

Models@run.time

Edited by

Uwe Aßmann¹, Nelly Bencomo², Betty H. C. Cheng³, and
Robert B. France⁴

1 TU Dresden, DE, uwe.assmann@tu-dresden.de

2 INRIA Paris-Rocquencourt, FR, nelly@acm.org

3 Michigan State University, US, chengb@cse.msu.edu

4 Colorado State University, US, france@cs.colostate.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 11481 “Models@run.time”. Research on models@run.time seeks to extend the applicability of models and abstractions to the runtime environment, with the goal of providing effective technologies for managing the complexity of evolving software behaviour while it is executing. The Dagstuhl Seminar “Models@run.time” brought together a diverse set of researchers and practitioners with a broad range of expertise, including MDE, software architectures, reflection, self-adaptive systems, validation and verification, middleware, robotics and requirements engineering.

Seminar 27. November – 02. December, 2011 – www.dagstuhl.de/11481

1998 ACM Subject Classification D.2 Software Engineering, D.2.10 Design, H.1 Models and Principles

Keywords and phrases Self-adaptive Systems, Feedback Loop, Assurance, Uncertainty, Requirements, Optimization, Adaptation

Digital Object Identifier 10.4230/DagRep.1.11.91

Edited in cooperation with Sebastian Götz


1 Executive Summary

Uwe Aßmann

Nelly Bencomo

Betty H. C. Cheng

Robert B. France

License  Creative Commons BY-NC-ND 3.0 Unported license
© Uwe Aßmann, Nelly Bencomo, Betty H. C. Cheng, and Robert B. France

To date, research on model-driven engineering (MDE) has mainly focused on the use of models during software development. This work has produced relatively mature techniques and tools that are currently being used in industry and academia to manage software complexity during development. Research on models@run.time seeks to extend the applicability of models and abstractions to the runtime environment, with the goal of providing effective technologies for managing the complexity of evolving software behaviour while it is executing.

As is the case for many software development models, a runtime model is often created to support reasoning. However, in contrast to development models, runtime models are used to reason about the operating environment and runtime behaviour for some purpose, for example, to determine an appropriate form of adaptation, and thus these models must capture abstractions of runtime phenomena. Different runtime dimensions need to be balanced when



Except where otherwise noted, content of this report is licensed under a Creative Commons BY-NC-ND 3.0 Unported license

Models@run.time, *Dagstuhl Reports*, Vol. 1, Issue 11, pp. 91–123

Editors: Uwe Aßmann, Nelly Bencomo, Betty H. C. Cheng, and Robert B. France



DAGSTUHL
REPORTS

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

adapting software at runtime, including efficient use of resources (time, memory, energy), context-dependencies (time, location, platform), and personalization concerns (quality-of-service specifications, profiles). The hypothesis can be stated as follows: Models@run.time that provide meta-information on these dimensions during execution enables the development of technologies that automate (1) runtime decision-making and (2) safe adaptation of runtime behaviour. Thus, we anticipate that this technology will play an integral role in the management of autonomic systems and self-adaptive systems.

The problems targeted by the models@run.time community are multi-faceted and thus tackling them requires expertise from a variety of research areas. The Dagstuhl Seminar models@run.time brought together a diverse set of researchers and practitioners with a broad range of expertise, including MDE, software architectures, reflection, self-adaptive systems, validation and verification, middleware, robotics and requirements engineering.

The following gives the objectives of the seminar and describes the extent to which they were met :

1. Objective: To identify and document the potential benefits of Models@run.time including benefits associated with their use in adaptive and autonomic systems. Extent Met: One of the working groups (Group 3: Uses and Purposes of M@RT) was charged with coming up with use cases that demonstrate the benefits associated with the use of models@run.time. The report produced by this group discusses the types of significant software systems that can benefit from use of models@run.time.
2. Objective: To reach a common understanding of the terminology and associated concepts that underpin the use of different models once a system is deployed. Extent Met: Each working group defined the terminology and the concepts used in the descriptions of their primary outcomes.
3. Objective: To identify a set of key research challenges that must be tackled to address the real-world problems posed by self-adaptive systems within the next five (5) years (the research roadmap). Extent Met: Each working group identified, discussed, and described the key challenges in its focus area. The identified challenges will be included in the roadmap we plan to publish.
4. Objective: To also identify associated technology transfer strategies to ensure that the research in this area has impact on industrial practice and associated methodologies and tool-sets. Extent Met: Group 2 (Runtime updating/adaptation mechanisms) focused on discussing and analysing the effectiveness of technologies that have been developed to support models@run.time.
5. Objective: To publish a collection of articles containing the roadmap, as well as papers from the participants. Extent Met: Plans have been put in motion for publishing peer-reviewed papers from participants in a LNCS State-of-the-Art Survey Volume on Models@run.time.

The seminar consisted of participant presentation and working group sessions. Monday and Tuesday morning were "speed dating" days, in which everyone (with the exception of some of the organizers, to better manage time) presented a 10-min introductory talk. The presenters covered a wide range of topics including adaptive cyber-physical systems, self-evolution, requirements-driven runtime adaptation, safe and trustworthy autonomous robots, adaptive and self-managing software, runtime variability and architectural reconfiguration at runtime. We also had two longer presentations by persons from industry (ERICSSON). The presenters were Joe Armstrong and Paer Emanuelsson. From Tuesday to Thursday participants worked in working groups that focused on particular aspects of models@run.time

research. At the start of each day the organizers summarized the activities that were to take place and the deliverables that were to be produced at the end of the day. At the end of each day, each group presented their deliverables.

On Wednesday evening we had a productive panel where the following questions were discussed: What are the compelling business models for models@run.time?, What are the killer applications? What are the obstacles to deployment of models@run.time systems?, What are key enabling technologies for models@run.time (e.g., standards or component models).

2 Table of Contents

Executive Summary

Uwe Aßmann, Nelly Bencomo, Betty H. C. Cheng, and Robert B. France 91

Overview of Talks

Challenges in Designing models@runtime Systems and Our Related Research Activities <i>Mehmet Aksit</i>	97
Programming systems that never stop <i>Joe Armstrong</i>	98
Multi-quality Dynamic Auto-Tuning for Cyber-Physical Systems <i>Uwe Aßmann</i>	98
Claim Monitoring for Tackling Uncertainty in Adaptive Systems <i>Nelly Bencomo</i>	99
Models@RT to support Interoperability <i>Amel Bennaceur</i>	99
Self-Evolution under the Hood <i>Walter Cazzola</i>	100
Challenges for Models@Runtime <i>Amit K. Chopra</i>	101
Urban-Scale, Dynamically Adaptable Systems <i>Siobhan Clarke</i>	101
A (Personal) Historical Perspective on Models@Run.time <i>Fabio M. Costa</i>	102
Models at RunTime for the Certification of Autonomous Assistive Robots <i>Kerstin I. Eder</i>	103
Flexible Telecom systems that never stop <i>Paer Emanuelsson</i>	105
Models@RT to support Interoperability <i>Nikolaos Georgantas</i>	105
Models at Runtime for Adaptive and Self-managing Software <i>Holger Giese</i>	106
Multi-Quality Auto-Tuning with Contract Negotiation <i>Sebastian Götz</i>	107
Runtime Monitoring of Java Bytecode with UML and OCL Models <i>Martin Gogolla</i>	107
Models@run.time for the proactive detection of QoS Problems <i>Lars Grunske</i>	108
SM@RT: A Model Driven Framework for Runtime Software Architecture <i>Gang Huang</i>	108
EAGLE: Engineering softwAre in the ubiquitous Globe by Leveraging uncErtainty <i>Paola Inverardi</i>	109

Models@RT to support Interoperability <i>Valerie Issarny</i>	109
Hyper-Agility: Handling Variability from Design-Time to Runtime <i>Jean-Marc Jezequel</i>	110
Multi-model adaptive control <i>Marin Litoiu</i>	110
Including humans in the Models@Run.Time loop <i>Brice Morin</i>	111
Requirements for Models at Runtime <i>Pieter J. Mosterman</i>	111
Models@Runtime for Self-Adaptation an Self-Protection <i>Liliana Pasquale</i>	112
Automatic Synthesis of Behaviour Protocols for Composable Web-Services <i>Patrizio Pelliccione</i>	112
Continuous Requirements Engineering for Self-Adaptive Software Systems <i>Anna Perini and Nauman Ahmed Qureshi</i>	113
A Model-based Framework for Dynamic Adaptive Systems <i>Andres J. Ramirez</i>	113
The need of M@RT in Event-driven Process-centric Decision Support <i>David Redlich</i>	114
Delta Modelling for Architectural Reconfiguration at Runtime <i>Bernhard Rumpe</i>	115
Run-time Model Projections for Software Failure Prediction <i>Giordano Tamburrelli</i>	115
Models@run.time in self-* systems <i>Matthias Tichy</i>	117
Safety Models@Run-Time in Open Adaptive Systems <i>Mario Trapp and Daniel Schneider</i>	117
A generalized view on models@run.time <i>Frank Trollmann</i>	118
SmarterContext: Realizing Dynamic Context Management by Exploiting Context Models@Runtime <i>Norha Milena Villegas and Hausi Müller</i>	118
Models at Runtime for Adaptive and Self-managing Software <i>Thomas Vogel</i>	119

Working Groups


Group 1: Assurance Using Models@Run.Time for Self-Adaptive Systems <i>B.H.C. Cheng, K.I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H.A. Müller, P. Pelliccione, A. Perini, N.A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, N.M. Villegas</i>	119
--	-----

Group 2: Runtime Updating/Adaptation Mechanisms <i>A. Bennaceur, M. Aksit, R. France, W. Cazzola, F.M. Costa, P. Emmanuelson, N. Georgantas, H. Gang, P.J. Mosterman, D. Redlich, A. Pierantonio, G. Tamburrelli, T. Vogel and M. Tichy</i>	120
Group 3: Uses and Purposes of M@RT Systems <i>U. Aßmann, S. Götz, J.-M. Jezequel, B. Morin and M. Trapp</i>	121
Group 4: Living With Uncertainty In the Age of Runtime Models <i>N. Bencomo, A.K. Chopra, S. Clarke, H. Giese, P. Inverardi, V. Issarny, L. Pasquale, and A.J. Ramirez</i>	122
Participants	123

3 Overview of Talks

3.1 Challenges in Designing models@runtime Systems and Our Related Research Activities

Mehmet Aksit (University of Twente, NL)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Mehmet Aksit

Runtime adaptable systems have become increasingly popular during the last decade, and there is a clear evidence that this trend will continue also in the future [1]. Designing effective and efficient models@runtime systems is far from trivial, since solutions to the design challenges demand combination of various techniques:

(i) there is a need for new languages and computational models that help in expressing such systems effectively. It has been shown that current object-oriented and aspect-oriented languages fall in short in this matter. Event-based languages offer considerable advantages [2,3]. Most runtime enforcement approaches (their objectives are similar to the ones of models@runtime) based on aspect-oriented languages support single base language (such as Java) and uni-process implementations and their expressive power is limited. We have developed various aspect-oriented (event-based) languages and systems (for example, Event-Composition Model and the EventReactor language) to overcome these limitations [4,5,6].

(ii) It is important to enforce the invariants of the models and the running systems and the conformance between the models and the systems. In [7], we show model checking of UML based models and conformance checking between the models and the running systems.

(iii) The control system(s) need to evaluate the running system from the perspective of various qualities (correctness, availability, memory usage, energy usage, performance etc.) and enforce optimal adaptation and control strategies. We have developed an architectural style and various techniques (for example based on multi-objective optimization) for this purpose [8,9,10].

(iv) We also believe that fuzzy-probabilistic techniques can be effective in dealing with uncertainty in such systems [11].


References

- 1 Aksit, M. and Choukair, Z. (2003) Dynamic Adaptive and Reconfigurable Systems Overview and Prospective Vision. In: Workshop on Distributed Auto-adaptive Reconfigurable Systems (DARES) – International Conference on Distributed Computing Systems (ICDCS), 23rd International Conference on Distributed Computing Systems Workshops (ICDCS'03 Workshops), 19-22 May 2003, Rhode Island, Providence, USA. pp. 84-89. IEEE Computer Society. ISBN 0-7695-1921-0
- 2 Malakuti Khah Olun Abadi, S. (2011) Event Composition Model: Achieving Naturalness in Runtime Enforcement. PhD thesis, University of Twente. CTIT Ph.D. thesis series no. 11-205 ISBN 978-90-365-3246-4
- 3 Bockisch, C.M. and Malakuti Khah Olun Abadi, S. and Aksit, M. and Katz, S. (2011) Making aspects natural: events and composition. In: Tenth International Conference on Aspect-Oriented Software Development, AOSD 2011, 21-25 Mar 2011, Porto de Galinhas, Brazil. pp. 285-299. ACM. ISBN 978-1-4503-0605-8
- 4 Malakuti Khah Olun Abadi, S. and Aksit, M. and Bockisch, C.M. (2011) Runtime Verification in Distributed Computing. Journal of Convergence, 2 (1). pp. 1-10. ISSN 2093-7741
- 5 Malakuti Khah Olun Abadi, S. and Aksit, M. and Bockisch, C.M. (2011) Distribution-Transparency in Runtime Verification. In: Ninth IEEE International Symposium on Parallel

- and Distributed Processing with Applications Workshops, ISPAW 2011, 26-28 May 2011, Busan, Korea. pp. 328-335. IEEE Communications Society. ISBN 978-1-4577-0524-3
- 6 Malakuti Khah Olun Abadi, S. and Bockisch, C.M. and Aksit, M. (2009) Applying the Composition Filter Model for Runtime Verification of Multiple- Language Software. In: The 20th annual International Symposium on Software Reliability Engineering,ISSRE 2009, 16-19 Nov 2009, Mysore, India. pp. 31-40. IEEE Computer Society. ISBN 978-0-7695-3878-5
 - 7 Ciraci, S. and Malakuti Khah Olun Abadi, S. and Katz, S. and Aksit, M. (2010) Checking the Correspondence Between UML models and Implementation. In: 1st International Conference on Runtime Verification, 1-4 Nov 2010, Malta. pp. 198-213. Lecture Notes in Computer Science 6418. Springer Verlag. ISSN 0302-9743 ISBN 978-3-642-16611-2
 - 8 Sozer, Tekinerdogan and Aksit. Optimizing Decomposition of Software Architecture for Local Recovery" for publication in Software Quality Journal.
 - 9 de Roo, A.J. and Sözer, H. and Aksit, M. (2011) Runtime Verification of Domain-Specific Models of Physical Characteristics in Control Software. In: Fifth IEEE International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27-29 Jun 2011, Jeju Island, Korea. pp. 41-50. IEEE Computer Society. ISBN 978-0-7695-4453-3
 - 10 de Roo, A.J. and Sözer, H. and Aksit, M. (2009) An Architectural Style for Optimizing System Qualities in Adaptive Embedded Systems using Multi- Objective Optimization. In: Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture, 14-17 Sep 2009, Cambridge, UK. pp. 349-352. IEEE Computer Society. ISBN 978-1-4244-4985-9
 - 11 Noppen, J.A.R. and van den Broek, P.M. and Aksit, M. (2008) Software development with imperfect information. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 12 (1). pp. 3-28. ISSN 1432-7643

3.2 Programming systems that never stop


Joe Armstrong (ERICSSON – Stockholm, SE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Joe Armstrong

How can we program systems that never stop? Erlang (a programming language) and more specifically OTP (a set of libraries) was designed specifically for building fault-tolerant systems. The systems we have built in Erlang have been running for many years and during this time have been upgraded many times without significant loss of service. In this context "many years" means of the order of ten years and "many times" means hundreds to thousands of times. These are large systems servicing hundred to millions of simultaneous sessions. I'll talk about the architectures of such systems and how they are designed for dynamic code and service upgrade and how this works in practice.

3.3 Multi-quality Dynamic Auto-Tuning for Cyber-Physical Systems

Uwe Aßmann (TU Dresden, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Uwe Aßmann

Cyber-physical systems of the future are safety-critical. They need verification of qualities but also need to be highly adaptive.

This requires completely new software architectures, based on dynamic multi-quality auto-tuning. Auto-tuning is an optimization method from high-performance computing which adapts an algorithm towards system and context parameters.





For cyber-physical systems, we suggest to extend auto-tuning to multiple qualities, based on quality contracts [1].

References

- 1 S. Götz, C. Wilke, M. Schmidt, S. Cech, and U. Aßmann. Towards Energy Auto-Tuning. In Proceedings of First Annual International Conference on Green Information Technology (GREEN IT), 2010.

3.4 Claim Monitoring for Tackling Uncertainty in Adaptive Systems

Nelly Bencomo (INRIA, FR)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Nelly Bencomo

Joint work of Bencomo, Nelly; Sawyer Pete; Welsh Kris

There is an increasing need for software systems that are able to adapt dynamically to changes in their environment.



However, a challenging characteristic of self-adaptive systems is that of uncertainty; a full understanding of all the environmental contexts they will encounter at runtime may be unobtainable at design time. Thus assumptions may have to be taken that risk being wrong, and this may lead to problems at runtime. We have developed REAssuRE, which uses the concept of claims to explicitly represent such assumptions in goal models of the system. We define a semantics for claims in terms of their impact on how alternative goal operationalisation strategies satisfy the system's non-functional requirements or soft goals. The implementation of REAssuRE includes automatic claim value propagation and goal model evaluation, using in-memory representations of the goal models and associated claims. We have demonstrated how claims can be monitored to verify claims at run time, and how falsified claims can trigger principled adaptation. We have evaluated REAssuRE using an adaptive flood warning system.

References

- 1 Kristopher Welsh, Pete Sawyer, Nelly Bencomo. *Towards Requirements Aware Systems: Run-time Resolution of Design-time Assumptions*. E26th IEEE/ACM International Conference On Automated Software Engineering, ASE 2011, Kansas, USA, 2011

3.5 Models@RT to support Interoperability

Amel Bennaceur (INRIA, FR)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Amel Bennaceur

Interoperability is a fundamental challenge for today's extreme distributed systems. Indeed, the high-level of heterogeneity in both the application layer and the underlying middleware and the conflicting assumptions that each system makes about its running environment

hinder the successful interoperation of independently-developed systems. Many solutions that aggregate the disparate systems in a non-intrusive way have been proposed.

These solutions use intermediary software entities, called mediators, to interconnect systems despite disparities in their data and/or interaction models by performing the necessary coordination and translations while keeping them loosely-coupled. Creating mediators requires a substantial development effort and a thorough knowledge of the application-domain, which is best understood by domain experts. Moreover, the increasing complexity of today's distributed systems, sometimes referred to as Systems of Systems, makes it almost impossible to develop 'correct' mediators manually. Therefore, formal approaches are used to automatically synthesize mediators. The notion of mediator is further realized using emergent middleware.


In this context, run-time models are used to capture meta-information about the heterogeneous systems including their interfaces and associated behaviour.

This is supplemented by ontological information to enable semantic interoperability in given application domains. We examine the nature of such run-time models coupled with consideration of the supportive synthesis algorithms that use these models to generate the appropriate mediators in order ensure interoperability in highly-heterogeneous environments.

This work takes place in the CONNECT project, which also examines how the models are derived, and how the system can adapt to underlying changes in context or issues related to the performance or behaviour of the system.

3.6 Self-Evolution under the Hood

Walter Cazzola (Università di Milano, IT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Walter Cazzola

No system escapes from the need of evolving either to fix bugs or to add new features. System evolution becomes particularly a problem in case the system cannot be stopped. The approach to solve this problem that is steadily gaining ground in the last few years is to grant reflective capability to the system that will introspect on its own model to plan its evolution when an update is necessary and drive the changes accordingly on its model and code.

Such an approach requires to fill the existing gap between the code and the model (often outdated and too loose) and to have an adequate support by the underlying framework (either the operating system or the virtual machine) [1]. The former is typically addressed by model driven engineering or reverse engineering techniques; the solution to the latter instead relies always on ad hoc solutions since the support to dynamic adaptation is quite limited.

To this respect, we developed JavAdaptor [3], a DSU approach which allows us to flexibly update running systems in an unanticipated manner.

JavAdaptor is a framework built on Java to deeply support dynamic evolution of Java applications; it permits to add/remove fields/methods (schema changes) and code adaptation with only a minimal lose of performance and without stopping a running application and above all the running application does not need to be pre-prepared to be updated by JavAdaptor (no special hooks are necessary in the code).

JavAdaptor, as it is, represents the perfect back-end for self-evolution when cooperates with an evolutionary planner that feeds it with the changes to apply to the running system.


Apart from presenting JavAdaptor and its characteristics, in this contribution, we are going to discuss its integration into a reflective architecture (a previous work from us named RAMSES [2]) to enable self-evolution driven by changes on the application model.

References

- 1 Walter Cazzola. Cogito, Ergo Muto! In *Proceedings of the Workshop on Self-Organizing Architecture (SOAR'09)*, pages 1–7, Cambridge, United Kingdom, September 2009.
- 2 Walter Cazzola, Ahmed Ghoneim, and Gunter Saake. Software Evolution through Dynamic Adaptation of Its OO Design. In *Objects, Agents and Features: Structuring Mechanisms for Contemporary Software*, Lecture Notes in Computer Science 2975, pages 69–84. Springer, July 2004.
- 3 Mario Pukall, Christian Kästner, Walter Cazzola, Sebastian Götz, Alexander Grebhahn, Reimar Schöter, and Gunter Saake. JavAdaptor — Flexible Runtime Updates of Java Applications. *Software—Practice and Experience*, 2012.

3.7 Challenges for Models@Runtime


Amit K. Chopra (*University of Trento – Povo, IT*)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Amit K. Chopra

My interest in models@runtime stems from my work with Tropos-like requirements models. These principal abstractions of these models are actors and their goals and commitments. I imagine them all to be runtime artefacts: actors would act toward satisfying their goals for which they would have to potentially enter into commitments with other actors. In this sense, I see an actor’s goal model as serving essentially as the program for the actor. Although many challenges need to be addressed for this vision to become reality, it illustrates my take on models@runtime: introduce a new layer of abstraction (the model) with its own interpreter. The function of the interpreter is to translate model-level concepts into the appropriate operational ones. For example, goals may be operationalised into procedures, constraints, and so on. The technical challenges relate to understanding the concerns to be addressed in high-level models; judiciously selecting and formalizing the concepts required so as to not end up with a concept soup; and understanding the assumptions that can be made of the operational layer and the nature of the API with which its services may be invoked. The biggest challenge though remains in making sure that we are engineering systems to user requirements, no matter how high-level the abstractions we employ.

3.8 Urban-Scale, Dynamically Adaptable Systems

Siobhan Clarke (*Trinity College – Dublin, IE*)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Siobhan Clarke

Joint work of Clarke, Siobhan; Popescu, Razvan; Staikopoulos, Athanasios; Brennan, Shane; Fritsch, Serena; Groba, Christin; O’Toole, Eamonn; Hannon, Paula;


URL <http://www.dsg.scss.tcd.ie/>

My research interests are in design and programming models for dynamically adaptable systems. Systems of interest include large-scale, multi-layer systems, and distributed embedded

systems, including wireless sensor networks. In particular, I am investigating systems to support smart and sustainable cities, where resources, such as energy, transport and water, need to be optimised, and knowledge available from sensors and devices in the environment can be harnessed to support citizens' daily lives. Models at runtime will provide the view of the system and its environment that will enable analysis and triggering of appropriate adaptations to ensure the required optimisations occur in a timely manner. Research challenges include how to extract and filter the appropriate runtime data from the large scope of information available, and reason over that data in manner timely enough to effect appropriate adaptations.

3.9 A (Personal) Historical Perspective on Models@Run.time

Fabio M. Costa (Universidade Federal de Goiás, BR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Fabio M. Costa

Joint work of Costa, Fabio M.; Blair, Gordon S.; Clarke, Peter J.

I started looking into the use of models at runtime when I was doing research on adaptive middleware for my PhD thesis at Lancaster University back in the year 2000. Reflection was a nice way to do adaptive middleware, but despite the use of meta-object protocols, it lacked a good approach to structure the self-representation of the base-level system. Then we came across with work from Kerry Raymond's group at DSTC, Australia, which proposed the concept of meta-information management, basically an approach for maintaining MOF-like repositories of meta-data (actually structured in the form of models) that could be accessed at runtime in a distributed environment [1]. To me, that was the seed to start thinking about ways to effectively use middleware models at runtime, and in the year 2000 we published a paper on the combination of reflection and meta-information management for adaptive middleware [2]. In that paper we proposed the design time use of models to generate middleware configurations, and, at runtime, the use of these same models as the causally connected self-representation of the middleware components that was maintained by the reflective meta-objects for the purposes of dynamic adaptation. This approach was realized as part of a middleware architecture (and a series of prototypes) called Meta-ORB, which also allowed the update of the runtime model and the creation of new model elements as a result of reflection. Now, after six successful editions of the Models@Run.time workshop, which built a very active community around the theme, and after this very interesting Dagstuhl seminar, it is good to see how the work of this great community has contributed to the development of the field to a point where it is gaining momentum and is starting to have an impact on the practice of building, maintaining and evolving software in many different application domains.

From the perspective of our research, some challenges that lie ahead include finding ways to deal with the complexity of models and developing efficient and safe (i.e., consistency-preserving) techniques to manipulate the models while the system is running, as well as to maintain the causal connection link without interfering with the performance of the system. The need to support user-centric models (models developed by non-technical users) and the requirement to manage distributed models are important additional challenges.

Our current research is looking into new approaches to use models at the core of adaptive/reflective and autonomic middleware technologies. We are particularly interested in the use of models to simulate and verify dynamic adaptations before they are actually


applied to the system. We are also looking into the development of a generic framework for model-driven middleware, using concepts such as layered architectures with models at different levels of abstraction, separation of behavioural and structural models (i.e., the model of the middleware system vs. the model run by the system), direct execution of (structural) models via interpretation, and the realization of functionality specified in user models via the dynamic selection and combination of existing underlying service providers. This generic framework is defined at the meta-model level and can be instantiated to realize the approach in different application domains. The approach originated from existing work on the Communication Virtual Machine middleware [3], and we are now working towards applying it to the domains of smart micro-grids and ubiquitous computing environments.

References

- 1 S. Crawley, S. Davis, J. Indulska, S. McBride, K. Raymond. 1997. Meta Information Management. In Proceedings 2 nd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'97).
- 2 Fabio M. Costa and Gordon S. Blair. 2000. Integrating Meta-Information Management and Reflection in Middleware. In Proceedings of the International Symposium on Distributed Objects and Applications (DOA '00). IEEE Computer Society, Washington, DC, USA, 133-.
- 3 Yali Wu, Andrew A. Allen, Frank Hernandez, Robert B. France, and Peter J. Clarke. A Domain-Specific Modeling Approach to Realizing User-Centric Communication. Journal of Software Practice and Experience (SP&E). Accepted February 2011.

3.10 Models at RunTime for the Certification of Autonomous Assistive Robots

Kerstin I. Eder (University of Bristol, GB)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Kerstin I. Eder

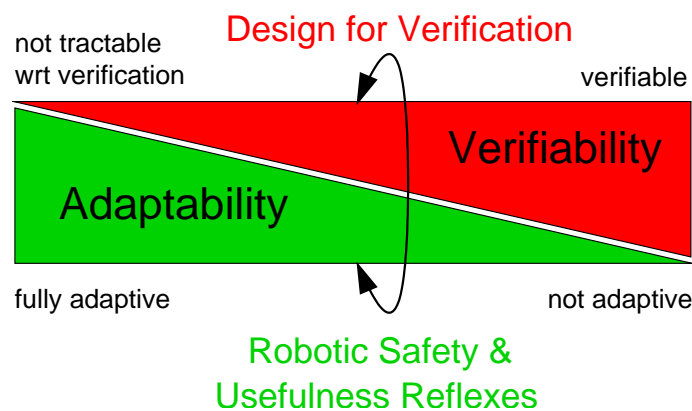
Human-assistive robots are machines designed to improve the quality of our lives by helping us to achieve tasks, e.g. a personal care robot helps a patient in a hospital during recovery. Such robots perform physical tasks within the personal space of a human, including shared manipulation of objects and even direct contact. While the actions a robot performs are largely the same, every single execution will be slightly different in detail from the last, e.g. a nursing assistant serving lunch provides differing degrees of support during a patient's recovery. This requires robots to adapt their behaviour to different situations. To be genuinely useful, some robots may need to be powerful and therefore are potentially dangerous.

The development of human-assistive robots introduces new ethical, legal and societal issues. One fundamental concern is whether human-assistive robots can be trusted by humans. Essential components of trustworthiness are *usefulness* and *safety*; both have to be demonstrated for humans to gain confidence in the trustworthiness of such robots and certainly before such robots pass product certification. *How this can be done is currently an open research question.*

At the Bristol Robotics Lab I am conducting research to understand the verification and certification needs arising out of the latest developments in Autonomous Assistive Robots. My aim is to develop design and verification techniques as well as overall methodologies

leading to certification to push the state of the art in this area. My first projects address this issue in the context of Human-Robot Interaction.

Design Verification is a process used to demonstrate the correctness of a design w.r.t. the requirements and specification. Techniques used for design verification rely upon the requirements, specification and design to fully define the functional behaviour of the system at the time of verification. Human-assistive robots, however, are designed to adapt their behaviour. There is typically no complete description of the entire range of possible behavioural adaptations, which renders traditional verification techniques useless in this context. Even if the full set of behavioural descriptions was available, it would be so large that traditional design-time verification techniques would quickly reach their limits. Fig. 1 shows *Adaptability vs. Verifiability* ranging from a system that is fully adaptive but has too many behaviours making verification no longer tractable (on the left) to a system that is easily verifiable but not adaptive.



■ **Figure 1** Adaptability vs. Verifiability

My research involves new techniques based on models at runtime to enable the verification of behavioural adaptations of human-assistive robots at runtime.

These models capture a consistent state of the agent initially (pre-verified warm start). Consistency is maintained at runtime. Based on the model the robot autonomously makes decisions. This enables the robot to dynamically adapt its behaviour to the situation such that it fulfils its requirements at all times.


The model contains a high-level description of the requirements and constraints that govern the functional (and non-functional) behaviour of the robot. It may be layered to reflect the robot's state, the state of the environment of the robot including the state of the humans/agents interacting with the robot, etc.

The key challenges for my research are currently:

- Establishing a set of high-level requirements that serve as invariant properties to be preserved during online behavioural adaptations, i.e. a set of invariant properties that form a “virtual cage”
- Online verification and validation (V&V) need to be tackled before certification: Can we design for online V&V? Can we build/adapt "correct-by-construction"?
- Finding a good compromise between the accuracy of the model to be a useful base for decision making and behavioural adaptation and computational complexity in terms of reasoning/processing based on the model.

3.11 Flexible Telecom systems that never stop

Paer Emanuelsson (Ericsson AB – Linköping, SE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Paer Emanuelsson

Telecom systems are large distributed systems with complex functionality that handle millions of users with real time performance. These systems evolve a lot during decades of years in operation. A system may only be unavailable for some minutes per year which makes the usual update procedures that stop the system impossible to use. Updates have to be made while the system is running.

We will briefly describe some different techniques used for achieving flexibility.

1. Parameters
2. Software update during runtime
 - a. patches, data cannot be updated
 - b. updates including data changes

1. Parameters

When it is known that there is need for variability a parameter is introduced.

By using parameters we avoid having lots of versions optimized for different markets and operators, which could lead to a maintenance nightmare. The biggest drawback with parameters is that changes have to be foreseen.

2a. Patches


Every software unit has a patch area, into which the patch can be loaded. The new software is activated by placing a jump to the new code.

2b. Updates including data changes

Software is run on two processors, and the second is a hot standby. While the first processor handles current traffic the new software is loaded onto the second processor. Runtime data is then transferred from the first processor to the second, which is specified in a data transformation language. This allows for arbitrary software changes. The data transformations necessary when data structures are changed can however be quite complex. The software doing the change from old to new version has to be designed for each change and the effort for specifying and performing the change can be larger than the development itself. It is also possible to have the new and old versions running simultaneously and transfer traffic only when needed – that is when old traffic does not end by itself after a period of waiting. Another way of handling data, which would allow for complex dependencies between different parts is to use a distributed real time database. Then transaction protected transformations can be performed.

3.12 Models@RT to support Interoperability

Nikolaos Georgantas (INRIA Le Chesnay, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Nikolaos Georgantas

Interoperability is a fundamental challenge for today's extreme distributed systems. Indeed, the high-level of heterogeneity in both the application layer and the underlying middleware and the conflicting assumptions that each system makes about its running environment

hinder the successful interoperation of independently-developed systems. Many solutions that aggregate the disparate systems in a non-intrusive way have been proposed.


These solutions use intermediary software entities, called mediators, to interconnect systems despite disparities in their data and/or interaction models by performing the necessary coordination and translations while keeping them loosely-coupled. Creating mediators requires a substantial development effort and a thorough knowledge of the application-domain, which is best understood by domain experts. Moreover, the increasing complexity of today's distributed systems, sometimes referred to as Systems of Systems, makes it almost impossible to develop 'correct' mediators manually. Therefore, formal approaches are used to automatically synthesize mediators. The notion of mediator is further realized using emergent middleware.

In this context, run-time models are used to capture meta-information about the heterogeneous systems including their interfaces and associated behaviour. This is supplemented by ontological information to enable semantic interoperability in given application domains. We examine the nature of such run-time models coupled with consideration of the supportive synthesis algorithms that use these models to generate the appropriate mediators in order to ensure interoperability in highly-heterogeneous environments.

This work takes place in the CONNECT project, which also examines how the models are derived, and how the system can adapt to underlying changes in context or issues related to the performance or behaviour of the system.

3.13 Models at Runtime for Adaptive and Self-managing Software

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

License  Creative Commons BY-NC-ND 3.0 Unported license
© Holger Giese

Software systems have to be continuously adapted to their changing requirements or environments. This is typically done by maintenance, while self-aware, context-aware, ultra-large-scale, or mission-critical software systems often have to be adapted during operation (self-adaptive software). Model-driven engineering and models at runtime play crucial roles in supporting maintenance, enabling self-adaptation, and integrating both, maintenance and self-adaptation.

In our talk, we outline our perspective on "models at runtime" as any models used on-line, i.e., internal to the running system, to represent running software, to represent the software's environment, or to manipulate or analyse any of the former two. Moreover, we sketch the benefits of models at runtime for maintenance, self-adaptation, and for an integrated setting of maintenance and self-adaptation. Considering different kinds of runtime models required for self-adaptation and self-adaptation activities as model operations, we discuss the role of a megamodel for specifying adaptation loops for self-adaptive software and operationalising the treatment of runtime models as means for monitoring, analysis, planning, and executing planned changes.

3.14 Multi-Quality Auto-Tuning with Contract Negotiation

Sebastian Götz (TU Dresden, DE)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Sebastian Götz

Main reference Sebastian Götz, Claas Wilke, Sebastian Cech and Uwe Aßmann. Architecture and Mechanism for Energy Auto-Tuning. To appear in: Sustainable Green Computing. Practices, Methodologies and Technologies. IGI Global. 2012

Building Software Architectures for self-optimizing, quality-aware systems, requires to explicitly express the non-functional concerns and how in particular they interleave using the notion of models at runtime. In our Multi-Quality Auto-Tuning approach we distinguish structure and variant models. The first are used to describe the types of software, hardware and users that constitute the system. The second are used to describe the actual state of the system and to prescribe target states of the system. To make non-functional concerns explicit we use contracts, which describe the relation between different elements of the system in terms of their non-functional requirements as well as provisions. All these models are and have to be used at runtime to reason about the system and thereby to identify the most efficient configuration.

The process of reasoning faces a trade-off between response time and intelligence.

The major challenge from my point of view is finding the right abstraction-level for the models and fast, intelligent reasoning/optimization techniques.

3.15 Runtime Monitoring of Java Bytecode with UML and OCL Models

Martin Gogolla (Universität Bremen, DE)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Martin Gogolla

Joint work of Hamann, Lars; Gogolla, Martin; Kuhlmann, Mirco


Main reference L. Hamann, M. Gogolla, M. Kuhlmann, “OCL-based Runtime Monitoring of JVM hosted Applications,” in Proc. 11th OCL Workshop (2011), ECEASST Vol. 44, 20 pages.

URL <http://journal.ub.tu-berlin.de/eceasst/article/view/623/677>

Implementations of object-oriented software usually contain a lot of technical classes. Thus, the central parts of an application, e.g., the business rules, may be hidden among peripheral functionality like user-interface classes or classes managing persistence. Our approach makes use of modern virtual machines and allows the developer to profile an application in order to achieve an abstract monitoring and quality assurance of central application components during runtime. We represent virtual machine bytecode in form of a so-called platform-aligned model (PAM) comprising OCL invariants and pre- and postconditions. We show a prototype implementation as an extension of the UML and OCL tool USE.

3.16 Models@run.time for the proactive detection of QoS Problems

Lars Grunske (TU Kaiserslautern, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Lars Grunske


Models@run.time are currently being used to evaluate quality demands and are especially suitable for performance, reliability, safety, and availability properties [2]. Our research is concerned with providing dedicated monitoring approaches to align the models with the running system [1,3]. Complemented with time-series analysis and change point detection techniques, these monitoring approaches allow for dynamic detection and proactive resolution of quality problems in modern software systems.

References

- 1 A. Amin, A. Colman, and L. Grunske. Using Automated Control Charts for the Runtime Evaluation of QoS Attributes. In *Proceedings of the 13th IEEE International High Assurance Systems Engineering Symposium*. IEEE Computer Society, 2011.
- 2 R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.
- 3 L. Grunske. An effective sequential statistical test for probabilistic monitoring. *Information and Software Technology*, 53(3):190 – 199, 2011.

3.17 SM@RT: A Model Driven Framework for Runtime Software Architecture

Gang Huang (Peking University, CN)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Gang Huang

Runtime software architectures (RSA) are architecture-level, dynamic representations of running software systems, which help monitor and adapt the systems at a high abstraction level. The key issue to support RSA is to maintain the causal connection between architecture and system, ensuring that the architecture represents the current system, and the modifications on architecture cause proper system changes.

We developed a model-driven framework, called SM@RT (supporting models at runtime), to construct and maintain RSA in an automated way. SM@RT does four contributions to RSA: First, SM@RT gives a formal definition of RSA with a set of meta models and their relationships; Second, SM@RT automatically generates all codes implementing the causal connection between the system and RSA after developers define the software architecture model they preferred, the management model of the target runtime system, and the causal connection between them using the above meta models; Thirdly, SM@RT provides an incremental bidirectional transformation-based engine for synchronizing multiple RSA models; Fourth, SM@RT provides QVT as a default architecture manipulation language for reading, writing and analysing the RSA in a programmatic way.


SM@RT is compliant with MOF and QVT, the dominant standards of modelling technology, and then almost all artefacts produced with SM@RT are reusable and integrable, which reduce much engineering work in practice.

SM@RT has been experimented on diverse systems, from JEE enterprise systems (ECPeRF, JPS and Rubis on JOnAS, JBoss and PKUAS), Java desktop systems (Eclipse GUI and XML Parser), to mobile systems (Android and PLASTIC/Window Mobile).

The future work of SM@RT will focus on seeking killer applications of models at runtime, in which the management is the core challenge (e.g. cloud) or the online construction and evolution are key enablement for business (e.g. internet of things or cyber-physical-systems).

3.18 EAGLE: Engineering softwAre in the ubiquitous Globe by Leveraging uncErtainty

Paola Inverardi (Università di L'Aquila, IT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Paola Inverardi

Joint work of Inverardi, Paola; Marco Autili; Vittorio Cortellessa; Davide Di Ruscio; Patrizio Pelliccione; Massimo Tivoli


Main reference M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, M. Tivoli, "EAGLE: engineering software in the ubiquitous globe by leveraging uncErtainty," in Proc. of 19th ACM SIGSOFT Symp./13th European Conf. on Foundations of Software Engineering (ESEC/FSE '11), pp. 488–491, 2011.

URL <http://dx.doi.org/10.1145/2025113.2025199>

In the next future we will be surrounded by a virtually infinite number of software applications that provide computational software resources in the open Globe. Users will be keen on producing their own piece of software, by also reusing existing software, to better satisfy their needs, therefore with a goal oriented, opportunistic use in mind. The produced software will need to be able to evolve, react and adapt to a continuously changing environment, while guaranteeing dependability. The strongest adversary to this view is the lack of knowledge on the software's structure, behaviour, and execution context. Despite the possibility to extract observational models from existing software, a producer will always operate with software artefacts that exhibit a degree of uncertainty in terms of their functional and non functional characteristics. We believe that uncertainty can only be controlled by making it explicit and by using it to drive the production process itself. In this paper, we introduce a novel paradigm of software production process that explores available software and assesses its degree of uncertainty in relation to the opportunistic goal G, assists the producer in creating the appropriate integration means towards G, and validates the quality of the integrated system with respect to G and the current context.

3.19 Models@RT to support Interoperability

Valerie Issarny (INRIA Le Chesnay, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Valerie Issarny

Interoperability is a fundamental challenge for today's extreme distributed systems. Indeed, the high-level of heterogeneity in both the application layer and the underlying middleware and the conflicting assumptions that each system makes about its running environment hinder the successful interoperation of independently-developed systems. Many solutions that aggregate the disparate systems in a non-intrusive way have been proposed.

These solutions use intermediary software entities, called mediators, to interconnect systems despite disparities in their data and/or interaction models by performing the necessary coordination and translations while keeping them loosely-coupled. Creating mediators requires a substantial development effort and a thorough knowledge of the application-domain, which is best understood by domain experts. Moreover, the increasing complexity of today’s distributed systems, sometimes referred to as Systems of Systems, makes it almost impossible to develop ‘correct’ mediators manually. Therefore, formal approaches are used to automatically synthesize mediators. The notion of mediator is further realized using emergent middleware.


In this context, run-time models are used to capture meta-information about the heterogeneous systems including their interfaces and associated behaviour.

This is supplemented by ontological information to enable semantic interoperability in given application domains. We examine the nature of such run-time models coupled with consideration of the supportive synthesis algorithms that use these models to generate the appropriate mediators in order ensure interoperability in highly-heterogeneous environments.

This work takes place in the CONNECT project, which also examines how the models are derived, and how the system can adapt to underlying changes in context or issues related to the performance or behaviour of the system.

3.20 Hyper-Agility: Handling Variability from Design-Time to Runtime

Jean-Marc Jezequel (IRISA – Rennes, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Jean-Marc Jezequel

Hyper-agility can be defined as the transposition of the Agile Manifesto at runtime to obtain systems able to adapt automatically to changes in their environment or their user requirements. We present an operational approach based on the use of models to separate concerns by abstracting specific aspects of reality. This approach has become quite popular in recent years for software analysis and design, relying on modelling languages of the UML family. Of course, the separation of concerns is of limited value if we cannot automatically reconstruct these concerns. The automatic composition of models makes it possible to effectively manage changes in the design or maintenance of software, especially in the context of product lines engineering. Beyond the resolution of this issue in the design phase, we show how the composition of models can also be used during the execution of a system to specify and manage dynamically adaptive software systems, here conceptualized as dynamic software product lines.

3.21 Multi-model adaptive control

Marin Litoiu (York University – Toronto, CA)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Marin Litoiu


An adaptive feedback loop in the cloud governs how and when resources (e.g., application server instances) are added to and/or removed from a cloud environment. The adaptive feedback loop can be implemented as a conventional control loop or as a set of heuristic rules.

In the control-theoretic approach, complex constructs such as tracking filters, estimators, regulators, and controllers are utilized. In the heuristic, rule-based approach, various alerts (e.g., events) are defined on instance metrics (e.g., CPU utilization), which are then aggregated at a global scale in order to make provisioning decisions for a given application tier.

This work provides an overview of our experiences designing and working with both approaches to construct an auto-scaler for simple applications. We enumerate different criteria such as design complexity, ease of comprehension, and maintenance upon which we form an informal comparison between the different methods. We conclude with a brief discussion of how these approaches can be used in the governance of resources to better meet a high-level goal over time.

3.22 Including humans in the Models@Run.Time loop


Brice Morin (SINTEF – Oslo, NO)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Brice Morin

Software systems are becoming more and more adaptive. While some software systems operate in a fully autonomic mode (typically, embedded systems), we foresee that the end-users have an important role to play in the new emerging adaptive systems such as Cyber-physical systems or Ambient Assisted Living systems. Such systems will still keep an certain part of autonomy, but their behaviour (core logic and adaptation logic) should be open, to some extent, to customization and personalization so that they can more easily be accepted by end-users. At design-time, designers only have a rough idea on how the system should behave and adapt, however the real requirements and needs will only be discovered at runtime, when actual end-users come into play. We believe that models@runtime could offer i) the right abstractions to end users to customize adaptive systems to their actual needs, and ii) assurance that their decisions will only be enacted if they do not jeopardize their (or the system's) safety.

3.23 Requirements for Models at Runtime


Pieter J. Mosterman (The MathWorks Inc. – Natick, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Pieter J. Mosterman

Model-Based Design has become critical to success in industry where embedded systems play an important role in the final products. For example, in automotive industry, the Chevy Volt includes over a million lines of code, which to a large extent have been automatically generated from models. While in a tightly controlled process of system design, models have proven themselves of unrivalled value, the requirements for the use of models at runtime are distinctly different. For example, models to be used for runtime adaptation must be defined for a much broader set of assumptions and these assumptions must be captured explicitly and precisely. Another example is the necessity to have models of potential system adaptations after deployment. The study of the idiosyncrasies that runtime use of models impose are the main objective of this work.

3.24 Models@Runtime for Self-Adaptation and Self-Protection

Liliana Pasquale (University of Limerick, IE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Liliana Pasquale


M@RT make possible to engineer systems that may adapt their behaviour in response to changes in the environment or in the requirements they are supposed to meet. As a matter of facts despite adaptation capabilities could be modelled with great detail at design time, anticipating all possible adaptations is not always feasible. To address this problem the requirements model of the system, which also includes the adaptation capabilities, is conceived as a run-time entity. My research applies requirements@runtime in two different application domains: adaptive service compositions and adaptive security.

For the first application domain, we use a live goal model to represent requirements (including adaptation capabilities) and track their changes. Note that changes in the requirements model cannot be learnt automatically, but must be always planned by the designer. These changes are propagated onto the running service instances (e.g., BPEL processes) through a suitable infrastructure, which is composed of a BPEL engine (to execute application instances), a set of data collectors (to monitor the environment and the applications execution state), monitors (to assess requirements) and adaptors (to modify the running application instances, for example, by changing their service components or by deploying new versions of the process).

On the other end, adaptive security is concerned on dynamically enabling a different set of security countermeasures when assets (to be protected) change unexpectedly, new threats arise, or undiscovered vulnerabilities are revealed. In our approach we relate the asset model to the requirements of the system, which are expressed through a goal model, and the objectives of an attacker, which are expressed through an anti-model. The asset, goals and threat models are conceived as a runtime entity and are used as input to build a causal network to analyse system security in different situations, and enable, when necessary, a set of countermeasures to mitigate the security threat. The relevant modifications that may arise at runtime (e.g., new or changing assets/threats/vulnerabilities) are dynamically discovered and tracked in the three models and consequently change the actual set of countermeasures that are enabled at runtime.

3.25 Automatic Synthesis of Behaviour Protocols for Composable Web-Services

Patrizio Pelliccione (Univ. degli Studi di L'Aquila, IT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Patrizio Pelliccione

Main reference A. Bertolino, P. Inverardi, P. Pelliccione, M. Tivoli, “Automatic synthesis of behavior protocols for composable web-services,” in Proc. of 7th Joint Meeting of the Europ. Software Engineering Conference and the ACM SIGSOFT Symp. on The Foundations of Software Engineering (ESEC/FSE '09), pp. 141–150, 2009.

URL <http://dx.doi.org/10.1145/1595696.1595719>

Web-services are broadly considered as an effective means to achieve interoperability between heterogeneous parties of a business process and offer an open platform for developing new composite web-services out of existing ones. In the literature many approaches have been proposed with the aim to automatically compose web-services. All of them assume that, along

with the web-service signature, some information is provided about how clients interacting with the web-service should behave when invoking it. We call this piece of information the web-service behaviour protocol.

Unfortunately, in the practice this assumption turns out to be unfounded.

To address this need, in this paper we propose a method to automatically derive from the web-service signature an automaton modelling its behaviour protocol. The method, called StrawBerry, combines synthesis and testing techniques. In particular, synthesis is based on data type analysis. The conformance between the synthesized automaton and the implementation of the corresponding web-service is checked by means of testing. The application of StrawBerry to the Amazon E-Commerce Service shows that it is practical and realistic.

3.26 Continuous Requirements Engineering for Self-Adaptive Software Systems

Anna Perini (CIT- FBK – Povo, IT) and Nauman Ahmed Qureshi (Fondazione Bruno Kessler – Trento, IT)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Anna Perini and Nauman Ahmed Qureshi

Joint work of Perini, Anna; Qureshi A. Nauman

Main reference N.A. Qureshi, I. Jureta, A. Perini, “Requirements engineering for self-adaptive systems: Core ontology and problem statement,” in 23rd Int’l. Conf. on Advanced Information Systems Engineering (CAiSE’11), Vol. 6741 of LNCS, pp. 33–47, Springer, 2011.

URL http://dx.doi.org/10.1007/978-3-642-21640-4_5

We recently proposed a formulation for the Requirements Problem of SAS (revisiting foundational work by Zave et. al., TOSEM’97, and a more recent one by Jureta et al., RE’08), as a dynamic problem (Qureshi at al., CAISE’11).

According to our formulation elements that contribute to the problem definition can change at run-time, for example, domain assumptions, context, resources, user’s preferences.

In our approach the requirements problem is represented through a goal- oriented model, which needs to be dynamically updated in order to reflect run- time changes in the requirements problem.

This motivates our interest in M@RT, and especially in continuous re-appraisal of requirements at run-time, which calls for effective and light-way methods for model representation and reasoning at run-time.

3.27 A Model-based Framework for Dynamic Adaptive Systems

Andres J. Ramirez (Michigan State University, US)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Andres J. Ramirez


Joint work of Ramirez, Andres J.; Cheng, Betty H.C.

A dynamically adaptive system (DAS) observes itself and its execution environment at run time to detect conditions that warrant adaptation. If an adaptation is necessary, then a DAS changes its structure and behaviour in order to continuously satisfy its requirements, even as its environment changes.

However, it is a challenging task to systematically and rigorously develop a DAS due to environmental uncertainty. In particular, it is often infeasible for a human to identify or enumerate all possible combinations of environmental conditions that a DAS might encounter throughout its lifetime. Nevertheless, a DAS must continuously satisfy its requirements despite the threat that this environmental uncertainty poses to its adaptation capabilities. For my dissertation research I have proposed a model-based framework that supports the specification, monitoring, and dynamic reconfiguration of a DAS to explicitly address uncertainty. Specifically, the proposed framework uses a goal-oriented requirements model to derive utility functions for requirements monitoring in a DAS. Using these functions, our framework then harnesses evolutionary computation techniques to identify interesting combinations of environmental conditions that may adversely affect the behaviour of a DAS and generates adaptations on-demand that not only transition the DAS to a target system configuration, but also preserve system consistency. We have demonstrated the capabilities of our model-based framework by applying it to two different case studies, one that involves an intelligent vehicle system that performs adaptive cruise control, lane keeping, and collision avoidance features, and another that involves the reconfiguration of a network of remote data mirrors.

3.28 The need of M@RT in Event-driven Process-centric Decision Support

David Redlich (Lancaster University, GB)

License  Creative Commons BY-NC-ND 3.0 Unported license
© David Redlich

Joint work of Redlich, David; Wasif Gilani

Main reference D. Redlich, W. Gilani, “Event-driven Process-centric Performance Prediction via Simulation,” in Business Process Management Workshops (BPM’11) Part I, vol. 99 of LNBIP, pp. 473–478, Springer, 2011.





URL http://dx.doi.org/10.1007/978-3-642-28108-2_46

Today’s fast, competitive and extremely volatile markets exert a great deal of pressure on businesses to react quicker against the changes, and sometimes even before the changes actually happen. A late action can potentially result in a legal compliance failure or violation of service level agreements (SLA’s). A business analyst needs to be notified before these failures and violations occur. We propose a simple approach that enables real-time and process-centric decision support in the form of performance prediction with the help of performance models at run time. To achieve this the ability of simulations to produce future-events, which are of the same type like the live-events generated by the really executed business process, is utilised. Live-events and simulated future-events can therefore be treated by a Complex-Event Processing (CEP) engine in the same way and parameter models representing the historic, current, and future performance of the business process can be easily computed or adapted.

Furthermore, we discussed further functional enhancements of the proposed architecture to support, for instance, automated business process management through self-adaptation.

3.29 Delta Modelling for Architectural Reconfiguration at Runtime

Bernhard Rumpe (RWTH Aachen, DE)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Bernhard Rumpe

Joint work of Haber, Arne; Rumpe, Bernhard; Schäfer, Ina

Given that logically or physically distributed systems such as cars, internet, clouds with attached mobiles etc. need to be designed, modelling languages for distributed architectures are inevitable. One such language, called MontiArc, resembles distributed communication of hierarchically decomposed components over explicit synchronous or asynchronous communication lines in the spirit of Automotive's function nets or UML's communication diagrams.

To describe variability of product lines in architectures, we extended this language for explicit modelling of deltas: structural changes of the architecture that extend functionality, signature, etc. but also allows to restructure communication lines.

So far, deltas have mainly been applied at design time to cut a concrete product from the possible configurations.

In this talk, we discuss how the Delta Modelling approach can be adapted to describe runtime re-configuration of software architectures.

While the modelling language basically remains the same, the underlying technical infrastructure, the way how to configure the architecture and in particular the dynamic re-configuration of the system during runtime significantly change.

3.30 Run-time Model Projections for Software Failure Prediction

Giordano Tamburrelli (Politecnico di Milano, IT)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Giordano Tamburrelli

Software is the driving engine of modern society. Most human activities—including critical ones—are either software enabled or entirely managed by software. As software is becoming ubiquitous and society increasingly relies on it, the adverse impact of unreliable or unpredictable software cannot be tolerated. Indeed, software systems must be able to evolve accordingly to their deployment environment to guarantee a seamless fulfilment of desired requirements and obtain a minimum downtime. In response to this challenge, current Software Engineering aims at designing self-adaptive systems able to react and reconfigure themselves minimizing human intervention and ideally guaranteeing a lifelong requirement fulfilment. To date, Software Engineering research in self-adaptive systems has produced promising initial results.

However, even if these findings provide an essential step towards a set of effective and efficient solutions for self-adaptation building these dependable systems is still unclear and requires further investigation. The most promising approaches to software self-adaptation rely on the introduction at run-time of software models (e.g., Markov models, automata, queuing networks, etc.), which represent the behaviour of the system under scrutiny. Such models are used to analyse and control the running implementation of the system with respect to desired requirements.

However, most current research efforts focus on enabling self-adaptation once the root event which triggered the adaptation already occurred. As a consequence, the system may

have been already compromised with a requirement violation. Designers must ensure that any critical requirement of the system continue to be satisfied before, during and after unforeseen changes. In current software engineering paradigms, systems do not anticipate events which may lead to failures, but only react accordingly to them. Let us consider for example the case in which a change itself is directly connected to a system requirement. In such scenarios the system requirement has been already violated because the occurred change or deviation implies a violation of the requirement. In this case self-adaptation boils down to a self-healing countermeasure to the occurred failure (i.e., reactive adaptation). Current approaches focus on this kind of adaptation and thus may occur into failures even if they adopt the proper countermeasures. They do not anticipate potentially dangerous changes. Run-time failure prediction, in our vision, is the enabling factor to tackle this problem.

Our current research challenge relies on the introduction of the novel concept of Run-Time Model Projection for Failure Prediction. By this we mean the ability of a software system at run-time to automatically forecast potentially dangerous events by reasoning on models which represent the expected future behaviour of the system (i.e., models projections) and thus work around predicted failures before their occurrence. This approach empowers self-adaptation capabilities of software systems obtaining an increased degree of dependability and availability.

Run-Time Model Projection for Failure Prediction is the ability of a system to anticipate failures by: (1) collecting at run-time relevant events occurred in the system internals as well as in its deployment environment, (2) performing quantitative and qualitative analyses of such events with their associated trends, and (3) building models representing a projection of the system behaviour in its near future. Indeed, by collecting these pieces of information and by building model projections it is possible to reason about the future compliance among the system and its requirement exploiting model-checking at run-time.

By analysing this compliance it is possible to predict future failures and put in action adaptation strategies that anticipate the incoming failure and work around it. The long-term vision of this approach is systems that are able to: (1) reason about potential changes, (2) anticipate them, and (2) reason on potential adaptations they might make. This process is driven by the events recorded in the internals of the system plus its environment and the account is given in terms of system requirements through model-checking.


We aim at contributing to the research on models at runtime for self-adaptive systems by building software models at run-time which do not represent the current behaviour of the system but its projection in the immediate future. Given these models, it is possible to automatically reason about the future fulfilment of desired requirements and potentially trigger proper adaptation strategies on time to steer the system and avoid a predicted failure. We investigate quantitative and qualitative algorithms which detect, record, correlate and analyse events occurred in the system and its environment. By reasoning on such events we estimate the future value of specific properties of the system (e.g., response time, incoming requests per unit of time, etc.) and synthesizes such estimates as updates for a software model of the system (initially produced at design-time). Indeed, we update the models at run-time coherently with these estimates obtaining model projections. By this we mean software models that aim at representing a prediction of the system in its near future. By reasoning on model projections it is possible to detect incoming failures before their occurrence. The research challenge tackled in this step relies on the accuracy and reliability of the model projections that is obtained with ad hoc techniques mainly based on statistical algorithms.

Furthermore, it is crucial in run-time model based approaches to have an effective and efficient reasoning engine. We plan to adopt as reasoning engines model-checking

tools, such as PRISM, which will be able to automatically prove the compliance of model projections against requirements. The introduction at run-time of model-checking is a promising approach for self-adaptive systems and offers several advantages. First of all, it allows a direct mapping of desired requirements against the model of the system. By this we mean that model checking allow not only to detect a requirement violation but also produce insights concerning the originating cause and about possible reconfigurations of the system which may solve the violation through what-if analyses. Secondly, model checking techniques rely on well-known and effective algorithms that are available off-shelf as powerful and open-source tools. Moreover, the recent research efforts on optimizing their performance in terms of execution time and space allow an efficient adoption at run-time, even if there is room for improvement.

3.31 Models@run.time in self-* systems

Matthias Tichy (Universität Paderborn, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Matthias Tichy

Using models@run.time is a key element to develop self-* applications. They can be used to assess the state of the system, to forecast system behaviour at runtime and to adapt system accordingly. We present applications of models@run.time in the area of self-healing distributed systems using graphs and graph transformations, self-adaptation of performance critical business information systems using performance annotated component models, self-adaptation in safety-critical embedded systems using failure propagation models and graph transformations and control engineering using models of the physics at run-time. Finally, we discuss the problem of ensuring the consistency of multiple models and dealing with uncertainty as key challenges in developing systems with models@run.time.

3.32 Safety Models@Run-Time in Open Adaptive Systems

Mario Trapp and Daniel Schneider (Fraunhofer IESE – Kaiserslautern, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Mario Trapp and Daniel Schneider

Joint work of Schneider, Daniel; Trapp, Mario

Main reference D. Schneider, M. Trapp, “A Safety Engineering Framework for Open Adaptive Systems,” in Proc. of the 5th IEEE Int’l Conf. on Self-Adaptive and Self-Organizing Systems, pp. 89–98, IEEE CS, 2011.


URL <http://dx.doi.org/10.1109/SASO.2011.20>

In embedded systems there is a clear trend towards open, context adaptive systems like cyber physical systems, ubiquitous computing, or ambient intelligence. All of these types of systems share the characteristic that neither their structure/behaviour nor the environment they run in are completely predictable at design time. On the one hand, this means that the systems must be able to adapt to these changing contexts at run time. On the other hand, this means that it is hardly possible to perform a complete quality assurance at design time. Particularly in the context of safety-critical systems this leads to new challenges. Models at run time provide a means to overcome this challenge. They enable the system to reason about its quality at run time and to adapt accordingly to assure the safety of the system in any given context situation. Regarding traditional safety engineering, there is a clear trend

to model-driven safety engineering and to modular certification based on models at design time. Combining these approaches with models at runtime to "safety models at runtime (SM@RT)" it is possible to shift certain safety engineering tasks to run time without losing the control over the systems' quality. Particularly, contracts based on safety certification models at run time seem to provide a very efficient means to ensure safety in open adaptive systems. In general, ensuring quality in adaptive systems poses several research challenges, M@RT build a sound basis to overcome these challenges.

3.33 A generalized view on models@run.time


Frank Trollmann (TU Berlin, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Frank Trollmann

The use of models@run.time is a suitable technique for the implementation of complex self-adaptive systems. The causal connection between the model and its system under study plays an important role in this approach. Although models@run.time are defined in to the scope of software engineering there are other systems which also utilize the causal connection. A comparison of models@run.time and such other approaches could lead to interesting new insights. For this purpose we introduce an abstracted understanding of models@run.time that can also be applied outside the scope of software engineering and makes these approaches comparable in the form of a mega model.

3.34 SmarterContext: Realizing Dynamic Context Management by Exploiting Context Models@Runtime

Norha Milena Villegas and Hausi Müller (University of Victoria, CA)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Norha Milena Villegas and Hausi Müller

Self-adaptive software systems adapt their behaviour to address changes in functional and non-functional requirements according to environmental conditions. Over the past decade the dynamic capabilities of these systems have proliferated and improved significantly. However, their real application is still limited due to a lack of mechanisms to address the uncertainty of the execution environment. The execution environment of an adaptive system is composed of external and internal context entities that can affect the desired system's behaviour. Thus, these entities and the interactions among them must be monitored to support decision making in the adaptation process. Nevertheless, as context information evolves over time, relevant entities and the corresponding monitoring requirements cannot be fully specified at design-time. Moreover, as the system is also evolving, monitoring requirements continuously change accordingly. Therefore, the effectiveness of self-adaptation is completely dependent on the adaptive capability of monitoring mechanisms to preserve context-awareness throughout the adaptation process. SmarterContext—our innovative approach to dynamic context management—exploits adaptive software techniques and RDF-based context models at runtime to optimize context-awareness and self-adaptation in service-oriented software systems.

SmarterContext is equipped with a self-adaptive context management infrastructure and an extensible context taxonomy based on the resource description framework (RDF) in order to support dynamic changes in context management strategies, through the deployment of new context management components to keep track of changes in relevant context at run-time. Our SmarterContext taxonomy includes a set of inference rules for supporting dynamic context representation and reasoning.

3.35 Models at Runtime for Adaptive and Self-managing Software

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

License  Creative Commons BY-NC-ND 3.0 Unported license
© Thomas Vogel

Software systems have to be continuously adapted to their changing requirements or environments. This is typically done by maintenance, while self-aware, context-aware, ultra-large-scale, or mission-critical software systems often have to be adapted during operation (self-adaptive software). Model-driven engineering and models at runtime play crucial roles in supporting maintenance, enabling self-adaptation, and integrating both, maintenance and self-adaptation.


In our talk, we outline our perspective on "models at runtime" as any models used on-line, i.e., internal to the running system, to represent running software, to represent the software's environment, or to manipulate or analyse any of the former two. Moreover, we sketch the benefits of models at runtime for maintenance, self-adaptation, and for an integrated setting of maintenance and self-adaptation. Considering different kinds of runtime models required for self-adaptation and self-adaptation activities as model operations, we discuss the role of a megamodel for specifying adaptation loops for self-adaptive software and operationalising the treatment of runtime models as means for monitoring, analysis, planning, and executing planned changes.

4 Working Groups

This section lists the abstracts of the break-out sessions. Each break-out group is in charge of a main section of the roadmap that is under preparation.

4.1 Group 1: Assurance Using Models@Run.Time for Self-Adaptive Systems

Betty H. C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pellicione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, Norha M. Villegas

License  Creative Commons BY-NC-ND 3.0 Unported license
© B.H.C. Cheng, K.I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H.A. Müller, P. Pellicione, A. Perini, N.A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, N.M. Villegas

Traditionally, software assurance is part of the software development process; i.e. assurance is a development-time concern. For software systems that change or evolve at runtime, as

in self-adaptive and self-managing systems, however, software assurance becomes a critical runtime concern. Continuous assurance throughout the entire software life-cycle provides unprecedented opportunities for monitoring, analyzing, guaranteeing, and predicting system properties and qualities throughout the operation of a software system. The fact that many variables that are free at design-time are bound at runtime provides new opportunities for verification and validation at runtime leading to assurance of critical system properties.


The goal of the assurance breakout group of Dagstuhl Seminar 11481 on Models@runtime was to identify research questions for software assurance using models at runtime to advance the state of the art in runtime assurance. Initially a set of valuable background starting points was established. The following is a summary of the most important research questions from the assurance breakout group:

- What development-time assurance methods and techniques (i.e., the entire spectrum from light-weight to heavy-weight approaches) and models (i.e., descriptive, prescriptive, constructive and predictive) readily extend to runtime?
- How do traditional assurance models and methods from domains such as performance, safety, and reliability extend to runtime?
- How can we extend reference models for self-adaptive and self-managing systems to include models and assurance at runtime (e.g., MAPE-K loop)?
- Can we leverage runtime assurance techniques from other disciplines (e.g., control theory)?
- Assuming models at runtime what reference architectures are appropriate for assurance at runtime?
- What are appropriate assurance reasoning techniques for different phases of the software life cycle (i.e., development, installation, load, and runtime) and how can assurance results from different life-cycle phases be combined to assure systems at runtime (e.g., incremental and compositional assurance)?
- How can we partition runtime assurance according the different types of runtime changes (e.g., dynamic context, changing requirements, or evolving models)?
- What are ideal applications to demonstrate the challenges and opportunities for assurance using models at runtime?

The research questions on runtime assurance will become part of a wider research roadmap dedicated to the development of models at runtime. One key challenge for this entire field is to develop effective training methods that equip traditional software engineers with the knowledge and skills to design and build software systems that change or evolve at runtime.

4.2 Group 2: Runtime Updating/Adaptation Mechanisms

Amel Bennaceur, Mehmet Aksit, Robert France, Walter Cazzola, Fabio M. Costa, Pär Emmanuelsen, Nikolaos Georgantas, Huang Gang, Pieter J. Mosterman, David Redlich, Alfonso Pierantonio, Giordano Tamburrelli, Thomas Vogel and Matthias Tichy

License  Creative Commons BY-NC-ND 3.0 Unported license


© A. Bennaceur, M. Aksit, R. France, W. Cazzola, F.M. Costa, P. Emmanuelsen, N. Georgantas, H. Gang, P.J. Mosterman, D. Redlich, A. Pierantonio, G. Tamburrelli, T. Vogel and M. Tichy

Runtime adaptation is becoming a fundamental property of today's complex open systems. Models at runtime (M@RT) systems offer promising abstractions, techniques and tools to manage the increasing complexity of adaptation and to handle the high dynamism of these systems. In this group, we discussed the issues and the solutions for runtime adaptation across different domains (e.g., fault-tolerance systems, dynamic tuning, energy optimization).

Despite the recent efforts from the software engineering community on M@RT, there is a lack of techniques and methods for the proper integration across domains, across layers (e.g., architectural, middleware, code), and across concerns (e.g., performance, security, reliability). We agreed on a conceptual architecture for adapting systems using M@RT and identified the different mechanisms used to: (1) create or update the system model according to the evolution of the system or to changes in its environment, (2) automatically reason on M@RT to produce appropriate adaptation strategies, (3) analyse and maintain different M@RT, each representing a specific view and/or abstraction, and (4) propagate changes from the model back to the system. Finally, we considered each mechanism, reviewed related work, and identified the open challenges in the form of short-term and long-term research goals. The output of this work leads to a fascinating set of research challenges both in terms of understanding the characteristics of M@RT and using them to create appropriate adaptation mechanisms in open settings, especially for the Internet of Things and Cyber-Physical systems. This is a vast and largely uncharted territory and we invite other researchers to join in the quest for suitable solutions for models at runtime.

4.3 Group 3: Uses and Purposes of M@RT Systems

Uwe Aßmann, Sebastian Götz, Jean-Marc Jezequel, Brice Morin and Mario Trapp


License  Creative Commons BY-NC-ND 3.0 Unported license
© U. Aßmann, S. Götz, J.-M. Jezequel, B. Morin and M. Trapp

The goal of this section is to understand how models@runtime are key enablers for modern software systems, to clarify their typical use cases and fundamental interests. Traditionally the development of software systems used to be split in distinct steps with a clear distinction between design activities and runtime execution. Safety critical embedded systems are for example designed and intensively validated at design time (e.g., using model checking) before they are actually deployed. At runtime, they have a predictable behaviour, time and resource consumption, which make it possible for certification bodies to approve these systems. The development of adaptive systems requires negotiating a trade off between adaptiveness and predictability. In many cases, system adaptation is restricted to a set of well-defined modes and reconfigurations defined at design-time. However, emerging types of systems—for example, cyber-physical systems—may need to adapt in ways that cannot be foreseen at design time and therefore require a higher degree of runtime adaptability, which hinders predictability. Hence, new approaches are needed to enable unanticipated adaptations while ensuring guarantees: This is, in our opinion, the ultimate purpose of models@runtime. The key characteristic of a M@RT system is its ability to project some aspects of the reality (e.g., its context, its behaviour, its goals) to the modelling space in order to enable tractable decision-making that produces decidable plans. This is basically separation of concerns applied in a disciplined way at runtime, and to some extent, is analogous to how human thinking works. This enables systems to reason about alternatives to reach their goals and to determine consequences of particular actions, rather than just learn and react (i.e., display animal-like behaviour) as is done in classical systems. This requires that the system be able to justify why a certain decision was made or not. The key advantage of M@RT systems over reflective software systems is decidability and tractability, which can reconcile users, domain experts, engineers (aware of the obvious need for runtime adaptivity), with certification bodies (which need stringent guarantees). Based on our recent experiences, we propose a

generic Reference Architecture (RA), which can be instantiated to cope with the specificities of different domains.

4.4 Group 4: Living With Uncertainty In the Age of Runtime Models

Nelly Bencomo, Amit K. Chopra, Siobhan Clarke, Holger Giese, Paolo Inverardi, Valerie Issarny, Liliana Pasquale, and Andres J. Ramirez

License  Creative Commons BY-NC-ND 3.0 Unported license
© N. Bencomo, A.K. Chopra, S. Clarke, H. Giese, P. Inverardi, V. Issarny, L. Pasquale, and A.J. Ramirez

Systems that operate in dynamic environments are inherently executing in situations of uncertainty. Uncertainty arises from multiple sources, such as the environment, the system itself, and different stakeholders, for example, end users, society including regulatory bodies, and others. Uncertainty emerges when there is a difference between useful information that exists in, for example, the original system and information available in the runtime model at a certain point in time. A runtime model provides an abstraction of information of interest to and in the executing system. It provides a knowledge base for decision-making related to behaviour optimized as far as possible within the bounds of some uncertainty. A runtime model to represent the uncertainty of interest can be specified at design time according to the possible goals of the system. At runtime, a monitor and analyser will instrument and refine information in the runtime model, which may reduce the level of uncertainty. A planner and executor must be able to deal with remaining uncertainty in a robust manner. In this group, we discussed research questions relating to the treatment of uncertainty with models at runtime.

Participants

- Mehmet Aksit
University of Twente, NL
- Joe Armstrong
ERICSSON – Stockholm, SE
- Uwe Aßmann
TU Dresden, DE
- Nelly Bencomo
INRIA, FR
- Amel Bennaceur
INRIA, FR
- Walter Cazzola
Università di Milano, IT
- Betty H. C. Cheng
Michigan State University, US
- Amit K. Chopra
University of Trento – Povo, IT
- Siobhán Clarke
Trinity College – Dublin, IE
- Fabio M. Costa
Univ. Federal de Goiás, BR
- Kerstin I. Eder
University of Bristol, GB
- Pär Emanuelsson
Ericsson AB – Linköping, SE
- Robert B. France
Colorado State University, US
- Nikolaos Georgantas
INRIA Le Chesnay, FR
- Holger Giese
Hasso-Plattner-Institut –
Potsdam, DE
- Sebastian Götz
TU Dresden, DE
- Martin Gogolla
Universität Bremen, DE
- Lars Grunske
TU Kaiserslautern, DE
- Gang Huang
Peking University, CN
- Paola Inverardi
Università di L'Aquila, IT
- Valerie Issarny
INRIA Le Chesnay, FR
- Jean-Marc Jezequel
IRISA – Rennes, FR
- Marin Litoiu
York University – Toronto, CA
- Brice Morin
SINTEF – Oslo, NO
- Pieter J. Mosterman
The MathWorks Inc. –
Natick, US
- Hausi A. Müller
University of Victoria, CA
- Liliana Pasquale
University of Limerick, IE
- Pankesh Patel
INRIA, FR
- Patrizio Pelliccione
Univ. degli Studi di L'Aquila, IT
- Anna Perini
CIT- FBK – Povo, IT
- Alfonso Pierantonio
Univ. degli Studi di L'Aquila, IT
- Nauman Ahmed Qureshi
Fondazione Bruno Kessler –
Trento, IT
- Andres J. Ramirez
Michigan State University, US
- David Redlich
Lancaster University, GB
- Bernhard Rumpe
RWTH Aachen, DE
- Daniel Schneider
Fraunhofer IESE –
Kaiserslautern, DE
- Giordano Tamburrelli
Politecnico di Milano, IT
- Matthias Tichy
Universität Paderborn, DE
- Mario Trapp
Fraunhofer IESE –
Kaiserslautern, DE
- Frank Trollmann
TU Berlin, DE
- Norha Milena Villegas
University of Victoria, CA
- Thomas Vogel
Hasso-Plattner-Institut –
Potsdam, DE

