

Model-driven Algorithms and Architectures for Self-Aware Computing Systems

Edited by

Samuel Kounev¹, Xiaoyun Zhu², Jeffrey O. Kephart³, and
Marta Kwiatkowska⁴

1 University of Würzburg, DE, samuel.kounev@uni-wuerzburg.de

2 VMware, Inc., US, xiaoyzhu@yahoo.com

3 IBM TJ Watson Research Center – Hawthorne, US, kephart@us.ibm.com

4 University of Oxford, GB, Marta.Kwiatkowska@cs.ox.ac.uk

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 15041 “Model-driven Algorithms and Architectures for Self-Aware Computing Systems”. The design of self-aware computing systems calls for an integrated interdisciplinary approach building on results from multiple areas of computer science and engineering, including software and systems engineering, systems modeling, simulation and analysis, autonomic and organic computing, machine learning and artificial intelligence, data center resource management, and so on. The Dagstuhl Seminar 15041 served as a platform to raise the awareness about the relevant research efforts in the respective research communities as well as existing synergies that can be exploited to advance the state-of-the-art, formulate a new research agenda that takes a broader view on the problem following an integrated and interdisciplinary approach, and establish collaborations between academia and industry.

Seminar January 18–23, 2015 – <http://www.dagstuhl.de/15041>

1998 ACM Subject Classification D.2 Software Engineering, D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification, D.2.5 Testing and Debugging, I.2 Artificial Intelligence, I.6 Simulation and Modelling

Keywords and phrases autonomic systems, self-adaptive, self-managing, model-driven, architecture-based, systems management, machine learning, feedback-based design

Digital Object Identifier 10.4230/DagRep.5.1.164

Edited in cooperation with Aleksandar Milenkoski (University of Würzburg, DE)


1 Executive Summary

Samuel Kounev

Jeffrey O. Kephart

Xiaoyun Zhu

Marta Kwiatkowska

License  Creative Commons BY 3.0 Unported license

© Samuel Kounev, Jeffrey O. Kephart, Xiaoyun Zhu, and Marta Kwiatkowska

Seminar Description

Self-aware computing systems are best understood as a subclass of autonomic computing systems. The term, autonomic computing, was first introduced by IBM in 2001. Expressing a concern that the ever-growing size and complexity of IT systems would soon become too



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

Model-driven Algorithms and Architectures for Self-Aware Computing Systems, *Dagstuhl Reports*, Vol. 5, Issue 1, pp. 164–196

Editors: Samuel Kounev, Xiaoyun Zhu, Jeffrey O. Kephart, and Marta Kwiatkowska



DAGSTUHL
REPORTS Dagstuhl Reports

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

difficult for human administrators to manage, IBM proposed a biologically-inspired solution. An analogy was drawn between the autonomic nervous system, which continually adjusts the heart and respiratory rates, pupil dilation, and other lower-level biological functions in response to conscious decisions made by individuals, and autonomic computing systems, which are envisioned as managing themselves in accordance with high-level objectives from humans.

In an effort to enlist the academic community in a worldwide effort to meet this grand challenge, Kephart and Chess laid out a vision of autonomic computing in an IEEE Computing article in 2003 [1]. The article postulated a multi-agent architecture for autonomic computing systems consisting of interacting software agents (called autonomic elements) that consume computational resources and deliver services to humans and to other autonomic elements, and used that architecture as a structure against which a diverse set of research challenges were defined. One of the major challenges from a scientific perspective was the definition of appropriate abstractions and models for understanding, controlling, and designing emergent behavior in autonomic systems. Many different components of IT systems could be autonomic elements – database management systems, load balancers, provisioning systems, anomaly detection system, etc. In addition to managing their own behavior in accordance with policies established by humans or other autonomic elements, they also manage their relationships with other autonomic elements.

The self-managing properties of autonomic computing systems, including self-optimization, self-configuration, self-healing and self-protection, are expected to arise not just from the intrinsic self-managing capabilities of the individual elements, but even more so from the interactions among those elements, in a manner akin to the social intelligence of ant colonies. Understanding the mapping from local behavior to global behavior, as well as the inverse relationship, was identified as a key condition for controlling and designing autonomic systems. One proposed approach was the coupling of advanced search and optimization techniques with parameterized models of the local-to-global relationship and the likely set of environmental influences to which the system will be subjected.

In the ensuing decade, there has been much research activity in the field of autonomic computing. At least 8000 papers have been written on the topic, and explicit solicitations for papers on autonomic computing can be found in the call for papers of at least 200 conferences and workshops annually, including the International Conference on Autonomic Computing (ICAC), now in its tenth year. The European government has funded autonomic computing research projects for several million euros via the FP6 and FP7 programs, and the US government has funded research in this field as well.

In a retrospective keynote at ICAC 2011, Kephart assessed the state of the field, finding through bibliometric analysis that progress in the field has been good but uneven [2]. While there has been a strong emphasis on self-optimization in its many forms, there have been considerably fewer works on other key autonomic properties such as self-configuration, self-protection and self-healing. An apparent reason for this imbalance is that benchmarks that quantify these properties and allow them to be compared across different systems and methods are still largely lacking. Another finding was that much work remains to be done at the system level. In particular, while there has been considerable success in using machine learning and feedback control techniques to create adaptive autonomic elements, few authors have successfully built autonomic computing systems containing a variety of interacting adaptive elements. Several authors have observed that interactions among multiple machine learners or feedback loops can produce interesting unanticipated and sometimes destructive emergent behaviors; such phenomena are well known in the multi-agent systems realm as

well, but insufficiently understood from a theoretical and practical perspective.

It is worth noting that there is a substantial sub-community within the autonomic computing field that applies feedback control to computing systems. FeBID (Feedback Control Implementation and Design in Computing Systems and Networks), a key workshop in this space, began in 2006 as a forum for describing advances in the application of control theory to computing systems and networks. In 2012, FeBID acquired a new name (Feedback Computing) to reflect a much broader and colloquial interpretation of “feedback”, in which the goals are no longer merely set points, and system models are not merely used to help transform or transduce signals, but may themselves be adapted through learning. The evolution of this sub-community of autonomic computing reflects a growing acceptance of the idea that, for an autonomic computing element or system to manage itself competently, it needs to exploit (and often learn) models of how actions it might take would affect its own state and the state of the part of the world with which it interacts.

Self-Aware Computing Systems

To understand how self-aware computing systems fit within the broader context of autonomic and feedback computing, we started with the following definition [4, 5] in the beginning of the seminar:

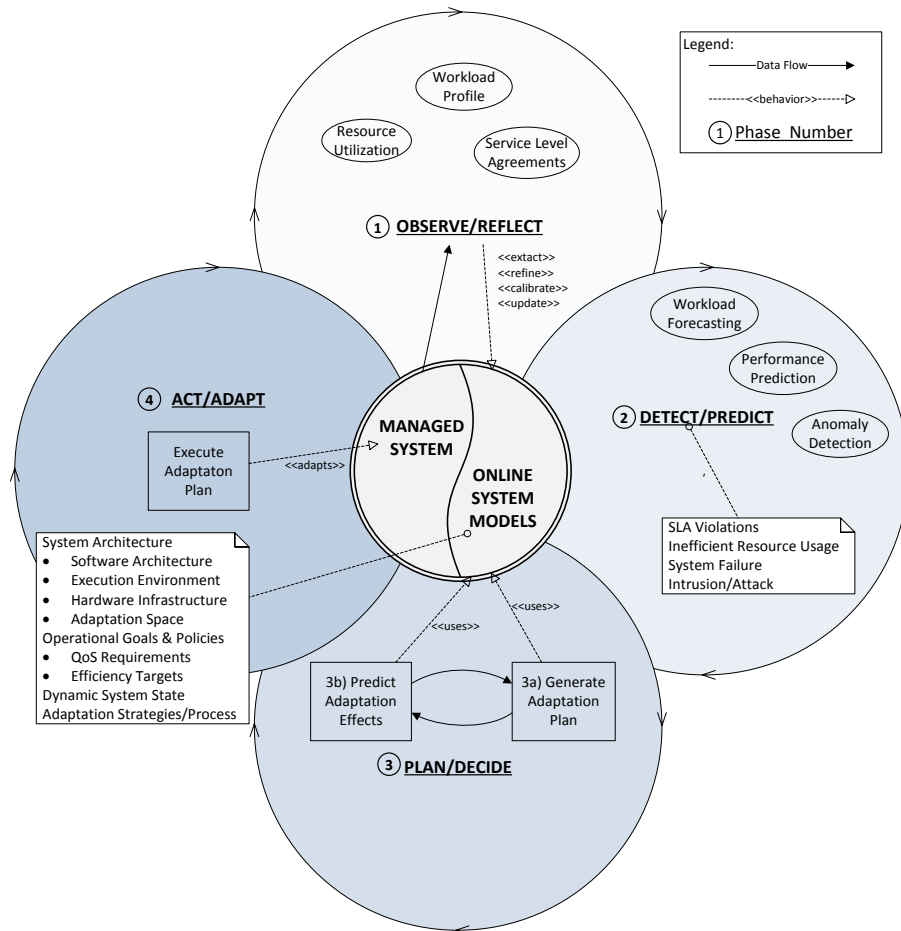
► **Definition 1.** A computing system is considered to be “self-aware” if it possesses, and/or is able to acquire at runtime, the following three properties, ideally to an increasing degree the longer the system is in operation:

- Self-reflective: Aware of its software architecture, execution environment, and hardware infrastructure on which it is running as well as of its operational goals (e.g., quality-of-service requirements, cost- and energy-efficiency targets),
- Self-predictive: Able to predict the effect of dynamic changes (e.g., changing service workloads) as well as predict the effect of possible adaptation actions (e.g., changing system configuration, adding/removing resources),
- Self-adaptive: Proactively adapting as the environment evolves in order to ensure that its operational goals are continuously met.

The three properties in the above definition are obviously not binary, and different systems may satisfy them to a different degree, however, in order to speak of “self-awareness”, all three properties must apply to the considered system.

To realize the vision of “self-aware” computing systems, as defined above, we advocated a holistic model-based approach where systems are designed from the ground up with built-in self-reflective and self-predictive capabilities, encapsulated in the form of online system architecture models. The latter are assumed to capture the relevant influences (with respect to the system’s operational goals) of the system’s software architecture, its configuration, its usage profile, and its execution environment (e.g., physical hardware, virtualization, and middleware). The online models are also assumed to explicitly capture the system’s operational goals and policies (e.g., quality-of-service requirements, service level agreements, efficiency targets) as well as the system’s adaptation space, adaptation strategies and processes.

Figure 1 presents our vision of a self-aware system adaptation loop based on the MAPE-K control loop [3] in combination with the online system architecture models used to guide the system adaptation at runtime. In the following, we briefly describe the four phases of the adaptation loop.



■ **Figure 1** Self-aware system adaptation loop.

Phase 1 (Observe/Reflect): In this phase, the managed system is observed and monitoring data is collected and used to extract, refine, calibrate, and continuously update the online system models, reflecting the relevant influences that need to be captured in order to realize the self-predictive property with respect to the system’s operational goals. In the context of this phase, expertise from software engineering, systems modeling and analysis, as well as machine learning, is required for the automatic extraction, refinement and calibration of the online models based on observations of the system at runtime.

Phase 2 (Detect/Predict): In this phase, the monitoring data and online models are used to analyze the current state of the system in order to detect or predict problems such as SLA violations, inefficient resource usage, system failures, network attacks, and so on. Workload forecasting combined with performance prediction and anomaly detection techniques can be used to predict the impact of changes in the environment (e.g., varying system workloads) and anticipate problems before they have actually occurred. In the context of this phase, expertise from systems modeling, simulation, and analysis, as well as autonomic computing and artificial intelligence, is required to detect and predict problems at different time scales during operation.

Phase 3 (Plan/Decide): In this phase, the online system models are used to find an adequate solution to a detected or predicted problem by adapting the system at runtime. Two steps are executed iteratively in this phase: i) generation of an adaptation plan, and ii) prediction of the adaptation effects. In the first step, a candidate adaptation plan is generated based on the online models that capture the system adaptation strategies, taking into account the urgency of the problem that needs to be resolved. In the second step, the effects of the considered possible adaptation plan are predicted, again by means of the online system architecture models. The two steps are repeated until an adequate adaptation plan is found that would successfully resolve the detected or predicted problem. In the context of this phase, expertise from systems modeling, simulation, and analysis, as well as autonomic computing, artificial intelligence, and data center resource management, is required to implement predictable adaptation processes.

Phase 4 (Act/Adapt): In this phase, the selected adaptation plan is applied on the real system at runtime. The actuators provided by the system are used to execute the individual adaptation actions captured in the adaptation plan. In the context of this phase, expertise from data center resource management (virtualization, cluster, grid and cloud computing), distributed systems, and autonomic computing, is required to execute adaptation processes in an efficient and timely manner.

Broader Notion of Self-aware Computing

As a result of the working group “Defining Self-aware Computing Systems”, a broader notion of self-aware computing was formulated:

► **Definition 2.** Self-aware computing systems are computing systems that:

1. *learn models* capturing *knowledge* about themselves and their environment (such as their structure, design, state, possible actions, and run-time behavior) on an ongoing basis and
2. *reason* using the models (for example predict, analyze, consider, plan) enabling them to *act* based on their knowledge and reasoning (for example explore, explain, report, suggest, self-adapt, or impact their environment)

in accordance with *higher-level goals*, which may also be subject to change.

For a detailed discussion of the interpretation of this definition, we refer the reader to Section 4.1.

Industrial Relevance

The envisioned novel algorithms and architectures for self-aware computing systems are of high relevance to the real-world problems faced by software developers and practitioners in the IT industry. Even though many of the specific problems have been researched upon within the aforementioned disciplines and communities, we believed the timing is right for adopting a broader integrated and interdisciplinary approach and exploiting synergies in the existing modeling and management approaches. The demand and the urgency for providing practical model-driven solutions to the described problems have never been higher, for the following reasons:

Large-scale, on-demand infrastructure: Although the cloud computing concept has been around for a long time, it wasn’t until the last few years did we see a wide availability and adoption of cloud computing platforms. Such platforms provide infrastructure-on-demand to business critical applications and high performance computing workloads. Such highly

dynamic, demand-driven environments make many existing automation schemes in computing systems inadequate, because they are mostly rule-based or heuristics-driven and cannot self-adapt to changes in both the infrastructure and the workloads.

Applications and workloads: The ever-increasing variety and complexity of modern applications and their workloads are placing more stress on computing systems and making many traditional management approaches obsolete. This is exacerbated by the extensive use of mobile devices and applications by an increasing population that produces new usage patterns and resource requirements.

Sensors and data: The numbers and types of sensors deployed in computing systems have never been greater, which lead to an explosion of runtime monitoring data that accurately capture the operating conditions of systems and software. Such data significantly enhance the chances for computing systems to Observe/Reflect (Phase 1) and to extract/refine/calibrate online system models that were difficult to learn otherwise, making a model-driven approach more feasible and reliable.

Need for automation: The IT industry is crying out ever so loud for automation technologies to help deal with the above challenges. Automation also helps reduce manual labor cost in management and administration and addresses the increasing gap between the number of skilled IT professionals and the industrial demand. There have been a growing number of startup companies that aim at developing automation solutions for capacity planning, provisioning and deployment, service level assurance, anomaly detection, failure/performance diagnosis, high availability, disaster recovery, and security enforcement. More research on modern-driven algorithms and architectures for self-aware computing can really feed into this new wave of innovations.

Organization of the Seminar

As inspired by the above described vision and approach towards its realization, we believed that the design of self-aware computing systems calls for an integrated interdisciplinary approach building on results from multiple areas of computer science and engineering including: i) software and systems engineering; ii) systems modeling, simulation and analysis; iii) autonomic and organic computing, machine learning and artificial intelligence; iv) data center resource management including virtualization, cluster, grid and cloud computing. This was the motivation of the research seminar. The list of invitees was carefully composed to provide a balance among these fields including both theoretical and applied research with participation from both academia and industry. We note that, in reality, each of the four mentioned communities is in fact comprised of multiple separate sub-communities although they have some overlap in their membership. While they can be seen as separate research communities, we consider them related in terms of their goals, with the difference being mostly in the specific focus of each sub-community and the employed scientific methods. The final participants of the seminar included representatives from each sub-community such that we cover the different relevant focus areas and scientific methodologies.

Achievements of the Seminar

This seminar has achieved its original goal of bringing together scientists, researchers, and practitioners from four different communities, including Software Engineering, Modeling and Analysis, Autonomic Computing, and Resource Management, in a balanced manner. The seminar program provided a basis for exchange of ideas and experiences from these different communities, offered a forum for deliberation and collaboration, and helped identify the technical challenges and open questions around self-aware computing systems. In summary, its achievements are mainly in the following two areas.

Identification of Synergies and Research Questions

By bringing together researchers from the above research fields and their respective communities, we avoid duplication of effort and exploit synergies between related research efforts.

During the seminar, we identified the following research questions and challenges that are of common interest to multiple communities:

- Design of abstractions for modeling quality-of-service (QoS) relevant aspects of systems and services deployed in dynamic virtualized environments. The abstractions should make it possible to capture information at different levels of detail and granularity allowing to explicitly model the individual layers of the system architecture and execution environment, context dependencies, and dynamic system parameters.
- Automatic model extraction, maintenance, refinement, and calibration during operation. Models should be tightly coupled with the system components they represent while at the same time they should abstract information in a platform-neutral manner.
- Efficient resolution of context dependencies including dependencies between the service deployment context and input parameters passed upon invocation, on the one hand, and resource demands, invoked third-party services, and control flow of underlying software components, on the other hand.
- Automatic generation of predictive models on-the-fly for online QoS prediction. The models should be tailored to answering specific online QoS queries. The model type, level of abstraction and granularity, as well as the model solution technique, should be determined based on: i) the type of the query (e.g., metrics that must be predicted, size of the relevant parts of the system), ii) the required accuracy of the results, iii) the time constraints, iv) the amount of information available about the system components and services involved.
- Efficient heuristics exploiting the online QoS prediction techniques for dynamic system adaptation and utility-based optimization.
- Novel techniques for self-aware QoS management guaranteeing service-level agreements (SLAs) while maximizing resource efficiency or minimizing energy cost.
- Standard metrics and benchmarking methodologies for quantifying the QoS- and efficiency-related aspects (e.g., platform elasticity) of systems running on virtualized infrastructures.

The above research questions and challenges were considered in the context of our holistic model-based approach and the self-aware system adaptation loop presented in the previous section. Answering these questions can help determine what system aspects should be modeled, how they should be modeled, how model instances should be constructed and maintained at runtime, and how they should be leveraged for online QoS prediction and proactive self-adaptation.

The online system models play a central role in implementing the four phases of the described system adaptation loop. The term “model” in this context is understood in a broad sense since models can be used to capture a range of different system aspects and modeling techniques of different type and nature can be employed (e.g., an analytical queuing model for online performance prediction, a machine learning model for managing resource allocations, a statistical regression model capturing the relationship between two different system parameters, a descriptive model defining an adaptation policy applied under certain conditions). At the seminar, we advocate a model-based approach that does not prescribe specific types of models to be employed and instead we use the term “online system models” to refer to all information and knowledge about the system available for use at runtime as part of the system adaptation loop. This includes both descriptive and predictive models.

Descriptive models describe a certain aspect of the system such as the system’s operational goals and policies (quality-of-service requirements and resource efficiency targets), the system’s software architecture and hardware infrastructure, or the system’s adaptation space and adaptation processes. Such models may, for example, be described using the Meta-Object-Facility (MOF) standard for model-driven engineering, heavily used in the software engineering community.

Predictive models are typically applied in three different contexts: i) to predict dynamic changes in the environment, e.g., varying and evolving system workloads, ii) to predict the impact of such changes on system metrics of interest, iii) to predict the impact of possible adaptation actions at runtime, e.g., application deployment and configuration changes. A range of different predictive modeling techniques have been developed in the systems modeling, simulation and analysis community, which can be used in the “detect/predict” phase of our adaptation loop, e.g., analytical or simulative stochastic performance models, workload forecasting models based on time-series analysis, reliability and availability models based on Markov chains, black-box models based on statistical regression techniques. Finally, models from the autonomic computing and machine learning communities can be used as a basis for implementing the “plan/decide” phase of our adaptation loop. Examples of such models are machine learning models based on reinforcement learning or analytical models based on control theory.

Two important goals of the seminar were to discuss the applicability of the various types of models mentioned above in the context of self-aware computing systems, and to evaluate the tradeoffs in the use of different modeling techniques and how these techniques can be effectively combined and tailored to the specific scenario. As discussed above, in each phase of the self-aware adaptation loop, multiple modeling techniques can be employed. Depending on the characteristics of the specific scenario, different techniques provide different tradeoffs between the modeling accuracy and overhead. Approaches to leverage these tradeoffs at runtime in order to provide increased flexibility will be discussed and analyzed.

Finally, the practical feasibility and associated costs of developing system architecture models was also extensively discussed. We also identified a major target of future research in the area of self-aware computing, which is to automate the construction of online system models and to defer as much as possible of the model building process to system runtime (e.g., the selection of a suitable model to use in a given online scenario, the derivation of adequate model structure by dynamically composing existing template models of the involved system components and layers, the parameterization of the model, and finally, the iterative validation and calibration of the model). Such an approach has the potential not only to help reduce the costs of building system architecture models, but also to bring models closer to the real systems and applications by composing and calibrating them at runtime based on

monitoring of the real observed system behavior in the target production environment when executing real-life operational workloads.

Impact on the Research Community

By bringing together the aforementioned four communities, the research seminar allowed for cross-fertilization between research in the respective area. It has raised the awareness of the relevant research efforts in the respective research communities as well as existing synergies that can be exploited to advance the state-of-the-art of the field of self-aware computing systems. The seminar has left to this Dagstuhl Report that provides an up-to-date point of reference to the related work, currently active researchers, as well as open research challenges in this new field. Given that a significant proportion of the proposed participants are from industry, the seminar also fostered the transfer of knowledge and experiences in the respective areas between industry and academia.

In addition to producing this joint report summarizing, we also found enough support and interest among the seminar participants to continue the collaboration through the following venues: i) writing a joint book to publish at Springer with chapter contributions from the seminar participants, ii) establish a new annual workshop on self-aware computing to provide a forum for exchanging ideas and experiences in the areas targeted by the seminar.

Overall, the seminar opened up new and exciting research opportunities in each of the related research areas contributing to the emergence of a new research area at their intersection.

References

- 1 Jeffrey O. Kephart, David M. Chess, “*The Vision of Autonomic Computing*,” in *IEEE Computer*, 36(1):41–50, 2003. DOI: 10.1109/MC.2003.1160055
- 2 Jeffrey O. Kephart, “*Autonomic Computing: The First Decade*”, in *Proc. of the 8th ACM Int’l Conf. on Autonomic Computing (ICAC’11)*, pp. 1-2, ACM, 2011. DOI: 10.1145/1998582.1998584
- 3 IBM Corporation, “*An Architectural Blueprint for Autonomic Computing*”, IBM White Paper, 4th Edition, 2006.
- 4 Samuel Kounev, “*Engineering of Self-Aware IT Systems and Services: State-of-the-Art and Research Challenges*”, in *Proc. of the 8th European Performance Engineering Workshop (EPEW’11)*, LNCS, Vol. 6977, pp. 10–13, Springer, 2011. DOI: 10.1007/978-3-642-24749-1_2
- 5 Samuel Kounev, Fabian Brosig, and Nikolaus Huber, “*Self-Aware QoS Management in Virtualized Infrastructures*”, in *Proc. of the 8th ACM Int’l Conf. on Autonomic Computing (ICAC’11)*, pp. 175–176, ACM, 2011. DOI: 10.1145/1998582.1998615

2 Table of Contents

Executive Summary

Samuel Kounev, Jeffrey O. Kephart, Xiaoyun Zhu, and Marta Kwiatkowska 164

Overview of Talks

Software Engineering for Self-Aware Computing <i>Samuel Kounev</i>	175
Modelling and Analysis (and towards Synthesis) <i>Marta Kwiatkowska</i>	175
Design-Time Analysis of Properties of Self-Adaptive Systems <i>Steffen Becker</i>	176
Helping Reflective Systems Build (Better) Self-models <i>Kirstie Bellman</i>	176
Analyzing Architecture-Based Self-Adaptation via Probabilistic Model Checking <i>Javier Camara</i>	177
Benchmark requirements for self-aware system <i>Lydia Y. Chen</i>	177
An Extensible Model-driven Architecture for Controlled Self-organisation <i>Ada Diaconescu</i>	178
Models in the middle and automated control synthesis: how to improve a software . . . engineer <i>Antonio Filieri</i>	179
Runtime Models for Dynamic Teams <i>Kurt Geihs</i>	179
Software Engineering for Self-Adaptive Systems & Self-Aware Computing <i>Holger Giese</i>	180
New Results on Property Specification Patterns <i>Lars Grunske</i>	180
Automated Synthesis of Service Choreographies <i>Paola Inverardi</i>	181
Self-* Datacenter Management for Business Critical Workloads <i>Alexandru Iosup</i>	181
Fairness in Data Stream Processing Under Overload <i>Evangelia Kalyvianaki</i>	182
The Descartes Modeling Language for Self-Aware Performance and Resource Management <i>Samuel Kounev</i>	183
Interplay of Design Time Optimization and Run Time Optimization <i>Anne Koziol</i>	183
Self-aware Computing in Industry 4.0 <i>Heiko Koziol</i>	183

Types of Computational Self-awareness	
<i>Peter Lewis</i>	184
Control Theory for Model-Based Software Engineering	
<i>Martina Maggio</i>	184
Janus: Optimal Flash Provisioning for Cloud Storage Workloads	
<i>Arif Merchant</i>	185
Projecting Disk Usage Based on Historical Trends in a Cloud Environment	
<i>Arif Merchant</i>	185
Evaluating Security Mechanisms in Dynamic Virtualized Environments	
<i>Aleksandar Milenkoski</i>	186
Adapting the Adaptation Logic	
<i>Felix Maximilian Roth</i>	186
Analysis of Mobile Offloading Strategies	
<i>Katinka Wolter</i>	187
Working Groups	
Working Group: “Defining Self-aware Computing Systems”	
<i>Samuel Kounev, Ada Diaconescu, Kirstie Bellman, Peter Lewis, Holger Giese, Javier Camara, Nelly Bencomo, Lukas Esterle, Henry Hoffmann, Hartmut Schmeck, Xin Yao, Sebastian Götz, and Andrea Zisman</i>	187
Working Group: “Quantification of Self-aware Systems: Metrics & Benchmarks”	
<i>Sara Bouchenak, Xiaoyun Zhu, Lydia Y. Chen, Evangelia Kalyvianaki, Eugenia Smirni, K. R. Jayaram, Alexandru Iosup, Kirstie L. Bellman, Anders Robertsson, Heiko Kozirolek, Steffen Becker, Christian Becker, Arif Merchant, and Aleksandar Milenkoski</i>	190
Working Group: “Generic Architectures for Collective Self-aware Computing Systems”	
<i>Kirstie Bellman, Nelly Bencomo, Ada Diaconescu, Lukas Esterle, Holger Giese, Sebastian Götz, Chris Landauer, Peter Lewis, and Andrea Zisman</i>	191
Working Group: “Benchmarking Self-aware Computing Systems”	
<i>Alexandru Iosup, Sara Bouchenak, Xiaoyun Zhu, Lydia Chen, Evangelia Kalyvianaki, K. R. Jayaram, Kirsty Bellman, Anders Robertson, Heiko Kozirolek, Steffen Becker, Eugenia Smirni, Arif Merchant, Aleksandar Milenkoski, and Felix Maximilian Roth</i>	193
Working Group: “Architecture and Reuse of Self-Aware Systems”	
<i>Anne Kozirolek, Christopher Landauer, Heiko Kozirolek, and Evangelia Kalyvianaki</i> .	194
Participants	196

3 Overview of Talks

3.1 Software Engineering for Self-Aware Computing

Samuel Kounev (University of Würzburg, DE)

License © Creative Commons BY 3.0 Unported license
© Samuel Kounev

In this talk, we present an overview of software engineering methods and models for self-aware computing. We first review projects in the area of self-aware computing, as well as related initiatives, discussing and contrasting existing notions of “self-aware computing”. We describe the most common types of models in the SE community, and how they are typically analyzed or otherwise used. Finally, we present an example of a modeling language (meta-model) specifically designed for self-aware performance and resource management in modern IT systems.

3.2 Modelling and Analysis (and towards Synthesis)

Marta Kwiatkowska (University of Oxford, GB)

License © Creative Commons BY 3.0 Unported license
© Marta Kwiatkowska

This presentation gave an overview of current research in model-based quantitative analysis, focusing on automated verification and the more recent shift towards synthesis. Probabilistic verification aims to establish the correctness of probabilistic models against safety, reliability and performance properties such as “the probability of an airbag failing to deploy within 0.02s” is less than 1 percent. A widely used tool for this purpose is the probabilistic model checker PRISM (www.prismmodelchecker.org). It is important to also consider the interaction between the system and its environment, which includes human users and other systems. The lecture focused on Markov decision process models, which model environmental uncertainty, and the corresponding verification algorithms for the temporal logic PCTL. It then introduced the synthesis problem, which aims to construct a strategy for the system so that a given quantitative goal, expressed in LTL, is satisfied. Since requirements may conflict, multi-objective synthesis was also discussed. Finally, the need for self-adaptation was discussed and the limitations of off-line verification in this regard, due to the need for the systems to take decisions based on current state and external events and to adapt/reconfigure so as to maintain the satisfaction of the objectives. Quantitative runtime verification based on the MAPE loop was put forward for this purpose. The lecture ended with an overview of future challenges. More information can be found in the following papers.

References

- 1 V. Forejt, and M. Kwiatkowska, G. Norman, D. Parker, “*Automated Verification Techniques for Probabilistic Systems*,” in Formal Methods for Eternal Networked Software Systems (SFM’11), LNCS, Vol. 6659, pp. 53–113, Springer, 2011.
- 2 R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, G. Tamburrelli, “*Dynamic QoS Management and Optimisation in Service-Based Systems*,” in IEEE Transactions on Software Engineering, 37(3):387–409, 2011.
- 3 R. Calinescu, C. Ghezzi, M. Kwiatkowska, R. Mirandola, “*Self-adaptive software needs quantitative verification at runtime*,” in Communications of the ACM, 55(9):69–77, 2012.

3.3 Design-Time Analysis of Properties of Self-Adaptive Systems

Steffen Becker (TU Chemnitz, DE)

License © Creative Commons BY 3.0 Unported license
© Steffen Becker

Joint work of Becker, Matthias; Lehrig, Sebastian; Becker, Steffen

Main reference M. Becker, S. Lehrig, S. Becker, “Systematically Deriving Quality Metrics for Cloud Computing Systems,” in Proc. of the 6th ACM/SPEC Int’l Conf. on Performance Engineering (ICPE’15), pp. 169–174, ACM, 2015.

URL <http://dx.doi.org/10.1145/2668930.2688043>

Analysing the properties of self-adaptive systems already at design time reduces the risks to implement systems with undesired behaviour that will be detected only in later stages or even in operation. Using an example based on Cloud Computing applications, the talk introduces metrics and analysis approaches we are currently researching. In particular, this includes new metrics for properties of self-adaptive systems. A simulation based approach can be used to analyse these properties using design time models.

3.4 Helping Reflective Systems Build (Better) Self-models

Kirstie Bellman (The Aerospace Corp. – Los Angeles, US)

License © Creative Commons BY 3.0 Unported license
© Kirstie Bellman

The purpose of this short talk was to stimulate discussion on the critical need to build self-modeling methods for self-aware systems. To have to provide all models to a system undercuts its ability to adapt and to operate within new environments (or to surprise us with new perceptions or understandings). A “self-aware” model is soon out of sync with its environment (especially new or dynamic ones) and itself if it cannot up-date and even, dramatically change its self-models (as well as environmental models), self-knowledge and behaviors. We described how we currently experiment with self-modeling in our CARS test bed (run by colleague Dr. Phyllis Nelson at Cal State Pomona) by giving the different robotic cars parameterized models which they “actively experiment” to fill in the values of the parameters for their own system. The results of their individualized self-models (e.g., how fast a specific car can go uphill or in grass or how well its different sensors work under different conditions, etc.) impact which strategies it will select to play different games in the test bed. We want to go beyond such templates to where in addition to some initial built-in information, the system has powerful methods for building models. One possibility is to use advanced data mining methods to discover new features and relationships both within the system and within its operational environments. In order for the system to build and evaluate self-models, it needs a way to test how its unique capabilities map into different operational environments and to learn the best way to integrate and to use its capabilities as well as learning its limitations.

Animals spend a great deal of energy and time exploring their environments and ‘playing’. This discussion emphasized that complex adaptive systems also need safe places to explore the interactions among their components, to try out new behaviors and to fail, and in so doing, to learn their best strategies for different operational conditions. It also allows the systems to look for gaps and errors in all of its models and essentially to practice ‘fire drills’, e.g., what to do when errors/problems/unexpected conditions. We then briefly discussed our current approaches to building the necessary language between the self-modeling system and its

human users and our first choices for mathematical methods that can support the development of self- models based on the data resulting from active experimentation (correlating one's own capabilities to different operational environments). We provided some references for our experiences in implementing computational reflection, one form of self-awareness, and self-adaptive architectures for different types of applications with our Wrappings approach (1989).

3.5 Analyzing Architecture-Based Self-Adaptation via Probabilistic Model Checking

Javier Camara (Carnegie Mellon University – Pittsburgh, US)

License © Creative Commons BY 3.0 Unported license
© Javier Camara

Joint work of Camara, Javier; Garlan, David; Moreno, Gabriel A.; Schmerl, Bradley;

Main reference J. Cámara, G. A. Moreno, D. Garlan, “Stochastic Game Analysis and Latency Awareness for Proactive Self-Adaptation,” in Proc. of the 9th Int’l Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS’14), pp. 155–164, ACM, 2014; pre-print available from project webpage.

URL <http://dx.doi.org/10.1145/2593929.2593933>

URL <http://acme.able.cs.cmu.edu/pubs/show.php?id=407>

Design decisions made during early development stages of self-adaptive systems tend to have a significant impact upon system properties (e.g., safety, QoS) during operation. However, understanding a priori the outcome of these decisions is difficult due to the high degree of uncertainty introduced by run-time changes (e.g., resource variability) in the execution context of such systems. Despite the fact that design decisions can be informed by artifacts (e.g., simulations, prototypes), these demand a significant effort to develop and tend to be available only during later stages of the development process. In this talk, I will describe an approach to enable the provision of assurances during early stages in the development cycle of self-adaptive systems. The approach is based on model checking of stochastic multiplayer games (SMGs), allowing developers to approximate the behavioral envelope of a self-adaptive system by analyzing best- and worst-case scenarios of alternative designs for self-adaptation mechanisms. Compared to other sources of evidence, such as simulations or prototypes, our approach provides developers with a preliminary understanding of adaptation behavior with little effort, and without the need to have any specific adaptation algorithms or infrastructure put in place. Moreover, this approach can be used complementarily with other sources of evidence and be used as a starting point to move from development-time to run-time assurances.

3.6 Benchmark requirements for self-aware system

Lydia Y. Chen (IBM Research GmbH – Zürich, CH)

License © Creative Commons BY 3.0 Unported license
© Lydia Y. Chen

There is a plethora of benchmarks developed by the academics and industries; however there is no such benchmark for evaluating the functionalities and performance of self-aware systems. Developing benchmarks suitable for self-aware system is deemed crucial and challenging. To such an end, one of key issues is how to produce representative workloads for self-aware

systems. For the ease of comprehension, let's consider the cloud datacenter as one of the live examples to illustrate such an issue. One would like to develop a self-aware resource management system for datacenter, so that the short-term resource provisioning and long-term capacity planning can be done in an autonomous fashion.

A valid benchmark specifically designed to evaluate the self-aware aspects shall first generate representative workload patterns. One could argue the definition of representative from the following perspectives. First, the workload patterns shall be identified from the real world applications and systems. Traces provided from the industries provide a rich base for such a need. Second, the workload patterns shall exhibit characteristics that can evoke the self-aware functions in the resource management systems. For example, some application workload has very flat pattern, e.g., constant arrival rate across control management horizon, and there exists no strong need for self-tuning resources. Thus, though such a workload pattern is commonly seen in some corporate datacenter, one may not incorporate such a pattern in benchmarks for the self-aware systems. Nevertheless, the second perspective depends very much on application and system which can have different requirements from each other. To summarize, we are in need to develop benchmarks for self-aware systems and their workload generation shall be representative of real world systems as well as inductive to self-awareness.

3.7 An Extensible Model-driven Architecture for Controlled Self-organisation

Ada Diaconescu (Telecom Paris Tech, FR)

License © Creative Commons BY 3.0 Unported license
© Ada Diaconescu

Joint work of Debbabi, Bassem; Diaconescu, Ada; Lalanda, Philippe
Main reference B. Debbabi, A. Diaconescu, P. Lalanda, "Controlling Self-Organising Software Applications with Archetypes," in Proc. of the IEEE 6th Int'l Conf. on Self-Adaptive and Self-Organizing Systems (SASO'12), pp. 69–78, IEEE CS, 2012.
URL <http://dx.doi.org/10.1109/SASO.2012.21>

Autonomic systems promise to help manage the increasing complexity of modern computing and communication systems. When introducing such autonomic functions, system designers must first determine the necessary balance between system control and runtime flexibility – between provable predictability (guarantees) and open adaptability (survival) of systems running in unpredictable environments.

Models provide formal means to define a system's objectives and runtime state – these offer necessary knowledge for controlling the system's self-* processes and constraining their outcomes. Dynamic extensibility and self-organisation provide means of opening and decentralising system control, in order to expand system adaptability, scalability and robustness.

We have been developing an approach that combines model-driven self-adaptation with decentralised self-organisation in order to construct autonomic systems that are highly adaptable, scalable and robust. A reusable and extensible meta-model and framework are available to help specify and develop autonomic systems in this manner. A prototype has been developed and experimented with in two application domains – smart home resource monitoring and e-health.

3.8 Models in the middle and automated control synthesis: how to improve a software. . . engineer

Antonio Filieri (University of Stuttgart, DE)

License © Creative Commons BY 3.0 Unported license

© Antonio Filieri

Joint work of Filieri, Antonio; Ghezzi, Carlo; Hoffmann, Henry; Leva, Alberto; Maggio, Martina;

Main reference A. Filieri, H. Hoffmann, M. Maggio, “Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees,” in Proc. of the 36th Int’l Conf. on Software Engineering (ICSE’14), pp. 299–310, ACM, 2014.

URL <http://dx.doi.org/10.1145/2568225.2568272>

Main reference A. Filieri, C. Ghezzi, A. Leva, M. Maggio, “Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements,” in Proc. of the 26th IEEE/ACM Int’l Conf. on Automated Software Engineering (ASE’11), pp. 283–292, IEEE, 2011.

URL <http://dx.doi.org/10.1109/ASE.2011.6100064>

Self-adaptation mechanisms empower software with the ability of withstanding unpredictable changes in its execution environment and handling uncertain knowledge about it. However, these mechanisms rarely provide formal guarantees about their effectiveness and dependability, limiting their applicability in practice.

Control theory has been concerned for decades with controlling of industrial plants aimed at the achievement of prescribed user goals. The mathematical grounding of control theory allows creating controllers effective and dependable by design, under reasonable assumptions about the environmental phenomena that may affect a system behavior.

Despite the conceptual similarity between controlling a plant and adapting software, the application of control theory to self-adaptive systems is very limited, with ad-hoc solutions hard to generalize. One of the main difficulties for a control theory of software is modeling a software behavior in a mathematical form suitable for control design.

This talk will recall two recent approaches for extracting mathematical models. First, the “model-in-the-middle” paradigm, exploiting established analytical quality models as pivot for filling the semantic gap between software design models and dynamic systems of equations. Second, a fully automated model inference and control synthesis technique considering software as a black box, which allow non experts to create software adaptation mechanisms with control theoretical guarantees.

3.9 Runtime Models for Dynamic Teams

Kurt Geihs (University of Kassel, DE)

License © Creative Commons BY 3.0 Unported license

© Kurt Geihs

Joint work of Geihs, Kurt; Niemczyk, Stefan;

Collaboration of autonomous actors (autonomous robots, intelligent agents etc.) requires a common understanding of the joint goals, context, and conditions, i.e. a common shared world model which is maintained at runtime and supports the self-awareness of the team. For example, in order to take concerted decisions on the allocation of tasks a team of soccer robots needs to know the position of the ball, the position of the opponents, the current strategy, etc. Here the required runtime model is known at design time and its structure is static. However, if we consider a more dynamic scenario, such as emergency response where the concrete environment and actors are not known in advance and several teams of heterogeneous autonomous robots as well as human rescue teams need to collaborate, then the question of building and maintaining a shared world model is a real challenge. The

basic research question that we are addressing is: How to create a shared world model for a dynamic group of heterogeneous actors in order to cooperate in unknown situations?

To establish and maintain a shared world model we create a runtime model reflecting the current context, the required information, and the currently available information sources. This model is composed dynamically at runtime by an ASP (Answer Set Programming) solver based on the semantic descriptions of the available system components. We use an OWL ontology to express the structure and semantics of information and system components. Autonomous actors share their semantic knowledge and integrate new elements into their own views. This enables the creation of a common understanding at runtime in order to establish a shared world model and to support the collaboration of autonomous actors in dynamic scenarios.

This research is part of a project cluster called NICER that focuses on ‘Networked Infrastructureless Cooperation for Emergency Response’. NICER is a joint research endeavour of TU Darmstadt, University of Kassel, and University of Marburg, supported financially by the state of Hesse in Germany.

3.10 Software Engineering for Self-Adaptive Systems & Self-Aware Computing


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

License  Creative Commons BY 3.0 Unported license
© Holger Giese

Self-Aware Computing is a very promising direction to enable systems to manage themselves in a more powerful manner than simple rule-based feedback loops can. In this presentation we will therefore first outline the overall landscape of software engineering for self-adaptive systems and our own work related to this. Then we will discuss which role self-aware computing can play in this landscape, but also identify which implications the landscape has for the required building blocks for self-aware computing solutions.

3.11 New Results on Property Specification Patterns

Lars Grunske (University of Stuttgart, DE)

License  Creative Commons BY 3.0 Unported license
© Lars Grunske

Effective system verification requires two important elements: a model that describes the operations and states of a (adaptive) system, and a set of properties that each implementation of this system must satisfy. Both specification need to be correct. However, for property specification its hard to guarantee that a given specification matches a software engineer’s intuition about the system in question. A solution is to use property specification patterns, which describes a generalized recurring system property type and provides a solution in form of a corresponding formal specification. This talk will present results from a unified framework [5, 1] for traditional [4], timed [3] and probabilistic [2] logic specifications.

References

- 1 M. Autili, L. Grunske, M. Lumpe, I. Meedeniya, P. Pelliccione, and A. Tang, “Property specification patterns wiki pages,” <http://ps-patterns.wikidot.com>, 2013.
- 2 L. Grunske, “Specification Patterns for Probabilistic Quality Properties,” in *Proceedings of the 30th International Conference on Software Engineering (ICSE’08)*. New York, NY, USA: ACM, 2008, pp. 31–40.
- 3 S. Konrad and B.H.C. Cheng, “Real-time Specification Patterns,” in *Proceedings of the 27th International Conference on Software Engineering (ICSE’05)*. New York, NY, USA: ACM, 2005, pp. 372–381.
- 4 M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, “Patterns in Property Specifications for Finite-State Verification,” in *Proceedings of the 21st International Conference on Software Engineering (ICSE’99)*. ACM, 1999, pp. 411–420.
- 5 M. Autili, L. Grunske, M. Lumpe, P. Pelliccione and A. Tang, “Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar,” in *IEEE Transactions on Software Engineering*, to appear .

3.12 Automated Synthesis of Service Choreographies

Paola Inverardi (University of L’Aquila, IT)

License © Creative Commons BY 3.0 Unported license
© Paola Inverardi

Joint work of Autili, Marco; Inverardi, Paola; Tivoli, Massimo;

Main reference M. Autili, P. Inverardi, M. Tivoli, “Automated Synthesis of Service Choreographies,” *IEEE Software*, 32(1):50–57, 2015.

URL <http://dx.doi.org/10.1109/MS.2014.131>

Future Internet research promotes the production of a distributed-computing environment that will be increasingly surrounded by a virtually infinite number of software services that can be composed to meet user needs. Services will be increasingly active entities that, communicating peer-to-peer, can proactively make decisions and autonomously perform tasks. Service choreography is a form of decentralized service composition that describes peer-to-peer message exchanges among participant services from a global perspective. In a distributed setting, obtaining the coordination logic required to realize a choreography is nontrivial and error prone. So, automatic support for realizing choreographies is needed. For this purpose, researchers developed a choreography synthesis tool. The Web extra at <http://www.di.univaq.it/marco.autili/synthesis/shortdemo/demo.htm> is a short demonstration of CHOReOSynt, a choreography synthesis tool.

3.13 Self-* Datacenter Management for Business Critical Workloads

Alexandru Iosup (TU Delft, NL)

License © Creative Commons BY 3.0 Unported license
© Alexandru Iosup

Joint work of Iosup, Alexandru; Epema, Dick; van Beek, Vincent; Fei, Lipu; Capota, Mihai; Shen, Siqi; Deng, Kefeng; Hegeman, Tim; Ghit, Bogdan; Yigitbasi, Nezhil

Multi-cluster datacenters, and, further, multi-datacenter infrastructure, are supporting increasing amounts and types of computer applications. Among the workloads of these datacenters, business-critical workloads, that is, workloads that support business decision and

intelligence, and that provide business and operational back-ends, are increasingly important (over 20% of the general IT load, according to an IDC study from 2010). Our goal is to build self-* resource managers for datacenters supporting high performance, efficient, and available business-critical and other services. We present in this talk new results in characterizing business-critical workloads running in multi-cluster multi-datacenters, and advances in scheduling such workloads using portfolio scheduling, scheduling by guessing (but not predicting) workload characteristics, and scheduling by enabling elasticity even for data-intensive workloads (for example, scheduling for elastic MapReduce frameworks).

References

- 1 Vincent van Beek, Siqi Shen, and Alexandru Iosup: Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. CCGRID 2015.
- 2 Bogdan Ghit, Nezh Yigitbasi, Alexandru Iosup, and Dick H. J. Epema: Balanced resource allocations across multiple dynamic MapReduce clusters. SIGMETRICS 2014.
- 3 Lipu Fei, Bogdan Ghit, Alexandru Iosup, and Dick H. J. Epema: KOALA-C: A task allocator for integrated multicluster and multicloud environments. CLUSTER 2014.
- 4 Kefeng Deng, Junqiang Song, Kaijun Ren, and Alexandru Iosup: Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds. SC 2013.

3.14 Fairness in Data Stream Processing Under Overload

Evangelia Kalyvianaki (City University – London, GB)

License © Creative Commons BY 3.0 Unported license

© Evangelia Kalyvianaki

Joint work of Kalyvianaki, Evangelia; Fiscato, Marco; Pietzuch, Peter;

Federated stream processing systems, which utilise nodes from multiple independent domains, can be found increasingly in multi-provider cloud deployments, large-scale grid systems and collaborative sensing applications. To pool resources from several sites and take advantage of local processing, submitted queries are split into query fragments, which are executed collaboratively by different sites. When supporting many concurrent users, however, queries may exhaust available processing resources, thus requiring constant load shedding. Given that individual sites have autonomy over how they allocate query fragments on their nodes, it is an open challenge how to ensure global fairness on processing quality experienced by queries in a federated scenario.

We describe THEMIS, a federated stream processing system for resource starved, multi-site deployments. It executes queries in a globally fair fashion and provides users with constant feedback on the experienced processing quality for their queries. THEMIS associates stream data with its source information content (SIC), a metric that quantifies the “value” of that data, as perceived by the user, based on the amount of source data used to generate it. We provide a distributed load shedding algorithm that implements fairness on the SIC values of result data. Our evaluation shows that, compared to a random shedding approach, THEMIS achieves a 33 percent more fair processing quality across queries as measured with the Jain’s index fairness metric. Our approach also incurs a low execution time overhead.

3.15 The Descartes Modeling Language for Self-Aware Performance and Resource Management

Samuel Kounev (University of Würzburg, DE)

License  Creative Commons BY 3.0 Unported license
© Samuel Kounev

The Descartes Modeling Language (DML) is a novel architecture-level language for modeling performance and resource management related aspects of modern dynamic software systems and IT infrastructures. Technically, DML is comprised of several sub-languages, each of them specified using OMG's Meta-Object Facility (MOF) and referred to as meta-model in OMG's terminology. The various sublanguages can be used both in offline and online settings for application scenarios like system sizing, capacity planning and trade-off analysis, as well as for self-aware resource management during operation. In this talk, we present an overview of DML and related tools for performance and resource management.

3.16 Interplay of Design Time Optimization and Run Time Optimization

Anne Koziolk (Karlsruhe Institute of Technology, DE)

License  Creative Commons BY 3.0 Unported license
© Anne Koziolk

The aim of designing self-aware systems is to enable the system to meet operational goals at runtime. A common goal is to maintain quality of service properties while minimizing costs of operation. We face an optimization problem. Software optimization is also a topic at design time, e.g. optimizing quality properties of software architectures.

In this talk, I would like to start a discussion on the interaction of design time optimization and runtime optimization. When designing self-aware systems, we should explore what are indeed completely unknown aspects so that the system needs to react in an autonomous way and what are parameters of the environment that vary, but that vary within known bounds so that we can optimize for different parameter combinations already at design time.

3.17 Self-aware Computing in Industry 4.0

Heiko Koziolk (ABB AG Research Center Germany – Ladenburg, DE)

License  Creative Commons BY 3.0 Unported license
© Heiko Koziolk

Industry 4.0 is a project in the high-tech strategy of the German government, which promotes the computerization of the manufacturing industry. The goal is the intelligent factory (Smart Factory), which is characterized by adaptability, resource efficiency and ergonomics as well as the integration of customers and business partners in business and value processes. Technological basis are cyber-physical systems and the Internet of Things. The talk outlines several application scenarios envisioned for Industry 4.0, where self-aware computing is expected to help improving production processes. Based on the Industry 4.0 vision, a number of challenges arise for self-aware computing, which are discussed in the talk.

3.18 Types of Computational Self-awareness

Peter Lewis (Aston University – Birmingham, GB)

License © Creative Commons BY 3.0 Unported license
© Peter Lewis

Joint work of Lewis, Peter R.; Faniyi, Funmilade; Bahsoon, Rami; Yao, Xin; Torresen, Jim; Chandra, Arjun
Main reference F. Faniyi, P. R. Lewis, R. Bahsoon, X. Yao, “Architecting Self-aware Software Systems,” in Proc. of the 11th Working IEEE/IFIP Conf. on Software Architecture (WICSA’14), pp. 91–94, IEEE, 2014.
URL <http://dx.doi.org/10.1109/WICSA.2014.18>

Novel computing systems are increasingly being composed of large numbers of heterogeneous components, each with potentially different goals or local perspectives, and connected in networks which change over time. Management of such systems quickly becomes infeasible for humans. As such, future computing systems should be able to achieve advanced levels of autonomous adaptive behaviour. In order for a system to effectively adapt itself in such a context, its ability to be self-aware becomes important.

There are several clusters of research in computer science and engineering which have used the term self-awareness explicitly. However, we are only now developing a common framework for describing the self-awareness capabilities of these systems. Questions remain about how to measure the benefits and costs that increased self-awareness brings.

In this talk, I shall begin by surveying definitions and current understanding of self-awareness in psychology. I will then describe how these concepts are being translated from psychology to the computing domain, and show how their explicit consideration may be beneficial in the engineering of adaptive computing systems. I will discuss how computational self-awareness may include knowledge of internal state, history, social or physical environment, goals, and perhaps even a system’s own way of representing or reasoning about these things.

3.19 Control Theory for Model-Based Software Engineering

Martina Maggio (Lund University, SE)

License © Creative Commons BY 3.0 Unported license
© Martina Maggio

Joint work of Maggio, Martina; Klein, Cristian; Papadopoulos, Alessandro Vittorio; Årzén, Karl-Erik
Main reference C. Klein, M. Maggio, K.-E. Årzén, F. Hernández-Rodríguez, “Brownout: Building More Robust Cloud Applications,” in Proc. of the 36th Int’l Conf. on Software Engineering (ICSE’14), pp. 700–711, ACM, 2014.
URL <http://dx.doi.org/10.1145/2568225.2568227>

Many software applications may benefit from the introduction of control theory at different stages of the development process. The requirements identification often translates directly into the definition of control goals. In fact, when these requirements are quantifiable and measurable, control theory offers a variety of techniques to complement the software design process with a feedback loop design process, that empowers the original software with self-adaptive capabilities and allows it to fulfill the mentioned quantifiable requirements.

The feedback loop design process consists in the definition of the “knobs”, what can be changed during the software life that affect the software behavior and the measurements of the goals. Control theory allows then to define models that describes the relationship between the values of the knobs and the measured values of the software behavior. These models are used to design decision loops and guarantee properties of the closed-loop systems.

In this talk I will briefly describe examples where model-based control allowed us to guarantee the satisfaction of specific properties, like synchronization between different nodes in a wireless sensor network and upper bounds on response times in a cloud application.

3.20 Janus: Optimal Flash Provisioning for Cloud Storage Workloads

Arif Merchant (Google Inc. – Mountain View, US)

License © Creative Commons BY 3.0 Unported license
© Arif Merchant

Joint work of Albrecht, Christoph; Merchant, Arif; Stokely, Murray; Waliji, Muhammad; Labelle, Francois; Coehlo, Nathan; Shi, Xudong; Schrock, Eric

Main reference C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, C. E. Schrock, “Janus: Optimal Flash Provisioning for Cloud Storage Workloads,” in Proc. of the USENIX Annual Technical Conf. (ATC’13), pp. 91–102, USENIX, 2013.

URL <https://www.usenix.org/conference/atc13/technical-sessions/presentation/albrecht>

Janus is a system for partitioning the flash storage tier between workloads in a cloud-scale distributed file system with two tiers, flash storage and disk. The file system stores newly created files in the flash tier and moves them to the disk tier using either a First-In-First-Out (FIFO) policy or a Least-Recently-Used (LRU) policy, subject to per-workload allocations. Janus constructs compact metrics of the cacheability of the different workloads, using sampled distributed traces because of the large scale of the system. From these metrics, we formulate and solve an optimization problem to determine the flash allocation to workloads that maximizes the total reads sent to the flash tier, subject to operator-set priorities and bounds on flash write rates. Using measurements from production workloads in multiple data centers using these recommendations, as well as traces of other production workloads, we show that the resulting allocation improves the hit rate by 47–76 percent compared to a unified tier shared by all workloads. Based on these results and an analysis of several thousand production workloads, we conclude that flash storage is a cost-effective complement to disks in data centers.

3.21 Projecting Disk Usage Based on Historical Trends in a Cloud Environment

Arif Merchant (Google Inc. – Mountain View, US)

License © Creative Commons BY 3.0 Unported license
© Arif Merchant

Joint work of Stokely, Murray; Mehrabian, Amaan; Albrecht, Christoph; Labelle, Francois; Merchant, Arif

Main reference M. Stokely, A. Mehrabian, C. Albrecht, F. Labelle, A. Merchant, “Projecting disk usage based on historical trends in a cloud environment,” in Proc. of the 3rd Workshop on Scientific Cloud Computing (ScienceCloud’12), pp. 63–70, ACM, 2012.

URL <http://dx.doi.org/10.1145/2287036.2287050>

Provisioning scarce resources among competing users and jobs remains one of the primary challenges of operating large-scale, distributed computing environments. Distributed storage systems, in particular, typically rely on hard operator-set quotas to control disk allocation and enforce isolation for space and I/O bandwidth among disparate users. However, users and operators are very poor at predicting future requirements and, as a result, tend to over-provision grossly. For three years, we collected detailed usage information for data stored in distributed file systems in a large private cloud spanning dozens of clusters on multiple continents. Specifically, we measured the disk space usage, I/O rate, and age of stored data for thousands of different engineering users and teams. We find that although the individual time series often have non-stable usage trends, regional aggregations, user classification, and ensemble forecasting methods can be combined to provide a more accurate prediction of future use for the majority of users.

We applied this methodology for the storage users in one geographic region and back-tested these techniques over the past three years to compare our forecasts against actual usage. We find that by classifying a small subset of users with unforecastable trend changes due to known product launches, we can generate three-month out forecasts with mean absolute errors of less than 12 percent. This compares favorably to the amount of allocated but unused quota that is generally wasted with manual operator-set quotas.

3.22 Evaluating Security Mechanisms in Dynamic Virtualized Environments

Aleksandar Milenkoski (University of Würzburg, DE)

License  Creative Commons BY 3.0 Unported license


© Aleksandar Milenkoski

Joint work of Milenkoski, Aleksandar; Vieira, Marco; Payne, Bryan D.; Antunes, Nuno; Kounev, Samuel; Avritzer, Alberto

The benefits of evaluation of security mechanisms (i.e., intrusion detection systems and access control systems) are manifold. For instance, one may compare multiple IDSEs in terms of their attack detection accuracy in order to deploy a mechanism which operates optimally in a given environment, thus reducing the risks of a security breach. Further, one may tune an already deployed mechanism by varying its configuration parameters and investigating their influence through evaluation tests. Many recent research works propose novel architectures of security mechanisms specifically designed to operate in dynamic virtualized environments. In this talk, we present recent advances in evaluating such mechanisms, including a method and a tool for generating workloads that contain virtualisation-specific attacks, and metrics that take elasticity, a feature of modern virtualised environments, into account. Finally, we present open challenges and requirements when it comes to evaluating security mechanisms deployed in self-adaptive virtualized systems with self-protecting features.

3.23 Adapting the Adaptation Logic

Felix Maximilian Roth (University of Mannheim, DE)

License  Creative Commons BY 3.0 Unported license

© Felix Maximilian Roth

Joint work of Roth, Felix Maximilian; Krupitzer, Christian; Becker, Christian

Self-adaptive systems are a response to the increasing complexity and size of information systems. They are able to adapt their behavior to changes in the context or system resources. Thus, they need to be self-aware in order to monitor their state. Furthermore, a self-adaptive system consists of managed resources and the adaptation logic. The managed resources realize functionality – they might be hardware or software components – whereas the adaptation logic controls the adaptation. For this purpose, the adaptation logic uses a feedback loop, such as IBM’s MAPE-loop. So far, many researchers have studied the field of adapting the managed resources. However, research in the adaptation of the adaptation logic is still at the beginning. In this talk I will briefly discuss why not only managed resources need to be adapted but also the adaptation logic and, furthermore, how this can be achieved.

3.24 Analysis of Mobile Offloading Strategies

Katinka Wolter (FU Berlin, DE)

License © Creative Commons BY 3.0 Unported license
 © Katinka Wolter
Joint work of Wolter, Katinka; Wu, Huaming; Wang, Qiushi;
Main reference Q. Wang, K. Wolter, “Reducing Task Completion Time in Mobile Offloading Systems through Online Adaptive Local Restart,” in Proc. of the 6th ACM/SPEC Int’l Conf. on Performance Engineering (ICPE’15), pp. 3–13, ACM, 2015.
URL <http://dx.doi.org/10.1145/2668930.2688041>

Mobile offloading migrates heavy computation from mobile devices to powerful cloud servers. It is a promising technique that can save energy of the mobile device while keeping job completion time low when cloud servers are available and accessible.

The benefit obtained by offloading greatly depends on whether it is applied at the right time in the right way. We use queueing models to minimize different metrics, such as the Energy-Response time Product (ERP) or a weighted sum of energy consumption and performance expressed in the Energy-Response time Weighted Sum (ERWS) metric. We consider different offloading policies (static and dynamic), where arriving jobs are processed either locally or remotely in the cloud. Offloading can be performed via different wireless technologies. We assume that the transmission techniques differ in energy requirement and speed.

We find that the dynamic offloading policy derived from the tradeoff offloading policy (TOP) outperforms other policies like the random selection of transmission channel by a significant margin. This is because the dynamic offloading policy considers the increase in each queue and the change in metric that newly arriving jobs bring in should they be assigned to that queue. The ERWS metric can be reduced further by considering either energy consumption or response time and it is minimal when optimising only energy consumption.

4 Working Groups

4.1 Working Group: “Defining Self-aware Computing Systems”

Samuel Kounev, Ada Diaconescu, Kirstie Bellman, Peter Lewis, Holger Giese, Javier Camara, Nelly Bencomo, Lukas Esterle, Henry Hoffmann, Hartmut Schmeck, Xin Yao, Sebastian Götz, and Andrea Zisman

License © Creative Commons BY 3.0 Unported license
 © Samuel Kounev, Ada Diaconescu, Kirstie Bellman, Peter Lewis, Holger Giese, Javier Camara, Nelly Bencomo, Lukas Esterle, Henry Hoffmann, Hartmut Schmeck, Xin Yao, Sebastian Götz, and Andrea Zisman

This working group discussed broader notions of self-aware computing systems accommodating the research communities focussing on the different aspects of self-aware computing (i.e., software and systems engineering, systems modeling, simulation and analysis, autonomic and organic computing, machine learning and artificial intelligence, data center resource management, and so on). The following definition of self-aware computing systems was formulated:

Self-aware computing systems are computing systems that:

1. *learn models capturing knowledge* about themselves and their environment (such as their structure, design, state, possible actions, and run-time behavior) on an ongoing basis and

2. *reason* using the models (for example predict, analyze, consider, plan) enabling them to *act* based on their knowledge and reasoning (for example explore, explain, report, suggest, self-adapt, or impact their environment) in accordance with *higher-level goals*, which may also be subject to change.

It is assumed that a self-aware system is built by an entity with some higher-level goals in mind. This entity may be a human (e.g., a developer) or a set of humans (e.g., a developer team), but it doesn't necessarily have to be. The entity that built the system may also be another computing system, at a higher-level, that generates code to build a new system for a given purpose.

The major distinctive characteristics of a self-aware computing system are: i) it must have the capability to learn models on an ongoing basis, capturing knowledge relevant to the purpose for which it is built, ii) it must be able to reason about this knowledge and act accordingly. Both the learning and reasoning part are driven by the system's goals, normally imposed by the entity that built the system. The goals are referred to as *higher-level goals* to emphasize that they are at a higher-level than the system itself and they are not under its control. Note that the system itself may generate its own goals (at lower levels) as part of its model learning and reasoning processes.

The term "model" is used here in a general sense and refers to any abstraction of the system and its environment that captures some knowledge and may be used for reasoning with respect to the system goals. In his general model theory, Stachowiak [1] identifies the following three features as essential for models: i) *mapping*: a model is always a model of some *original* (which can be a model itself), ii) *reduction*: a model always *abstracts* from the original by reflecting only a subset of its attributes, and iii) *pragmatic*: a model only replaces the original for a certain *purpose*. Usually, we further distinguish *descriptive* models that capture the originals as they are from *prescriptive* models that describe envisioned futures (planned originals). Descriptive models, in our context, describe a given system aspect that may be relevant with respect to the system's higher-level goals. We further distinguish *predictive models* that support more complex reasoning such as, for example, predicting the system behavior under given conditions or predicting the impact of a considered possible adaptation action.

Some examples of different types of models capturing various aspects that may be relevant in a given scenario include:

- a descriptive model capturing the system's resource landscape and software architecture and their performance-relevant parameters
- a descriptive model describing the system's possible adaptation actions (degrees-of-freedom at run-time)
- a prescriptive model describing how to act in a given situation (e.g., after a component failure)
- a descriptive model describing the system's goals and policies (e.g., service level agreements)
- a predictive statistical regression model capturing the influence of user workloads on the system resource consumption
- a predictive stochastic model allowing to predict the system performance for a given user workload and resource allocation
- a control theory model used to guide the system behavior

We stress that the term "learn" does not imply that all information based on which models are derived is obtained at system run-time. Models may be derived both from

static information provided by the entity that built the system, as well as from dynamic information that is gathered and maintained at run-time. Typically, a combination of both would be expected, for example, a system may be built with integrated skeleton models whose parameters are estimated using monitoring data collected at run-time. The model learning is expected to happen on an *ongoing basis* during operation meaning that models should be continuously refined and calibrated in order to better fulfill the purpose for which they are used.

The term “reason” in the definition refers to any type of model-based analysis that goes beyond applying explicitly programmed, i.e., hard-coded, rules or simple heuristics. Depending on the considered type of system and its respective goals, different types of reasoning may be relevant. For example, in the context of an IT system that has been designed to guarantee certain performance requirements, the following types of reasoning may be relevant:

- predict an IT system’s performance (e.g., response time) for a given workload and resource allocation (e.g., number of servers)
- forecast the system load (e.g., number of users or requests sent per unit of time) in a future time horizon
- predict the expected impact of a given system adaptation action (e.g., adding system resources) on the end-to-end system performance

An example of reasoning in the context of a cyber-physical system for traffic management may be to analyze the traffic situation in order to provide a recommendation which route to take for a given target destination.

By stressing the role of model learning and model-based reasoning, driven by higher-level goals, we distinguish the term self-aware computing from related terms such as autonomic computing or self-adaptive systems. Although, in most cases, it would be expected that the system uses the learned models to reason *and self-adapt* to changes in the environment, self-adaptation (often referred to as *self-expression* in this context) is not strictly required. In this way, we accommodate cases where all adaptation actions must be supervised and authorized by an entity outside of the system, such as the entity that built the system or a human system user. For example, in mission-critical cognitive computing applications, systems may provide recommendations on how to act, however, the final decision on what specific action to take is often made by a human operator.

References

- 1 H. Stachowiak, “*Allgemeine Modelltheorie*,” Springer, Wien, 1973.

4.2 Working Group: “Quantification of Self-aware Systems: Metrics & Benchmarks”

Sara Bouchenak, Xiaoyun Zhu, Lydia Y. Chen, Evangelia, Kalyvianaki, Evgenia Smirni, K. R. Jayaram, Alexandru Iosup, Kirstie L. Bellman, Anders Robertsson, Heiko Koziolok, Steffen Becker, Christian Becker, Arif Merchant, and Aleksandar Milenkoski

License © Creative Commons BY 3.0 Unported license

© Sara Bouchenak, Xiaoyun Zhu, Lydia Y. Chen, Evangelia Kalyvianaki, Evgenia Smirni, K. R. Jayaram, Alexandru Iosup, Kirstie L. Bellman, Anders Robertsson, Heiko Koziolok, Steffen Becker, Christian Becker, Arif Merchant, and Aleksandar Milenkoski


This working group discussed both metrics and benchmarks for self-aware systems. Most of the participants had a background in cloud computing or storage systems and viewed the topic from that perspective. A few participants also emphasized the broader perspective of self-aware systems outside of the cloud computing domain. The group first discussed criteria for the quantification of self-aware systems. Metrics need to span multiple non-functional properties, such as performance, timeliness, scalability, dependability, trustworthiness, security, safety, costs, adaptability and agility. Benchmarks need to be representative for a certain application domain, exhibit dynamic behavior (i.e., changing at runtime) and provide different failure scenarios that can trigger adaptations. The participants mentioned that several papers on good criteria for constructing benchmarks for cloud computing systems exist.

A few metrics for self-aware system were brought in a brainstorming session: it could be measured how much information is processed by a self-aware system to make informed adaptation actions. Measuring the adaptation overhead in terms of duration, cost, and quality could help comparing similar self-aware systems in an application domain. The “level of self-awareness” is another proposed metric, which is however is hard to quantify. Stability could be measured for systems with lots of adaptation actions. A more extended metric would be “number of surprising finding” made by a self-adaptive systems, i.e., suggestions or decisions for adaptation actions that were not obvious for human observers.

Only a few benchmarks for self-aware or self-adaptive systems were known to the group. But several participants pointed out that these were usually benchmarks originally not designed for self-adaptive systems, e.g., cloud computing benchmarks that were extended with dynamically changing properties at runtime. For example the community website self-adaptive.org list two benchmarks: ZNN.com (a webserver system providing a simplified news site), and ATRP (automated traffic routing problem). Outside the cloud computing domain only a few benchmarks were known. In conclusion, the area of quantification of self-aware systems appeared to the participants as a fruitful research area still in initial stages, which bears many topics for future research.

4.3 Working Group: “Generic Architectures for Collective Self-aware Computing Systems”

Kirstie Bellman, Nelly Bencomo, Ada Diaconescu, Lukas Esterle, Holger Giese, Sebastian Götz, Chris Landauer, Peter Lewis, and Andrea Zisman

License  Creative Commons BY 3.0 Unported license
 © Kirstie Bellman, Nelly Bencomo, Ada Diaconescu, Lukas Esterle, Holger Giese, Sebastian Götz, Chris Landauer, Peter Lewis, and Andrea Zisman

The purpose of this working group was to discuss the architectural aspects relevant to collectives of self-aware computing systems, meaning, of several self-aware computing systems interacting in some way, such that they may also themselves comprise a self-aware system.

We discussed where the goals for such a collective could come from, in terms of both sources external and internal to the collective. Further, we considered how one could reason about or control the dynamics of the collective’s goals, which might include, for example how to induce their individual goals cohering into a mutually beneficial set. We discussed the case where the individual systems coordinate or cooperate in order to achieve a common goal, when their goals might lead them to competition, and also situations where the systems could pursue their goals in parallel without interference.

One important point emerged that we must not assume that systems inside a collective will all be alike or have the same levels of self-awareness. In fact, one objective of an individual system, for example upon joining a collective, might be the discovery of other members’ levels of self-awareness and capabilities. One challenge is to coordinate such a collective given these different levels of awareness (which can include no awareness).

In this context, three key challenges were identified by the group:

1. Whether we could define a meta-architecture for collectives of self-aware systems, which might express the space of possible concrete architectures for a (self-aware) collective of self-aware systems.
2. How high level goals coming from outside the collective could be decomposed to individual goals for component systems.
3. How the types and levels of self-awareness at the component system level and the global system level relate to each other. For example, if a certain level of self-awareness is desired at the global level, what form of self-awareness at the component system level is required?

In attempting to tackle challenge 1, and with a focus on how this might support solutions to challenge 2, we first identified some of the essential interfaces (or input/output ports) that self-aware systems should expose in order to be able to interact with one another. These include:

- Input goal(s): to receive goals from external entities, such as human administrators, users, or other systems;
- Output evaluation(s): to report (to the goal-requiring entities) the extent to which their goals have been achieved;
- Output goal(s): to require goals to be achieved by other systems;
- Input evaluation(s): to obtain reports from the self-aware systems which it has required goals from;
- Input/output communication: to manage relations with other systems.

Concerning the self-aware systems’ connectivity within collectives, it was noted that the output goal interface of one self-aware system can be connected to the input goal interface of another system, or of several other systems. Notably, it can also be connected to the system’s

own input goal interface, triggering exploratory play behaviour, which is considered essential for the system's proactive learning process. It was also noted that a self-aware system can in turn be composed of other self-aware systems, and so on, in a recursive manner.

Based on these observations, we next focused on the range of seemingly viable integration alternatives of self-aware systems, from different perspectives. This resulted in a first attempt to provide an organisational taxonomy comprising the main axes of possible variations. The purpose was to go towards defining a meta-architecture from which concrete architecture instances of collective self-aware system could be instantiated, by selecting precise values from each one of the variability axes. Three such variability axes were identified:

- System types: representing various levels of self-awareness, ranging from reactive systems to meta-self-aware systems;
- Inter-system relations: including cooperation, competition, ignorance and parasitism;
- System inter-connection patterns: including hierarchy, centralised and peer-to-peer.

Considering challenge 3, and in terms of the implications of having possibly different levels of self-awareness in components of a collective system, as well as different levels apparent between global and local levels, we discussed the learning of individuals versus the learning of the collective as a whole. We discussed the concept of “active experimentation” in the CARS architecture as an example of creating suitable test-beds or ‘places’ that a system can experiment, learn what its boundaries are, learn where it fails in different operational contexts, and because of which subsystems; or in the case of collectives, learn which systems it can integrate or not in different ways. We also discussed how a collective could learn strategies for different goals that it could potentially achieve and then decide at runtime whether it can achieve those goals or better offload (some of) them to others for instance. Furthermore we briefly explored the effects of individual components leaving or joining the collective during runtime on the level of self-awareness of the collective and its capability to achieve its high-level goals.

We further distinguished between the scope of a component system's self-awareness being limited to the individual component system, or also being aware of the wider collective. In the second case, a feedback loop between the global system and local components' self-awareness properties will have significant impact on many design choices in self-aware collectives.

The group was productive in identifying core challenges involved in the architecture and design of collectives of self-aware systems. The main research challenges identified included the meta-architecture specification, the decomposition of goals and knowledge across self-aware systems within a collective, and the possible relations between the levels of self-awareness of a collective and of the systems within the collective. These challenges, and the results of this working group's subsequent discussions have provided a base for a future collaboration among most participants for writing a corresponding chapter in a book dedicated to self-aware computing.

4.4 Working Group: “Benchmarking Self-aware Computing Systems”

Alexandru Iosup, Sara Bouchenak, Xiaoyun Zhu, Lydia Chen, Evangelia Kalyvianaki, K. R. Jayaram, Kirsty Bellman, Anders Robertson, Heiko Koziolk, Steffen Becker, Evgenia Smirni, Arif Merchant, Aleksandar Milenkoski, and Felix Maximilian Roth

License © Creative Commons BY 3.0 Unported license

© Alexandru Iosup, Sara Bouchenak, Xiaoyun Zhu, Lydia Chen, Evangelia Kalyvianaki, K. R. Jayaram, Kirsty Bellman, Anders Robertson, Heiko Koziolk, Steffen Becker, Evgenia Smirni, Arif Merchant, Aleksandar Milenkoski, and Felix Maximilian Roth

The purpose of this working group was to discuss how to benchmark self-aware computing systems. We discussed the types of problems that appear when benchmarking self-aware computing systems, what are the aspects/bottlenecks to benchmark in self-aware computing systems, what are the unique benchmarking aspects of self-aware computing systems (also relative to non-self-aware computing systems). Each aspect was discussed from a conceptual perspective matched to one or several use cases or examples.


Six important challenges emerged:

1. New tools to evaluate and benchmark self-aware computing systems. There is a general lack of workloads for benchmarking. New workloads should include input data, for the new data-intensive applications that are now common in datacenters and other application scenarios. New workloads should include various patterns of load intensity, including burstiness. How to represent realistic scenarios, e.g., for a specific industry, without introducing undue complexity?
2. New metrics and metric bundles to benchmark self-aware computing systems. These systems need to be self-* in maintaining or optimizing multiple aspects, such as performance, availability, energy consumption, financial and human-resource costs, security, etc.
3. Quantifying the type and level of self-awareness and self-adaptation. Characterizing the transient nature of the operation of these systems is challenging. What is elasticity? How to quantify performance variability? Which metrics for self-configuration, self-expression, etc.? What overheads to consider? etc.
4. Quantifying the type and level of self-protection and self-healing. Measurement and metrics of intrusion-detection scenarios. Measurement and metrics for performance isolation. etc.
5. Using benchmarks online, to improve or even provide self-awareness. How can system properties be uncovered online? How can this knowledge be used online?
6. Benchmarking the benchmarks. What is the coverage provided by (current) benchmarks, in terms of workload representativeness, data aging, etc.? How to define and refine pain points/bottlenecks of self-aware systems? What type and to what level of dynamicity is the benchmark addressing? etc.

The discussion was productive in identifying and sharing metrics, use cases, benchmarking goals, and in identifying and formulating five key challenges for the future of benchmarking self-aware computing systems. Most participants are currently collaborating in topics related to this meeting. In particular, we are collaborating in writing a chapter on benchmarking in a book dedicated to self-aware computing.

4.5 Working Group: “Architecture and Reuse of Self-Aware Systems”

Anne Kozirolek, Christopher Landauer, Anne Kozirolek, Heiko Kozirolek, and Evangelia Kalyvianaki

License  Creative Commons BY 3.0 Unported license
© Anne Kozirolek, Christopher Landauer, Heiko Kozirolek, and Evangelia Kalyvianaki

This working group discussed the architecture of self-aware systems and reuse as a selected aspect of the life-cycle of such systems. Additionally we touched on the theoretical limitations of self-awareness.

The group first discussed the autonomic control loop with its MAPE-K phases in the context of self-aware systems. We found that self-aware systems distinguish themselves from general self-adaptive systems or autonomic systems by focussing on the “Monitor” and “Analyze” phases. Additionally, the “Knowledge” in a self-aware system includes knowledge about the system itself and even about its MAPE-K loop. For a specific control loop for self-aware systems, one might consider to specialize the “Knowledge” by “Self-Knowledge”, “Reflective Models” or just “Models”. We also noted that for different applications, some parts of MAPE-K loop are not explicitly available or modelled (in code or in the running system). Especially the “Execute” step is not required in self-aware system, as our working definition does not strictly require the system to act upon its observations and analyses.

The group also talked about the models and controllers in the Wrappings approach of Kirstie L. Bellman and Christopher Landauer, where the phases of MAPE-K are not explicitly distinguished. In Wrappings, models and controllers are the same. The controller is also interpreted based on a model. As an example, a system changes itself by bringing in new resources. Still selection and execution process. To cite from one of their papers, Wrappings is based on two key, complementary parts: (1) explicit, machine-interpretable descriptions meta-knowledge of all software, hardware, and other computational resources in a Constructed Complex System (organized into Wrapping Knowledge Bases), and (2) active integration processes that use that information to select, adapt, and combine these resources for particular problems (the Problem Managers). The integration process needs to be guided by constraints to do something useful and it is the task of the designer to define these constraints.

As a second topic, we discussed reuse as an aspect of the life-cycle of self-aware systems. Starting again with the MAPE-K picture, we hypothesized that the system to manage by the MAPE-K loop (the autonomic manager in the original IBM terms) can be exchanged and everything else can be reused to some extent. Models might have to be updated so that they match the new system. The part in the Knowledge describing the awareness of the MAPE-K loop is likely to be highly reusable across different systems. Also analyses can be reusable. For example, a performance analysis approach that learns the appropriate performance model (whether or not the system has exponential distributed service times, what the relevant queues in the system are, etc).

We considered whether the system be changed in other ways than defined by the actuators. One opinion was that there is no way to in general know the side effects of such unanticipated changes, the manager cannot change code in general. At the same time, one could exchange the whole system.

Continuing with reuse, we found that controllers might be reusable. Additionally, monitoring hooks are reusable (actually there are many monitoring frameworks available).

One challenge when reusing are the dependencies among the MAPE steps, as a subsequent step needs the right output from the previous step. It might be practical to have specialized

models, such as a model of web server applications. Such a model can learn specific parameters at run time. It is reusable for different types of web servers, but not beyond that.

This lead us to the question of how generalizable and expressive the models can be. We hypothesized that the models should not be too general. At some point, one would reach the generality of a Turing machine, which is not helpful in practice as it is not analysable.

Continuing on that thought, we hypothesized that if there was a model that can be aware of any system, it would be supported by the architecture. A potential limitation of the self-awareness lies in the used formalism of the model (as one does not want to be as general as Turing machine, there are some limitations of what such models can express). Additionally, there may be limitations in what you can monitoring and execute. Finally, we hypothesized that a self-aware system will not be able to (1) infer what its code does based on measurements and (2) that it will not be able to be fully aware of itself, because the model would have to be at least as complex as the system (cf. law of requisite variety) but at the same time we need models that are abstract and analyzable. To conclude this part of the discussion, we identified the need to more closely study the theory of self-aware systems, their theoretical capabilities and limitations.

Participants

- Tarek F. Abdelzaher
Univ. of Illinois – Urbana, US
- Artur Andrzejak
Universität Heidelberg, DE
- Christian Becker
Universität Mannheim, DE
- Steffen Becker
TU Chemnitz, DE
- Kirstie Bellman
The Aerospace Corp. – Los Angeles, US
- Nelly Bencomo
Aston Univ. – Birmingham, GB
- Sara Bouchenak
University of Grenoble, FR
- Javier Camara
Carnegie Mellon University – Pittsburgh, US
- Giuliano Casale
Imperial College London, GB
- Lydia Y. Chen
IBM Res. GmbH – Zürich, CH
- Ada Diaconescu
Telecom Paris Tech, FR
- Lukas Esterle
Universität Klagenfurt, AT
- Antonio Filieri
Universität Stuttgart, DE
- Francesco Gallo
University of L'Aquila, IT
- Kurt Geihs
Universität Kassel, DE
- Holger Giese
Hasso-Plattner-Institut – Potsdam, DE
- Sebastian Götz
TU Dresden, DE
- Lars Grunske
Universität Stuttgart, DE
- Henry Hoffmann
University of Chicago, US
- Paola Inverardi
University of L'Aquila, IT
- Alexandru Iosup
TU Delft, NL
- K. R. Jayaram
IBM TJ Watson Research Center – Yorktown Heights, US
- Evangelia Kalyvianaki
City University – London, GB
- Joost-Pieter Katoen
RWTH Aachen, DE
- Jeffrey O. Kephart
IBM TJ Watson Research Center – Yorktown Heights, US
- Samuel Kounev
Universität Würzburg, DE
- Anne Koziulek
KIT – Karlsruher Institut für Technologie, DE
- Heiko Koziulek
ABB AG Forschungszentrum Deutschland – Ladenburg, DE
- Marta Kwiatkowska
University of Oxford, GB
- Philippe Lalanda
Université de Grenoble, FR
- Chris Landauer
The Aerospace Corporation – Los Angeles, US
- Peter Lewis
Aston Univ. – Birmingham, GB
- Martina Maggio
Lund University, SE
- Ole Mengshoel
Carnegie Mellon University – Silicon Valley, US
- Arif Merchant
Google Inc. – Mountain View, US
- Aleksandar Milenkoski
Universität Würzburg, DE
- Anders Robertsson
Lund University, SE
- Felix Maximilian Roth
Universität Mannheim, DE
- Hartmut Schmeck
KIT – Karlsruher Institut für Technologie, DE
- Evgenia Smirni
College of William and Mary – Williamsburg, US
- Katinka Wolter
FU Berlin, DE
- Xin Yao
University of Birmingham, GB
- Xiaoyun Zhu
VMWare, Inc. – Palo Alto, US
- Andrea Zisman
The Open University – Milton Keynes, GB

