Report from Dagstuhl Seminar 13511

Software Engineering for Self-Adaptive Systems: Assurances

Edited by

Rogerio de Lemos[1], David Garlan[2], Carlo Ghezzi[3], and Holger Giese[4]

1 **University of Kent, GB,** R.Delemos@kent.ac.uk
2 **Carnegie Mellon University – Pittsburgh, US,** garlan@cs.cmu.edu
3 **Politecnico di Milano, IT,** carlo.ghezzi@polimi.it
4 **Hasso-Plattner-Institut – Potsdam, DE,** holger.giese@hpi.uni-potsdam.de

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 assurances, i.e., the provision 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 self-adaptive systems is challenging since run-time changes introduce a high degree of uncertainty during their operation. In this seminar, we discussed the problem of assurances for self-adaptive systems from four different views: criteria for assurances, composition and decomposition of assurances, feedback loop and assurances, and perpetual provisioning of assurances.

Seminar 15.–19. December, 2013 – www.dagstuhl.de/13511
1998 ACM Subject Classification D.2.10 [Software Engineering] Design, D.2.11 [Software Engineering] Software Architectures
Keywords and phrases software engineering, self-adaptive systems, assurances, criteria, feedback loop, decentralization
Digital Object Identifier 10.4230/DagRep.3.12.67

1 Executive Summary

Rogerio de Lemos
David Garlan
Carlo Ghezzi
Holger Giese

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, etc.).

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 assurances: that is, providing evidence that systems satisfy their functional and non-functional requirements during operation. Specifically, assurances involve not only system dependability, but also resilience with respect to changes that may occur in the system, its environment, or its goals. The provision of assurances for self-adaptive systems, which should be done tandem with their development, operation and evolution, is difficult since run-time changes (e.g., resource variability) introduce a high degree of uncertainty that is atypical in more conventional systems.

This Dagstuhl Seminar has focused on the topic of obtaining assurances for self-adaptive software systems. Self-adaptive systems has been studied independently within different research areas of software engineering, including requirements engineering, modelling, architecture and middleware, event-based, component-based and knowledge-based systems, testing, verification and validation, as well as software maintenance and evolution [1, 2]. On the other hand, the topic of assurances for 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. The major challenge when combining self-adaptability and dependability is how to obtain assurances regarding the uncertainty of changes that may affect the system, its environment or its goals.

References

1 B. H. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos. 08031 Abstracts Collection – Software Engineering for Self-Adaptive Systems. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, number 08031 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany.

2 R. de Lemos, H. Giese, H. Müller, and M. Shaw. 10431 Report – Software Engineering for Self-Adaptive Systems. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems*, number 10431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany.

2 Table of Contents

3 Key Topics on Assurances

The aim of the seminar was not so much to be comprehensive concerning the topics associated with assurances for self-adaptive software systems, but to be focused on key and challenging topics.

3.1 Composition and Decomposition of Assurances

Contributors: *J. Andersson, B. Schmerl, L. Baresi, Y. Brun, M. B. Cohen, A. Gorla, C. M. F. Rubira, T. Vogel, F. Zambonelli*

In software development, system goals are often provided assurances. An *assurance* is "a reasoned and compelling argument, supported by a body of evidence, that a system, service or organisation will operate as intended for a defined application in a defined environment" [1]. Examples of goals include enforcing certain system properties, such as safety, security, and reliability. The most widely accepted approaches for assuring suchgoals are based on producing evidence, and then arguing that a system meets its specification based on that evidence. An *assurance case* [2] is composed of subgoals, strategies, contexts, and evidence, tied together into an argument justifying that the goal will be met. For example, safety cases are *"a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment"* [3, p. 8].

We are interested in assurances for engineering a particular kind of software system: self-adaptive systems. In such systems, self-adaptation may be used as a mechanism for achieving a particular goal, or may be a mechanism in spite of which a goal must be satisfied. We are particularly interested in the process of building self-adaptive systems out of self-adaptive components (and further, recursively building larger self-adaptive systems from self-adaptive subsystems). Thus, we aim to understand (1) how assurances of self-adaptive components compose to form an assurance of a system composed of those components, and (2) how an assurance of a system can be decomposed into assurances its components must satisfy. While the challenges of composition and decomposition are neither unique to *assurances* within self-adaptive systems engineering [4, 5], nor to *self-adaptive* systems, composing and decomposing assurances within such systems poses special challenges, on which we focus here.

1. **How to accommodate a changing environment?** In traditional software development, assurance cases often make assumptions about fixed goals and environments. By contrast, self-adaptive systems often relax these assumptions. This creates a challenge for composition and decomposition of assurances cases of self-adaptive components and systems. For example, for composition, it is often insufficient to consider the assurance case solely within the scope of the component, and must instead be considered within the larger scope of the system. Unfortunately, this breaks the component abstraction and complicates reasoning about assurances.
2. **What are potential sources of evidence?** As some self-adaptive components can be made up of a separate managed part and managing part [6], understanding how assurances on one part affect or can result in assurances on the whole poses a challenge. At the very least, these pose multiple sources of evidence, the combination of which must be understood.

   Furthermore, what kinds of evidence can be provided by the different activities of self-adaptation. For example, what kinds of evidence can monitoring provide? Is this type of evidence different to the type that analysis can provide? What kinds of assurances can emerge from more biological forms of self-adaptation? Evidence may also come from other engineering activities, e.g., design decisions, validation and verification, or the engineering process. These all impact how assurances can be composed and decomposed. For example, design decisions such as the choice of architectural style may provide some evidence for assurance cases.

3. **What strategies can be used to combine assurances when self-adaptive systems are composed?** To address this challenge, we must answer further questions such as "How to design assurances so that they can be composed with one another?", "Which properties of assurances lend themselves to composition, and which do not?", "How can assurance composition strategies be reused?", "How can evidence from assurance cases be shared and reused among products or components?", and "How do composition and decomposition of assurance cases affect the amounts of evidence needed to support those cases?"

To begin to address these challenges, we plan to use a well-known method for constructing assurance cases known as the Goal Structuring Notation (GSN) [1, 7]. This notation allows engineers to codify an assurance case as a set of *strategies*, *contexts*, *assumptions*, *justifications*, and *solutions*. This structure serves as the argument for the assurance of a specific goal. It is likely that this notation will need to be adjusted to apply to address the first two challenges.

To address the third challenge, Bate and Kelly [8] outline a strategy for composing assurance arguments. However, this technique imposes constraints on the underlying system architecture. No comprehensive techniques exist today for composing assurances in a more general system-of-systems scenario, but this work may form a basis to build upon.

References

1 Goal Structuring Notation (GSN) community standard version 1, November 2011. Available at http://goalstructingnotation.info.

2 Peter Bishop, Robin Bloomfield, and Sofia Guerra. The future of goal-based assurance cases. In *Proc. Workshop on Assurance Cases*, pages 390–395, 2004.

3 Robin Bloomfield, Bishop Peter, Claire Jones, and P. Froome. *ASCAD — Adelard Safety Case Development Manual.* Adelard, 3 Coborn Road, London E3 2DA, UK, 1998.

4 Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science (LNCS)*, pages 48–70. Springer-Verlag, 2009. doi:10.1007/978-3-642-02161-9_3.

5 Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. On patterns for decentralized control in self-adaptive systems. In Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science (LNCS)*, pages 76–107. Springer-Verlag, 2013. doi:10.1007/978-3-642-35813-5_4.

6 Danny Weyns, Sam Malek, and Jesper Andersson. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):8:1–8:61, May 2012. doi:10.1145/2168260.2168268.

7 T. P. Kelly and J. A. McDermid. Safety case construction and reuse using patterns. In Peter Daniel, editor, *Safe Comp*, pages 55–69. Springer, London, 1997. URL: http://dx.doi.org/10.1007/978-1-4471-0997-6_5, doi:10.1007/978-1-4471-0997-6_5.

8 Iain Bate and Tim Kelly. Architectural considerations in the certification of modular systems. *Reliability Engineering & System Safety*, 81(3):303 – 324, 2003. doi:10.1016/S0951-8320(03)00094-2.

3.2 Feedback Loop and Assurances

Contributors: *H. A. Müller, N. M. Villegas, V. Cortellessa, A. Filieri, H. Giese, G. Karsai, A. Leva, M. Pezzè, R. Rouvoy, E. Rutten, M. Shaw, G. Tamura*

Feedback loops are cornerstones of software-intensive self-adaptive systems (SASs). In this chapter, we study the relationships existing between feedback loops and the types of assurance SASs can provide focusing on the conceptual rather implementation level of the feedback model. The review includes, on the one hand, how feedback loops contribute to providing assurances about the controlled system and, on the other hand, how assurances improve the realisation of feedback loops in SASs. To set the stage for the discussion of concrete examples and their challenges, we first study the parallels between traditional engineering control theory [1, 2] and the more recent research on feedback control of software computing systems (e.g., MAPE-K loops) [3, 4] in the realm of SASs in order to establish a common vocabulary of key concepts, such as the disturbance affecting the system or the control actions used by the controller to adapt the system. This provides a basis to discuss concrete examples that allow us to illustrate open challenges in the engineering of SASs [5]. For each of these examples, we identify the main concepts related to the shared vocabulary and highlight the need and purpose of feedback loops. Furthermore, we outline the related assurances, including concrete assurance goals, techniques employed for assurances, and how goals and techniques relate to the feedback loops present in SASs. By studying concrete examples, we posit key challenges for assurance research related to feedback loops in self-adaptive software.

1. **Identification of the core phenomena to control**: in control theory, system transfer functions based on differential equations are the models on which principles and properties of the phenomena to control are described and analysed [1, 6, 2]. Depending on the behavioural characteristics of the target system, a controller can be defined to make the system behave as desired. The desired behaviour is usually specified by either providing a reference input the system should follow (e.g., PID controllers), or as an optimisation problem (e.g., Model Predictive Control) [1]. The characteristics of a controller are usually defined by adjusting its parameters, which have special significance to generate the signals that will adapt the system depending on how far measured outputs are from the corresponding reference inputs. The controllers are indeed designed as (parametric) transfer functions that generate the control signals that drive the target system towards accomplishing its goals. In software systems the identification of the core phenomena to control is typically a complex task. This is mainly because, in contrast to physical systems, software systems still lack general methods to model the multi-dimensional and non-linear relationships between system goals and adaptation mechanisms [6, 7, 8]. For example, an adaptation mechanism can reconfigure the system by applying an architectural pattern with the goal of improving the system performance. However, it is still an open challenge to model the exact effect of this pattern in the performance of the system. Many research questions remain open in the identification and modelling of the core phenomena to control in software systems. For example, how to model explicitly and accurately the relationship among system goals, adaptation mechanisms, and the effects produced by

actuators? 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 to some "real physics" inside the software system? What techniques can we use for this? How far can we go by modelling SAS systems mathematically? What are the limitations?

2. **Composition and incrementality**: performing validation and verification (V&V) tasks over the entire system—at runtime, to guarantee desired properties and goals, is often infeasible due to prohibitive computational costs. Therefore, the assurance of SAS systems requires composable V&V tasks that can be applied incrementally along the adaptation loop. With respect to composition, relevant research questions include: what are the problems that require the composition of assurance mechanisms? What are suitable techniques to realise the composition of V&V tasks? What approaches can we borrow from testing? How can we reuse or adjust them for the assurance of SAS systems? Regarding incrementality: in which cases is it useful? How can incrementality be realised? How to characterise increments, and their relationship to system changes?
3. **Synergy between control and software engineering**: despite the efforts to apply control theory to the engineering of SASs, evidenced for instance in [5, 9, 10, 6, 3, 11, 12], the assurance of SASs still demands effective ways of integrating control theory and software engineering. Research questions to advance towards this direction include: what are the best practices from both sides that can be exploited in the assurance of SAS systems? How to construct reliable controllers leveraging formal model techniques used in software engineering? Can we define and/or use anti-patterns in control-based assurance of SAS systems? How to apply off-line V&V mechanisms at runtime? How to close the semantic gap between control in physical systems and control in software systems? Can we define a model in the middle of these two worlds? Would these models be domain or problem specific? Can we characterise (at least some of) these problems and corresponding suitable models? How can we educate software engineers in the application of control to the design of software systems?
4. **Management of viability zones**: the viability zone of a SAS system can be defined as the set of possible states in which the system operation is not compromised [13]. Moreover, it can be characterised in terms of relevant context attributes and corresponding desired values [14], and can change with context changes. In effect, the viability zone of a target system under adaptation constantly varies with and along adaptation dimensions. The dynamic nature of viability zones is a relevant research challenge in the assurance of SAS systems because it implies adjusting the domain coverage not only of design and adaptation realisation but also of V&V tasks. Open research questions in this aspect include: what are runtime models that can be used for the incremental and dynamic derivation of software artefacts for implementing V&V tasks? How to maintain the causal connection between viability zones, adapted system, and its corresponding V&V software artefacts? How to adapt these artefacts at runtime?

References

1 Karl Johan Astrom and Bjorn Wittenmark. *Adaptive Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

2 Richard M. Murray, editor. *Control in an Information Rich World*. Society for Industrial and Applied Mathematics, 2003. URL: http://epubs.siam.org/doi/abs/10.1137/1.9780898718010, arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9780898718010, doi:10.1137/1.9780898718010.

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

4 Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003. URL: http://dx.doi.org/10.1109/MC.2003.1160055, `doi:10.1109/MC.2003.1160055`.

5 Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science (LNCS)*, pages 48–70. Springer-Verlag, 2009. `doi:10.1007/978-3-642-02161-9_3`.

6 Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th IEEE/ACM International Conference on Software Engineering*, ICSE 2014, page (to appear), New York, NY, USA, 2014. ACM.

7 T. Patikirikorala, A. Colman, J. Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 33–42, June 2012. `doi:10.1109/SEAMS.2012.6224389`.

8 Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 80–89, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/1988008.1988020, `doi:10.1145/1988008.1988020`.

9 Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 283–292, Washington, DC, USA, 2011. IEEE Computer Society. URL: http://dx.doi.org/10.1109/ASE.2011.6100064, `doi:10.1109/ASE.2011.6100064`.

10 A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Reliability-driven dynamic binding via feedback control. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Symposium on*, pages 43–52, June 2012. `doi:10.1109/SEAMS.2012.6224390`.

11 Hausi Müller, Mauro Pezzè, and Mary Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2Nd International Workshop on Ultra-large-scale Software-intensive Systems*, ULSSIS '08, pages 23–26, New York, NY, USA, 2008. ACM. URL: http://doi.acm.org/10.1145/1370700.1370707, `doi:10.1145/1370700.1370707`.

12 Gabriel Tamura, Norha M. Villegas, HausiA. Muller, JoãoPedro Sousa, Basil Becker, Gabor Karsai, Serge Mankovskii, Mauro Pezzè, Wilhelm Schäfer, Ladan Tahvildari, and Kenny Wong. Towards practical runtime verification and validation of self-adaptive software systems. In Rogerio Lemos, Holger Giese, Hausi. Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 108–132. Springer Berlin Heidelberg, 2013. URL: http://dx.doi.org/10.1007/978-3-642-35813-5_5, `doi:10.1007/978-3-642-35813-5_5`.

13 Jean-Pierre Aubin, Alexandre Bayen, and Patrick Saint-Pierre. *Viability Theory: New Directions*. Springer, 2011. URL: http://hal.inria.fr/inria-00636570, `doi:10.1007/978-3-642-16684-6`.

14 Norha M. Villegas. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD thesis, University of Victoria, Canada, February 2013.

3.3 Perpetual Provisioning of Assurances

Contributors: *G. Tamburrelli, D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J. M. Jezequiel, S. Malek, R. Mirandola, M. Mori*

The breakout group focused first of finding a clear and unanimous definition of "assurances for self-adaptive systems" and then focused on (1) studying solutions for assurances of self-adaptive systems together with benchmark criteria, and (2) defining a unifying case study to be used by practitioners and academics to assess, challenge and compare their solutions. After a debate we came up with the following definition: "Assurances for self-adaptive systems means providing evidences for requirements compliance (functional and non-functional) on and off-line". With perpetual assurances, the group refers to the perpetual process of evidence provision for assurance that blends system and human activities throughout the live cycle of a self-adaptive system. Given this premise, the group concentrated on two aspects discussed in two subgroups.

One subgroup discussed key challenges of perpetual assurances for self-adaptive systems, requirements for solutions, realisation techniques, mechanism to make solutions suitable, and benchmark criteria to compare solutions. Identified challenges include: uncertainty, incompleteness, heterogeneity of systems, dynamism (changes in requirements, environment ...), scale of systems, distribution, and user/resource constraints. The following requirements of the solution were discussed: dealing with uncertainty and incompleteness, dealing with variants, timeliness, overhead, and scalability (efficiency), traceability of adaptation decisions (convey the rationale and evidences), user-in-the-loop, and reactive vs. proactive(speculative) vs. predictive provision of assurances. Discussed techniques to provide assurances are model checking, testing, simulation and statistical analysis, proving, runtime verification, and sanity checks. We identified possible mechanisms for making the techniques suitable: incrementally, compositionality, hierarchical reasoning (fallback mechanisms), parallelism/off-loading, parameterisation, abstraction, learning, and caching. Key benchmark criteria identified were: (1) does the approach provides evidence with respect to (can your approach handle); (2) does your approach provide evidence based on the current state, the past, projection in the future (or any combination of these; (3) how efficient (effective) is your approach in providing evidence, e.g., wrt. timeliness given a set of system reconfigurations, and overhead (memory, CPU).

The other subgroup discussed the definition of the case study. We decided do focus on the specific domain of Service Oriented Architectures for several reasons: popularity, flexibility, and incremental complexity. First the subgroup identified three distinct areas of assurances: assurances w.r.t. the services, the users, and the requirements. As a consequence of this initial assessment, the definition of the case study went through several refinement steps that included the definition of a series of challenges, techniques, and mechanisms that the case study implementers may exploit. This process culminated with the definition of an initial list of scenarios and a list of categories that detailed the case study. This refinement process and the scenario characterisation is still an on-going process and is part of the future work of the group. The final goal of the subgroup is to have a repository of one or more concrete case study that may be used as concrete benchmarks for the community.

The joint group discussed the results of subgroup discussions and decided to elaborate further on solutions for perpetual assurances (challenges, requirements, techniques and mechanisms, and benchmark criteria) and case studies that allow benchmarking. To that end, the group plans to write a joint book chapter that integrates both aspects.

4 Outcomes

The two concrete outcomes from this seminar will be a challenges paper and a new book. The challenges paper, which will follow the same format from previous roadmap papers will be structured according to the topics to be identified during the seminar, namely: criteria for assurances, composition and decomposition of assurances, feedback loop and assurances, and perpetual provisioning of assurances. For each topic, the objective is to summarise the current state-of-the-art, discuss its limitations, and identify future challenges for the field.

The book will contain state-of-the-art contributions from participants of the seminar and some invited contributions. In addition to these contributions, the roadmap paper will be the introductory chapter of the book, which should be followed by four chapters containing extended versions of the topics discussed in the roadmap paper. The book will be published by Springer as Lecture Notes in Computer Science volume on their State-of-the-Art series.

5 Overview of Talks

5.1 Self-Adaptive Authorisation Infrastructures: Managing malicious behaviour

Chris Bailey (University of Kent, GB)

Authorization infrastructures are an integral part of any network where electronic resources require protection. As networks expand and organizations begin to federate access to their resources, authorization infrastructures become increasingly challenging to manage. This talk outlines recent works in regards to self-adaptive security, specifically self-adaptive authorization. This explores the automatic adaptation of authorization assets (such as access control policies and subject access rights) for the handling of malicious user behavior. We identify the core motivation for our work, our current progress, and a short adaptation scenario that demonstrates key aspects and challenges of self-adaptive authorization. Finally, we identify our evaluation approach and discuss how our work applies to self-adaptive assurances, in regards to verification and validation of adaptations.

5.2 A Fine-grained Autonomic Management Solution for Multi-layered Systems

Luciano Baresi (Technical University of Milan, IT)

The service paradigm, together with virtualization technology, is imposing a profound re-thinking of many complex software systems. Providing virtual infrastructures as services is gaining more and more momentum, and is imposing a more cohesive view of the different layers that constitute a software system. Applications, service platforms, and virtualized infrastructures have become tightly integrated. It is now possible to change a software's quality of service (QoS) by changing the configuration of the virtual machines it uses. We can even instantiate new virtual machines to address load problems, or move our software from one infrastructure to another if its quality is not acceptable. Even if one might say that this is nothing new, the key distinctive feature is the ease with which the different parameters can be changed, and the different runtime executors (e.g., virtual machines) added or modified to impact a system's behavior. Originally, monitoring of service-oriented systems was only performed at the application layer, and the lower layers were considered to be a constant. Recently, however, cross-layer monitoring is imposing itself as a promising and challenging research problem. Available technologies provide users with means to tackle the problem of the quality of service of these applications by digging down into the different layers. However, the more complexity we want to tame, the more sophisticated our solutions must become. If we take advantage of the ease and low impact of installing software probes within the different layers, the amount of data we collect can grow very rapidly. This calls for efficient and precise methods for their management. We need a customizable and extensible way to collect, aggregate, and analyze data, in order to identify the causes of anomalous behaviors. This talk introduces a new approach for the cross-layer monitoring of complex service-based systems. The approach proposes two main novel contributions. We present mlCCL, a novel and extensible declarative language that designers can use to define (i) the various data they want to collect from the layers in their system, (ii) how to aggregate them to build higher-level knowledge, and (iii) how to analyze them to discover undesired behaviors. The talk also presents ECoWare (Event Correlation Middleware), an event correlation and aggregation middleware that supports mlCCL specifications. It provides advanced data aggregation and analysis features, and can be used to probe systems. Ecoware also provides a dashboard for reasoning on multiple dimensions of a running system at the same time, and for performing drill-down analyses to discover the causes of a revealed anomaly.

5.3 Artificial software diversity: automatic synthesis of program sosies

Benoit Baudry (INRIA Rennes – Bretagne Atlantique, FR)

Recent work have exploited the plastic properties of software to develop unsound program transformations. These transformations modify the program's behavior while staying in acceptable correctness boundaries with the aim of improving some qualitative attribute (e.g., response-time, fault-tolerance, etc.). This work reports on a novel form of unsound transformation, which consists in generating program 'sosies'. Sosies of a program P are variants that exhibit exactly the same observable behavior, through different execution paths ('sosie' is the French word for look-alike). We use the test suite of the program as the specification of acceptable correctness and we experiment transformations that replace code in the program by other code that comes from the same program. We define transformations at different levels of granularity (expression, statement, block). Here we report on the feasibility of automatic synthesis of sosies, at different granularities, on a set of open source Java programs of very different sizes. We show that transformations, which consider a part of the program's semantics are more effective for sosie synthesis than pure random transformations.

5.4 Bayesian Artificial Intelligence for Tackling Uncertainty in Self-Adaptive Systems: The Case of Dynamic Decision Networks

Nelly Bencomo (Aston University – Birmingham, GB)

In recent years, there has been a growing interest towards the application of artificial intelligence approaches in software engineering (SE) processes. In the specific area of SE for self-adaptive systems (SASs) there is a growing research awareness about the synergy between SE and AI. However, just few significant results have been published. We report and discuss our own experience using Dynamic Decision Networks (DDNs) to model and support decision-making in SASs while explicitly taking into account uncertainty. In this session we talk about the application of our DDN-based approach to the case of an adaptive remote data mirroring system. We discuss results, implications and potential benefits of the DDN to enhance the development and operation of self-adaptive systems, by providing mechanisms to cope with uncertainty and automatically make the best decision. We also discuss the ongoing work on a Bayesian definition of surprise as the basis for quantitative analysis to measure degrees of uncertainty and deviations of self- adaptive systems from normal behavior. A surprise measures how observed data affects the models or assumptions of the world during runtime. The key idea is that a surprising event can be defined as one that causes a large divergence between the belief distributions prior to and posterior to the event occurring. In such a case the system may decide either to adapt accordingly or to flag that an abnormal situation is happening.

5.5 Inferring models for verification

Yuriy Brun (University of Massachusetts – Amherst, US)

Model inference – constructing a model of an implementation based on execution information – can help greatly with ensuring that requirements are satisfied, enable exploration and evaluation of potential adaptations, and predicting the effects of adaptations.

5.6 Self-Adaptive Software Assurance through Continual Verification of Non-Functional Properties

Radu Calinescu (University of York, GB)

Software systems are used in business-critical and safety-critical applications from domains ranging from e-commerce and e-government to finance and healthcare. Many of these systems must comply with strict non-functional requirements while evolving in response to changes in their environment and requirements. My talk will describe how such compliance can be achieved through the run-time use of quantitative verification, and discuss the challenges that must be overcome to make this continual verification feasible for real-world software systems.

5.7 Failure Avoidance using Feature Locality

Myra B. Cohen (University of Nebraska – Lincoln, US)

Despite the best efforts of software engineers, faults still escape into deployed systems. Developers need time to prepare and distribute fixes, and in the interim, deployments must either avoid failures or endure their consequences. Configurable software, software in which features can be added or removed at run-time, are known to suffer from failures that appear only under certain feature combinations, and these failures are particularly challenging for testers, who must find suitable configurations as well as inputs to detect them. We believe that these failures are well suited for avoidance by self-adaptation. This talk discusses that possibility by leveraging a phenomenon we call feature-locality that allows us to use historical data to predict failure-prone configurations and hence reconfiguration workarounds. We have evidence from two case studies that feature locality exists, and that our algorithms can improve time to failure avoidance as more and more history is incorporated. We have implemented a version of our avoidance algorithms within the Rainbow framework and show preliminary results of this study.

5.8 Applying Model Differences to Automate Performance-Driven Refactoring of Software Models

Vittorio Cortellessa (University of L'Aquila, IT)

Identifying and removing the causes of poor performance in software systems are complex problems, and these issues are usually tackled after software deployment only with human-based means. Performance anti patterns can be used to harness these problems since they capture design patterns that are known leading to performance problems, and they suggest refactoring actions that can solve the problems. This talk introduces an approach to automate software model refactoring based on performance antipatterns. A Role-Based Modeling Language is used to model antipattern problems as Source Role Models (SRMs), and antipattern solutions as Target Role Models (TRMs). Each (SRM, TRM) pair is represented by a difference model that encodes refactoring actions to be operated on a software model to remove the corresponding antipattern. Differences are applied to software models through a model transformation automatically generated by a higher-order transformation. The approach is shown at work on an example in the e-commerce domain.

5.9 Assurance of Autonomous Adaptive Systems: (Some) Lessons Learned

Bojan Cukic (West Virginia University – Morgantown, US)

Over the past decade, we have been involved in design, verification and validation of self-adaptive systems in two very different domains: avionics and border management. While the first is safety critical and adaptation addresses unanticipated aircraft failures, the second is critical for national security. In border management, adaptation adjusts the accuracy of traveler identification with the throughput of the border crossing under changing arrival patterns and security requirements. Although these domains appear to have little in common, the assurance arguments we made about systems and adaptation appear to point to common underlying principles. The first commonality is the importance of the choice of an appropriate level of detail available in the model of the system that explores adaptation options (the controller). The strength of assurance arguments is inextricable from the granularity the model supports. We also conclude that the overall goal of assurance of self-adaptive systems is maintaining the assurance case made prior to system deployment. In other words, any adaptation should support existing assurance arguments even if system's functionality or non-functional properties evolve. Finally, the selection of assurance techniques used in case studies points to the diversity. As assurance goals are system specific, the techniques to achieve them vary too.

5.10 Software engineering for self-adaptive software: motivational talk

Carlo Ghezzi (Technical University of Milan, IT)

Research on self-adaptive software systems must mature in a way that assurances can be given on dependability of adaptation. Dependability assurance means that one must be able to show satisfaction of the following fundamental argument (FA): S (specification of the software) and E (environment assumptions and properties) entail satisfaction of R (the requirements)

FA was first stated by Jackson and Zave in their foundational work on requirements. Because of changes in D (and R), FA must be shown to hold not only at design time, but it must be continuously checked also at run time. If the run-time check is done by reasoning on models (of S and D), then a violation of FA must lead to a change in S (and hence in the running software). Possibly, the change in S must be accomplished in a self-managed manner.

5.11 Assurance for Self-Adaptive Software and Models

Holger Giese (Hasso-Plattner-Institut – Potsdam, DE)

This presentation discusses, which role development-time models and runtime models can play for the assurance of self-adaptive software. On the one hand the role of models to assure the proper functioning of the adaptation algorithm is addressed. On the other hand also the roles of model for ensuring the correct implementation is covered. Furthermore, concrete research results for the different cases were presented to outline concrete results that have been achieved for these different roles of models for self-adaptive software.

5.12 Fully Decentralized Service Assembly under Non Functional Requirements

Vincenzo Grassi (University of Rome "Tor Vergata", IT)

Joint work of Grassi, Vincenzo; Mirandola, Raffaela; Marzolla, Moreno

We present a fully decentralized solution to the adaptive self-assembly of distributed services. The proposed solution is able to build and maintain an assembly of services, guaranteeing the fulfillment of both functional requirements, and non functional requirements concerning global quality of service (QoS) and structural properties. The key aspect of our solution is the use of a gossip protocol to achieve decentralized information dissemination and decision making. Simulation experiments show the effectiveness of our approach in terms of robustness, and convergence speed.

5.13 Runtime Quality Problem Detection Techniques with Statistical Techniques: Theory and Practical Applications

Lars Grunske (Universität Stuttgart, DE)

Software systems may suffer at runtime from changes in their operational environment or/and requirements specification, so they need to be adapted to satisfy the changed environment or/and specifications [5]. The research community has developed a number of approaches to building adaptive systems that respond to these changes such as Rainbow [6]. Additionally currently, several approaches have been proposed to monitor QoS attributes at runtime with the goal of reactively detecting QoS violations (e.g. [9]).

The presentation will introduce some reactive [1, 3, 8] and proactive [2, 4] detection techniques.

References

1 Ayman Amin, Alan Colman, and Lars Grunske. Using Automated Control Charts for the Runtime Evaluation of QoS Attributes. In Proc. of the 13ht IEEE Int. High Assurance Systems Engineering Symposium, pages 299–306. IEEE Computer Society, 2011.

2 Ayman Amin, Alan Colman, and Lars Grunske. An Approach to Forecasting QoS Attributes of Web Services Based on ARIMA and GARCH Models. In Proc. of the 19th Int. Conf. on Web Services, pages 74–81. IEEE, 2012.

3 Ayman Amin, Alan Colman, and Lars Grunske. Statistical Detection of QoS Violations Based on CUSUM Control Charts. In Proc. of the 3rd ACM/SPEC Int. Conf. on Performance Engineering, pages 97–108. ACM, 2012.

4 Ayman Amin, Lars Grunske, and Alan Colman. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. In Proc. of the 27th IEEE/ACM Int. Conf. on Automated Software Engineering, pages 130–139. IEEE, 2012.

5 Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. IEEE Trans. Software Eng., 37(3):387–409, 2011.

6 David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10):45–54, 2004.

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

8 Lars Grunske, Pengcheng Zhang: Monitoring probabilistic properties. ESEC/SIGSOFT FSE 2009: 183–192

9 Raffaela Mirandola and Pasqualina Potena. A QoS-based framework for the adaptation of service-based systems. Scalable Computing: Practice and Experience, 12(1), 2011.

5.14 Pluggable Verification for Models at Runtime

Jean-Marc Jezequel (University of Rennes, FR)

We propose an approach to integrate the use of ad hoc verifiers (e.g. time-related stochastic properties) in a continuous design process based on models at runtime. Assurance guarantees are an important aspect of component-based architectures, for instance in distributed, volatile networks of computation nodes. The models at runtime approach eases the management of such architectures by maintaining abstract models of architectures synchronized with the physical, distributed execution platform. For self-adapting systems, prediction of quality attributes such as delays and throughput of a component assembly is of utmost importance to take adaptation decision and accept evolutions that conform to non functional specifications. To this aim we propose a modular way of defining metamodel extensions to capture quality attributes. Model transformations are then executed at runtime to process these extensions and connect to off-the-shelf verification engines. The result of the verification is then fed back into the decision engine to help it choose the right reconfiguration model, before it is enacted on the component platform level. Of course one challenge is that both the transformation machinery and the verification engine are fast enough to be used at runtime.

5.15 Can RE Contribute to SAS Assurance?

Zhi Jin (Peking University, CN)

When systems will run in open, changing, uncertain execution and/or interaction environments, self-adaptation becomes a must-be requirement. That is the expectation to the systems to cope with variable resources and variable interactors that may cause system errors, while maintaining the business goals envisioned by the engineers and expected from the users. The task of the requirement phase is to identify such a requirement and explore the requirement into sufficient detailed specification that is ready for system design. This talk proposes an environment-based methodologies for engineering the self- adaptation requirements, that is the first step assurance. The following are the three features: (1) Identifying the requirement should start from the environment conceptualisation that allowing to identifying the uncertainty and changing patterns of the environment; (2) Modelling the system-to-be as discrete control system which captures the capabilities of monitoring and detecting the potential changes in both environment and system itself; (3) Defining precisely the specification of the run-time mechanism for allowing the system to be self- adaptive.

5.16 Self-adaptivity vs. Latent Software Defects: Software Health Management

Gabor Karsai (Vanderbilt University, US)

The complexity of software systems is reaching the point where latent defects remain in deployed systems. To detect, isolate, and mitigate the effects of such defects new paradigms are needed. One such paradigm is software health management that is interesting application area of self- adaptive software techniques. Software health management borrows the language and techniques from system health management (frequently used in aerospace vehicles), where anomaly detection, fault diagnostics, and reconfiguration are used to remove faults from systems while in operation. The talk describes a software framework that was created to enable developers to design and implement software health management functions in their systems.The framework is based on a software component model, local and global health managers. The latter incorporates a system-wide fault diagnostics component as well as a reasoner engine that computes architectural adaptations of the component architecture to mitigate the effect of component faults. The framework is supported by a model-driven development toolchain.

5.17 Challenges in Autonomous Vehicle Validation

Philip Koopman (Carnegie Mellon University, US)

Creating safe Transportation Cyber-Physical Systems (CPSs) presents new challenges as autonomous operation is attempted in unconstrained operational environments. The extremely high safety level required of such systems (perhaps one critical failure per billion operating hours) means that validation approaches will need to consider not only normal operation, but also operation with system faults and in exceptional environments. Additional challenges will need to be overcome in the areas of rigorously defining safety requirements, trusting the safety of multi-vendor distributed system components, tolerating environmental uncertainty, providing a realistic role for human oversight, and ensuring sufficiently rigorous validation of autonomy technology.

5.18 Control-theoretical computing system design

Alberto Leva (Technical University of Milan, IT)

In the last years, we have been addressing several problems related to computing systems by adopting a control-theoretical approach. With respect to the mainstream research on the matter, our work has a relevant peculiarity: instead of taking the system "as is" and just adding a control layer on top of it, we try to isolate the parts of the overall problem that can be formally stated as control ones, and design parts of the considered system consequently. To explain with an example relative to task scheduling, we do not take an existing and fully functional scheduler and use a feedback controller to adapt its tuning parameters (e.g., priorities). On the contrary, we build the entire scheduler as a controller, that directly decides the CPU times to be allotted to the tasks. Applying this idea to different contexts, ranging from time synchronisation to service composition and dynamic binding, we found that isolating the "core" problems – in the sense above – tends to allow for the application of simple modelling and control methods. We also noticed that once formalised this way, many heterogeneous problems come to assume a significantly uniform mathematical structure. We finally observed that some properties of interest for control systems are potentially very keen to be re-formulated as "assurances" in the computing systems sense. As a result, besides favouring efficiency, the proposed approach also eases the formal assessment of some properties of interest. We therefore conjecture that by setting up (wherever possible and convenient) a sound correspondence between said properties – that are natively expressed in control-theoretical terms – and assurances in the computing system sense, significant benefits can be achieved.

5.19 Adaptation in Software Defined Infrastructures

Marin Litoiu (York University – Toronto, CA)

To be efficient and robust, an adaptive feedback loop has to aggregate the contributions of different software layers. For example, the performance of an application can be controlled not only by tuning the parameters of the application but also by adjusting the underlying computing, storage, network or hardware infrastructures. With the advent of Software Defined Infrastructure, those adjustment can be effected at runtime. In this presentation, we introduce the Smart Applications on Virtual Infrastructure (SAVI), a Canadian project that builds an experimental Software Defined Infrastructure testbed. As an example of problems that we can solve more efficiently with the cross layer approach are the Low and Slow Distributed Denial of Service attacks. Those attacks are becoming a serious issue because, due to low resource consumption and slow ramping, they are hard to detect. A possible solution is to identify the attack at the application layer and then to use the underlying layers to mitigate the attack.

5.20 Toward the Making of Software that Learns to Manage Itself

Sam Malek (George Mason University – Fairfax, US)

A self-managing software system is capable of adjusting its behavior at runtime in response to changes in the system, its requirements, or the environment in which it executes. Self-management capabilities are sought-after to automate the management of complex software in many computing domains, including service-oriented, mobile, cyber-physical and ubiquitous settings. While the benefits of such software are plenty, its development has shown to be much more challenging than the conventional software.

At the state of the art, it is not an impervious engineering problem in principle to develop a self-adaptation solution tailored to a given system, which can respond to a bounded set of conditions that are expected to require automated adaptation. However, any sufficiently complex software system once deployed in the field is subject to a broad range of conditions and many diverse stimuli. That may lead to the occurrence of behavioral patterns that have not been foreseen previously: in fact, those may be the ones that cause the most critical problems, since, by definition, they have not manifested themselves, and have not been accounted for during the previous phases of the engineering process. A truly self-managing system should be able to cope with such unexpected behaviors, by modifying or enriching its adaptation logic and provisions accordingly.

In this talk, I will first provide an introduction to some of the challenges of making software systems self-managing. Afterwards, I will provide an overview of two research projects in my group that have tackled these challenges through the applications of automated inference techniques (e.g., machine learning, data mining). The results have been promising, allowing the software engineers to empower a software system with advanced self-management capabilities with minimal effort. I will conclude the talk with an outline of future research agenda for the community.

5.21 On the uncertainties in the modeling of self-adaptive systems

Raffaela Mirandola (Technical University of Milan, IT)

The complexity of modern software systems has grown enormously in the past years with users always demanding for new features and better quality of service. The satisfaction of non-functional requirements like performance and availability is of paramount importance if a product hopes to be considered in the marketplace. Model-based evaluation techniques at software design-time have been proposed to ensure the delivering of software that meets its non-functional requirements. However, since a large part of modern software is embedded in dynamic execution contexts where requirements, environment assumptions, and usage profiles continuously change, this quality assessment at design time becomes more difficult. As an answer to dynamic execution context, self-adaptive systems have been adopted. Self-adaptation endows a system with the capability to accommodate its execution to different contexts in order to achieve continuous satisfaction of requirements. Often, self- adaptation process also makes use of runtime model evaluations to decide the changes in the system. However, even at runtime, context information that can be managed by the system is not complete or accurate; i.e, it is still subject to some uncertainties. This work motivates the need for the consideration of the concept of uncertainty in the model-based evaluation as a primary actor, classifies the avowed uncertainties of self-adaptive systems, and illustrates examples of how different types of uncertainties are present in the modeling of system characteristics for availability requirement satisfaction.

5.22 A Software Lifecycle Process For Data-intensive Self-adaptive Systems

Marco Mori (University of Namur, BE)

Joint work of Cleve, Anthoy; Inverardi, Paola; Mori, Marco

Nowadays ubiquitous software systems have to meet user expectations while considering an ever-changing environment. The increasing space of possible contexts and the limited capacity of mobile devices make no longer possible to incorporate all necessary software alternatives and the required data for all possible contexts. Thus, upon variations to user task, user role, user preferences or physical environment, the current software alternative and its required data have to be reconfigured. In order to prevent incorrect system behaviours, reconfigurations should avoid inconsistencies of both data and software by providing assurance to the uncertainty of changes affecting the system.

In this talk we present a generic lifecycle process for self-adaptive systems which supports predictable and unpredictable system evolutions and different notions of inconsistencies which apply to code artifacts and system requirements. Consequently we introduce the problem of supporting adaptivity of data belonging to a global database and we improve the former definition of assurance by considering consistency of database schema and instances. We claim the need for a new lifecycle process which has to consider together adaptivity of software with adaptivity of data occurring in a consistent predictable and unpredictable manner.

5.23 Managing Viability Zone Dynamics for the Assurance of Self-Adaptive Systems

Hausi A. Mueller (University of Victoria, CA)

We define the viability zone of a self-adaptive software (SAS) system as the set of possible states in which the system operation is not compromised, that is, the set of states where the 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. Viability zones are N-dimensional. Therefore, 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 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. Viability zones 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. 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. 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 to keep the adaptive system inside its viability zone. We believe that managing viability zones at runtime is crucial for the assurance of self-adaptive systems.

5.24 Self-Adaptive Cloud Controllers

Mauro Pezze (University of Lugano, CH)

Cloud technologies are rapidly substituting classic computing solutions and challenge the community with new problems. In our research we focus on cloud controllers and we work on novel solutions for self-adaptive cloud controllers based on Kriging models. While in classic software engineering solutions, scheduling problems are mostly hidden from the application developers, in Cloud based applications the responsibility of allocating the required resources is assigned to the developers and depends on the application requirements and the nature of the cloud. General-purpose Cloud schedulers provide sub-optimal solutions to the problem with respect to application-specific solutions that we call cloud controllers. We are investigating the use of surrogate models, and in particular Kriging models, that present interesting properties to support adaptive control.

5.25 Feedbacks Control Loops as 1st Class Entities – The SALTY Experiment

Romain Rouvoy (Université de Lille I, FR)

Joint work of Collet, Philippe; Krikava, Filip; Rouvoy, Romain; Seinturier, Lionel
Main reference F. Krikava, P. Collet, R. France, "ACTRESS: Domain-Specific Modeling of Self-Adaptive Software Architectures," in Proc. of the 9th DADS Track of the 29th ACM Symp. on Applied Computing, ACM, 2014.
URL http://salty.unice.fr/

This talk shortly reports on the results of the SALTY R&D project (https://salty.unice.fr) funded by the French funding agency (ANR). The key outcome of this project is a software framework that covers both design-time and runtime support for integrating self-adaptive behaviours into potentially complex legacy systems. SALTY therefore provides a reflective domain-specific model to externalise and make explicit the control layer of legacy systems. While the adoption of a domain-specific model leverages the mapping on different middleware stacks (FraSCAti or Akka in our case studies), it also acts as a pivot, within a modular toolchain, to implement design-time verifications and to inject runtime guards. Future case studies of this approach will cover green computing, crowd- sensing, and big data systems.

5.26 Model-driven infrastructure for reliable service compositions using dynamic software product lines

Cecilia Mary Fischer Rubira (UNICAMP, BR)

Joint work of Rubira, Cecilia Mary Fischer; Nascimento; Amanda S. Castor; Fernando
Main reference A. Nascimento, C. Rubira, F. Castor, "ArCMAPE: A Software Product Line Infrastructure to Support Fault-Tolerant Composite Services," in Proc. of the 15th IEEE Int'l Symposium on High Assurance Systems Engineering (HASE'14), pp. 41–48, IEEE CS, 2014.
URL http://doi.ieeecomputersociety.org/10.1109/HASE.2014.15

A number of solutions use software fault tolerance techniques based on design diversity to create fault-tolerant composite services that leverage functionally equivalent services. Nevertheless, these solutions are not able to adapt themselves at runtime to cope with dynamic changes of user requirements and fluctuations in the quality of services (QoS). We propose a self-adaptive solution, called ArCMAPE, that leverages ideas from Software Product Line Engineering to support fault-tolerant composite services. In particular, we specify a feature model and product line architecture to capture the common and variable features among a number of software fault tolerance techniques based on design diversity. ArCMAPE provides software com- ponents implementing the common features; and a foundation on which plug-in components, or variable components, can be easily added to realise the target variable features. At runtime, ArCMAPE dynamically instantiates software fault tolerance techniques tailored to the specific needs of different clients and contexts by employing feature-based runtime adaptations. Outcomes obtained from an empirical study suggest the feasibility and efficiency of our solution to support self-adaptive, fault- tolerant composite services. We discuss the obtained outcomes and present directions for future work.

5.27 Modular Discrete Control for Adaptive Software Systems

Eric Rutten (INRIA Grenoble – Rhône-Alpes, FR)

This talk will present a synthesis on our work on safe design of controllers for autonomic computing systems, using techniques originally conceived in Control thoery, more specifically Discrete event Systems. Our approach is supported by a programming language from the family of reactive languages, and based on a formalism of Labelled Transition Systems. Applications concern control of logical and synchronization aspects in reconfigurable FPGA-based architectures, coordination of multiple loops in Data-center management, and smart-environments in the Internet of Things.

5.28 Managing Non-Functional Uncertainty via Model-Driven Adaptivity

Giordano Tamburrelli (University of Lugano, CH)

Modern software systems are often characterized by uncertainty and changes in the environment in which they are embedded. Hence, they must be designed as adaptive systems. We propose a framework that supports adaptation to non-functional manifestations of uncertainty. Our framework allows engineers to derive, from an initial model of the system, a finite state automaton augmented with probabilities. The system is then executed by an interpreter that navigates the automaton and invokes the component implementations associated to the states it traverses. The interpreter adapts the execution by choosing among alternative possible paths of the automaton in order to maximize the system's ability to meet its non-functional requirements. To demonstrate the adaptation capabilities of the proposed approach we implemented an adaptive application inspired by an existing worldwide distributed mobile application and we discussed several adaptation scenarios.

5.29 Managing Viability Zone Dynamics for the Assurance of Self-Adaptive Systems

Gabriel Tamura (Universidad Icesi, CO)

We define the viability zone of a self-adaptive software (SAS) system as the set of possible states in which the system operation is not compromised, that is, the set of states where the 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. Viability zones are N-dimensional. Therefore, 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 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.

Viability zones 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. 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. 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 to keep the adaptive system inside its viability zone. We believe that managing viability zones at runtime is crucial for the assurance of self-adaptive systems.

5.30 Managing Viability Zone Dynamics for the Assurance of Self-Adaptive Systems

Norha Milena Villegas Machado (Universidad Icesi, CO)

We define the viability zone of a self-adaptive software (SAS) system as the set of possible states in which the system operation is not compromised, that is, the set of states where the 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. Viability zones are N-dimensional. Therefore, 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 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.

Viability zones 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. 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. 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 to keep the adaptive system inside its viability zone. We believe that managing viability zones at runtime is crucial for the assurance of self-adaptive systems.

5.31 Modeling Self-Adaptive Software

Thomas Vogel (Hasso-Plattner-Institut – Potsdam, DE)

Self-adaptive software typically uses a self-representation to reflect on its adaptable parts. While runtime models are employed for such a self-representation, runtime models and model-driven engineering methods may additionally be employed to specify, execute, and adjust (on-line and off-line) the individual adaptation activities of a feedback loop as well as the whole feedback loop. Besides leveraging flexibility, keeping models alive at runtime and considering them as first class entities support the provisioning of assurances for on-line and off-line adaptation based on such models. Therefore, a formal underpinning of modeling languages and models, the composition and decomposition of models and assurances, and assurances for the models themselves are critical aspects that, among others, have to be addressed.

5.32 ActivFORMS: Active FORmal Model for Self-adaptation

Danny Weyns (Linnaeus University – Växjö, SE)

Self-adaptation enables a software system to adapt itself at runtime to deal with uncertainties, such as dynamic operating conditions that were difficult to predict or unanticipated changes of goals. Self-adaptation is realized with a feedback loop, which typically consists of monitor, analysis, plan, and execution functions. To provide guarantees of the adaptations, state of the art approaches propose to equip the feedback loop with formal models of the managed system, its environment and goals. However, existing approaches do not systematically formalize and verify the behavior of the adaptation functions themselves. Furthermore, there is limited attention for adaptation of unanticipated changes. We propose ActivFORMS (Active FORmal Model for Self-adaptation) that uses a formal model of the complete feedback loop. This model is directly executed at runtime by a virtual machine realizing adaptation. ActivFORMS assures that the verified system goals are met at runtime, and the approach enables dynamic updates of self-adaption behaviors to support unanticipated changes.

5.33 Reconciling self-adaptation and self-organization towards effective assurances

Franco Zambonelli (University of Modena, IT)

Two complementary software engineering approaches currently exist to make complex software systems adaptive. Self-adaptation approaches attack the problem by engineering proper feedback loops around components and systems, so as to make them adaptive by explicit design. Self-organization approaches, on the other hand, attack the problems by trying to mimic in software the capabilities of collective adaptation of natural systems, so as to make systems adaptive by emergence. A key challenge is that, while self-organization can be much more effective, assurances can be better achieved via self-adaptation. Accordingly, I analyze the issue of reconciling the two approaches towards a novel, innovative, approach that to synthesize the benefits of both approaches, efficiency and assurance.

5.34 Runtime Testing of Self-Adaptive Systems

Carlos Eduardo da Silva (Federal University of Rio Grande do Norte, BR)

In our previous work we have tackled the problem of dynamically generating adaptation plans in order to deal with situation not foreseen during development. We have developed a framework for the dynamic generation of processes that factors out common process generation mechanisms and provides explicit customisation points to tailor process generation capabilities to different application domains. Such framework has been focused on the planning aspects of the MAPE-K loop, and has benn applied in two different domains, namely, architectural reconfiguration and integration testing. We have also considered the problem of dynamic modifying the mechanisms responsible for generating processes.

Currently, we have been looking further on the aspects related to testing at run-time as the means of providing assurances about self-adaptive systems. We have also been looking on "feature phases", that is, MAPE-K loops that deals with different features (concerns) of each of the phases of the main MAPE-K loops, investigating patterns of interaction between MAPE-K loops and their phases as the means to position run-time testing in the MAPE-K loop.

5.35 Architecting Resilience: Handling Malicious and Accidental Threats

Rogerio de Lemos (University of Kent, GB)

Resilience is the persistence of service delivery that can justifiably be trusted, when facing changes. While architecting is the art and science of creating and building complex systems, and which covers the following basic activities: scope, structure and certification. One important aspect of resilience is the provision of assurances, and these are obtained by building arguments about system resilience. However in order to build arguments, one needs to collect, structure and analyse evidence. In self-adaptive systems, evidence can be obtained either at development-time or run-time.

This talk has covered three contributions. In the first contribution, we describe how for self-adaptive software systems integration testing can be performed at run-time. This activity should be implemented as a feedback control loop, which should be associated with the analysis phase of the autonomic MAPE-K loop.

The second contribution is related to a stepwise progress for the provision of assurances about the resilience of self-adaptive software systems, and it covers the following topics: (i) resilience evaluation based on environmental stimuli in which probabilistic model-checking is used for obtaining levels of confidence, (ii) resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems, (iii) robustness evaluation of controllers by injecting faults into the probes of Rainbow, (iv) effectiveness of architecture-based self-adaptation by evaluating the effort of evolving industrial middleware into a architectural-based self-adaptive software system, finally (v) robustness-driven resilience evaluation of self-adaptive software systems in which system properties are evaluated by injecting faults.

The third contribution concerns an approach based on self-adaptation as a means to improve the management of malicious behaviour, by adapting authoris ation policies and access rights. The goal is to adapt to mitigate malicious behaviour, and prevent future attacks.

References

1 J. Camara, R. de Lemos, M. Vieira, R. Almeida, R. Ventura. Architecture-Based Resilience Evaluation for Self-Adaptive Systems. Computing Journal (Special "Software Architecture for Code Testing and Analysis") 95(8). 2013. pp. 689–722.

2 J. Camara, R. de Lemos. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012). Zurich, Switzerland. June 2012. pp. 53–62.

3 J. Camara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, R. Ventura. Evolving an Adaptive Industrial Software System to Use Architecture-Based Self-Adaptation. Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013). San Francisco, CA, USA. May 2013. pp. 13–22.

4 J. Camara, R. de Lemos, N. Laranjeiro, R. Ventura, M. Vieira. Robustness Evaluation in Self-Adaptive Software Systems. Latin American Symposium on Dependable Computing (LADC 2013). Rio de Janeiro, RJ, Brazil. April 2013. pp. 1–10.

5 C. Bailey, D. W. Chadwick, R. de Lemos, K. W. S. Sui. Enabling the Autonomic Management of Federated Identity Providers. 7th International Conference on. Autonomous Infrastructure, Management and Security (AIMS 2013). June 2013, UPC Barcelona, Spain. 2013. pp. 100–111.

6 C. E. da Silva and R. de Lemos. Dynamic plans for integration testing of self-adaptive software systems. Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011). Honolulu, HI, USA. May 2011. pp. 148–157.

Participants

- Jesper Andersson
Linnaeus University – Växjö, SE
- Chris Bailey
University of Kent, GB
- Luciano Baresi
Technical University of Milan, IT
- Benoit Baudry
INRIA Rennes – Bretagne Atlantique, FR
- Nelly Bencomo
Aston Univ. – Birmingham, GB
- Yuriy Brun
University of Massachusetts – Amherst, US
- Radu Calinescu
University of York, GB
- Javier Cámara
Carnegie Mellon University, US
- Myra B. Cohen
Univ. of Nebraska – Lincoln, US
- Vittorio Cortellessa
University of L'Aquila, IT
- Bojan Cukic
West Virginia University – Morgantown, US
- Carlos Eduardo da Silva
Federal University of Rio Grande do Norte, BR
- Rogerio de Lemos
University of Kent, GB
- Antonio Filieri
Universität Stuttgart, DE
- Carlo Ghezzi
Technical University of Milan, IT
- Holger Giese
Hasso-Plattner-Institut – Potsdam, DE
- Alessandra Gorla
Universität des Saarlandes – Saarbrücken, DE
- Vincenzo Grassi
University of Rome "Tor Vergata", IT
- Lars Grunske
Universität Stuttgart, DE
- Paola Inverardi
Univ. degli Studi di L'Aquila, IT
- Jean-Marc Jezequel
University of Rennes, FR
- Zhi Jin
Peking University, CN
- Gabor Karsai
Vanderbilt University, US
- Philip Koopman
Carnegie Mellon University, US
- Seok-Won Lee
Ajou University, KR
- Alberto Leva
Technical University of Milan, IT
- Marin Litoiu
York University – Toronto, CA
- Sam Malek
George Mason University – Fairfax, US
- Raffaela Mirandola
Technical University of Milan, IT
- Marco Mori
University of Namur, BE
- Hausi A. Müller
University of Victoria, CA
- Mauro Pezzè
University of Lugano, CH
- Romain Rouvoy
Université de Lille I, FR
- Cecilia Mary Fischer Rubira
UNICAMP, BR
- Eric Rutten
INRIA Grenoble – Rhône-Alpes, FR
- Bradley Schmerl
Carnegie Mellon University, US
- Mary Shaw
Carnegie Mellon University, US
- Giordano Tamburrelli
University of Lugano, CH
- Gabriel Tamura
Universidad Icesi, CO
- Norha Milena Villegas Machado
Universidad Icesi, CO
- Thomas Vogel
Hasso-Plattner-Institut – Potsdam, DE
- Danny Weyns
Linnaeus University – Växjö, SE
- Franco Zambonelli
University of Modena, IT