Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 11, Issue 11, 2022 | ISSN: 1000-3428

Challenges in Ensuring Reliability and Adaptability: A Research Agenda for Self-adaptive Systems

Dr. Fabián Bustamante
Northwestern University, USA.

Abstract. The important concern for modern software systems is to become more cost-effective, while being versatile, flexible, resilient, dependable, energy-efficient, customisable, configurable and self-optimising when reacting to run-time changes that may occur within the system itself, its environment or requirements. One of the most promising approaches to achieving such properties is to equip software systems with self-managing capabilities using self-adaptation mechanisms. Despite recent advances in this area, one key aspect of self-adaptive systems that remains to be tackled in depth is the provision of assurances, *i.e.*, the collection, analysis and synthesis of evidence that the system satisfies its stated functional and non-functional requirements during its operation in the presence of self-adaptation. The provision of assurances for selfadaptive systems is challenging since run-time changes introduce a high degree of uncertainty. This paper on research challenges complements previous roadmap papers on software engineering for self-adaptive systems covering a different set of topics, which are related to assurances, namely, perpetual assurances, composition and decomposition of assurances, and assurances obtained from control theory. This research challenges paper is one of the many results of the Dagstuhl Seminar 13511 on *Software Engineering for Self-Adaptive Systems: Assurances* which took place in December 2013.

This is a post-peer-review, pre-copyedit version of an article published in Lecture Notes in Computer Science Vol. 9640. The final authenticated version is available online at: https://doi.org/10.1007/978-3-319-74183-3_1

1 Introduction

Repairing faults, or performing upgrades on different kinds of software systems have been tasks traditionally performed as a maintenance activity conducted off-line. However, as software systems become central to support everyday activities and face increasing dependability requirements, even as they have increased levels of complexity and uncertainty in their operational environments, there is a critical need to improve their resilience, optimize their performance, and at the same time, reduce their development and operational costs. This situation has led to the development of systems able to reconfigure their structure and modify their behaviour at run-time in order to improve their operation, recover from failures, and adapt to changes with little or no human intervention. These kinds of systems typically operate using an explicit representation of their own structure, behaviour and goals, and appear in the literature under different designations (*e.g.*, self-adaptive, self-healing, self-managed, self-*, autonomic). In particular, self-adaptive systems should be able to modify their behavior and/or structure in response to their perception of the environment and the system itself, and their goals.

Self-adaptive systems have been studied independently within different research areas of software engineering, including requirements engineering, modelling, architecture and middleware, event-based, component-based and knowledgebased systems, testing, verification and validation, as well as software maintenance and evolution [14,30]. On the one hand, in spite of recent and important advances in the area, one key aspect of self-adaptive systems that poses important challenges yet to be tackled in depth is the provision of assurances, that is, the collection, analysis and synthesis of evidence for building arguments that demonstrate that the system satisfies its functional and non-functional requirements during operation. On the other hand, the topic of assurances for

How to cite: Dr. Fabián Bustamante (2022). Challenges in Ensuring Reliability and Adaptability: A Research Agenda for Self-adaptive Systems. Journal of Computer Engineering, 11(11), 37–63. DOI: https://doi.org/10.5281/zenodo.19346948

software-based systems has been widely investigated by the dependability community, in particular when considered in the context of safety-critical systems. For these types of systems there is the need to build coherent arguments showing that the system is able to comply with strict functional and non-functional requirements, which are often dictated by safety standards and general safety guidelines [7]. However, distinct from conventional systems in which the assurances are provided in tandem with development, the provision of assurances for self-adaptive systems should also consider their operation because run-time changes (*e.g.*, resource variability) introduce a high degree of uncertainty.

In self-adaptive systems, since changes and uncertainty may affect the system during its operation, it is expected that assurances also need to be perpetually revised depending on the type and number of changes, and how the system selfadapts to these changes in the context of uncertainties. In order to allow the continuous revision of assurances, new arguments need to be formed based on new evidence or by composing or decomposing existing evidence. Concepts and abstractions for such evidence can be obtained from classical control theory for providing reasoning about whether the feedback loop enabling self-adaption is able to achieve desired properties. Moreover, feedback loops supported by processes should provide the basis for managing the continuous collection, analysis and synthesis of evidence that will form the core of the arguments that substantiate the provision of assurances.

The goal of this paper on the provision of assurances for self-adaptive systems is to summarize the topic state-of-the-art and identify research challenges yet to be addressed. This paper does not aim to supersede the previous roadmap papers on software engineering of self-adaptive systems [14,30], but rather to complement previous papers by focusing on assurances. Though assurances were lightly treated in the former papers, this paper goes in more depth on topics that pose concrete challenges for the development, deployment, operation, evolution and decommission of self-adaptive systems, and identifies potential research directions for some of these challenges.

In order to provide a context for this paper, in the following, we summarize the most important research challenges identified in the two previous roadmap papers. In the first paper [14], the four topics covered were: *modeling dimensions*, where the challenge was to define models that can represent a wide range of system properties; *requirements*, where the challenge was to define languages capable of capturing uncertainty at the abstract level of requirements; *engineering*, where the challenge was to make the role of feedback control loops more explicit; *assurances*, where the challenge was how to supplement traditional V&V methods applied at requirements and design stages of development with run-time assurances. In the second paper [30], the four topics covered were: *design space*, where the challenge was to define what is the design space for self-adaptive systems, including the decisions the developer should address; *processes*, where the challenge was to define innovative generic processes for the development, deployment, operation, and evolution of self-adaptive systems; *decentralization of control loops*, where the challenge was to define architectural patterns for feedback control loops that would capture a varying degree of centralization and decentralization of the loop elements; *practical run-time verification and validation (V&V)*, where the challenge was to investigate V&V methods and techniques for obtaining inferential and incremental assessments for the provision of assurances.

For the motivation and presentation of the research challenges associated with the provision of assurances when engineering self-adaptive systems, we divide this paper into three parts, each related to one of the identified key research challenges. For each key research challenge, we present a description of the topic related to the challenge, and suggest future directions of research around the identified challenges to the topic. The three key research challenges are: perpetual assurances (Section 2), composition and decomposition of assurances

(Section 3), and what can we learn from control theory regarding the provision of assurances (Section 4). Finally, Section 5 summarizes our findings.

2 Perpetual Assurances

Changes in self-adaptive systems, such as changes of parameters, components, and architecture, are shifted from development time to run-time. Furthermore, the responsibility for these activities is shifted from software engineers or system administrators to the system itself. Hence, an important aspect of the software engineering process for self-adaptive systems, in particular business or safety critical systems, is providing new evidence that the system goals are satisfied during its entire lifetime, from inception to and throughout operation until decommission. The state of the art advocates the use of formal models as one promising approach to providing evidence for goal compliance. Several approaches employ formal methods to provide such guarantees by construction. In particular, recent research suggests the use of probabilistic models to verify system properties and support decision-making about adaptation at run-time. However, providing assurances for the goals of self-adaptive systems that must be achieved during the entire lifecycle remains a difficult challenge. This section summarizes a background framework for providing assurances for self-adaptive systems that we term "perpetual assurances for self-adaptive systems." We discuss uncertainty as a key challenge for perpetual assurances, requirements for solutions addressing this challenge, realization techniques and mechanisms to make these solutions effective, and benchmark criteria to compare solutions. For an extensive description of the background framework for perpetual assurance, we refer the reader to [46].

2.1 Uncertainty as a Key Challenge for Perpetual Assurances

We use the following working definition for *perpetual assurances for self-adaptive systems*:

Perpetual assurances for self-adaptive systems mean providing evidence for requirements compliance through an enduring process that continuously provides new evidence by combining system-driven and humandriven activities to deal with the uncertainties that the system faces across its lifetime, from inception to operation in the real world.

Thus, providing assurances cannot be achieved by simply using off-line solutions, possibly complemented with on-line solutions. Instead, we envision that perpetual assurances will employ a continuous process where humans and the system jointly and continuously derive and integrate new evidence and arguments required to assure stakeholders (*e.g.*, end users and system administrators) that the requirements are met by the self-adaptive system despite the uncertainties it faces throughout its lifetime.

A primary underlying challenge for this process stems from *uncertainty*. There is no agreement on a definition of uncertainty in the literature. Here we provide a classification of sources of uncertainty based on [33] using two dimensions of a taxonomy for uncertainty proposed in [35], *i.e.*, *location* and *nature*. *Location* refers to where uncertainty manifests in the description (the model) of the studied system or phenomenon. We can specialize it into: (i) input parameters, (ii) model structure, and (iii) context. *Nature* indicates whether the uncertainty is due to the lack of accurate information (*epistemic*) or to the inherent variability of the phenomena being described (*aleatory*). Recognizing the presence of uncertainty and managing it can mitigate its potentially negative effects and increase the level of assurance in a self-adaptive system. By ignoring uncertainties, one could draw unsupported claims on system validity or generalize them beyond their bounded scope.

Table 1 shows our classification of uncertainty based on [33], which is inspired by recent research on self-adaptation, *e.g.*, [18,21]. Each source of uncertainty is classified according to the location and nature dimensions of the taxonomy.

Table 1. Classification of sources of uncertainty based on [33].

Source of Uncertainty		Classification	
		Location	Nature
Simplifying assumptions	System	Structural/context	Epistemic
Model drift		Structural	Epistemic
Incompleteness		Structural	Epistemic /Aleatory
Future parameters value		Input	Epistemic
Adaptation functions		Structural	Epistemic /Aleatory
Automatic learning		Structural/input	Epistemic /Aleatory
Decentralization		Context/structural	Epistemic
Requirements elicitation	Goals	Structural/input	Epistemic /Aleatory
Specification of goals		Structural/input	Epistemic /Aleatory
Future goal changes		Structural/input	Epistemic /Aleatory
Execution context	Context	Context/structural/input	Epistemic
Noise in sensing		Input	Epistemic /Aleatory
Different sources of information		Input	Epistemic /Aleatory
Human in the loop	Human	Context	Epistemic /Aleatory
Multiple ownership		Context	Epistemic /Aleatory

Sources of uncertainty are structured in four groups: (i) uncertainty related to the *system* itself; (ii) uncertainty related to the system *goals*; (iii) uncertainty in the execution *context*; and (iv) uncertainty related to *human* aspects. Uncertainty in its various forms represents the ultimate source of both motivations for and challenges to perpetual assurance. Uncertainty manifests through *changes*. For example, uncertainty in capturing the precise behavior of an input phenomenon to be controlled results in assumptions made during the implementation of the system. Therefore, the system must be calibrated later when observations of the physical phenomenon are made. This in turn leads to changes in the implemented control system that must be scrutinized to derive assurances about its correct operation.

2.2 Requirements for Solutions that Realize Perpetual Assurances

The provision of perpetual assurance for self-adaptive systems must cope with a variety of uncertainty sources that depend on the purpose of self-adaptation and the environment in which the assured self-adaptive system operates. To this end, they must build and continually update their assurance arguments through the integration of two types of assurance evidence. The first type of evidence corresponds to system and environment components not affected significantly by uncertainty, and therefore can be obtained using traditional off-line approaches [28]. The second type of evidence is associated with the system and environment components

affected by the sources of uncertainty summarized in Table 1. This type of evidence is required for each of the different functions of adaptation: from sensing to monitoring, analyzing, planning, executing, and activating [15]. The evidence must be synthesized at run-time, when the uncertainty is treated, *i.e.*, reduced, quantified or resolved sufficiently to enable such synthesis.

Table 2 summarizes the requirements for perpetual assurance solutions. R1– R7 are functional requirements to treat uncertainty. R8–R10 are non-functional requirements to provide assurance that are timely, non-intrusive and auditable.

2.3 Approaches to Perpetual Assurances

Several approaches have been developed in previous decades to check whether a software system complies with its requirements. Table 3 below gives an overview of these approaches organized in three categories: human-driven approaches (manual), system-driven (automated), and hybrid (manual and automated).

We briefly discuss one representative approach of each group. *Formal proof* is a human-driven approach that uses a mathematical calculation to prove a sequence of related theorems that refer, or are based upon, a formal specification of the system. Formal proofs are rigorous and unambiguous, but can only be produced by experts with both detailed knowledge about how the self-adaptive system works and significant mathematical experience. As an example, in [51] the authors formally prove a set of theorems to assure safety and liveness properties of self-adaptive systems. The approach is illustrated for data stream components that modify their behavior in response to external conditions through the insertion and removal of filters. *Run-time verification* is a system-driven approach that is based on extracting information from a running system to detect whether certain properties are violated. Run-time verification is less complex than traditional formal verification because only one or a few execution traces are analyzed at a time. As an example, in [40] the authors introduce an approach for estimating the probability that a temporal property is satisfied by a run of a program. *Model checking* is a well-known hybrid approach that allows designers to check that a property holds for all reachable states in a system. Model checking can be applied off-line or on-line, and can only work in practice on a high-level abstraction of an adaptive system or on one of its components. For

Table 2. Summary requirements.

Requirement	Brief description
R1: Monitor uncertainty	A perpetual assurance solution must continually observe the sources of uncertainty affecting the self-adaptive system
R2: Quantify uncertainty	A perpetual assurance solution must use its observations of uncertainty sources to continually quantify and potentially mitigate the uncertainties affecting its ability to provide assurance evidence
R3: Manage overlapping uncertainty sources	A perpetual assurance solution must continually deal with overlapping sources of uncertainty and may need to treat these sources in a composable fashion
R4: Derive new evidence	A perpetual assurance solution must continually derive new assurance evidence arguments
R5: Integrate new evidence	A perpetual assurance solution must continually integrate new evidence into the assurance arguments for the safe behavior of the assured self-managing system

R6: Combine new evidence	A perpetual assurance solution may need to continually combine new assurance evidence synthesized automatically and provided by human experts
R7: Provide evidence for the components and activities that realize R1-R6	A perpetual assurance solution must provide assurance evidence for the system components, the human activities, and the processes used to meet the previous set of requirements
R8: Produce timely updates	The activities carried out by a perpetual assurance solution must produce timely updates of the assurance arguments
R9: Limited overhead	The activities of the perpetual assurance solution and their overheads must not compromise the operation of the assured self-adaptive system
R10: Auditable arguments	The assurance evidence produced by a solution and the associated assurance arguments must be auditable by human stakeholders

Table 3. Approaches for assurances.

Assurances approach category	**Examples**
Human-driven approaches	Formal proof Simulation
System-driven approaches	Run-time verification Sanity checks Contracts
Hybrid approaches	Model checking Testing

example, [25] models MAPE loops of a mobile learning application as timed automata and verifies robustness requirements expressed in timed computation tree logic using the UPPAAL tool. In [10] QoS requirements of service-based systems are expressed as probabilistic temporal logic formulae, which are automatically analyzed at run-time to identify optimal system configurations.

2.4 Mechanisms for Turning Perpetual Assurances into Reality

Turning the approaches of perpetual assurances into a working reality requires aligning them with the requirements discussed in Section 2.2. For the functional requirements (R1 to R7), the central problem is how to build assurance arguments based on the evidence collected at run-time, which should be composed with the evidence acquired throughout the lifetime of the system, possibly by different approaches. For the quality requirements (R8 to R10) the central problem is to make the solutions efficient and to support the interchange of evidence between the system and its users. Efficiency needs to take into account the size of the self-adaptive system and the dynamism it is subjected to. An approach for perpetual assurances is efficient if it is able to: (i) provide results (assurances) within defined time constraints (depending on the context of use); (ii) consume an acceptable amount of resources, so that the resources consumed over time (*e.g.*, memory size, CPU, network

bandwidth, energy, etc.) remain within a limited fraction of the overall resources used by the system; and (iii) scale well with respect to potential increases in size of the system and the dynamism it is exposed to.

It is important to note that orthogonal to the requirements for perceptual assurance in general, the level of assurance that is needed depends on the requirements of the self-adaptive system under consideration. In some cases, combining regular testing with simple and time-effective run-time techniques, such as sanity checks and contract checking, will be sufficient. In other cases, more powerful approaches are required. For example, model checking could be used to verify a safe envelope of possible trajectories of an adaptive system at design time, and verification at run-time to check whether the next change of state of the system keeps it inside the pre-validated envelope. We briefly discuss two classes of mechanisms that can be used to provide the required functionalities for perpetual assurances and meet the required qualities: decomposition mechanisms and model-driven mechanisms.

Decomposition Mechanisms. The first class of promising mechanisms for the perpetual provisioning of assurances is based on the principle of decomposition, which can be carried out along two dimensions:

1. Time decomposition, in which: (i) some preliminary/intermediate work is performed off-line, and the actual assurance is provided on-line, building on these intermediate results; (ii) assurances are provided with some degree of approximation/coarseness, and can be refined if necessary.
2. Space decomposition, where verification overhead is reduced by independently verifying each individual component of a large system, and then deriving global system properties through verifying a composition of its component-level properties. Possible approaches that can be used are: (i) flat approaches, that exploit only the system decomposition into components; (ii) hierarchical approaches, where the hierarchical structure of the system is exploited; (iii) incremental approaches targeted at frequently changing systems, in which re-verification are carried out only on the minimal subset of components affected by a change.

Model-based Mechanisms. For any division of responsibilities between human and systems in the perpetual assurance process, an important issue is how to define in a traceable way the interplay between the actors involved in the process. Model-driven mechanisms support the rigorous development of a selfadaptive system from its high-level design up to its running implementation, and they support traceable modifications of the system by humans and/or of its self-adaptive logic, *e.g.*, to respond to modifications of the requirements. In this direction, [45] presents a domain-specific language for the modeling of adaptation engines and a corresponding run-time interpreter that drives the adaptation engine operations. The approach supports the combination of on-line machine-controlled adaptations and off-line long-term adaptations performed by humans to maintain and evolve the system. Similarly, [24] proposes an approach called ActivFORMS in which a formal model of the adaptation engine specified in timed automata and adaptation goals expressed in timed computation tree logic are complemented by a virtual machine that executes the verified models, guaranteeing at run-time compliance with the properties verified off-line. The approach supports on-the-fly deployment of new models by human experts to deal with new goals.

2.5 Benchmark Criteria for Perpetual Assurances

We provide benchmark criteria for comparing four key aspects of perpetual assurance approaches: approach capabilities, basis of evidence for assurances, stringency of assurances, and performance. The criteria, shown in Table 4, cover both functional and quality requirements for perpetual assurances approaches.

Table 4. Summary of benchmark aspects and criteria for perpetual assurances.

Benchmark	Benchmark Criteria	
	Criteria	**Description**

Aspect		
Capabilities of approaches to provide assurances	Variability	Capability of an approach to handle variations in requirements (adding, updating, deleting goals), and the system (adding, updating, deleting components)
	Inaccuracy & incompleteness	Capability of an approach to handle inaccuracy and incompleteness of models of the system and context
	Competing criteria	Capability of an approach to balance the tradeoffs between utility (*e.g.*, coverage, quality) and cost (*e.g.*, time, resources)
	User interaction	Capability of an approach to handle changes in user behavior (preferences, profile)
	Handling alternatives	Capability of an approach to handle changes in adaptation strategies (*e.g.*, pre-emption)
Basis of assurance benchmarking	Historical data only	Capability of an approach to provide evidence over time based on historical data
	Projections in the future	Capability of an approach to provide evidence based on predictive models
	Combined approaches	Capability of an approach to provide evidence based on combining historical data with predictive models
	Human evidence	Capability of an approach to complement automatically gathered evidence by evidence provided by humans
Stringency of assurances	Assurance rational	Capability of the approach to provide the required rational of evidence for the purpose of the system and its users (*e.g.*, completeness, precision)
Performance of approaches	Timeliness	The time an approach requires to achieve the required evidence
	Computational overhead	The resources required by an approach (*e.g.*, memory and CPU) for enacting the assurance approach
	Complexity	The scope of applicability of an approach to different types of problems

Several criteria from Table 4 directly map to a requirement from Table 2. For example, 'Timeliness' directly links to requirement R8 ('Produce timely updates'). Other criteria correspond to multiple requirements. For example, Human evidence links to R5 ('Integrate new evidence'), R7 ('Provide evidence for human activities that realize

R5'), and R10 ('Auditable arguments'). Other arguments only link indirectly to requirements from Table 2. This is the case for the 'Handling alternatives' criterion, which corresponds to the solution for selfadaptation, which may provide different levels of support for the requirements of perpetual assurances.

2.6 Research Challenges

Assuring requirements compliance of self-adaptive systems calls for an enduring process where evidence is collected over the lifetime of the system. This process for the provision of perpetual assurances for self-adaptive systems poses four key challenges.

First, we need a better understanding of the nature of uncertainty for software systems and of how this translates into requirements for providing perpetual assurances. Additional research is required to test the validity and coverage of this set of requirements.

Second, we need a deeper understanding of how to monitor and quantify uncertainty. In particular, how to handle uncertainty in the system, its goal and its environment remains to a large extent an open research problem.

Third, the derivation and integration of new evidence pose additional hard challenges. Decomposition and model-based reasoning mechanisms represent potential approaches for moving forward. However, making these mechanisms effective is particularly challenging and requires a radical revision of many existing techniques.

Last but not least, to advance research on assurances for self-adaptive systems, we need self-adaptive system exemplars (*e.g.* [28,47]) that can be used to assess the effectiveness of different solutions.

3 Composing and Decomposing Assurances

Assuring a self-adaptive system in all the configurations that it could possibly be in, under all the environments it can encounter, is challenging. One way to address this challenge is to understand how to *decompose assurances* so that an entire revalidation is not required at run-time time when the system changes. Another source of challenges for assuring self-adaptive systems is when they are composed together to create larger systems (for example, having multiple adaptive systems in autonomous vehicles, or having multiple adaptive systems managing a building). Typically, assurances are also required for this systemsof-systems context. We therefore need ways to *compose assurances* that do not require complete revalidation of each of the constituent parts.

For safety-critical systems there is a large body of work on constructing safety cases [8], or more generally assurance cases [6], that allow engineers to build assurance arguments to provide confidence that a system will be safe (in addition to other qualities). How these assurance cases are constructed for safety-critical systems can shed some light on how to provide assurances for self-adaptive systems. Typically, building assurance cases involve decomposing top level goals into argumentation structures that involve sub-goals, strategies for achieving the goals, and defining evidence that can be collected to show that the goals are achieved. For example, a safety case presents a structured demonstration that a system is acceptably safe in a given context – *i.e.*, it is a comprehensive presentation of evidence linked by argument to a claim. Structuring evidence in such a way means that an expert can make a judgment that the argument makes sense and thus, if the evidence in the case is provided, have confidence that the system is acceptably safe. Assurance cases are a generalization of safety cases to construct arguments that are about more than just safety.

Assurance cases themselves can be composed together to provide assurances about a system with multiple goals, to reuse some assurances for goals in similar systems, or to provide assurances in systems-of-systems contexts.

We can therefore use work on assurance case construction and composition as a guide to how to decompose and compose assurances for self-adaptive systems. For an extensive description of the ideas presented in this section, we refer the reader to [37].

3.1 Assurances in Self-Adaptive Systems

While the focus of much of the research in self-adaptive systems to date has been to engineer systems that can maintain stated goals, especially in the presence of uncertain and changing environments, there is existing research in assurances for self-adaptive systems that either addresses how to compose assurances, or can be used as part of an argument in assurance cases. We categorize existing work into the following areas:

Evidence types and sub-goals for use in assurance case decomposition. Each of the classic activities of self-adaptation—monitoring, analysis, planning, and execution—have existing techniques that help to provide evidence for goals that can be used in assurance cases. For example, [11,12] provide theoretical foundations based on information theory for determining if a selfadaptive system has enough information to diagnose problems. In [1], contextual goals can be used to identify modeling requirements. Models and simulations can provide evidence about whether adaptation should be done. Models (particularly performance models) have been used in [16,31], for example. Formal models have also been used to provide evidence that adaptations will achieve desired results, for example probabilistic models in [19,20,38].

Assurance composition based on the MAPE-K loop. Once assurances have been decomposed, and evidence sources have been identified, we need ways to recompose the assurances. Fortunately, in self-adaptive systems, there is

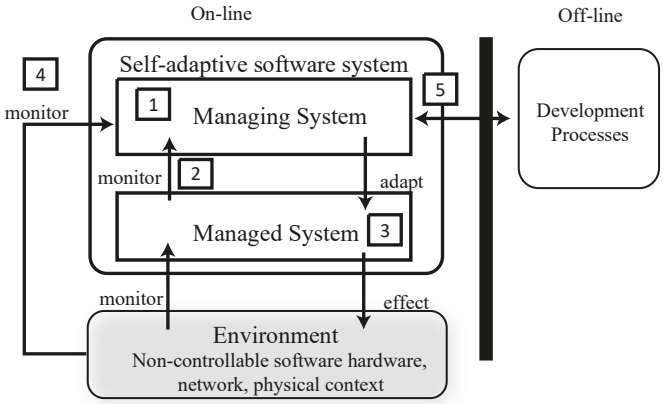

Fig.1. Architectural Reference Model for Self-Adaptive Software Systems.

research that takes advantage of the common MAPE-K pattern used for constructing self-adaptive systems. The integration of V&V tasks into the MAPE-K loop is discussed in [41]. Both [41] and [48] discuss the different modes of self-adaptive systems (called viability zones) that can guide what assurance techniques can be used in and between each mode.

Combining these approaches with work on assurance cases can lead to a principled way of designing, structuring, and adapting assurances for self-adaptive systems.

Decomposing Assurances in Self-Adaptive Systems. Assurance cases naturally guide how to decompose assurances into subgoals and evidence. For selfadaptive systems there are a number of challenges centered on

(a) what types of evidence can be provided, and (b) where that evidence might come from. A reference model for self-adaptive systems, depicted in Figure 1, can be used as a guide for addressing these challenges. This reference model can help to identify architectural concerns for self-adaptation that require assurances and that should be included in the argumentation structure. For example, we must provide convincing evidence that the managing system:

- makes a correct decision about when and how to adapt the managed system (cf.in Figure 1),
- correctly monitors the managed systemand that and the environment are correct such that the managing system can rely on them, the assurances and assumptions provided for the managed system and environment are
- correctly adapts the managed system, which in turn must change according to this adaptation,
- correctly interacts with the development processthrough which engineers directly adapt the managed system, or change the managing system itself (*e.g.*, to add new adaptation strategies).

This model can guide strategy selection and evidence placement for both functional and extra-functional goal decomposition. Within this, though, there are additional issues to consider when determining the argumentation structures:

Time and Lifecycle. Evidence for self-adaptive systems can be generated during development or at run-time. On-line techniques can be embedded in the managing systems to provide evidence at run-time. Off-line techniques are traditionally used during development or maintenance to provide static evidence. The managing system may, however, involve off-line techniques at run-time as an alternative for resource demanding evidence generation. The work presented in [36] discusses how one might use off-line formal verification results when reusing components; such work could also inspire formal verification reuse during run-time.

Independence and additivity. The inherent flexibility in a self-adaptive system suggests that argumentation structures be generated and adapted dynamically to reflect system adaptations. One aspect that promotes dynamic argumentation is independent evidence, which can be added to support arguments. We discuss this matter in more detail below. However, it is clear that if two or more evidence types are equally strong, the more independent should be favored. This will reduce the complexity of combining evidence for parts when assuring compositions.

Evidence completeness. Complete evidence is key for assurance cases. The argumentation structure should provide convincing arguments. For a selfadaptive system, completeness is affected by time and lifecycle concerns. As mentioned above, some evidence is not generated before run-time, thus complete evidence will be unavailable during development. In fact, completeness can change during the lifecycle. Evidence violations may, for instance, trigger adaptations, which implies that from time to time, evidence may transition from complete to incomplete and back to complete after a successful adaptation.

Composing Assurances in Self-Adaptive Systems. The argumentation structures with goals, strategies, and evidence have a close analogy with validation & verification. For example, individual decomposed assurance cases have similarities with unit tests, while composing assurances is more closely aligned with integration and system testing. Unit tests are run without the global view of the system, and may either over- or under-approximate system behavior, while integration and system tests consider the environment under which they are run. Test dependencies, a subfield of software testing, provides some pointers to root causes for dependencies and consequences on a system's testability and design for testability is mentioned as a viable resolution strategy. Dependencies between goals, strategies, and evidence have similar negative effects on the assurance case composition. We exemplify some causes and their effects in our analysis of three composition scenarios below.

1. Combining assurance cases for different goals of the same system: Consider the managing system in Figure 1 and two goals, one for self-optimization and one for self-protection. The decomposition strategy described above generates two separate argumentation structures, one for each goal. We may not find the two top-level goals conflicting, however, parts in the argumentation structures may have explicit or implicit dependencies. For example, the inner components in the MAPE-K loop create numerous implicit and explicit inter-dependencies, which impact composability negatively.
2. Combining assurance cases for two different systems: In this case, we consider a situation where two independently developed systems are composed. We need to examine goal, evidence, and resource dependencies the composition creates. For example, some goals may subsume other goals, *i.e.*, the weakest claim needs to be replaced with the strongest claim. Further analysis of resource and evidence dependencies will indicate if evidence is independent and may be reused or if new evidence is required for the composition.
3. Combining multiple assurances for multiple systems composed in a systemsof-systems context: This is the extreme case of scenario 2. The analysis will be more complex and hence, also conflict resolution and evidence generation.

These issues are also challenging for the assurance community. Work in assurance case modularization [26,50] can address both assurance case decomposition and composition through the use of contracts. Contracts and modularization of assurance cases will help with independence and additivity. Furthermore, [7] points out that assurance case decomposition is seldom explicit and that the assurance case community needs to develop rigorous decomposition strategies. These efforts should be tracked so that insights can be transferred to self-adaptive systems.

3.2 Research Challenges

In this section we have proposed that assurance cases can be used to guide decomposition and composition of assurances for self-adaptive system. We have shown how self-adaptive systems themselves might help in informing how to decompose and compose assurance cases, and suggested that the assurance case community is addressing some of the challenges raised. However, there a number of challenges that arise when trying to apply assurance cases to self-adaptation, which researchers in this area should investigate further:

Uncertainty. Self-adaptive systems are often self-adaptive because they are deployed in environments with uncertainty. This uncertainty affects the types of evidence that can be collected to support assurances, the ways in which the evidence can be collected, and even the specification of the assurance case itself. For example, goals in assurance cases need to specify the environmental assumptions under which they are valid, but for self-adaptive systems we need some way to make uncertainty about these assumptions first-class.

Assurance case dependencies: Goals, strategies, and evidence create a complex dependency web that connects argumentation structures. This web impacts how we derive and combine assurance cases negatively. A better understanding of the nature of these dependencies and how to mitigate their consequences will improve current practice for assurance case decomposition and composition. Existing work on testability and reuse of validation and verification results could be the point of departure.

Adaptation assurances. When conditions change and the system adapts, an assurance may describe how quickly or how well it adapts. For example, increased demand may trigger the addition of a web server. An assurance may state that when the per-server load exceeds a threshold, the system adapts within two minutes by adding web servers and the per-server load falls below the threshold within five minutes. This assurance may hold at all times, or may be expected to hold only when the demand increases but then remains constant.

Automatable assurance cases. Assurance cases rely on human judgment to discern whether the argument and rationale actually makes the case given the evidence. One of the aims of self-adaptation is to eliminate or at least reduce the involvement of humans in the management of a software system. To accomplish this, self-

adaptation requires ways to computationally reason about assurance cases, and a logic to judge whether an assurance case is still valid, what changes must be made to it in terms of additional evidence, etc.

Adaptive assurances. As just alluded to, self-adaptation may require the assurance cases themselves to adapt. For example, replacing a new component in the system may require replacing evidence associated with that component in the assurance case. Changing goals of the system based on evolving business contexts will likely involve changes to the assurance cases for those goals. Automatable assurance cases are an initial step to addressing this challenge, but approaches, rules, and techniques for adapting the assurance cases themselves are also needed.

Assurance processes for self-adaptive software systems. One overarching challenge is the design of adequate assurance processes for self-adaptive systems. Such a process connects the system's goals, the architecture, and implementation realizing the goals to the assurance cases' argumentation structures, its strategies, evidence types, and assurance techniques. This challenge requires that parts of the design and assurance process that were previously performed off-line during development time must move to run-time and be carried out on-line in the system itself. The assurance goals of a system are dependent on a correct, efficient, and robust assurance process, which employs on-line and off-line activities to maintain continuous assurance support throughout the system lifecycle. Currently, such processes are not sufficiently investigated and understood.

Reassurance. If we are able to move the evaluation of assurance cases to runtime, the challenge arises in how to reassure the system when things change. Reassurance may need to happen when environment states, or the state of the system itself, change. Which part of the assurance case needs to be reevaluated? For composition, where the composition itself is dynamic, we need ways to identify the smallest set of claims (goals) that have to be reassured when two systems are composed? Which evidence needs to be re-established, and which can be reused?

4 Control Theory and Assurances

To realize perpetual assurances for adaptive systems requires effective run-time instrumentation to regulate the satisfaction of functional and non-functional requirements, in the presence of context changes and uncertainty (cf. Table 1). Control theory and feedback loops provide a number of powerful mechanisms for managing uncertainty in engineering adaptive systems [34]. Basically, feedback control allows us to manage uncertainty by monitoring the operation and environment of the system, comparing the observed variables against static or dynamic values to achieve (*i.e.*, system goals), and adjusting the system behavior to counteract disturbances that can affect the satisfaction of system requirements and goals. While continuous control theory suffices for purely physical systems, for cyber physical systems with significant software components a mix of discrete and continuous control is required. Moreover, adaptive systems require adaptive control where controllers must be modified at run-time. Many exciting and challenging research questions remain in applying control theory and concepts in the realm of self-adaptive systems.

The work presented in this paper is fundamentally based on the idea that, even for software systems that are too complex for direct application of classical control theory, the concepts and abstractions afforded by control theory can be useful. These concepts and abstractions provide design guidance to identify important control characteristics, as well as to determine not only the general steps but also the details of the strategy that determines the controllability of the resulting systems. This in turn enables careful reasoning about whether the control characteristics are in fact achieved in the resulting system.

Feedback loops have been adopted as cornerstones of software-intensive selfadaptive systems [9,27,30]. Building on this, this paper explores how classical feedback loops, as defined by control theory, can contribute to the design of self-adaptive systems, particularly to their assurances. The proposed approach focuses on the abstract characteristics of classical feedback loops —including their formulation and afforded assurances, as well as the analysis required to obtain those assurances. The approach concentrates on the conceptual rather than

the implementation level of the feedback-loop model. We investigated the relationships among desired properties that can be ensured by control theory applied in feedback loops (*e.g.*, stability, accuracy, settling time, or efficient resource use), the ways computational processes can be adjusted, the choice of the control strategy, and the quality attributes (*e.g.*, responsiveness, latency, or reliability) of the resulting system.

On the one hand, we discuss how feedback loops contribute to providing assurances about the behavior of the controlled system, and on the other hand, how the implementation of feedback loops in self-adaptive systems improves the realization of assurances in them. To set the stage for identifying concrete challenges, we first reviewed the major concepts of traditional control theory and engineering, then studied the parallels between control theory [3,34] and the more recent research on feedback control of software systems (*i.e.*, MAPE-K loops and hierarchical arrangements of such loops) [22,27] in the realm of selfadaptive systems. To gain a good understanding of the role that feedback loops play in the providing assurances for self-adaptive systems, the following books and seminal papers [2,3,9,17,27,41,43] are recommended.

In the next sections, we introduce basic concepts and properties that can be borrowed from control theory to provide assurances for self-adaptive systems. Then, we discuss research challenges and questions identified in analyzing classical feedback loops to guarantee desired properties for self-adaptive systems. For an extensive description of the ideas presented in this section, we refer the reader to [32].

4.1 Feedback Control

In a simple feedback control system, a process P (*i.e.*, the system to be adapted or managed in the self-adaptive systems realm) has a tuning parameter u (*e.g.*, a knob, which is represented by the little square box in Fig. 2) that can be manipulated by a controller C to change the behavior of the process, and a tracked metric y that can be sensed in some way. The system is expected to maintain the tracked metric y at a reference level y_r (*i.e.*, reference input), as illustrated in Fig. 2. Being a function that depends on time, C compares the value of y (subject to possible signal translation in the transducer) to the desired value y_r. The difference is the tracking or control error. If this error is significantly enough, the controller changes parameter u to drive the process in such a way as to reduce the tracking error. The tuning parameter u is often called the "control input" and the tracked metric y is often called the "measured output."

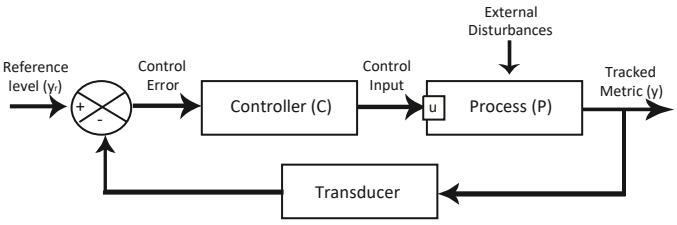

Fig.2. The feedback loop control model.

Control theory depends on the definition of reference control points to specify system behavior and corresponding explicit mathematical models. These models describe the Process (P), the Controller (C) and the overall feedback system. Control theory separates between the design of the process to be controlled and the design of the controller. Given a model of a process, a controller is designed to achieve a particular goal (*e.g.*, stability, robustness). Control theory also provides assurances when the process and the whole system are described by models.

Feedback, as implemented in controlled systems as described above, is especially useful when processes are subject to unpredictable disturbances. In computing environments, disturbance examples include, among others, system loads, such as those implied by the number of users or request arrival rates, and variable hit rates

on system caches. Feedback can also be useful when the computation in the process itself is unpredictable and its accuracy or performance can be tuned by adjusting its parameters.

4.2 Adaptive and Hierarchical Control

For systems that vary over time or face a wide range of external disturbances, it is impossible to design one controller that addresses all those changes. In these cases, there is the need to design a set of controllers (*i.e.*, parameterized controllers). When the current controller becomes ineffective, we switch to a new controller or adjust its parameters. When the controller is updated while the system runs, this control strategy is referred to as *adaptive control*. This control strategy requires additional logic that monitors the effectiveness of the controller under given conditions and, when some conditions are met, it re-tunes the controller to adapt it to the new situation. The control community has investigated adaptive control since 1961 according to ˚Astr¨om and Wittenmark [3]. They provide a working definition for adaptive control that has withstood the test of time: "An adaptive controller is a controller with adjustable parameters and a mechanism for adjusting the parameters." This definition implies *hierarchical control*: arrangements of two (or more) layers of control loops, usually three-layered architectures.

Control theory offers several approaches for realizing adaptive control. Two of them are *model reference adaptive control* (MRAC) and *model identification adaptive control* (MIAC) [17]. MRAC and MIAC feature an additional controller that modifies the underlying controller that affects the target system. This higher-level controller, also referred to as the "adaptation algorithm", is specified at design time for MRAC (*e.g.*, using simulation) and identified at run-time (*e.g.*, using estimation techniques) for MIAC. In practice, given the dynamic nature of the reference model in MIAC, this approach is used more often for highly uncertain scenarios. The strategies for changing the underlying controller range from changing parameters (*e.g.*, three parameter gains in a PID controller) to replacing the entire software control algorithm (*e.g.*, from a set of predefined components).

In the seventies, the AI and robotics communities introduced three-layer intelligent hierarchical control systems (HICS) [39]. The *Autonomic Computing Reference Architecture* (ACRA), proposed by Kephart and Chess [23,27] for the engineering of autonomic systems, is the most prominent reference architecture for hierarchical control. self-adaptive systems based on ACRA are defined as a set of hierarchically structured controllers. Using ACRA, software policies and assurances are injected from higher layers into lower layers. Other notable three-layer hierarchical control reference models include the Kramer and Magee model [29], DYNAMICO by Villegas *et al.* [44], and FORMS by Weyns *et al.* [49]. Please refer to Villegas *et al.* [42] for a more detailed discussion of these hierarchical control models for self-adaptive systems. The models at run-time (MART) community has developed extensive methods and techniques to evolve models dynamically for adaptive and hierarchical control purposes [5,15].

4.3 Control Theory Properties

Exploiting control theory to realize effective assurances in self-adaptive systems implies that special attention needs to be paid to the selection of a "control strategy" that contributes to guaranteeing desired *properties*. From this perspective, the control strategy is even more critical than the mechanisms used to adjust the target system. Furthermore, a property that is properly achieved effectively becomes an assurance for the desired system behavior. We submit that the lessons learned from control theory in the assurance of desired properties is a promising research direction. Here, we present an overview of control theory properties (Villegas *et al.* comprehensively define these properties in the context of self-adaptive systems [43]). These properties, even if

not captured in formalized models for self-adaptive systems, must be considered by their designers along the system's engineering lifecycle.

Broadly, control theory studies two types of control loops: open and closed loops. Open loop models focus only on the controlled or managed system, that is, the outputs of the controlled system (*i.e.*, measured outputs) are not considered to compute the control input. In contrast, in closed loop models control inputs are computed from measured outputs.

Properties of the Open Loop Model. In the open loop model, the most important properties are stability, observability and controllability.

Stability means that for bounded inputs (commands or perturbations), the system will produce bounded state and output values. Unfortunately, perturbations are the enemy of stability in open loop self-adaptive systems, because the system is not set up to recognize how much the perturbations affect the system. If the open self-adaptive systems is not stable, it can be stabilized through the design of a suitable controller. However, by analyzing the stability of the open system, we must understand the source of instability and design the controller appropriately.

Observability is the property of the model that allows to find, or at least estimate, the internal state variables of a system from the tracked output variables.

Controllability (or state controllability) describes the possibility of driving the open system to a desired state, that is, to bring its internal state variables to certain values [13]. While observability is not a necessary condition for designing a controller for self-adaptive systems, the controllability property is. Even if we do not have an explicit open loop model, a qualitative analysis should be performed.

Properties of the Closed Loop Model. When an explicit model of the open loop is available, the closed loop model can be synthesized mathematically to achieve the properties the designer aims to guarantee. In general, the controller is designed to achieve stability, robustness and performance.

Stability refers to whether control corrections move the open system state, over time, toward the reference value or level. A system is unstable if the controller causes overcorrections that never decrease, or that increase without limit, or that oscillates indefinitely. Instability can be introduced by making corrections that are too large in an attempt to achieve the reference level quickly. This leads to oscillating behaviors in which the system overshoots the reference value alternately to the high side and the low side.

Robust stability, or robustness, is a special type of stability in control theory. Robust stability means that the closed loop system is stable in the presence of external disturbances, model parameters and model structure uncertainties.

Performance is another important property in closed loop models and can be measured in terms of rise time, overshoot, settling time, steady error or accuracy [2,4].

4.4 Research Challenges

In this section, we have briefly described and analyzed how control theory properties can be used to guide the realization of assurances in the engineering of self-adaptive systems. However, for this realization we identify a number of challenges that require further research work.

Control theory challenges. We argue that the concepts and principles of control theory and the assurances they provide, at least in abstract form, can be applied in the design of a large class of self-adaptation problems in software systems. Part of these problems correspond to scenarios in which it is possible to apply control theory

directly to self-adaptive systems, that is, by mathematically modeling the software system behavior, and applying control theory techniques to obtain desired properties, such as stability, performance and robustness. These properties automatically provide corresponding assurances about the controlled system behavior. However, there are still no clear guidelines about the limitations of control theory as directly applied to self-adaptive systems in the general case.

Another set of problems corresponds to scenarios where it is infeasible to build a reasonably precise mathematical model, but instead, it is possible to create an approximated operational or even qualitative model of the selfadaptive system behavior. In this case, the formal definitions and techniques of control theory may not apply directly, but understanding the principles of control theory can guide the sorts of questions the designer should answer and take care of while designing an self-adaptive system.

Many challenges on the application of feedback control to perpetual assurances in self-adaptive systems arise from the following research questions:

- How can we determine whether a given self-adaptive system will be stable?
- How quickly will the system respond to a change in the reference value? Is this fast enough for the application? Are there lags or delays that will affect the response time? If so, are they intrinsic to the system or can they be optimized?
- What are the constraints for external disturbances and how do they affect self-adaptive systems design?
- Can we design a robust controller to achieve robust stability or use adaptive or hierarchical control?
- How accurately and periodically shall the system track the reference value? Is this good enough for the application domain?
- How much resources will the system spend in tracking and adjusting the reference value? Can this amount be optimized? What is more important: to minimize the cost of resources or the time for adjusting the reference values? Can this tradeoff be quantified?
- How likely is that multiple control inputs are needed to achieve robust stability?

Modeling challenges. These challenges concern the identification of the control core phenomena (*e.g.*, system identification or sampling periods). The analysis of the system model should determine whether the "knobs" have enough power (command authority) to actually drive the system in the required direction. Many open research questions remain, for example:

- How do we model explicitly and accurately the relationship among system goals, adaptation mechanisms, and the effects produced by controlled variables?
- Can we design software systems having an explicit specification of what we want to assure with control-based approaches? Can we do it by focusing only on some aspects for which feedback control is more effective?
- Can we improve the use of control, or achieve control-based design, by connecting as directly as possible some real physics inside the software systems?
- How far can we go by modeling self-adaptive systems mathematically? What are the limitations?
- How can we ensure an optimal sampling rate over time? What is the overhead introduced by oversampling the underlying system?
- Can we control the sampling rate depending on the current state of the self-adaptive system?
- How can we ensure an optimal sampling rate over time? What is the overhead introduced by oversampling the underlying system?
- Can we control the sampling rate depending on the current state of the self-adaptive system?

Run-time validation and verification (V&V) challenges. Run-time V&V tasks are crucial in scenarios where controllers based on mathematical models are infeasible. Nonetheless, performing V&V tasks (*e.g.*, using model checking) over the entire system—at run-time, to guarantee desired properties and goals, is often infeasible due to prohibitive computational costs. Therefore, other fundamental challenges for the assurance of

self-adaptive systems arise from the need of engineering incremental and composable V&V tasks [41]. Some open research questions on the realization of run-time V&V are:

- Which V&V tasks can guarantee which control properties, if any, and to what extent?
- Are stability, accuracy, settling-time, overshoot and other properties composable (*e.g.*, when combining control strategies which independently guarantee them)?
- What are suitable techniques to realize the composition of V&V tasks?
- Which approaches can be borrowed from testing? How can these be reused or adjusted for the assurance of self-adaptive systems?
- Regarding incrementality: in which cases is it useful? How can incrementality be realized? How increments are characterized, and their relationship to system changes?

Control strategies design challenges. As mentioned earlier, uncertainty is one of the most challenging problems in assuring self-adaptive systems. Thus, it is almost impossible to design controllers that work well for all possible values of references or disturbances. In this regard, models at run-time as well as adaptive and hierarchical (*e.g.*, three-layer hierarchies) control strategies are of paramount importance to self-adaptive systems design [30]. Relevant research questions include:

- How to identify external disturbances that affect the preservation of desired properties? What about external disturbances affecting third party services?
- How do we model the influence of disturbances on desired properties?
- How to deal with complex reference values? In the case of conflicting goals, can we detect such conflicting goals a priori or a posteriori?
- Can we identify the constraints linking several goals in order to capture a more complex composite goal (reference value)?
- Feedback control may also help in the specification of viability zones for self-adaptive systems. In viability zones desired properties are usually characterized in terms of several variables. How many viability zones are required for the assurance of a particular self-adaptive software system? Does each desired property require an independent viability zone? How to manage trade-offs and possible conflicts among several viability zones?
- How to maintain the causal connection between viability zones, adapted system, and its corresponding V&V software artifacts?

5 Summary and Conclusions

In this section, we present the overall summary of the identified key research challenges for the provision of assurances for self-adaptive systems. Though the theme of assurances is quite specific, the exercise was not intended to be exhaustive. Amongst the several topics involved by the challenges on the provision of assurances when engineering self-adaptive systems, we have focused on three major topics: perpetual assurances, composition and decomposition of assurances, and assurances inspired by control theory. We now summarize the most important research challenges for each topic.

- *Perpetual Assurances* — provision of perpetual assurances during the entire lifecycle of a self-adaptive system poses three key challenges: how to obtain a better understanding of the nature of uncertainty in software systems and how it should be equated, how to monitor and quantify uncertainty, and how to derive and integrate new evidence.
- *Composing and Decomposing Assurances* — although assurance cases can be used to collect and structure evidence, the key challenge is how to compose and decompose evidence in order to build arguments. There are two reasons for that: first, there is the need to manipulate different types of evidence and their respective

assumptions because of the uncertainty permeating selfadaptive systems; and second, whenever a system adapts, it is expected that its associated assurance cases adapt, preferably autonomously because of the reduced involvement of humans in managing a self-adaptive system. Another challenge is the need to provide overarching processes that would allow us to manage assurance cases, both during development time and run-time, since assurance cases are dynamic and should be updated whenever the system self-adapts.

- *Control Theory Assurances* — although synergies have been identified between control theory and self-adaptive systems, the challenge that remains is the definition of clear guidelines that would facilitate the direct application of control theory principles and practices into self-adaptive systems. As a result, adapted properties from control theory could be used as evidence for the provision of assurances. Since modelling is a key aspect in control systems, the challenge is to identify, in the context of self-adaptive systems, the variables to be monitored and controlled, suitable control strategies to model for each case, and how to implement these models directly in the adaptation mechanisms to fulfil the goals of the system. In order to deal with uncertainties, and inspired by practical control systems, there is the need to consider hierarchical structures of controllers, which should be supported by models that should be adaptable (*i.e.*, adaptive control). Due to the dynamic aspects of self-adaptive software systems, there is a need to perform the verification and validation tasks in an incremental and composable way, both at design and run-time.

There are several aspects that permeate the identified research challenges, but uncertainty is a key factor in the provision of assurances for self-adaptive systems. For example, there is uncertainty associated with the generation and composition of evidence that is used to build assurance arguments. In some contexts the only way to deal with uncertainty is to make assumptions —for example, assumptions on the number and type of changes, assumptions about the context in which the system operates, and assumptions associated with the produced evidence. The validity of the assumptions need to be perpetually evaluated while providing assurances. How to manage assumptions considering the limited involvement by humans during run-time of self-adaptive systems is a research challenge.

The autonomous way in which the provision of assurances ought to be managed is also considered a research challenge. For that, a separate feedback control loop might be needed to perpetually collect, structure and analyze the evidence. The role of this feedback control loop is not directly related to the services provided by the system, but to the management of the assurance arguments that justify the ability of the system to provide its intended service and its associated quality. Although a feedback control loop is an excellent mechanism to handle uncertainty, it should be considered under a set of assumptions, which also need to be evaluated during run-time.

Furthermore, considering the complexity of the task at hand, processes should be incorporated into the feedback control loop in order to manage the perpetual provision of assurances, which should depend, for example, on the trust level required by the system, the kind of evidence that the system is able to generate, and how this evidence can be composed in order to build assurance arguments. If there are any changes in the evidence or its assumptions, the controller should automatically evaluate the validity of the assurance arguments. Equally, if trust levels associated with the system goals change, the controller should evaluate the arguments of the assurance case, and if required, new evidence ought to be generated and composed in order to rebuild assurance arguments.

The identified research challenges are specifically associated with the three topics related to the provision of assurances when engineering self-adaptive systems, which were addressed in this paper. These are challenges that our community must face because of the dynamic nature of self-adaptation. Moreover, the ever changing nature of these type of systems requires to bring uncertainty to the forefront of system design. It is this uncertainty that challenges the applicability of traditional software engineering principles and practices, but motivates the search for new approaches when developing, deploying, operating, evolving and decommissioning self-adaptive systems.

References

1. Ali, R., Griggio, A., Franz´en, A., Dalpiaz, F., Giorgini, P.: Optimizing monitoring requirements in self-adaptive systems. In: Enterprise, Business-Process and Information Systems Modeling, pp. 362–377. Springer (2012)
2. ˚Astro¨m, K.J., Murray, R.M.: Feedback Systems. An introduction for scientists and engineers (2008)
3. ˚Astro¨m, K., Wittenmark, B.: Adaptive Control. Addison-Wesley series in Electrical Engineering: Control Engineering, Addison-Wesley (1995)
4. Balzer, B., Litoiu, M., Mu¨ller, H., Smith, D., Storey, M.A., Tilley, S., Wong, K.: 4th International Workshop on Adoption-Centric Software Engineering. In: Proceedings of the 26th International Conference on Software Engineering. pp. 748–749. ICSE 2004, IEEE Computer Society, Washington, DC, USA (2004)
(2009)
5. Blanchette Jr., S.: Assurance cases for design analysis of complex system of systems software. Tech. rep., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (Apr 2009)
6. Bloomfield, R., Bishop, P.: Safety and assurance cases: Past, present and possible future–an adelard perspective. In: Making Systems Safer, pp. 51–67. Springer London (2010)
7. Bloomfield, R., Peter, B., Jones, C., Froome, P.: ASCAD — Adelard Safety Case Development Manual. Adelard, 3 Coborn Road, London E3 2DA, UK (1998)
8. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Mu¨ller, H., Pezz`e, M., Shaw, M.: Engineering Self-adaptive Systems through Feedback Loops. In: Software Engineering for Self-adaptive Systems, pp. 48–70. Springer (2009)
9. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic qos management and optimization in service-based systems. IEEE Trans. Softw. Eng. 37(3), 387–409 (May 2011)
10. Casanova, P., Garlan, D., Schmerl, B., Abreu, R.: Diagnosing architectural runtime failures. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 103–112. SEAMS '13 (2013)
11. Casanova, P., Garlan, D., Schmerl, B., Abreu, R.: Diagnosing unobserved components in self-adaptive systems. In: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 75–84. SEAMS 2014 (2014)
12. Checiu, L., Solomon, B., Ionescu, D., Litoiu, M., Iszlai, G.: Observability and Controllability of Autonomic Computing Systems for Composed Web Services. In: Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI 2011). pp. 269–274. IEEE (2011)
13. 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., Mu¨ller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software Engineering for SelfAdaptive Systems: A Research Roadmap, pp. 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
14. Cheng, B., Eder, K., Gogolla, M., Grunske, L., Litoiu, M., Mu¨ller, H.A., Pelliccione, P., Perini, A., Qureshi, N., Rumpe, B., Schneider, D., Trollmann, F., Villegas, N.M.: Using models at runtime to address assurance for self-adaptive systems. In: Bencomo, N., France, R., Cheng, B., Amann, U. (eds.) Models@run.time, Lecture Notes in Computer Science, vol. 8378, pp. 101–136. Springer International Publishing (2014)
15. Cheng, S.W., Garlan, D., Schmerl, B., Sousa, J.a.P., Spitznagel, B., Steenkiste,
P.: Using architectural style as a basis for self-repair. In: Bosch, J., Gentleman, M., Hofmeister, C., Kuusela, J. (eds.) Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture. pp. 45–59. Kluwer Academic Publishers (2531 August 2002)
16. Dumont, G., Huzmezan, M.: Concepts, Methods and Techniques in Adaptive Control. In: IEEE American Control Conf. (ACC). vol. 2, pp. 1137–1150 (2002)
17. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: de Lemos, R., Giese, H., Mu¨ller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers, pp. 214–238. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
18. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: 33rd International Conference on Software Engineering (ICSE). pp. 341–350 (May 2011)

19. Filieri, A., Tamburrelli, G.: Probabilistic verification at runtime for self-adaptive systems. In: Ca´mara, J., de Lemos, R., Ghezzi, C., Lopes, A. (eds.) Assurances for Self-Adaptive Systems, Lecture Notes in Computer Science, vol. 7740, pp. 30–59.
Springer Berlin Heidelberg (2013)
20. Garlan, D.: Software engineering in an uncertain world. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. pp. 125–128.
FoSER '10 (2010)
21. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
22. IBM Corporation: An Architectural Blueprint for Autonomic Computing. Tech. rep., IBM Corporation (2006)
23. Iftikhar, M.U., Weyns, D.: Activforms: Active formal models for self-adaptation.
In: Proceedings of the 9th International Symposium on Software Engineering for
Adaptive and Self-Managing Systems. pp. 125–134. SEAMS 2014 (2014)
24. Gil de la Iglesia, D., Weyns, D.: Guaranteeing robustness in a mobile learning application using formally verified MAPE loops. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 83–92. SEAMS '13 (2013)
25. Kelly, P.: Managing complex safety cases. In: 11th Safety Critical System Symposium (SSS'03. pp. 99–115. Springer-Verlag (2003)
26. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer 36(1), 41–50 (2003)
27. Kit, M., Gerostathopoulos, I., Bures, T., Hnetynka, P., Plasil, F.: An architecture framework for experimentations with self-adaptive cyber-physical systems. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 93–96. SEAMS '15 (2015)
28. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. In: FOSE 2007: 2007 Future of Software Engineering. pp. 259–268. IEEE Computer
Society, Washington, DC, USA (2007)
29. de Lemos, R., Giese, H., Mu¨ller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., G¨oschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer,
J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos,
J., Nierstrasz, O., Pezz`e, M., Prehofer, C., Scha¨fer, W., Schlichting, R., Smith, D.B., Sousa, J.P., Tahvildari, L., Wong, K., Wuttke, J.: Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, pp. 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
30. Litoiu, M.: A performance analysis method for autonomic computing systems. ACM Trans. Auton. Adapt. Syst. 2(1) (Mar 2007)
31. Litoiu, M., Shaw, M., Tamura, G., Villegas, N.M., Mu¨ller, H.A., Giese, H., Rouvoy, R., Rutten, E.: What can control theory teach us about assurances in self-adaptive software systems? In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III. Springer Berlin Heidelberg, Berlin,
Heidelberg (2017)
32. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In: Mistrik, I., Ali, N., Kazman, R., Grundy, J., Schmerl, B. (eds.) Managing Trade-Offs in Adaptable Software Architectures, pp. 45 – 77. Morgan Kaufmann, Boston (2017)
33. Murray, R.M.: Control in an Information Rich World: Report of the Panel on Future Directions in Control, Dynamics, and Systems. SIAM (2003)
34. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. pp. 3–14. ICPE '14 (2014)
35. Redondo, R.P.D., Arias, J.J.P., Vilas, A.F.: Reusing verification information of incomplete specifications. In: Component-based Software Engineering Workshop. Lund, Sweden (2002)
36. Schmerl, B., Andersson, J., Vogel, T., Cohen, M.B., Rubira, C.M.F., Brun, Y., Gorla, A., Zambonelli, F., Baresi, L.: Challenges in composing and decomposing assurances for self-adaptive systems. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)

37. Schmerl, B., Ca´mara, J., Gennari, J., Garlan, D., Casanova, P., Moreno, G.A., Glazier, T.J., Barnes, J.M.: Architecture-based self-protection: Composing and reasoning about denial-of-service mitigations. In: HotSoS 2014: 2014 Symposium and Bootcamp on the Science of Security. Raleigh, NC, USA (8-9 April 2014)

38. Shibata, T., Fukuda, T.: Hierarchical intelligent control for robotic motion. IEEE Transactions on Neural Networks 5(5), 823–832 (Sep 1994)

39. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers, pp. 193–207. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

40. Tamura, G., Villegas, N.M., Mu¨ller, H.A., Sousa, J.P., Becker, B., Pezz`e, M., Karsai, G., Mankovskii, S., Scha¨fer, W., Tahvildari, L., Wong, K.: Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 108–132. Springer (2013)

41. Villegas, N., Tamura, G., Mu¨ller, H.: Chapter 2 - architecting software systems for runtime self-adaptation: Concepts, models, and challenges. In: Mistrik, I., Ali, N., Kazman, R., Grundy, J., Schmerl, B. (eds.) Managing Trade-Offs in Adaptable Software Architectures, pp. 17 – 43. Morgan Kaufmann, Boston (2017)

42. Villegas, N., Mu¨ller, H., Tamura, G., Duchien, L., Casallas, R.: A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and SelfManaging Systems (SEAMS 2011). pp. 80–89. ACM (2011)

43. Villegas, N.M., Tamura, G., Mu¨ller, H.A., Duchien, L., Casallas, R.: DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 265–293. Springer (2013)

44. Vogel, T., Giese, H.: Model-driven engineering of self-adaptive software with eurema. ACM Trans. Auton. Adapt. Syst. 8(4), 18:1–18:33 (Jan 2014)

45. Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.M., Malek, S., Mirandola, R., Mori, M., Tamburrelli, G.: Perpetual assurances in self-adaptive systems. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)

46. Weyns, D., Calinescu, R.: Tele assistance: A self-adaptive service-based system examplar. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 88–92. SEAMS ’15, IEEE Press, Piscataway, NJ, USA (2015)

47. Weyns, D., Iftikhar, M.U., 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. pp. 67–79. C3S2E ’12 (2012)

48. Weyns, D., Malek, S., Andersson, J.: Forms: Unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans. Auton. Adapt. Syst. 7(1), 8:1–8:61 (May 2012)

49. Ye, F., Kelly, T.: Contract-based justification for cots component within safety critical applications. In: Cant, T. (ed.) Ninth Australian Workshop on SafetyRelated Programmable Systems (SCS 2004). CRPIT, vol. 47, pp. 13–22. ACS, Brisbane, Australia (2004)

50. Zhang, J., Cheng, B.H.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software 79(10), 1361 – 1369 (2006)