Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems

Gabriel Tamura[1], Norha M. Villegas[2], Hausi A. Müller[3], João Pedro Sousa[4], Basil Becker[5], Gabor Karsai[6], Serge Mankovskii[7], Mauro Pezzè[8], Wilhelm Schäfer[9], Ladan Tahvildari[10], and Kenny Wong[11]

[1] University of Lille 1-LIFL-INRIA, France,
Los Andes University and Icesi University, Colombia
gabriel.tamura@inria.fr
[2] University of Victoria, British Columbia,
Canada, and Icesi University, Colombia
nvillega@cs.uvic.ca
[3] University of Victoria, British Columbia, Canada
hausi@cs.uvic.ca
[4] George Mason University, USA
jpsousa@gmu.dot.edu
[5] Hasso Plattner Institute at the University of Potsdam, Germany
basil.becker@hpi.uni-potsdam.de
[6] Vanderbilt University, USA
gabor.karsai@vanderbilt.edu
[7] CA Inc., Canada
serge.mankovskii@ca.com
[8] University of Milano Bicocca, Italy and University of Lugano, Switzerland
mauro.pezze@unisi.ch
[9] University of Paderborn, Germany
wilhelm@upb.de
[10] University of Waterloo, Canada
ltahvild@uwaterloo.ca
[11] University of Alberta, Canada
kennyw@ualberta.ca

Abstract. Software validation and verification (V&V) ensures that software products satisfy user requirements and meet their expected quality attributes throughout their lifecycle. While high levels of adaptation and autonomy provide new ways for software systems to operate in highly dynamic environments, developing certifiable V&V methods for guaranteeing the achievement of self-adaptive software goals is one of the major challenges facing the entire research field. In this chapter we (i) analyze fundamental challenges and concerns for the development of V&V methods and techniques that provide certifiable trust in self-adaptive and self-managing systems; and (ii) present a proposal for including V&V operations explicitly in feedback loops for ensuring the achievement of software self-adaptation goals. Both of these contributions provide valuable starting points for V&V researchers to help advance this field.

R. de Lemos et al. (Eds.): Self-Adaptive Systems, LNCS 7475, pp. 108–132, 2013.

1 Introduction

Software validation and verification (V&V) concerns the quality assessment of software products throughout their lifecycle. Its goal is to ensure that the software product satisfies its functional requirements and expected quality attributes [1–3]. Over the past decade, many self-adaptive approaches and systems have been proposed by researchers from the software engineering for adaptive and self-managing systems (SEAMS) community, with multiple adaptation purposes [4, 5]. Certainly, many of the proposed self-adaptive software (SAS) systems have been designed to operate in highly dynamic socio-technical ecosystems where requirements, models, and contexts change at runtime [6]. This wide spectrum of system types, adaptation concerns, and dynamic goals has made it difficult to develop general runtime V&V methods. Unsurprisingly, V&V of SAS systems running in safety-critical environments is particularly challenging [7].

For inherently non-adaptive systems, that is, systems based on stable and well-known system execution conditions, many V&V methods, techniques and tools have been developed to be applied at design time. However, the quality assessment of SAS systems is challenging, not only because their adaptation objectives may vary according to environmental conditions at runtime, but also because the systems evolve to satisfy their evolving dynamic requirements. In this realm, V&V tasks—traditionally applied at design-time—are required to certify structural and behavioral aspects in the different phases of the adaptation process. In addition, these tasks must be performed at runtime in the two essential parts of a SAS system, namely, the adaptation mechanism, and the target system.

Besides the SEAMS community, there are several other communities dealing with runtime V&V, although not necessarily for SAS systems. During the past decade, the real-time verification (RV) community has run a workshop concerned with the monitoring and analysis of system executions.[1] The longer term goal of RV, already stated at RV 2001, is to investigate whether the use of lightweight formal methods applied during the execution of programs is a viable complement to the current heavyweight methods proving programs' correctness always before their execution, such as model checking and theorem proving, among others. Dynamic analysis, or the analysis of data gathered from a running program, has great potential for self-adaptive systems because it relies on direct monitoring mechanisms that expose the system's actual behavior [8]. The Models@runtime workshop, which emerged from the model-driven software development community, aims to use model-driven techniques for validating and monitoring runtime behavior. The Requirements@runtime workshop—collocated with the Requirements Engineering (RE) conference—aims to explore the potential of runtime abstractions and models of requirements, to be used as a practical means to address the challenges posed by volatile or poorly-understood environmental contexts.[2] In many ways, these workshops and conferences focus on different aspects

[1] International Conference on Runtime Verification
`http://runtime-verification.org/`

[2] `http://www.comp.lancs.ac.uk/~bencomo/RRT/`

of runtime V&V such as requirements, models, properties, instrumentation, and dynamic analysis.

Naturally, for the non-adaptive parts of a self-adaptive system, the traditional V&V methods can be used effectively. For the adaptive parts, runtime V&V methods are needed to guarantee self-adaptation objectives, independently of what is adapted. In general, SAS systems feature mechanisms based on the idea of the feedback loop [9]. We aptly termed the foundational science for runtime V&V methods *control science.* Control science can be defined as a systematic way to study certifiable V&V methods and tools to allow humans to trust decisions made by self-adaptive systems. In a 2010 report, Dahm identified control science as a top priority for the US Air Force (USAF) science and technology research agenda for the next 20 years [10]. Certifiable V&V methods and tools are critical for the success of autonomous, autonomic, smart, self-adaptive and self-managing systems.

One systematic approach to control science for adaptive systems is to study V&V methods for the mechanisms that sense the dynamic environmental conditions and the target system behavior, and act in response to these conditions by answering the questions *what*, *when* and *how* to adapt. In this paper we use the answers to these questions as key factors to determine *when* and *where* to perform V&V activities in the context of feedback loops—the common core of SAS systems.

This roadmap chapter focuses on research challenges concerning runtime V&V for the adaptive parts of self-adaptive systems in highly dynamic environments. In particular, we (i) analyze the cases in which the objectives, the system, or the monitoring infrastructure of a SAS system must be adapted; and (ii) propose how to make V&V tasks explicit in the feedback loop model elements, using results obtained by the aforementioned communities in this research field. Our goal is to provide researchers with a vision of open challenges in V&V for SAS systems, and discuss opportunities not only for proposing new runtime V&V techniques, but also for building on top of existing ones. In addition, our proposal for the explicitness of V&V tasks provides solid starting points for V&V researchers from other communities to deploy different techniques and methods for improving the trustworthiness of self-adaptive and self-managing systems.

The remaining sections of this chapter are organized as follows. Section 2 describes a concrete industrial case study that we use to illustrate the concepts, concerns and challenges discussed in this chapter. Section 3 outlines several challenges that arise from the differences in V&V requirements between software developed with traditional methods and self-adaptive software, and presents selected V&V drivers for self-adaptive software. Section 4 presents a refinement of the general feedback adaptation loop to propose a model that explicitly involves V&V tasks to address some of the previously identified challenges. This section also presents approaches from SEAMS-related research communities that provide valuable contributions for the assessment of SAS systems. Finally, Section 5 concludes the chapter.

2 Application Example

This section introduces our concrete industrial case study.[3] In this application example, self-adaptation is exploited to implement SOA governance mechanisms to enforce service level agreements (SLAs), such as those on performance, availability and confidentiality, in a cloud computing infrastructure [11]. In SOA and cloud-based systems QoS are highly affected by, and dependent on changing situations. On the one hand, SLAs may be violated at any time during system execution due to changes in the situation of relevant context entities such as computational infrastructure components and users. On the other hand, as businesses and users' requirements are evolving continuously, contracted QoS conditions (i.e., adaptation goals) may be frequently re-negotiated.

In this example, a performance SLA has been negotiated in terms of three throughput service level objectives (SLOs) to guarantee three different levels of system capacity in a cloud-based e-commerce platform: normal, medium and high load. These SLOs are observed on the bottleneck-operation of the system, *ProcessingPurchaseOrders*, and measured in terms of number of transactions per time unit. A *normal capacity* is required for a regular load of the shopping platform. A *medium capacity* is required when special offers are placed on social networks promoting them. A *high capacity* must be guaranteed to deal with the highest peak load of the platform caused by shopping seasons such as "Black Friday".

Governing the efficiency of the service-oriented infrastructure to optimize operational costs is a major concern for the retailers of this example. Hence, a self-adaptive mechanism based on service component architecture (SCA) reconfiguration was implemented to ensure quality of service (QoS) requirements in the service-oriented system. The *adaptation goal* for this dynamic service-oriented infrastructure is the contracted system capacity, in terms of the performance SLA. *Short settling time* and *consistency* correspond to the *adaptation properties* to be preserved. Adaptation properties in this application example are borrowed from the catalog proposed by researchers from the SEAMS community [5]. Settling time is defined as the time required for the adaptive system to achieve the desired state. Consistency guarantees the structural and behavioral integrity of the managed system with respect to the respective SCA integrity constraints [12], after its adaptation.

Use Case 1: Controlling the Elasticity of the E-Commerce Platform. As efficiency is a major concern, the capacity of the system must be either increased, or decreased according to the context situations that determine the expected load of the system. To accomplish this, context monitors must keep track of the popularity of special offers placed on social networks, as well as the day of the year to determine the applicable shopping season.

[3] This example is based on the IBM Centre for Advanced Studies (CAS) Canada project: "Managing Dynamic Context to Optimize Smart Interactions and Smart Services" `https://www-927.ibm.com/ibm/cas/cassis/viewReport?REPORT=747`.

Use Case 2: Re-negotiating Adaptation Goals at Runtime. After the e-commerce platform has been in operation, a new set of SLOs is added to the performance SLA. These new SLOs define different thresholds of response time that must be guaranteed according to the classification of the e-commerce platform's customers. Customers are classified as regular and premium users. A particular *maximum response time* threshold applies to regular customers. For premium customers, the maximum response time must correspond to 90% of the threshold defined for regular customers. Response time thresholds can be re-negotiated at runtime.

This example is used in the following sections to discuss runtime V&V concerns and research challenges in SAS systems, and illustrate the need for applying V&V tasks at runtime.

3 V&V Drivers for Self-Adaptive Software Systems

In this section we analyze and discuss drivers or key factors to consider when performing V&V tasks for SAS systems. We identify these drivers by (i) comparing how V&V for software that is adaptable at runtime differs from V&V for software that is immutable at runtime; and (ii) analyzing concerns that arise when dealing with three types of context changes that have been addressed by SAS systems, namely, in the objectives, the system, and the monitoring infrastructure to be adapted.

The goal of this section is twofold. First, it analyzes the classic V model for software development, and in particular its V&V activities, from the perspective of SAS systems. Second, it presents V&V drivers that we identified by analyzing three foundational questions concerning assurances for SAS systems: *when* to perform V&V tasks? *what* must V&V tasks validate and verify? and *where* in the adaptation cycle must these V&V tasks be performed? In light of these drivers, we identify research avenues in the form of research problems and opportunities to integrate V&V methods and techniques into the engineering of self-adaptive software systems.

3.1 The Classic V Model for System Development

Figure 1 illustrates how V&V activities are enacted in traditional software engineering to ensure that, at the different levels of system development, the software satisfies a given set of requirements.

This set of requirements is usually specified in advance of system development, allowing the definition of the corresponding complete *problem space*. From these requirements, a *solution space* is delimited and a solution derived and conceived in the form of a software architecture, which is refined into a software design. Both, architecture and design, are expressed as *models* (formal, semi-formal or informal), which can be verified at design time on their functional properties (e.g., correctness) with respect to the initial set of requirements. From this design, the software is materialized as units of code, which are gradually integrated, verified and tested

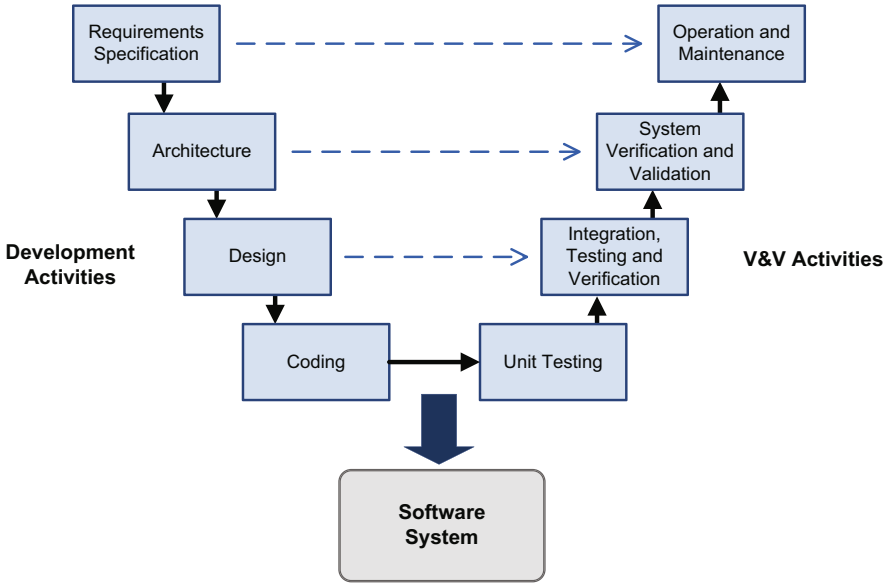

Fig. 1. The classic V model for system development (adapted from [13]). Each development phase is subject to a corresponding V&V phase—horizontal layers—as the software is built and integrated.

until the final system is obtained. Finally, this system is verified and validated as a whole before its deployment in production environments [14].

In this general lifecycle, the software quality is guaranteed by different V&V strategies applied in its different phases, even though several variations may provide additional assurances. For instance, despite the described flow of activities following the solid arrows in the depicted V-model, the dashed arrows allow V&V to be performed on the artifacts produced by any of the development activities (e.g., requirements or design models).

Among the V&V strategies that have been used, the software testing methods are the ones most commonly used in industry. Software testing methods can be very effective both in revealing failures and assessing the reliability of software systems, but cannot provide evidence of the absence of faults [13, 15]. More rigorous and effective strategies to reason about the program correctness employ model checking, graph-based, and other model-based software testing and verification methods [16–19]. However, these V&V methods have focused generally on design time. Therefore, the assessment and certification of system properties after changes occurring during system execution, either for ensuring the satisfaction of changing requirements, or for re-certifying system properties after adaptation, require not only traditional V&V methods adjusted to be applied at runtime, but also the adoption of non-traditional ones to be applied in the different adaptation phases of SAS systems.

Another important difference between these two types of software systems is the lifecycle phases in which V&V tasks are performed. In the classic V model,

V&V tasks are performed by software developers before deployment into production environments. However, in the adaptation process, the system architecture, design and implementation are evaluated, reconsidered and reconfigured at runtime by the system itself, according to relevant context changes. Hence, V&V tasks must be performed by the adaptation mechanism during the adaptation process. This has three important consequences. First, after the software initial release is in execution, the software development lifecycle phases (i.e., architecture, design and implementation) are in fact "absorbed", at least to some extent, by the adaptation (i.e., self-reconfiguration) process. Second, the boundaries between these phases, now amalgamated in the adaptation process, are blurred [20]. Third, the target system may be adapted and reach a state that was unforeseen at design time, and thus, the system has not been verified for that state. In simple ("switching") systems with a few possible adaptation results, this can be verified at design-time, but for a system with a very large number of resulting states this is unfeasible. Therefore, for SAS systems, in addition to the V&V tasks performed at design time, the system itself needs to apply different V&V methods at runtime. At this point, two questions arise: (i) what V&V methods are the most adequate to be applied at runtime? and (ii) at which specific moments in the adaptation process should these methods be applied?

These are certainly challenging questions, given that, additionally, many aspects of self-adaptive systems are impossible to assess at design time, due to their strong dependency on the actual execution environments. A recent US Air Force research agenda posits that developing certifiable V&V methods for highly adaptive systems is one of the major challenges facing the entire field. Understanding the inherent properties of adaptation mechanisms for software systems, and the ways in which these properties can be guaranteed may require a large part of the decade, if not more [10]. In the following sections we address these questions.

3.2 The Viability Zone of Self-Adaptive Software Systems

We define the *viability zone* of a SAS system as the set of possible system states in which the system operation is not compromised [21]. That is, the set of states where the SAS system's requirements and desired properties (i.e., adaptation goals) are satisfied. Viability zones can be characterized in terms of relevant context attributes and corresponding desired values. These context attributes correspond to either measurements of internal variables of the target system or the adaptation mechanism, or environmental variables whose variations can take the system outside its viability zone. A particular SAS system may have more than one associated viability zone (e.g., one for each adaptation goal). The global viability zone of a SAS system thus results from the composition of these partial viability zones. Moreover, existing viability zones can be added, replaced or adjusted by adding or removing variables of interest at runtime.

In the case of our application example, the initial viability zone is defined in terms of the performance SLA, and the three throughput SLOs (normal, medium and high capacity—cf. Use Case 1 in Sect. 2). These three SLOs constitute three different levels of system capacity that can been interpreted as three viability

sub-zones. The variables that characterize the e-commerce platform's viability zone correspond to the actual throughput of the *ProcessingPurchaseOrders* operation, the popularity of special offers placed in a social network (including whether an offer has been placed), and the shopping season, all of them to be monitored at runtime. Seasons are characterized in three groups: regular, medium (e.g., Valentine's Day), and high seasons (e.g., Christmas and Black Friday). Another associated viability zone in this example is used to control the short settling-time adaptation property. This zone is defined by a single-variable that is monitored to keep track of the time the e-commerce platform takes to reconfigure the system to obtain the desired throughput. Furthermore, after the re-negotiation of the performance SLA, a new viability zone must be computed at runtime to control the response time SLOs (cf. Use Case 2).

V&V under Viability Zone Dynamics. It can be argued that our definition of viability zone coincides with that of the *solution space* used in traditional (i.e., non-adaptive) software systems. However, from the previous examples it is clear that the viability zone can change with context changes, as opposed to the solution space concept, which is assumed to be fixed.

In effect, the viability zone of a target system under adaptation constantly varies along adaptation dimensions. These variations take place every time the adaptation operation modifies either the target system architecture (e.g., adding or removing components and connectors) or the controller itself (e.g., modifying its parameters or replacing the control algorithm), thus introducing new, or removing existing variables and associated domain types.

Therefore, not only are runtime V&V methods required to cope with the viability zone dynamics problem, but these V&V methods also need to be automatically generated according to the modifications that result from dynamic adaptation. Thus, to extend the V&V coverage of the expanded viability zone, runtime models are required for the incremental derivation of software artifacts for V&V monitoring and checking.

In the aforementioned example, understanding its viability zone dynamics is crucial for the self-adaptive e-commerce platform V&V tasks. In fact, the adaptation mechanism together with its V&V tasks can be interpreted as an optimization problem, where the optimal solution is chosen among those within the viability zone, based on the system capacity policies, as proposed by Balasubramanian *et al.* [22]. First, transitions between viability sub-zones are associated to an adaptation policy (adaptation strategy). For instance, when the system is approaching the threshold between a lower and a higher load—going from the lower to the higher, the corresponding adaptation task must be triggered to increase the system processing capacity accordingly (e.g., by deploying new components for scalable processing). Similarly, the system capacity must be reduced when it goes from a higher load to a lower one. In both cases, as the software component structure is modified as a result of the adaptation, the SCA structural conformance property must be verified at runtime on the resulting system. Second, changes in viability zones (e.g., changes in variables' thresholds, and addition or replacement of variables in adaptation dimensions) may affect not only the adaptation strategy, but

also the monitoring infrastructure, since these changes are caused by changes in adaptation goals. Finally, runtime V&V tasks aim to keep the adaptive system inside its viability zone, even when viability zones are subject to changes at runtime. The way how V&V tasks contribute to achieve this goal depends on the nature of the system and its requirements. For instance, for safety-critical applications, runtime V&V must check if the system will trespass the boundaries of its viability zone as a result of an adaptation, before instrumenting it in the running system. In those cases where self-adaptation is interpreted as an optimization problem, V&V tasks can be used both, before the adaptation, and after it. Before the adaptation, to restrict the alternatives to consider, to those within the viability zone. After the adaptation, to ensure that the solution is satisfying the new requirements under possibly changed context situations.

3.3 What: Requirements and Adaptation Properties

We identified the underlying V&V questions in the domain of SAS systems as *what*, *where*, and *when* to validate. This subsection focuses on the *what* to validate question. The answer to this question relates to the identification of adaptation goals (e.g., non-functional requirements of the target system) and adaptation properties (e.g., desired characteristics of the adaptation mechanism). Explicit adaptation goals and properties are crucial for the specification of suitable V&V models for SAS systems, and the identification of corresponding metrics. Moreover, having an explicit mapping between adaptation goals and properties, and relevant context is required to ensure the coherence between V&V tasks and the relevant context variables that characterize the system's viability zone. In our application example, we address this mapping by defining *context-driven SLAs* [11]. As proposed in [11], context-driven SLAs are machine readable specifications of SLAs, in the form of contextual resource description framework (RDF)[4] graphs, that not only state contracted conditions explicitly (e.g., the throughput and response time SLOs), but also the context variables, and context monitoring strategies required to keep track of the system behavior and its viability zone (e.g, sensors and monitoring conditions to measure throughput, response time, settling time, and the popularity of special offers, as well as identify shopping seasons).

Properties and Metrics. V&V concerns for self-adaptation certification can be classified according to the two constitutive parts of a SAS system. The first relates to the certification of the target system, while the second to the certification of the adaptation mechanism [5]. After the 2010 Dagstuhl Seminar on Software Engineering for Self-Adaptive Systems, researchers from the SEAMS community conducted an extensive analysis of self-adaptive approaches and developed a framework for evaluating self-adaptive systems, where desired properties of the target system (i.e., adaptation goals) and the adaptation mechanisms (i.e., adaptation properties) are identified explicitly and defined in terms of quality attributes [5].

[4] `http://www.w3.org/RDF/`

Several of the identified adaptation properties were borrowed from control theory [9, 23] and re-interpreted for self-adaptive software. Moreover, they classified adaptation properties according to *how* and *where* these properties are observed (cf. Table 1). Concerning how they are observed, some properties can be evaluated using static verification techniques, while others require dynamic verification and runtime monitoring (i.e., runtime V&V). With respect to where they are observed, properties can be evaluated on either the target system, or the adaptation mechanism. However, most properties can only be observed directly on the target system even when they are used to evaluate the adaptation mechanism.

Table 1. Classification of adaptation properties according to how and where they are observed [5]

Adaptation Property	Property Verification Mechanism	Where the Property is Observed
Stability	Dynamic	Target system
Accuracy	Dynamic	Target system
Settling Time	Dynamic	Both
Small Overshoot	Dynamic	Target system
Robustness	Dynamic	Adaptation Mechanism
Termination	Static	Adaptation Mechanism
Consistency	Both	Target system
Scalability	Dynamic	Both
Security	Dynamic	Both

Having no well defined and explicit metrics that can be used to assess properties, it is impossible to realize the vision of runtime V&V. Nevertheless, even though the importance of having such explicit metrics seems obvious, an important barrier for the assessment of dynamic software systems is the lack of accurate metrics to evaluate adaptive software [4]. Therefore, more research is required on the definition of applicable domain-specific metrics that effectively provide the means for evaluating relevant properties of dynamic software systems. Some examples of metrics and corresponding mappings to adaptation properties used in actual self-adaptive implementations and research initiatives, where non-functional requirements are a major concern, are summarized in the evaluation framework for self-adaptive software proposed by Villegas *et al.* [5].

An important challenge for V&V of SAS is to investigate innovative mechanisms that enable the application of techniques such as model checking, compositional verification, program synthesis, and dynamic analysis and monitoring to asses these properties at runtime. Another important research concern is the management of trade-offs that may arise from the need to ensure multiple properties—trade-offs among multiple viability zones.

Dependency on Runtime Monitoring. Besides using different representation models for target system behaviour, traditional V&V also uses controlled simulation environments. However, given the difficulties for building models to

predict self-adaptive system behavior for every possible operational situation, and the impossibility of characterizing these situations in simulation environments, V&V of context-dependent properties requires information gathered at runtime. For instance, in mission-critical systems, only with actual runtime measurements it is possible to determine confidently whether the target system is within its viability zone [24]. Understanding and characterizing which properties of self-adaptive software are critically dependent on runtime information is crucial for realizing V&V in SAS effectively.

Uncertainty in Self-Adaptation. Context dependent requirements usually involve uncertainty. Uncertainty can be both a challenge and an opportunity. In safety-critical systems uncertainty is a tough challenge that exacerbates verification tasks significantly [19, 24]. In other scenarios such as e-commerce applications, uncertainty is an opportunity, since the system can provide better service to customers by leveraging the context information that arise from the interactions between the users and the system, as well as from users' situations [25].

The adaptive nature of the execution environment in SAS systems makes uncertainty one of the most difficult challenges to be addressed by V&V researchers. An interesting research opportunity is to tailor feedback loop-based mechanisms used to manage uncertainty in modern control theory to context-aware SAS systems [26]. Similarly, the rich literature on engineering adaptive mechanisms for flight control systems inspires many researchers. In particular, Schumann and Gupta proposed a V&V method to calculate safety regions for adaptive systems around the current state of operation based on a Bayesian statistical approach [27]. With this approach, they can provide a confidence measurement on the probable accuracy of the system's model under a particular situation.

We argue for the exploitation of viability zones as useful mechanisms to manage uncertainty in the assurance of SAS systems. From this perspective, the management of uncertainty problem focusses on determining explicit boundaries for the SAS system's viability zones and controlling the target system accordingly (cf. Sect. 3.2).

3.4 Where: Separation of Concerns

We distinguish two system levels in SAS systems: the target system to be dynamically adapted according to context changes, and the adaptation mechanism. For runtime V&V it is critical to understand the extent of the separation of these two levels. This separation of concerns allows us to characterize, investigate, and analyze V&V research problems for self-adaptive software effectively, by focusing specifically on the respective concerns of each level.

Although the discussion in this chapter is applicable to both feedback and feedforward control in computing systems [9], we focus on feedback control since runtime V&V depends on online measurements from the target system and the adaptation mechanism. That is, measured outputs are important for making adaptive system quality decisions at runtime. Moreover, as feedforward control takes also environmental disturbances—external context—into account, subsequently

we use the terms feedback loop and control loop interchangeably. Following the feedback loop abstraction from the V&V perspective, the target system is an open loop for which the adaptation mechanism provides the elements to close the loop. In other words, the target software system itself is unaware of both context conditions and self-performance, with respect to the satisfaction of its own functional and non-functional (context-dependent) requirements. Thus, given that the objective of V&V is to guarantee the quality of a system, and this quality is expressed as the fulfillment of its requirements, in SAS systems V&V tasks must be incorporated as part of the adaptation loop. This implies that, in addition to the common context monitoring elements considered in feedback adaptation loops, additional components dedicated to verification and testing of the target system itself are required. At runtime, these components could, for instance, perform partial and incremental model checking on the next most probable states with respect to the current system state. Referred to our application example, the property to be verified could be the structural conformance of the reconfigured software application, with respect to the SCA structural constraints.

In addition, the separation of concerns between the target system and the adaptation mechanism implies different possible V&V interactions among these two system levels. Each of these interactions affects, in different ways, the ultimate goal of self-adaptation: the continued and effective operation of the target system services under varying context conditions. Of course, a general requirement is that the adaptation mechanism executes as unobtrusively and independently as possible from the target system. This observation has two implications. First, the target system functionalities must execute uninterruptedly for as long as possible while the adaptation mechanism performs the required adaptations on these functionalities. Moreover, the target system is expected to remain functional even if the V&V fails (i.e., if it indicates that the new system state is invalid). This implies that the adaptation mechanism must also run without interruptions. Second, unavailability of the adaptation mechanism should not cause unavailability of the target system. However, at some point, it is reasonable to expect that the adaptation mechanism, and even the target system itself, will require a shut down for maintenance or correcting system failures.

3.5 When: V&V in the Adaptation Process

Traditional V&V strategies involve checking and testing before system deployment under presumably well-defined conditions of system operation. This process of checking and testing is often automated using model checking, theorem proving, and testing tools. For context-dependent requirements, traditional V&V activities and certification techniques, designed to be applied before system deployment on fully specified requirements, are neither sufficient nor applicable. On the one hand, these formal V&V methods are often too expensive to be executed regularly at runtime when the system adapts due to their time and space complexity. On the other hand, context-dependent variables are unbound at design time, but bound at runtime. Thus, performing V&V on these variables at runtime is valuable to reduce the verification space significantly, even when the

SAS system viability zone varies with context changes. From this perspective, it is crucial to determine precisely when in the adaptation process these V&V operations are to be performed to guarantee the system properties and prevent unsafe operation. As previously mentioned, the lack of effective runtime V&V methods is considered one of the biggest obstacles and major challenges for the wide adoption of self-adaptive software applications in industry [10].

In addition, the considerations discussed in the previous section (i.e., the where) require the analysis of at least the following questions with respect to *when* to perform V&V tasks:

i. What properties can be exclusively verified at design time (executing neither the target system nor the adaptation mechanism)?
ii. What properties can be exclusively verified or tested at system configuration time?
iii. What properties can be exclusively verified or tested at runtime?
iv. What properties can be verified or tested either at design time, configuration time, or at runtime?

For instance, a machine-learning-based adaptive mechanism, such as the one proposed by Elkhodary *et al.* [28], could be checked for training coverage with respect to pre-defined adaptation cases at configuration time, before its deployment in production. However, the effectiveness of learned adaptations should be verified at runtime, based on information gathered from the actual adapted system behavior.

The answers to these questions are highly interdependent. For example, an approach aimed at verifying stability (what)—a behavioral adaptation property of the adaptation mechanism—may require the assessment of performance quality factors such as latency, throughput and capacity [5]. These factors assume runtime (when) monitoring on the target system (where). Stability is defined as the convergence of the subject system behavior toward a desired state. Moreover, many of the design concerns, such as availability, performance, survivability, fault tolerance and security, are highly interdependent and evolve at discrete points in time. It is critical to separate these concerns at design as well as at runtime.

In our application example, the performance SLA and its SLOs (throughput and response time—cf. Use Case 1 and Use Case 2 in Sect. 2), as well as the settling time and SCA structural conformance properties constitute the *what* to validate. Regarding the *where* question, throughput and response time must be observed on the target system, settling time must be observed on both the adaptation mechanism and the target system, whereas the SCA structural conformance, on the target system. Finally, concerning the *when* question, V&V tasks to ensure these requirements and properties must be performed at runtime.

In the following section we give some answers to the *when* and *where* questions by extending the feedback-loop elements with V&V responsibilities.

4 Making V&V Explicit in the Self-Adaptation Loop

So far, we have analyzed four key V&V drivers for SAS systems that pose major research challenges for SEAMS-related communities: (i) the viability zone and its dynamics; (ii) what to validate and verify, and its dependency on context information; (iii) where to validate—closely related to the separation of concerns between the target system and the adaptation mechanism; and (iv) when to perform V&V in SAS with respect to the adaptation loop.

To advance SAS goal assurance, we argue for the integration of runtime V&V tasks in the adaptation process. Accordingly, this section presents our proposal for making V&V tasks explicit in the elements of feedback adaptation loops, as for example in the MAPE-K loop [29]. Moreover, we discuss runtime V&V enablers (i.e., requirements at runtime, models at runtime, and dynamic context monitoring), which provide effective support to materialize V&V assurances for self-adaptation. Our proposal, depicted in Fig. 2, clearly answers *when* and *where* concrete V&V tasks can be implemented in the adaptation loop, using these enablers. The V&V enablers—dashed boxes in this figure—also provide a guide for other SEAMS-related research communities to contribute with runtime V&V methods for SAS systems. With this proposal we contribute to the convergence of these research communities towards the realization of suitable assurance mechanisms for SAS systems.

Applying this proposal to our application example, we use requirements at runtime to represent machine-readable specifications of the performance SLA, and its throughput and response-time SLOs. In this way, runtime validators and

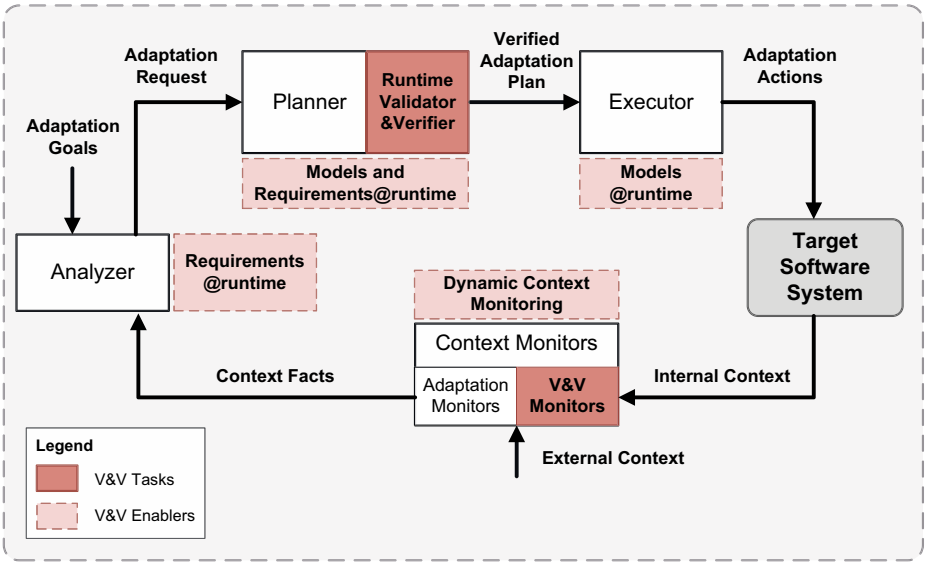

Fig. 2. Runtime V&V tasks made explicit as common elements in the engineering of self-adaptive software systems. Dashed boxes represent runtime V&V enablers.

verifiers can have access to the requirements and properties defined as adaptation goals that must be ensured by the adaptation process. Then, we use models at runtime to represent, the software architecture of the e-commerce platform to be adapted, the adaptation strategies, and the context monitoring strategies. Throughout the adaptation process, planners, runtime validators and verifiers, and executors use service component architecture (SCA) models to manipulate and adapt the system's software architecture, as well as to verify properties such as the SCA structural conformance, as realized in [30]. Similarly, we use models at runtime to represent the information gathered by context monitors as contextual RDF graphs. We exploit this form of representing context information to characterize the e-commerce platform state with respect to its viability zone, and perform inferences on this information to ensure SLAs at runtime [11].

4.1 Runtime V&V Tasks

We identify two particular elements in the adaptation loop that initiate runtime V&V tasks: *runtime validators & verifiers* (associated to the `Planner` element), and *V&V monitors* (associated to `Context Monitors` elements).

The Runtime Validator & Verifier. The responsibility of the *runtime validator & verifier* elements is to verify each of the outputs (i.e., adaptation plans) produced by the adaptation planner with respect to the properties of interest. The instrumentation of an adaptation plan on a given system execution state implies a change of the system state. Thus, the verification of these properties can be performed before or after instrumenting the plan.

In the case of our application example, concerning the SCA structural conformance property, the produced reconfiguration plans modify the target system's software architecture to obtain a new software structure to satisfy the agreed SLOs. To prevent execution failures, these plans must be verified, *before* instrumenting them, in such a way that the resulting structures satisfy the SCA integrity constraints (e.g., components, connectors, wires, and bindings). However, if the adaptation plan is for affecting the performance SLO, the corresponding verification should be performed *after* its instrumentation, with the new system's performance measurements. Moreover, on the new target system structure, partial and incremental verification could be performed also *in advance* on the most probable states that are immediately adjacent to the one generated by the adaptation plan. These states could be computed with statistical approaches such as the proposed in [27]. In addition, similar verification could be performed on the controller algorithm, if this is object of adaptation, such as in self-tuning control approaches [31, 32].

Nonetheless, different performance and synchronization issues between the executor and the runtime validator & verifier elements may appear. An example of this occurs when considering the previously introduced *in advance* partial and incremental V&V of the structural conformance property. In this case, the idea is to perform V&V not only on the state produced by the adaptation plan, but also on the most probable states that can immediately follow it, as a result of further

adaptation processes. Thus, runtime V&V elements could verify the property of interest on these states either *at the same time*, or *after* instrumenting the plan to reach the produced state. In other words, if the function computing the next most probable states is correct, the system structure to be obtained with the produced adaptation plan had to be verified in the previous adaptation. Alternatively, the execution of this "advance" runtime verification can be delayed, and even scheduled for later execution, for instance if this verification is highly time-consuming and can compromise the performance SLO of the target system.

Finally, in those cases in which the system state is represented and maintained explicitly (i.e., having a stateful representation via, for example, reflection or models at runtime), and the system is modified by an adaptation, this explicit state has to be transformed or updated accordingly. This is especially critical if the state is maintained in a volatile data structure, whose layout changes when the system is reconfigured. That is, if system information is represented in one form in its state variables, and then the system is reconfigured such that this information is represented in a different form, then the old values from the old structure must be mapped into new values in the new structure. Moreover, for the system operation to continue safely and uninterruptedly, it is crucial to (i) make the new state (information and structure) persistent (e.g., for recovery purposes); and (ii) be able to initialize the new system with the old information mapped into the new state. Hence, in these cases V&V must be performed not only on the adaptation process, but also on the state-mapping from the old structure to the new one.

The V&V Monitors. *V&V monitors* are responsible for monitoring and enforcing the V&V tasks performed by the runtime validator & verifier elements. Referring to the V&V tasks assigned to the runtime validator & verifier elements in the example of the previous subsection, we could use the V&V monitors to perform the aforementioned "advance" runtime verification. As outlined in the previous subsection, this is a verification task that can be scheduled by the *runtime validator & verifier* elements for later execution, to be performed on the most probable states to the current one in execution.

Assurance of Runtime V&V Tasks. Derived from the previous discussion, we identify the following questions, which pose additional challenges for ensuring the effectiveness of V&V tasks.

What if V&V fails or provides a negative answer? To prevent the target system from reaching inconsistent states and avoid catastrophic situations, one first strategy is to guarantee the atomicity property in the adaptation process, as defined in [5, 33]. That is, to guarantee that the adaptation process is an atomic operation that finishes and successfully modifies the target system, or it fails and the target system is left unmodified in its previous safe state. The verification of the atomicity and termination properties is a challenging problem, given that they should be guaranteed internally by the planner, and possibly requiring interactions with the executor. The use of models at runtime for modeling the

target system is crucial for guaranteeing these properties, for instance as realized by Tamura *et al.* [30].

How to validate "snapshots" and transitions between states without affecting the target system? V&V tasks must not affect the desired behaviour of the adaptive system. Therefore, we identify another kind of properties—properties of runtime V&V methods, intended to support the safe integration of traditional V&V techniques and mechanisms into the adaptation loop. These properties include *sensitivity, isolation, incrementality, and composability.*

As stated by González *et al.*, sensitivity and isolation refer to the level of runtime validation that a particular SAS system can support [34]. On the one hand, sensibility defines the degree to which V&V tasks (e.g., runtime testing operations) interfere with the running target system. That is, the degree to which runtime V&V may affect the satisfaction of system requirements and adaptation goals. Instances of factors that can affect runtime test sensitivity are (i) component state—not only because runtime validation tasks are influenced by the actual state of the system, but also because the state of the system can be altered as a result of V&V operations; (ii) component interactions—as the runtime testability of a component may depend on the testability of the components it interacts with; (iii) resource limitations—because runtime V&V may affect non-functional requirements on the target system, such as performance at undesirable levels; and (iv) availability—as runtime validation can be performed depending on whether testing tasks require exclusive usage of components with high availability requirements.

On the other hand, they also define isolation as the means to counteract runtime test sensitivity. Techniques for implementing test isolation include (i) state separation (e.g., blocking the component operation while testing takes place or performing testing on cloned components); (ii) interaction separation (e.g., blocking component interactions that may be propagated due to results of test invocations); (iii) resource monitoring (e.g., indicating that testing must be postponed due to resource unavailability); and (iv) scheduling (e.g., planning V&V executions when the target system and involved components are less used).

4.2 Runtime V&V Enablers

Runtime V&V techniques for SAS systems require special support to deal with the dynamic nature of this kind of systems in the assurance of adaptation goals. We classify this support in three main categories, as follows:

i. Enablers for the management of adaptation properties and requirements at runtime;
ii. Enablers for the exploitation of models at runtime; and
iii. Enablers for dynamic context monitoring.

Clearly, these categories correspond to challenges of the *Models@runtime* [35] and *Requirements@runtime* [36] communities, rather than research challenges of

V&V communities. Nevertheless, given that runtime V&V tasks for SAS rely on this support, with this categorization we aim to provide valuable guidance, not only for V&V researchers to understand the support that runtime V&V for SAS requires, but also for SEAMS-related researchers to visualize how could they attack runtime V&V challenges.

Requirements and Adaptation Properties at Runtime. The first category of support required for runtime V&V concerns the specification of *what* must be validated and verified. That is, the specification of the adaptation properties and system requirements the adaptation process must guarantee. In either case, V&V methods and techniques must determine whether the software product satisfies its requirements, especially after performing adaptation operations. These requirements and properties, expressed using different notations and formalisms, constitute the actual reference specifications for V&V tasks to accomplish their mission. Thus, requirements and adaptation goals must be available as machine-processable specifications (cf. Requirements@runtime in Fig. 2) to be used by adaptation analyzers, monitors, validators and verifiers. Furthermore, to minimize the impact of runtime V&V tasks on the adaptive system, support for tracing changes on requirements and properties is also required to identify what to validate and verify incrementally. Manipulating requirements and adaptation properties during the adaptive system execution poses interesting research questions such as the ones being addressed by the *Requirements@run.time* research community [36].

Models at Runtime. Having machine-processable models at runtime of the target system provides adaptation controllers, monitors, validators and verifiers with up-to-date structural and behavioral representations of the target system, and their relationships with adaptation properties and goals. Recalling our application example, after the renegotiation of the performance SLA (cf. Use Case 2 in Sect. 2), the runtime representations of the system and its requirements must change accordingly. That is, a new requirement is added to the context-driven SLA specification, as well as the corresponding monitoring strategy, using a contextual RDF graph. As a result, not only adaptation components, but also V&V tasks and monitors will have up-to-date representations of the new goals that must be ensured, and the corresponding context entities to be monitored.

Classifications of models at runtime vary from coarse-grained to fine-grained models, from structural to behavioural models, from dynamic to static models [35]. In this endeavor, researchers from the *Models@run.time* community are tackling important challenges [37, 38]. An instance of these challenges is model evolution, which concerns with the management of changes in models over time.

Model Evolution. Having an explicit representation of the target system, the properties to be preserved, and the relationships between these properties and adaptation mechanisms is critical for the assessment of SAS systems at runtime. At design time, models provide a meta-level representation of these concerns. At execution time, instances of these design time models, models at runtime, provide up-to-date representations of the system to V&V operations. These online

representations support decision making on the preservation of desired properties. Since SAS systems are continuously changing, the effectiveness of runtime V&V tasks depends on the timely coherence between the actual system state and its runtime models. Model evolution support is therefore required to preserve the coherence of runtime models with respect to the system and its environment.

Model evolution for SAS systems can borrow relevant ideas from control-based approaches. These approaches include model reference adaptive control (MRAC) and model identification adaptive control (MIAC) [31, 39]. MRAC and MIAC not only separate adaptation models from adaptation controllers, but also V&V models from V&V tasks. As illustrated in Fig. 3, MRAC and MIAC enable a basic level of model evolution by modifying the adaptation and V&V models at runtime. The main difference between MRAC and MIAC, from the perspective of runtime V&V, is that in MRAC changes in models are controlled by users, whereas in MIAC changes in models are managed by executors as defined by runtime validators and verifiers. Changes in models (cf. label ChM: Change model) may cause changes not only in adaptation controllers, but also in V&V tasks (cf. labels SCh: Send changes, and AC: Adapt controllers). Therefore, runtime support is required to adapt runtime validators and verifiers accordingly (cf. label AC-VV). Changes in V&V models could be triggered by adaptation mechanisms. In any case, these changes must be subject to V&V operations. From a software engineering perspective, the probabilistic approach to model synchronization proposed by Epifani *et al.* constitutes a good MIAC approach to model evolution [32].

Different model evolution mechanisms can be applied depending on the modeling technique used. For example, to synchronize UML models with corresponding systems, the model-driven engineering community provides model transformation techniques applicable at runtime [40]. In systems relying on probabilistic models, synchronization of models is realized by changing the model's parameters at runtime. One key challenge in probabilistic models is to synchronize the measured probabilities with the probabilities used in the model [32].

Dynamic Context Monitoring. The third category of V&V support corresponds to runtime context monitoring. Context monitoring is crucial to optimize the assessment of dynamic software systems as the effectiveness of V&V methods is highly dependent on the information provided by context sources [41]. These context information sources must be consistent with the actual system adaptation properties and requirements. Thus, for V&V tasks to succeed in the assessment of a SAS system, it must understand the situations of relevant context entities and their implications for the preservation of system properties and requirements, even when these requirements and properties vary over time.

Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject that can affect the behaviour of the system and/or its interaction with the user. Context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and

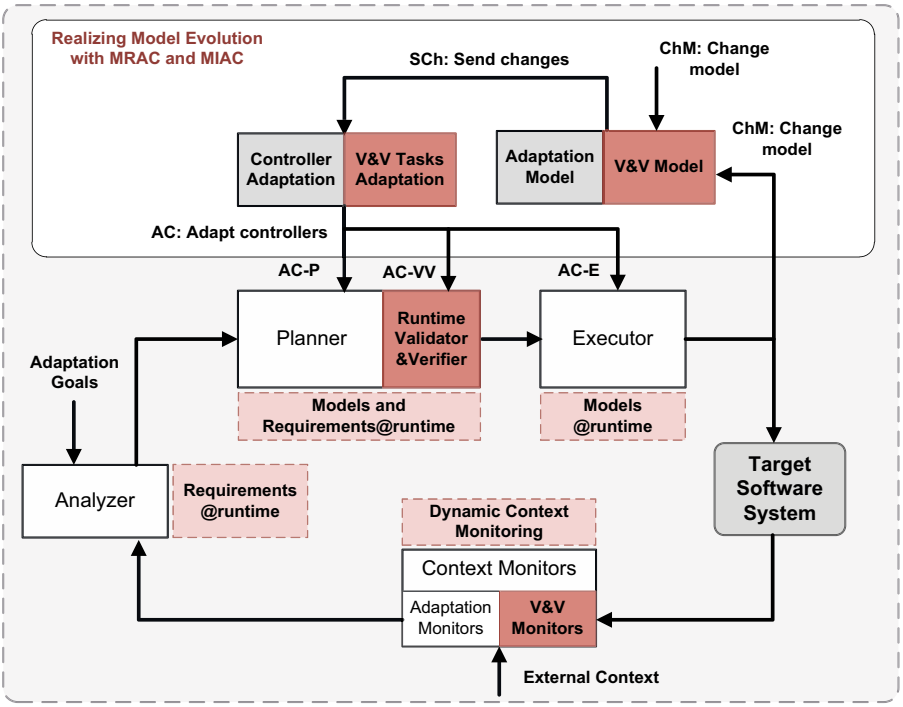

Fig. 3. MRAC and MIAC [31, 39] reinterpreted to realize model evolution in self-adaptation with explicit V&V tasks

maintained to support its dynamic evolution [42]. Based on this definition, and from the perspective of runtime V&V, runtime monitoring must support context representation and monitoring to characterize the system's state with respect to its viability zone, taking into account the dynamic nature of viability zones. Regarding context representation, operational specifications of context information must be able to represent semantic dependencies among properties and requirements to be satisfied, V&V strategies, and the environmental situations that have impact on the system behaviour and the assessment tasks. Hence, an important challenge refers to context representation such that these specifications can adapt dynamically, according to changes in V&V concerns. With respect to context management, several challenges arise from the perspective of the context information lifecycle, that is context acquisition, handling, provisioning, maintenance, and disposal [42]. One of these challenges is the instrumentation of monitoring infrastructures with dynamic capabilities to deploy new monitoring strategies at runtime according to changes in V&V concerns (e.g., to deploy new sensors or reasoning strategies based on changes in adaptation goals and properties dynamically [11, 25, 43]). In our application example, the monitoring infrastructure must be adapted at runtime to deploy new context sensors and monitoring conditions provided with the new response time SLO that resulted from contract re-negotiation (cf. Use Case 2 in Sect. 2).

Runtime monitoring could help alleviate issues concerning the application of traditional V&V techniques at runtime. An instance of such issues is the state explosion problem inherent in model checking techniques. In adaptive software systems, the uncertainty of the execution environment, the dynamic nature of system requirements, and the continuous adaptation of systems exacerbate the problem. From this perspective, we hypothesize that if we are able to characterize the current state of a system at a specific time during its execution, the number of system states to be checked could be significantly smaller. At design time many variables are free or not bounded, thus all of their possible significant values must be checked. In contrast, at runtime, variables are bound using the actual system state and the situation of relevant environmental (internal and external) entities. In other words, the number of possible states for the system to maintain within its viability zone is considerably reduced by the current and next most probable context situations. This is precisely where *context monitoring* plays a crucial role in the assessment of self-adaptive software systems. Nevertheless, due to the uncertainty inherent in dynamic software systems, it is infeasible to specify context requirements in advance exhaustively. Moreover, since context is evolving over time, monitoring requirements—entities to be monitored and monitoring conditions—are also continuously evolving. Therefore, the application of traditional V&V techniques to the assessment of self-adaptive systems at runtime depends on the dynamic capabilities of the runtime monitoring instrumentation.

5 Conclusions

In this roadmap chapter we (i) discussed key challenges for the development of certifiable runtime V&V methods that can certify adaptation mechanisms in the achievement of their adaptation goals; and (ii) presented how to make explicit and integrate runtime V&V methods as concrete tasks to be performed by elements of the adaptation process. Certifiable V&V methods and tools are critical for the success of autonomous, autonomic, smart, self-adaptive and self-managing systems.

We defined control science as a systematic way to study certifiable runtime V&V methods and tools to allow humans to trust decisions made by self-adaptive systems. For the first contribution, we analyzed critical differences between SAS systems and non-adaptive ones. From these differences, we identified and discussed key factors and challenges to consider when tailoring existing V&V methods, or developing new ones, to be applied at runtime in SAS systems. For the second, we discussed and illustrated different possibilities to integrate these V&V methods as responsibilities for the adaptation process elements. To enable this integration, we analyzed how to exploit some of the foundational ideas developed by SEAMS-related research communities to support the application of V&V methods at runtime.

While self-adaptation confers obvious benefits to systems with high levels of adaptability and autonomy, its wide adoption by industry is still limited due to a lack of self-applicable validation and verification methods at runtime. The positive impact that self-adaptive software can have on our society is potentially

huge. Nevertheless, without the necessary and sufficient trustworthy certification methods the negative impact can be potentially huge too. Consequently, research and development in runtime assurance techniques is critical to guarantee that adaptation mechanisms will not cause the target system to produce undesired, nor catastrophic results.

Therefore, our motivation for this chapter was to provide researchers with a vision of open challenges in V&V for SAS systems, and discuss opportunities not only for proposing new runtime V&V techniques, but also for building on top of existing ones. In addition, our proposal for making V&V tasks explicit in the adaptation loop provides solid starting points for V&V researchers from other communities to deploy different techniques and methods for improving the trustworthiness of self-adaptive and self-managing systems. For this, we analyzed runtime assessment concerns from the perspective of *when* in the adaptation process, and in which of the two parts of an adaptive system (i.e., the *where*)—the target system or the adaptation mechanism—the V&V tasks must be implemented and performed.

The questions discussed in this chapter have uncovered key research problems that require collaborative efforts among different software engineering research communities. In particular, models at runtime, requirements at runtime, validation and verification, and context monitoring have in the assessment of adaptive software a unique opportunity to advance the state-of-the-art software engineering for self-adaptive systems. With our contributions in this chapter we aim to provide researchers from various runtime V&V communities with research avenues that can shape the development of certifiable assurance techniques, as required for the engineering of trustworthy SAS systems.

Acknowledgments. This chapter was motivated by stimulating discussions during Dagstuhl Seminar 10431 on Software Engineering for Self-Adaptive Systems at Schloss Dagstuhl in October 2010. This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada under the Strategic Networks Grants Program (NETGP 397724-10) and Collaborative Research and Development program (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation, CA Inc., Icesi University (Cali, Colombia), and Ministry of Higher Education and Research of Nord-Pas de Calais Regional Council and FEDER under Contrat de Projets Etat Region (CPER) 2007-2013.

References

1. IEEE: 1012-1998: IEEE Standard for Software Verification and Validation. Technical report, Institute of Electrical and Electronics Engineers (2005)
2. IEEE: Industry Implementation of International Standard ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes. Technical report, IEEE (1996)
3. Bourque, P., Dupuis, R.: Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE Computer Society (2005)

4. Salehie, M., Tahvildari, L.: Self-Adaptive Software: Landscape and Research Challenges. ACM Transactions on Autonomous and Adaptive Systems 4, 14:1–14:42 (2009)
5. Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R.: A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. In: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 80–89. ACM, New York (2011)
6. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
7. Schafer, W., Wehrheim, H.: The Challenges of Building Advanced Mechatronic Systems. In: 2007 Future of Software Engineering (FOSE 2007), pp. 72–84. IEEE Computer Society, Washington, DC (2007)
8. Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A Systematic Survey of Program Comprehension through Dynamic Analysis. IEEE Transactions on Software Engineering (TSE) 35, 684–702 (2009)
9. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
10. Dahm, W.J.A.: Technology Horizons a Vision for Air Force Science & Technology During 2010-2030. Technical report, U.S. Air Force (2010)
11. Villegas, N.M., Müller, H.A., Tamura, G.: Optimizing Run-Time SOA Governance through Context-Driven SLAs and Dynamic Monitoring. In: 2011 IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011), pp. 1–10. IEEE (2011)
12. Beisiegel, M., Blohm, H., Booz, D., Edwards, M., Hurley, O., et al.: Service Component Architecture, Assembly Model Specification. Specification Version 1.0, Open Service Oriented Architecture (OSOA) Collaboration (2007)
13. Thayer, R.H., Bailin, S.C., Dorfman, M.: Software Requirements Engineering, 2nd edn. IEEE Computer Society Press, Los Alamitos (1997)
14. Dorfman, M.: System and Software Requirements Engineering, pp. 7–22. IEEE Computer Society Press Tutorial, IEEE Computer Society Press (1990)
15. Pezzè, M., Young, M.: Software Test and Analysis: Process, Principles and Techniques. John Wiley and Sons, Hoboken (2008)
16. Gat, E.: Autonomy Software Verification and Validation might not be as Hard as it Seems (AeroConf 2004). In: 2004 IEEE Aerospace Conference, pp. 3123–3128 (2004)
17. Bucchiarone, A., Pelliccione, P., Vattani, C., Runge, O.: Self-Repairing Systems Modeling and Verification Using AGG. In: 8th IEEE/IFIP Joint Working International Conference on Software Architecture (WICSA) and 3rd European Conference on Software Engineering (ECSA), pp. 181–190. IEEE (2009)
18. Bose, P., Quilling, M.: Model-Based Analysis of Autonomous Self-Adaptive Cooperating Robots. In: 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), pp. 57–63. IEEE Computer Society, Washington, DC (2008)

19. Murray, R.M. (ed.): Control in an Information Rich World: Report of the Panel on Future Directions in Control, Dynamics, and Systems. Society for Industrial and Applied Mathematics, Philadelphia (2003)
20. Baresi, L., Ghezzi, C.: The Disappearing Boundary between Development-time and Run-time. In: FSE/SDP Workshop on Future of Software Engineering Research (FoSER 2010), pp. 17–22. ACM, New York (2010)
21. Aubin, J., Bayen, A., Saint-Pierre, P.: Viability Theory: New Directions. Springer, Heidelberg (2011)
22. Balasubramanian, S., Desmarais, R., Müller, H.A., Stege, U., Venkatesh, S.: Characterizing Problems for Realizing Policies in Self-Adaptive and Self-Managing Systems. In: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 70–79. ACM, New York (2011)
23. Jacklin, S.A., Lowry, M.R., Schumann, J.M., Gupta, P.P., Bosworth, J.T., Zavala, E., Kelly, J.W.: Verification, Validation, and Certification Challenges for Adaptive Flight-Critical Control System Software. In: American Institute of Aeronautics and Astronautics AIAA Guidance Navigation and Control Conference and Exhibit. American Institute of Aeronautics and Astronautics, pp. 1–10 (2004)
24. Crum, V.W., Buffington, J.M., Tallant, G.S., Krogh, B., Plaisted, C., Prasanth, R., Bose, P., Johnson, T.: Verification & Validation of Intelligent and Adaptive Control Systems. In: IEEE Aerospace Conference (AeroConf. 2004), pp. 68–77. IEEE Computer Society (2004)
25. 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: 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2011), pp. 200–214. IBM Corp, Markham (2011)
26. Murray, R.M., Åström, K.J., Boyd, S.P., Brockett, R.W., Stein, G.: Future Directions in Control in an Information Rich World. IEEE Control Systems 23, 20–33 (2003)
27. Schumann, J., Gupta, P.: Bayesian Verification & Validation Tools for Adaptive Systems: Report on Principle of Operation and Prototypical Implementation of Bayesian Envelope Tool for Neural Networks. Technical report, National Aeronautics and Space Administration, NASA (2006)
28. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: A Framework for Engineering Self-Tuning Self-Adaptive Software Systems. In: 18th ACM International Symposium on Foundations of Software Engineering, FSE 2010, pp. 7–16. ACM, (2010)
29. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer 36(1), 41–50 (2003)
30. Tamura, G., Casallas, R., Cleve, A., Duchien, L.: QoS Contract-Aware Reconfiguration of Component Architectures Using E-Graphs. In: Barbosa, L.S. (ed.) FACS 2010. LNCS, vol. 6921, pp. 34–52. Springer, Heidelberg (2010)
31. Dumont, G., Huzmezan, M.: Concepts, Methods and Techniques in Adaptive Control. In: 2002 IEEE American Control Conference (ACC 2002), Anchorage, AK, USA, vol. 2, pp. 1137–1150 (2002)
32. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model Evolution by Run-Time Parameter Adaptation. In: 31st International Conference on Software Engineering (ICSE 2009), pp. 111–121. IEEE (2009)
33. Léger, M., Ledoux, T., Coupaye, T.: Reliable Dynamic Reconfigurations in a Reflective Component Model. In: Grunske, L., Reussner, R., Plasil, F. (eds.) CBSE 2010. LNCS, vol. 6092, pp. 74–92. Springer, Heidelberg (2010)

34. González, A., Piel, E., Gross, H.G.: A Model for the Measurement of the Runtime Testability of Component-Based Systems. In: 2009 International Conference on Software Testing Verification and Validation Workshops (ICSTW), pp. 19–28. IEEE (2009)
35. Bencomo, N., Blair, G., France, R., Muñoz, F., Jeanneret, C.: 4th International Workshop on Models@run.time. In: Ghosh, S. (ed.) MODELS 2009. LNCS, vol. 6002, pp. 119–123. Springer, Heidelberg (2010)
36. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-Aware Systems. A Research Agenda for RE For Self-Adaptive Systems. In: 18th International Requirements Engineering Conference (RE 2010), pp. 95–103. IEEE (2010)
37. Blair, G., Bencomo, N., France, R.: Models@run.time. IEEE Computer 42, 22–27 (2009)
38. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: 2007 Future of Software Engineering (FOSE 2007). IEEE Computer Society (2007)
39. Müller, H.A., Kienle, H.M., Stege, U.: Autonomic Computing Now You See It, Now You Don't. In: De Lucia, A., Ferrucci, F. (eds.) ISSSE 2006-2008. LNCS, vol. 5413, pp. 32–54. Springer, Heidelberg (2009)
40. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental Model Synchronization for Efficient Run-Time Monitoring. In: Ghosh, S. (ed.) MODELS 2009. LNCS, vol. 6002, pp. 124–139. Springer, Heidelberg (2010)
41. Goldsby, H., Cheng, B., 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)
42. Villegas, N.M., Müller, H.A.: Managing Dynamic Context to Optimize Smart Interactions and Services. In: Chignell, M., Cordy, J., Ng, J., Yesha, Y. (eds.) The Smart Internet. LNCS, vol. 6400, pp. 289–318. Springer, Heidelberg (2010)
43. Villegas, N.M., Müller, H.A.: Context-driven Adaptive Monitoring for Supporting SOA Governance. In: 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010). CMU/SEI-2011-SR-008, Pittsburgh: Carnegie Mellon University (2011)