but it also means that the adaptation process and the policies governing the adaptation are protected from malicious attacks (e.g., preventing attackers from hijacking its adaptation mechanisms and policies) [Ais03, BJY11, OMH+11]. Adaptation methods, data, policies and certificates must be properly protected to ensure confidentiality, authenticity, and trusted communication of the entire adaptation process and its drivers. The components of every MAPE-MART loop depicted in Figure 2 must also be protected accordingly.

While an SAS is expected to make its adaptation decisions autonomously, a key question is how and how much to empower users with privacy and data security control (e.g., when user context is involved in adaptation decisions). The Surprise [MTVM12] approach (i) allows users to configure access permissions to their sensitive personal information to third parties, selectively and with different levels of granularity; (ii) supports changes in these configurations at runtime to add or remove third parties or permissions, and (iii) realizes partial encryption to share non-sensitive data with third parties who have not been explicitly authorized access, while protecting user identity. The Surprise approach is an exemplar of the application of M@RT to the preservation of privacy and security policies in user-driven SASs.

Security assurance, like other assurance goals at runtime, relies on the definition of high-level policies that must be preserved during adaptation. To achieve this security assurance, the Self-Adaptive Authorization Framework (SAAF) uses a feedback loop that continuously monitors the decisions made by the system's authorization pro-

cess [BCdL11] . The knowledge gained is used to adjust the authorization policy at runtime, making it more restrictive to constrain user behavior or loosening it to endorse users. Dynamic conflict resolution is particularly important in the context of security assurance but many existing approaches, e.g. [HMPB00], resolve conflicts using priority levels assigned at design time. Instead, the ATNAC (Adaptive Trust Negotiation and Access Control) framework [RZN05] allows access control policies to be dynamically adjusted depending on a set of trust-associated attributes observed at runtime. Formal methods have also been used successfully in this context. For example, the Willow Architecture [KHW+01], a dynamic reconfiguration framework for critical distributed systems, enables systems to continue working with reduced functionality while under a security attack. The use of formal methods enables autonomous handling of conflicts at runtime during reconfiguration.

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 evaluate states of user interfaces automatically [IH01]. Quade *et al.* [QBL+11] introduced an approach that evaluates the usability of the current state of a user interface using M@RT. 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 modeling 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 M@RT play a major role in the assurance of functional and non-functional assurance criteria. The goal of this section is to provide a catalogue of "killer applications" useful to motivate case studies on the assurance of SASs where M@RTare used as a foundation.

Kaleidoscope. Kaleidoscope [9]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. As such, Kaleidoscope must act as a proxy server that is used to store and forward multimedia content 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 must adapt at runtime from one configuration variant to another in order to provide the best quality video to users concurrently and reliably. The broadcast is fetched from a video source via TV cable (e.g., TV broadcast) or either wired or wireless (e.g., Webcast) Internet connection.

Latency and capacity (i.e., bandwidth) are important assurance criteria in Kaleidoscope since high-quality video streaming is a major functional requirement. To guarantee functional requirements under the desired quality conditions, Kaleidoscope must

[9] `http://www.savinetwork.ca`

adapt itself by reconfiguring its network and software architecture to minimize latency and maximize capacity. In this scenario, M@RT are useful for a variety of purposes. For example, predictive models can be used to anticipate latency and required capacity in the near future to perform preventive adaptations and thus avoid the violation of the desired qualities. Another example is the use of runtime formal models such as those exploited in rule-based analysis and verification to guarantee the reliable re-configuration of the system.

Autonomous Vehicle Service. Google driverless cars are now licensed in California, Florida and Nevada.[10] Google engineers and scientists achieved this amazing feat in a short five years after DARPA formulated the Great and Urban Challenges on autonomic cars.[11]

It is speculated that driverless cars could come from and go to parking lots, or deliver packages. 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 have greater access to services using an autonomic vehicle.

Increasingly, cars are being equipped with 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", they typically 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, especially for understanding relevant context situations: models are required to represent entities that affect the behavior 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 manageable at runtime. Another category of important models are those that specify typical vehicle behavior used to understand unusual behavioral patterns.

Models for autonomous vehicle software are typically developed implicitly and coded manually into the running system. In order to rigorously address the assured behavior of these systems, these models need to be managed explicitly and rigorously throughout the software life cycle, including at runtime.

Autonomous Agricultural Operations. *Precision agriculture*[12] is an approach to realize a comprehensive farming management concept. One of the main issues addressed

[10] `http://www.forbes.com/sites/ptc/2013/11/06/why-google-and-others-see-a-future-with-driverless-cars/print/`

[11] `http://www.tartanracing.org/challenge.html`

[12] `https://www.ispag.org`

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 each location. This level of customization can be achieved through the use of different information sources such as GPS, satellite imagery, and IT systems. More recently, efforts have been underway 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 comprising several harvesters but only one is operated by a human, autonomous tractors that 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 fulfill (partially) autonomous tasks. The assurance and certification of important properties, such as safety and security are clearly critical in this context. Furthermore, traditional assurance techniques are not applicable without significant modifications. A first step to this problem is to shift parts of the assurance measures into runtime. This strategy can be achieved by means of suitable M@RT 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)* is intended to use these technologies to assist users with disabilities in their daily tasks, such as monitoring health conditions and detecting emergency situations.[13] Software applications in this domain are not only critical, but also highly dynamic. On the one hand, human lives can be compromised. On the other hand, every home is different and can contain different devices that could be leveraged 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 differ considerably 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 an effective and safe manner.

In addition to variations 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 environmental conditions. Therefore, the system is required to adapt itself according to current users and their environment. Moreover, these systems must be sufficiently flexible to support future extensions, such as the integration of new sensors or actuators for new applications. Most importantly, these adaptations must be performed seamlessly and reliably to guarantee user safety.

To deal with these complex dynamics, AAL software requires M@RT to reason about users and their context in order to correctly and safely deliver services. Moreover, it is important to maintain a causal connection between these models and both the target systems and adaptation mechanisms. Given the potential risks to human lives, assurance is a major concern that must be guaranteed to prevent hazardous operation before, during, and after adaptation [ZC06a, VMT+11b]. M@RT can be essential in the management

[13] `http://www.aal-europe.eu`

of AAL software for capturing the environment, monitoring the user interaction, and reasoning about possible adaptive behavior and their impact.

The Guardian Angels Project. In the context of AAL, the "Guardian Angels for a Smarter Planet" project[14] is a good example to illustrate the potential benefits from using M@RT to address SAS assurance. The following details are based on information from the Publications Office of the European Union[15]:

The overarching objective of the Guardian Angels Flagship Initiative is to provide information and communication technologies to assist people in all stages of life. Guardian Angels are envisioned as 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 systems will provide assistance from infancy through old age. A key feature of these Guardian Angels will be their zero power requirements as they will scavenge for energy. Example services include individual health support tools, local monitoring of ambient conditions for dangers, and emotional applications. Scientific challenges for supporting their research challenges include 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.[16] The development of M@RT can support runtime decision making and certification for this important and innovative application area.

7 Conclusions

This chapter presented a research agenda for assurance at runtime with M@RT as a foundation. It grew out of stimulating discussions among the participants of the 2011 Schloss Dagstuhl Seminar on Models@run.time. In particular, we report on the findings of the breakout group Assurance@run.time as well as online discussions among the authors over the past two years while writing this chapter.

In an instrumented, interconnected and intelligent world, self-adaptive software systems proliferate. A key goal is to provide assurance at runtime when such systems adapt at runtime due to changes in their execution environment or their requirements. Traditionally software engineering, as opposed to control engineering, has concentrated on design-time assurance. Thus, a key challenge for the software engineering community is to develop runtime assurance techniques for self-adaptive systems that provide high performance, high confidence, and reconfigurable operation in the presence of uncertainties. One of the most promising avenues of research in this area is to use M@RT

[14] `http://www.ga-project.eu`

[15] Publications Office of the European Union: FET Flagship Pilots, Community Research and Development Information Service (CORDIS), http://cordis.europa.eu/fp7/ict/programme/fet/flagship/6pilots_en.html, 2012.

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

as a foundation for developing runtime assurance techniques. Of all the subfields of assurance, V&V has probably made the most progress in transitioning design time models and techniques to runtime. While not all design-time assurance tasks can be transitioned to runtime, a significant opportunity exists to conduct assurance tasks at runtime, thereby making the overall SAS more resilient, reliable, responsive, secure, and cost-effective. One of the most formidable challenges for researchers in the field of runtime software assurance is to investigate techniques that guarantee certifiable trust for highly-adaptive systems.

This research agenda on runtime assurance techniques provides excellent starting points for research communities dealing with SASs, including Models@runtime, Run-time V&V, Requirements engineering@runtime, SEAMS, SASO (International Conference on Self-Adaptive and Self-Organizing Systems), and ICAC (International Conference on Autonomic Computing). Given the increasing use of SAS for high-assurance application domains, such as intelligent vehicles, power grid management, telecommunication infrastructure, financial systems, healthcare management systems, etc., it is paramount that these communities and related communities work together to address the assurance of SASs. M@RT is a key enabling technology to accelerate progress in this area.

References

[ABH+10] Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in Event-B. Software Tools for Technology Transfer (STTT) 12(6), 447–466 (2010)

[Abr88] Abrial, J.R.: The B Tool. In: Bloomfield, R.E., Marshall, L.S., Jones, R.B. (eds.) VDM 1988 VDM — The Way Ahead. LNCS, vol. 328, pp. 86–87. Springer, Heidelberg (1988)

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

[ACR+11] Autili, M., Cortellessa, V., Di Ruscio, D., Inverardi, P., Pelliccione, P., Tivoli, M.: Eagle: Engineering software in the ubiquitous globe by leveraging uncertainty. In: Proceedings of the 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011), pp. 488–491 (2011)

[AIP07] Autili, M., Inverardi, P., Pelliccione, P.: Graphical scenarios for specifying temporal properties: An automated approach. Automated Software Engineering (ASE 2007) 14, 293–340 (2007)

[Ais03] Aissi, S.: Runtime environment security models. Intel Technology Journal 7(1), 60–67 (2003)

[ALRL04] Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing (TDSC) 1(1), 11–33 (2004)

[ASaP13] Angelopoulos, K., Silva Souza, V.E., Pimentel, J.A.: Requirements and architectural approaches to adaptive software systems: A comparative study. In: Proceedings of the 8th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013), pp. 23–32 (2013)

[ASM80] Abrial, J.-R., Schuman, S.A., Meyer, B.: A specification language. In: McNaughten, R., McKeag, R.C. (eds.) On the Construction of Programs, pp. 343–406. Cambridge University Press (1980)

[AW94] Aström, K.J., Wittenmark, B.: Adaptive Control, 2nd edn. Addison-Wesley (1994)

[AZ12] Abeywickrama, D.B., Zambonelli, F.: Model checking goal-oriented requirements for self-adaptive systems. In: Proceedings of the Engineering of Computer Based Systems (ECBS 2012), pp. 33–42 (2012)

[BAMP81] Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. In: Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1981), pp. 164–176 (1981)

[BBF09] Blair, G., Bencomo, N., France, R.B.: Models@run.time. IEEE Computer 42, 22–27 (2009)

[BBG+06] Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: Proceedings of the 28th ACM/IEEE International Conference on Software Engineering (ICSE 2006), pp. 72–81 (2006)

[BCD97] Blair, G., Coulson, G., Davies, N.: Adaptive middleware for mobile multimedia applications. In: Proceedings of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 1997), pp. 259–273 (1997)

[BCdL11] Bailey, C., Chadwick, D.W., de Lemos, R.: Self-adaptive authorization framework for policy based RBAC/ABAC models. In: Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2011), pp. 37–44 (2011)

[BDM+11] Balasubramanian, S., Desmarais, R., Müller, H.A., Stege, U., Venkatesh, S.: Characterizing problems for realizing policies in self-adaptive and self-managing systems. In: Proceedings of the 6th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 70–79 (2011)

[Ber08] Berry, D.M.: Ambiguity in natural language requirements documents. In: Martell, C. (ed.) Monterey Workshop 2007. LNCS, vol. 5320, pp. 1–7. Springer, Heidelberg (2008)

[BG10] Baresi, L., Ghezzi, C.: The disappearing boundary between development-time and run-time. In: Proceedings of the Workshop on Future of Software Engineering Research (FoSER 2010), pp. 17–22. ACM (2010)

[BHTV06] Baresi, L., Heckel, R., Thöne, S., Varró, D.: Style-based modeling and refinement of service-oriented architectures. Software and System Modeling 5(2), 187–207 (2006)

[BJ78] Bjorner, D., Jones, C.B. (eds.): The Vienna Development Method: The Meta-Language. LNCS, vol. 61. Springer, Heidelberg (1978)

[BJY11] Bauer, A., Jürjens, J., Yu, Y.: Run-time security traceability for evolving systems. Computer Journal 54(1), 58–87 (2011)

[BKLW95] Barbacci, M., Klein, M.H., Longstaff, T.A., Weinstock, C.B.: Quality attributes. Technical Report CMU/SEI-95-TR-021, CMU/SEI (1995)

[BLG11] Barna, C., Litoiu, M., Ghanbari, H.: Autonomic load-testing framework. In: Proceedings of the 8th ACM/IEEE International Conference on Autonomic Computing (ICAC 2011), pp. 91–100 (2011)

[BLPV95] Bormann, J., Lohse, J., Payer, M., Venzl, G.: Model checking in industrial hardware design. In: Proceedings of the 32nd ACM/IEEE Conference on Design automation (DAC 1995), pp. 298–303 (1995)

[BPS10] Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 18th IEEE International Requirements Engineering Conference (RE 2010), pp. 125–134 (2010)

[BPVR09] Bucchiarone, A., Pelliccione, P., Vattani, C., Runge, O.: Self-repairing systems modeling and verification using AGG. In: Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture (WICSA/ECSA 2009), pp. 181–190 (2009)

[BSG+09] Brun, Y., et al.: Engineering self-adaptive systems through feedback loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009)

[BWS+10] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: Requirements as runtime entities. In: Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE 2010), pp. 199–202 (2010)

[CdL12] Cámara, J., de Lemos, R.: Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In: Proceedings of the 7th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012), pp. 53–62 (2012)

[CE82] Gupta, M., Rao, R.S., Pande, A., Tripathi, A.K.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds.) CCSIT 2011, Part I. CCIS, vol. 131, pp. 318–328. Springer, Heidelberg (2011)

[CGK+11] Calinescu, R., Grunske, L., Kwiatkowska, M.Z., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Transactions on Software Engineering (TSE) 37(3), 387–409 (2011)

[CGP01] Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2001)

[CGP02] Chandra, S., Godefroid, P., Palm, C.: Software model checking in practice: An industrial case study. In: Proceedings of the 24th ACM/IEEE International Conference on Software Engineering (ICSE 2002), pp. 431–441 (2002)

[CHC96] Cho, S.M., Hong, H.S., Cha, S.D.: Safety analysis using coloured Petri nets. In: Proceedings of the Asia Pacific Software Engineering Conference (APSEC 1996), pp. 176–193 (1996)

[Cle96] Clements, P.C.: A survey of architecture description languages. In: Proceedings of the 8th IEEE International Workshop on Software Specification and Design (IWSSD 1996), pp. 16–26 (1996)

[CNdRW06] Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: Proceedings of 14th IEEE International Requirements Engineering Conference (RE 2006), pp. 59–68 (2006)

[CPL09] Chung, L., do Prado Leite, J.C.S.: On non-functional requirements in software engineering. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. Conceptual Modeling: Foundations and Applications, pp. 363–379. Springer, Heidelberg (2009)

[CSBW09] Cheng, B.H.C., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 468–483. Springer, Heidelberg (2009)

[CVM14] Castañeda, L., Villegas, N.M., Müller, H.A.: Self-adaptive applications: On the development of personalized web-tasking systems. In: Proceedings of the 9th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014 (in press, 2014)

[Dah10] Dahm, W.J.A.: Technology Horizons a Vision for Air Force Science & Technology During 2010-2030. Technical report, U.S. Air Force (2010)

[DDKM08] Dawson, D., Desmarais, R., Kienle, H.M., Muller, H.A.: Monitoring in adaptive systems using reflection. In: Proceedings of the 3rd ACM/IEEE International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008), pp. 81–88 (2008)

[DMIS13] Di Marco, A., Inverardi, P., Spalazzese, R.: Synthesizing self-adaptive connectors meeting functional and performance concerns. In: Proceedings of the 8th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013), pp. 133–142 (2013)

[EER+10] Ehrig, H., Ermel, C., Runge, O., Bucchiarone, A., Pelliccione, P.: Formal analysis and verification of self-healing systems. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 139–153. Springer, Heidelberg (2010)

[EGMT09] Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: Proceedings of the 31st ACM/IEEE International Conference on Software Engineering (ICSE 2009), pp. 111–121 (2009)

[EKM11] Esfahani, N., Kouroshfar, E., Malek, S.: Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011), pp. 234–244 (2011)

[EM13] Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Self-Adaptive Systems. LNCS, vol. 7475, pp. 214–238. Springer, Heidelberg (2013)

[FA11] Fraser, G., Arcuri, A.: Evosuite: Automatic test suite generation for object-oriented software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011), pp. 416–419 (2011)

[FDB+08] Fleury, F., Dehlen, V., Bencomo, N., Morin, B., Jezequel, J.M.: Modeling and validating dynamic adaptation. In: Proceedings of the International Workshop on Models@RunTime (M@RT 2008), pp. 97–108 (2008)

[FDC14a] Fredericks, E.M., Devries, B., Cheng, B.H.C.: AutoRELAX: Automatically RELAXing a goal model to address uncertainty. Empirical Software Engineering (in press, 2014)

[FDC14b] Fredericks, E.M., Devries, B., Cheng, B.H.C.: Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In: Proceedings of the 9th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014 (in press, 2014)

[FF95] Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE 1995), pp. 140–147 (1995)

[FFvLP98] Feather, M.S., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behavior. In: Proceedings of the 9th International Workshop on Software Specification and Design (IWSSD 1998), pp. 50–59. IEEE (1998)

[FGT11] Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering (ICSE 2011), pp. 341–350 (2011)

[FRC13a] Fredericks, E.M., Ramirez, A.J., Cheng, B.H.C.: Towards run-time testing of dynamic adaptive systems. In: Proceedings of the 8th ACM/IEEE International Symposium on Software Engineering for Self-Adaptive Systems (SEAMS 2013), pp. 169–174 (2013)

[FRC13b] Fredericks, E.M., Ramirez, A.J., Cheng, B.H.C.: Validating code-level behavior of dynamic adaptive systems in the face of uncertainty. In: Ruhe, G., Zhang, Y. (eds.) SSBSE 2013. LNCS, vol. 8084, pp. 81–95. Springer, Heidelberg (2013)

[GCH+04] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer 37(10), 46–54 (2004)

[GCZ08] Goldsby, H.J., Cheng, B.H.C., Zhang, J.: AMOEBA-RT: Run-time verification of adaptive software. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 212–224. Springer, Heidelberg (2008)

[GH04] Gomaa, H., Hussein, M.: Software reconfiguration patterns for dynamic evolution of software architectures. In: Proceedings of the Fourth IEEE/IFIP Working Conference on Software Architecture (WICSA 2004), pp. 79–88 (2004)

[GHL10] Giese, H., Hildebrandt, S., Lambers, L.: Toward bridging the gap between formal semantics and implementation of triple graph grammars. In: Proceedings of the Workshop on Model-Driven Engineering, Verification, and Validation (MODEVVA 2010), pp. 19–24. IEEE (2010)

[Gor01] Gordon, D.F.: APT agents: Agents that are adaptive, predictable, and timely. In: Rash, J.L., Rouff, C.A., Truszkowski, W., Gordon, D.F., Hinchey, M.G. (eds.) FAABS 2000. LNCS (LNAI), vol. 1871, pp. 278–293. Springer, Heidelberg (2001)

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

[GSLI11] Ghanbari, H., Simmons, B., Litoiu, M., Iszlai, G.: Exploring alternative approaches to implement an elasticity policy. In: Proceedings of the IEEE International Conference on Cloud Computing (CLOUD 2011), pp. 716–723 (2011)

[GT06] Giese, H., Tichy, M.: Component-based hazard analysis: Optimal designs, product lines, and online-reconfiguration. In: Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166, pp. 156–169. Springer, Heidelberg (2006)

[GZ09] Grunske, L., Zhang, P.: Monitoring probabilistic properties. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2009), pp. 183–192 (2009)

[Har87] Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)

[HDPT04] Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)

[HMPB00] Hashii, B., Malabarba, S., Pandey, R., Bishop, M.: Supporting reconfigurable security policies for mobile programs. Computer Networks 33(1-6), 77–93 (2000)

[HT04] Harel, D., Thiagarajan, P.S.: Message sequence charts. UML for Real, 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 (1990)

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

[Inv07] Inverardi, P.: Software of the future is the future of software? In: Montanari, U., Sannella, D., Bruni, R. (eds.) TGC 2006. LNCS, vol. 4661, pp. 69–85. Springer, Heidelberg (2007)

[IPT09] Inverardi, P., Pelliccione, P., Tivoli, M.: Towards an assume-guarantee theory for adaptable systems. In: Proceedings of the 4th ACM/IEEE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009), pp. 106–115 (2009)

[IST11] Inverardi, P., Spalazzese, R., Tivoli, M.: Application-layer connector synthesis. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 148–190. Springer, Heidelberg (2011)

[Jen03] Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. EATCS Series, vol. 1. Springer (2003)

[JT96] Jonsson, B., Tsay, Y.-K.: Assumption/guarantee specifications in linear-time temporal logic. Theoretical Computer Science 167(1-2), 47–72 (1996)

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

[KHW+01] Knight, J.C., Heimbigner, D., Wolf, A.L., Carzaniga, A., Hill, J., Devanbu, P., Gertz, M.: The Willow architecture: Comprehensive survivability for large-scale distributed applications. Technical Report CU-CS-926-01, Department of Computer Science, University of Colorado (2001)

[KM90] Kramer, J., Magee, J.: The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering (TSE) 16(11), 1293–1306 (1990)

[KM07] Kramer, J., Magee, J.: Self-managed systems: An architectural challenge. In: Future of Software Engineering (FOSE 2007), pp. 259–268. IEEE (2007)

[LAL+03] Lu, Y., Abdelzaher, T., Lu, C., Sha, L., Liu, X.: Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In: Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003), p. 208 (2003)

[MAB+02] Murray, R.M., Aström, K.J., Boyd, S.P., Brockett, R.W., Burns, J.A., Dahleh, M.A.: Control in an information rich world (2002)

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

[MB03] Menascé, D.A., Bennani, M.N.: On the use of performance models to design self-managing computer systems. In: Proceedings of the 29th International Computer Measurement Group Conference (CMG 2003), pp. 7–12 (2003)

[MG10] Meedeniya, I., Grunske, L.: An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In: Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE 2010), pp. 229–238 (2010)

[MKS09] Müller, H.A., Kienle, H.M., Stege, U.: Autonomic computing: Now you see it, now you don't—design and evolution of autonomic software systems. In: De Lucia, A., Ferrucci, F. (eds.) ISSSE 2006-2008. LNCS, vol. 5413, pp. 32–54. Springer, Heidelberg (2009)

[MPP09] Morandini, M., Penserini, L., Perini, A.: Operational semantics of goal models in adaptive agents. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems—Volume 1 (AAMAS 2009), pp. 129–136. IFAAMAS (2009)

[MPS08] Müller, H.A., Pezzè, M., Shaw, M.: Visibility of control in adaptive systems. In: Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008). ACM (2008)

[MTVM12] Muñoz, J.C., Tamura, G., Villegas, N.M., Müller, H.A.: Surprise: User-controlled granular privacy and security for personal data in smartercontext. In: Proceedings of the 22nd Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012), pp. 128–142. ACM (2012)

[Mur89] Murata, T.: Petri nets: Properties, analysis and applications. Proceedings to the IEEE 77(4), 541–580 (1989)

[MV14] Müller, H., Villegas, N.: Runtime evolution of highly dynamic software. In: Mens, T., Serebrenik, A., Cleve, A. (eds.) Evolving Software Systems, pp. 229–264. Springer (2014)

[NPB+09] Nguyen, C., Perini, A., Bernon, C., Pavón, J., Thangarajah, J.: Testing in multi-agent systems. In: Proceedings of the 10th International Workshop on Agent Oriented Software Engineering (AOSE 2009), pp. 180–190 (2009)

[NPT+09] Nguyen, C., Perini, A., Tonella, P., Miles, S., Harman, M., Luck, M.: Evolutionary testing of autonomous software agents. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems—Volume 1 (AAMAS 2009), pp. 521–528. IFAAMAS (2009)

[OGT+] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An architecture-based approach to self-adaptive software. IEEE Intelligent Systems, 14

[OMH+11] Ouedraogo, M., Mouratidis, H., Hecker, A., Bonhomme, C., Khadraoui, D., Dubois, E., Preston, D.: A new approach to evaluating security assurance. In: Proceedings of the 7th IEEE International Conference on Information Assurance and Security (IAS 2011), pp. 215–221 (2011)

[PGGB13] La Manna, V.P., Greenyer, J., Ghezzi, C., Brenner, C.: Formalizing correctness criteria of dynamic updates derived from specification changes. In: Proceedings of the 8th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013), pp. 63–72 (2013)

[PIM09] Pelliccione, P., Inverardi, P., Muccini, H.: CHARMY: A framework for designing and verifying architectural specifications. IEEE Transactions on Software Engineering (TSE) 35, 325–346 (2009)

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

[PPS09] Peled, D., Pelliccione, P., Spoletini, P.: Model Checking. In: Wiley Encyclopedia of Computer Science and Engineering, 6th edn., 5-Volume Set, vol. 3, pp. 1904–1920. John Wiley (2009)

[PS11] Pasquale, L., Spoletini, P.: Monitoring fuzzy temporal requirements for service compositions: Motivations, challenges, and experimental results. In: Proceedings of the International Workshop on Requirements Engineering for Systems, Services and Systems of Systems (RESS 2011), pp. 63–69. IEEE (2011)

[PSWTH11] Priesterjahn, C., Sondermann-Wölke, C., Tichy, M., Hölscher, C.: Component-based hazard analysis for mechatronic systems. In: Proceedings of the IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2011), pp. 80–87 (2011)

[PT09] Priesterjahn, C., Tichy, M.: Modeling safe reconfiguration with the fujaba real-time tool suite. In: Proceedings of the 7th International Fujaba Days, pp. 20–14 (2009)

[PTBP08] Pelliccione, P., Tivoli, M., Bucchiarone, A., Polini, A.: An architectural approach to the correct and automatic assembly of evolving component-based systems. Journal of Systems Software 81, 2237–2251 (2008)

[QBL+11] Quade, M., Blumendorf, M., Lehmann, G., Roscher, D., Albayrak, S.: Evaluating user interface adaptations at runtime by simulating user interaction. In: Proceedings of the 25th BCS Conference on Human Computer Interaction (HCI 2011), pp. 497–502 (2011)

[QJP11] Qureshi, N.A., Jureta, I.J., Perini, A.: Requirements engineering for self-adaptive systems: Core ontology and problem statement. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 33–47. Springer, Heidelberg (2011)

[QLP11] Qureshi, N.A., Liaskos, S., Perini, A.: Reasoning about adaptive requirements for self-adaptive systems at runtime. In: Proceedings of the 2nd International Workshop on Requirements@run.time (RE@run.time 2011), pp. 16–22 (2011)

[QP10] Qureshi, N.A., Perini, A.: Requirements engineering for adaptive service based applications. In: Proceedings of the 18th IEEE International Requirements Engineering Conference (RE 2010), pp. 108–111 (2010)

[QS82] Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)

[RC10a] Ramirez, A.J., Cheng, B.H.C.: Adaptive monitoring of software requirements. In: Proceedings of the Workshop on Requirements at Run Time (RE@RunTime 2010), pp. 41–50. IEEE (2010)

[RC10b] Ramirez, A.J., Cheng, B.H.C.: Design patterns for developing dynamically adaptive systems. In: Proceedings of the 5th ACM/IEEE Workshop on Software Engineering for Adaptive and Self-Managed Systems (SEAMS 2010), pp. 49–58 (2010)

[RC11] Ramirez, A.J., Cheng, B.H.C.: Automatic derivation of utility functions for monitoring software requirements. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 501–516. Springer, Heidelberg (2011)

[RCBS12] Ramirez, A.J., Cheng, B.H.C., Bencomo, N., Sawyer, P.: Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 53–69. Springer, Heidelberg (2012)

[RCM10] Ramirez, A.J., Cheng, B.H.C., McKinley, P.K.: Adaptive monitoring of software requirements. In: Proceedings of the First International Workshop on Requirements@run.time (RE@run.time 2010), pp. 41–50 (2010)

[RFJB12] Ramirez, A.J., Fredericks, E.M., Jensen, A.C., Cheng, B.H.C.: Automatically RELAXing a goal model to cope with uncertainty. In: Fraser, G., Teixeira de Souza, J. (eds.) SSBSE 2012. LNCS, vol. 7515, pp. 198–212. Springer, Heidelberg (2012)

[RJC12] Ramirez, A.J., Jensen, A.C., Cheng, B.H.C.: A taxonomy of uncertainty for dynamically adaptive systems. In: Proceedings of the 7th ACM/IEEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012), pp. 99–108 (2012)

[Rus07] Rushby, J.: Just-in-time certification. In: Proceedings of the 12th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2007), pp. 15–24 (2007)

[Rus08] Rushby, J.: Runtime certification. In: Leucker, M. (ed.) RV 2008. LNCS, vol. 5289, pp. 21–35. Springer, Heidelberg (2008)

[RZN05] Ryutov, T., Zhou, L., Neuman, C.: Adaptive trust negotiation and access control. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT 2005), pp. 139–146 (2005)

[SAS14] Proceedings of the ACM/IEEE International Conference on Self-Adaptive and Self-Organizing Systems (2007-2014)

[SBT11] Schneider, D., Becker, M., Trapp, M.: Approaching runtime trust assurance in open adaptive systems. In: Proceedings of the 6th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 196–201 (2011)

[SCF+06] Sottet, J.-S., Calvary, G., Favré, J.-M., Coutaz, J., Demeure, A., Balme, L.: Towards model driven engineering of plastic user interfaces. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 191–200. Springer, Heidelberg (2006)

[SMB+09] Simanta, S., Morris, E., Balasubramaniam, S., Davenport, J., Smith, D.B.: Information assurance challenges and strategies for securing SOA environments and web services. In: Proceedings of the 3rd IEEE Annual Systems Conference (SysCon 2009), pp. 173–178 (2009)

[SSLRM11] Souza, V.E.S., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 60–69 (2011)

[ST13] Schneider, D., Trapp, M.: Conditional safety certification of open adaptive systems. ACM Transactions on Autonomous and Adaptive Systems 8(2), 1–20 (2013)

[Tam12] Tamura, G.: 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 (2012)

[TCCD12] Tamura, G., Casallas, R., Cleve, A., Duchien, L.: QoS contract-aware reconfiguration of component architectures using e-graphs. In: Barbosa, L.S., Lumpe, M. (eds.) FACS 2010. LNCS, vol. 6921, pp. 34–52. Springer, Heidelberg (2012)

[TVM+12] Tamura, G., et al.: Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 108–132. Springer, Heidelberg (2013)

[TVM+13] Tamura, G., Villegas, N.M., Müller, H.A., Duchien, L., Seinturier, L.: Improving context-awareness in self-adaptation using the dynamico reference model. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013), pp. 153–162 (2013)

[Uni99] International Telecomunication Union. ITU-T Recommendation Z.100: Specification and Description Language, SDL (1999)

[Vil13] Villegas, N.M.: Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems. PhD thesis, Department of Computer Science, University of Victoria, Canada (February 2013)

[vLDL98] van Lamsweerde, A., Darimont, R., Letier, E.: Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering (TSE) 24(11), 908–926 (1998)

[VMM+11] Villegas, N.M., Müller, H.A., Muñoz, J.C., Lau, A., Ng, J., Brealey, C.: A dynamic context management infrastructure for supporting user-driven web integration in the personal web. In: Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2011), pp. 200–214. ACM (2011)

[VMT11a] Villegas, N.M., Müller, H.A., Tamura, G.: Optimizing run-time SOA governance through context-driven SLAs and dynamic monitoring. In: Proceedings of the IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011), pp. 1–10 (2011)

[VMT+11b] Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R.: A framework for evaluating quality-driven self-adaptive software systems. In: Proceedings of the 6th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 80–89 (2011)

[VTM+12] Villegas, N.M., Tamura, G., Müller, H.A., Duchien, L., Casallas, R.: DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 265–293. Springer, Heidelberg (2013)

[Wel02] Wells, L.: Performance analysis using coloured Petri nets. In: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2002), pp. 217–224 (2002)

[WIdlIA12] Weyns, D., Usman Iftikhar, M, de la Iglesia, D.G., Ahmad, T.: A survey of formal methods in self-adaptive systems. In: Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering (C3S2E 2012), pp. 67–79. ACM (2012)

[WMA10] Weyns, D., Malek, S., Andersson, J.: Forms: A formal reference model for self-adaptation. In: Proceedings of the 7th IEEE International Conference on Autonomic Computing (ICAC 2010), pp. 205–214 (2010)

[WSB+09] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.-M.: RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th IEEE International Requirements Engineering Conference (RE 2009), pp. 79–88 (2009)

[WSB11] Welsh, K., Sawyer, P., Bencomo, N.: Towards requirements aware systems: Run-time resolution of design-time assumptions. In: Proceedings of the 26th ACM/IEEE International Conference on Automated Software Engineering (ICSE 20011), pp. 560–563 (2011)

[ZC05] Zhang, J., Cheng, B.H.C.: Specifying adaptation semantics. In: Proceedings of the 2005 Workshop on Architecting Dependable Systems (WADS 2005), pp. 1–7. ACM (2005)

[ZC06a] Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: Proceedings of the 28th ACM/IEEE International Conference on Software Engineering (ICSE 2006), pp. 371–380 (2006)

[ZC06b] Zhang, J., Cheng, B.H.C.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software (JSS) 79(10), 1361–1369 (2006); Architecting Dependable Systems

[ZCYM05] Zhang, J., Cheng, B.H.C., Yang, Z., McKinley, P.K.: Enabling safe dynamic component-based software adaptation. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems III. LNCS, vol. 3549, pp. 194–211. Springer, Heidelberg (2005)

[ZGC09] Zhang, J., Goldsby, H.J., Cheng, B.H.C.: Modular verification of dynamically adaptive systems. In: Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development (AOSD 2009), pp. 161–172 (2009)

[ZJ97] Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Transactions on Software Engineering Methodology (TOSEM) 6(1), 1–30 (1997)