Monitoring for Control in Role-oriented Self-Adaptive Systems

Ilja Shmelkin
ilja.shmelkin@tu-dresden.de
Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany

ABSTRACT

Self-adaptive Systems (SASs) are one way to address the ever-growing complexity of software systems by allowing the system to react on changes in its operating environment. In today's systems, self-adaptation is typically realized with a control loop, for which the MAPE-K feedback loop is a prominent example. Research uses the notion of patterns to describe the distribution and decentralization of individual control loop components or control loops and their underlying *managed subsystems*. While there are some well-accepted standards about which components a managed subsystem has to implement so that it can interact with the control loop, research still lacks best practices for communication *within* and *across* control loops. This paper aims to identify several research challenges that exist currently in this domain. Furthermore, ideas on upcoming research to create distributed SASs that rely on roles, benchmarking and inter- and intra-loop communication for control loops will be presented. Furthermore, ongoing work on a self-adaptive distributed benchmarking application will be discussed. Finally, an evaluation strategy will be presented to provide evidence for viable results to the community.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; • **Networks** → *Network performance analysis.*

KEYWORDS

self-adaptive systems, roles, control loops, monitoring, metrics

ACM Reference Format:
Ilja Shmelkin. 2020. Monitoring for Control in Role-oriented Self-Adaptive Systems. In *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387939.3391598

1 INTRODUCTION

Self-adaptive systems (SASs) can adapt themselves by reacting on changes in their operating environment, their goals, failure of individual components, resource demand, and many more. Typically SASs consist of multiple components interacting with each other in a way that is defined during system design time. Most of them use *control loops* to reason about and organize adaptations of an underlying *managed subsystem* of which the MAPE or MAPE-K (Monitor, Analyse, Plan, Execute, and Knowledge) control loop is one prominent example [7]. The authors use the terms *autonomic manager* to refer to the (intelligent) control loop and *managed resources* for the system which comprises the application logic and provides the actual SASs functionality. Other researchers use different terms (e.g. *architecture layer*, *reflective subsystem* or *adaptation engine* for the autonomic manager and *system layer*, *base-level subsystem* or *core function* for the managed resources) to make the same distinction [9, 26, 35]. As Weyns et al. [36] state, it makes sense to avoid confusion by sticking to the terms *managing subsystem* (i.e. the control loop) and the managed subsystem. Further, they clarify the distinction between a *distributed* and a *decentralized* SAS from which the former refers to the physical distribution of an SAS across multiple nodes which are connected with each other through a computer network, and the latter defines how control decisions within an SAS are made, based on information communicated between individual components of the control loop. To better understand the decentralization aspect of SASs, the authors introduce *patterns* that describe the communication paths between controlling components of the SAS, whereby the components that communicate can be individual parts of a control loop (e.g. only monitoring components communicate with other monitoring components), individual control loops with each other, nested loops [11] or even sub-loops where the interaction is not bound to the logical sequence of the MAPE-K control loop [31].One important task in this research field is to establish standards to accelerate progress and allow new researchers and students to gain a foothold in the community of SASs. While there are some well-accepted standards about which components a managed subsystem has to implement [36] (e.g. *sensors* and *effectors*, however different names exist) so that they can interact with the managing subsystem, research still lacks standardization for communication *within* and, in cases where the SAS is decentralized, *across* control loops [33, 36] and therefore introduces potential bottlenecks. Creating standardized communication interfaces and protocols could be one way to address this problem. Another, so far not well-addressed task is to create suitable benchmarking applications so that researchers can compare their work on new SAS architectures. Although such benchmarks exist [6, 34, 37], they are commonly ignored by the community.

The remainder of this paper is structured as follows. Section 2 presents the idea of using the notion of *roles* to incorporate adaptivity into software (i.e. the managed subsystem). Section 3 shows a general outline of how the author plans to solve the just introduced tasks in his dissertation, already ongoing work as well as a strategy for the evaluation of results. Section 4 concludes this paper.

SEAMS '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7962-5/20/05…$15.00
https://doi.org/10.1145/3387939.3391598

2 ROLE-ORIENTED SOFTWARE INFRASTRUCTURES

While the community predominantly agrees on the use of control loops in managing subsystems, there are multiple approaches to design the underlying managed subsystems. One of them is centered around the notion of roles, which was introduced by Bachmann way back in 1977 [2]. Roles encapsulate dynamic behavior of static *players*, which, in theory, can be used to build adaptivity into software by allowing dynamic binding and unbinding of roles during run time. More recent notions of roles allow capturing context-dependent properties of objects [14, 15, 21]. Since, for a long time, there was no common understanding of roles, Kühn et al. [22] denoted 26 classifying role features (based on 15 role features from the work of Steimann [27]), from which a subset can provide adaptivity during run time [28]. Even though there are many modeling languages supporting roles [2, 4, 10, 12, 14, 17, 25, 27], they commonly use different views on relationships and their context-dependency. This research, therefore, sticks to more recent approaches for role-based modeling, which focus heavily on the context-dependent nature [10, 14, 25], some of them allowing relationships to play roles themselves, as Kühn et al. [21] conclude. This enlarges the flexibility by allowing to stack roles. The approaches above use different naming conventions for expressing context-dependency (e.g. *environment*, *institution*, *ensemble*, and *compartment*) from which, as Kühn et al. [21] state, the term *compartment* encompasses all other notions.

Since classic object-oriented programming languages are not sufficient for creating adaptive role-based applications, a group of role-based programming languages emerged [3, 5, 13, 18, 23, 24], from which SCROLL[1] [23] supports most of the role features and therefore represents the go-to programming language for roles. Neither SCROLL nor the other languages natively allow for creating distributed role-based applications, which is a serious shortcoming since today's systems focus less on monolithic applications but more on heavily distributed, loosely coupled services that operate towards a common goal.

Taing et al. recently tackled this topic by creating a prototypical implementation of the concept *dynamic instance binding* [29] called LyRT (and its successor Role4j[2]) and combining it with a distributed adaptation protocol [32] to allow for the adaptation of a distributed role-based system during run time. Although the adaptation protocol implemented a mechanism how to handle adaptation failure on the network level, consistent behavior of objects engaging in ongoing method executions had to be guaranteed on instance level when dynamic changes took place, allowing for a *quiescent* state [19] before and after distributed adaptations had taken place. Neither this state could be guaranteed in a distributed environment, nor were proper interfaces for the integration of the network protocol into the MAPE control loop defined, which opens space for further research. Nevertheless, as stated earlier, roles have several benefits in modeling and implementation of adaptive systems and therefore will be used as a basis for the author's thesis.

[1] https://github.com/max-leuthaeuser/SCROLL
[2] https://github.com/nguonly/role4j

3 THESIS SCOPE

As there is currently no state-of-the-art approach to implement working distributed role-based SASs, this work is dedicated to advance this field of research. Although a role-based approach is chosen, the author's contribution is expected to be applicable in a more general way throughout the domain of SASs. The following section will show multiple interesting topics with research potential which will be addressed in the author's dissertation, however, the overall focus remains on monitoring. Later, a currently ongoing structured literature review will be explained. Finally, an example application based on role-playing actors will be introduced.

3.1 Monitoring in Distributed Self-Adaptive Applications

The previous section has shown that recent research has provided a local runtime system [28] as well as a network protocol to distribute adaptations [32]. Both approaches combined allow to create a distributed role-based application, however, there is no guarantee that the application will run robustly since neither the protocol nor the local runtime can assure a quiescent state of the local instances. Furthermore, the combined approach does not follow the full logical sequence of the MAPE-K control loop and therefore does not allow for self-adaptivity yet as only the execution of an already available adaptation plan [28, 32] was addressed. To solve this issue it is required to extend the provided functionality by adding appropriate distributed monitoring and analyze mechanisms to collect meaningful data and further analyze this data allowing to reason about the possibility to adapt the distributed application, as figure 1 shows. To allow monitoring of SASs in a distributed scenario the following questions have to be answered:

(1) Which additional data has to be collected specifically for SASs (also regarding roles) compared to traditional systems? See subsection 3.2.
(2) In which format do we represent monitored data (i.e. metrics)? See subsection 3.2.
(3) How do we transport monitored metrics between nodes in a decentralized scenario? See subsection 3.3.
(4) Do we need to persistently store, and if yes, where and how long do we store monitored data? See subsection 3.3.
(5) Does the linear and non-linear pair correlation of metrics provide additional information?
(6) Do we need to consider the grade of distribution and decentralization [36] (i.e. different patterns) of an SAS?

To reason about adequate analysis mechanisms research first needs to overcome the uncertainty posed by those questions. Any claims made regarding that would be highly hypothetical and therefore are retained for future publications. Next, to make self-adaptivity more accessible to software developers, research needs to define standardized interfaces for communication within the MAPE control loop, which will be discussed in subsection 3.3 in more detail.

3.2 Towards a Standardized Metric Representation

Monitoring individual components of the SASs infrastructure is already part of most approaches [30], however, when collecting data

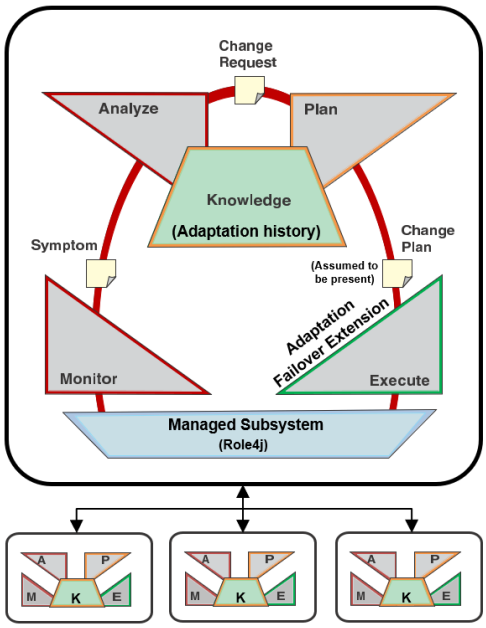

Figure 1: Image altered from [7]. Current state of research in role-based distributed SASs. Author's planned contribution marked in red. Partially available components marked in orange. Available components marked in green.

no naming conventions or standardized data structures are used. As control loops follow a defined logical sequence of actions it is possible to represent gathered data as time series. For years, leading companies in the domains of time series monitoring and datastores (e.g. Prometheus, InfluxDB) have been struggling to define an open standard for representing and transmitting metrics at scale. Although multiple attempts exist (i.e. Metrics 2.0[3], OpenMetrics[4]), Metrics 2.0 is only used by a handful of systems and OpenMetrics has yet to be released, however, each associated GitHub repository did not receive an update in years. One of the author's planned contributions is to advance those attempts to answer questions 1) and 2) posed in the previous subsection by proposing a standardized representation of metrics for SASs.

3.3 Inter- and Intra-loop Communication in Control Loops

As stated earlier, the communication between individual components is a potential bottleneck for decentralized SASs. This gets even worse when physical distribution across multiple nodes is introduced. Therefore, it is not only required to reason about adequate monitoring metrics and mechanisms, but also about standardized interfaces and communication protocols between control loop components. A common solution for all SASs is unlikely as their structure can differ significantly (i.e. different patterns [36] may be present). This challenge does not only apply to role-based SASs but any SAS. This subject was not addressed by the SASs community yet as Weyns and others state in multiple articles [33, 36]. Therefore, one anticipated contribution of this work is to find principled solutions to decentralized self-adaptation by answering questions 3) to 6) posed in subsection 3.1 and proposing standardized interfaces and communication protocols for SASs that rely on control loops. Another crucial aspect to consider is *pre-processing* for knowledge gain on the instance level to minimize the communication footprint.

3.4 Structured Literature Review

The term structured refers to the way how the vast amount of existing papers on different subjects in the domain of SASs is handled. The author's strategy on this matter was to start by examining well-accepted survey papers and books in this domain [8, 16, 20, 26], however, work older than 15 years was not considered in the first place to avoid possible misdirection in the early stages of the author's dissertation. Since a part of the overall topic examined by the *research training group "Role-based Software Infrastructures for continuous-context-sensitive Systems"* (*RoSI RTG*) is to investigate the practical applicability of roles in software during run time, meaningful dissertation theses [28, 32] in the domain of SASs coupled with the aspect of roles were kept in mind while extracting useful information and cross-references out of the surveys or books respectively. The next task was to identify researchers in the community who regularly contribute new knowledge to the domain of SASs which applies to the niches the author's thesis contributes to (i.e. patterns for SASs based on roles, communication *within* control loop elements, communication, and architectures in SASs in general, benchmarking of SASs). People like that are D. Weyns, J. Andersson, S. Malek, H. Giese, S-W. Cheng, however, this list is incomplete as the structured literature review is still in progress. While reading new articles, the contributions of the people listed get prioritized. While continuing to search for new literature is a perpetual process, a more detailed overview of the review is planned as a contribution by the end of the third quarter of 2020.

3.5 Benchmarking Application with Role-Playing Actors

As stated before, the community lacks suitable benchmarking applications. This intensifies even more in the domain of role-based scenarios for which no adequate option exists that incorporates any of the best practices for benchmarking of SASs [6], opening yet another task for research. During the first months of research activity, it turned out that an example application to test the applicability of roles in adaptive software during run time would be beneficial. Therefore, in collaboration with other research fellows from the RoSI RTG, an example project was created. The feature-requirements we put up for the application were the following, however, we plan to extend them to allow for full self-adaptivity in the future. In the beginning, the example application should only be able to activate and deactivate roles dynamically during run time, as well as run in a distributed environment (i.e. multiple different virtual machines). Furthermore, this allowed us to investigate a currently not examined approach, namely the creation of distributed systems with multiple role-playing actors (based on the actor model [1]), which exchange messages network-transparently, introducing the ability to create concurrent role-based SASs in the

[3] http://metrics20.org/
[4] https://openmetrics.io/

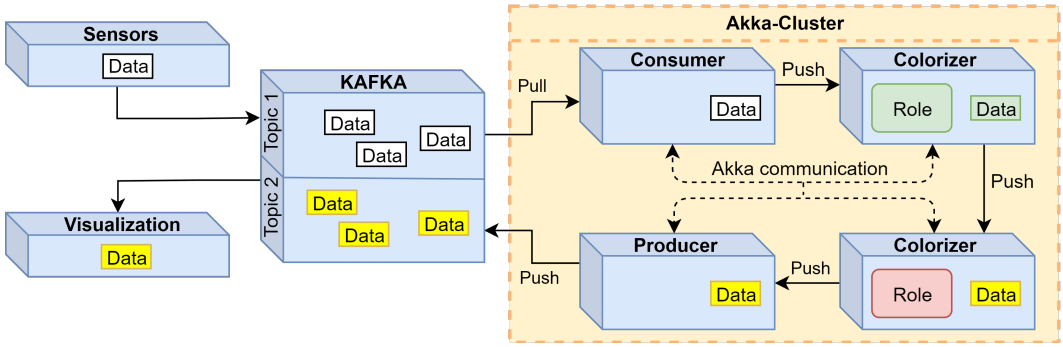

Figure 2: GPS dataflow within an adaptive example application on the basis of role-playing actors. Actors operate concurrently within the Akka-Cluster. Description: → shows GPS dataflow, ⇢ shows adaptation transactions (within the Akka-Cluster).

future. For this, we used *Akka*[5], an implementation of the actor model in Scala/Java, among other technologies. A more detailed description of the concept of using the actor model in distributed self-adaptive applications is planned for another contribution and therefore will not be addressed in more detail in this paper.

The overall application context is settled in the domain of stream processing. More precisely, we tried to build an adaptive stream processing pipeline in which the actual function of each individual operator (i.e. actor) is determined by the roles bound during run time rather than during the development of the pipeline. This allows exchanging specific parts of the pipeline's functionality while the system is operating. That, however, introduces possible data corruption, hence creates the requirement for a structured adaptation process. We used roles to enable adaptation and used actors to engage in dynamic adaptations, as depicted in figure 2.

We created a pipeline consisting of four *operators* (i.e. actors of an Akka-Cluster), from which one reads GPS data out of a Kafka[6] topic and one writes the processed GPS data back to another Kafka topic respectively. The remaining operators process the provided GPS data by adding an additive color code to it. The color code can be either plain red, plain green or plain blue and is solely determined by the role bound to the operator. Since the color codes are additive, different combinations of roles bound to the operators during run time result in a different colorization of the output data. The output data finally gets visualized by a web service[7], painting a colorized line in real-time, which represents the GPS track processed by the pipeline. Still, the application does not allow for self-adaptivity yet since no monitoring occurs, hence opening an intersection between both proposed research topics.

3.6 Evaluation Strategy

Since there are different types of contributions, also different evaluation strategies apply for each of them. The following subsection summarizes those strategies. The claims made involve a high amount of engineering effort and are therefore highly hypothetical at this moment.

[5]https://akka.io/
[6]https://kafka.apache.org/
[7]https://rosimon.shmelkin.net/karte/

3.6.1 Communication and Pre-processing. When creating SASs or software systems in general, performance and reliability play important roles, hence they need to be monitored. By introducing physical distribution across several nodes the performance of a system can suffer from different conditions (e.g. insufficient bandwidth, long response times) which can create performance bottlenecks. Therefore, in order to evaluate if a system provides proper function under heavy load, simulating unusual resource demand (e.g. "slashdot effect") can be a suitable benchmark scenario and therefore will be used on different applications like TAS [34], Znn.com [6] and the application introduced in subsection 3.5 to compare new findings in communication and pre-processing of metrics.

3.6.2 Benchmarking distributed SAS. To conduct the experiments mentioned in the last paragraph it is necessary to create benchmarks that allow the testing of SASs in a distributed environment. Since it is beneficial to advance the usage of a common toolset in the community of SASs, the mentioned benchmark applications will be extended to allow for distributed operation. As TAS, as well as Znn.com, were developed in Java, they would even allow a re-implementation in Scala or SCROLL to efficiently test those applications against one another. Furthermore, that would allow comparing role-based SASs with more traditional approaches as well as give an insight into the applicability of role-playing actors in real-life scenarios.

4 CONCLUSION

The complexity of information systems keeps growing continuously. One way to handle this complexity are SASs. In this paper, several challenges for SASs were named which need to be addressed in future research. Furthermore, an outlook was given about how the author plans to investigate possible solutions and how those solutions will be evaluated. The stage of this research has to be ranked as very early as only four months passed since it commenced. All claims and statements were made to the best of the author's knowledge and belief.

ACKNOWLEDGMENTS

This work is funded by the German Research Foundation (DFG) within the Research Training Group Role-based Software Infrastructures for continuous-context-sensitive Systems (GRK 1907).

REFERENCES

[1] Gul A Agha. 1985. *Actors: A model of concurrent computation in distributed systems.* Technical Report. MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB.

[2] Charles W Bachman and Manilal Daya. 1977. The role concept in data models. In *Proceedings of the third international conference on Very large data bases-Volume 3.* VLDB Endowment, 464–476.

[3] Matteo Baldoni, Guido Boella, and Leendert Van Der Torre. 2006. powerJava: ontologically founded roles in object oriented programming languages. In *Proceedings of the 2006 ACM symposium on Applied computing.* ACM, 1414–1418.

[4] Stephanie Balzer and Thomas R Gross. 2011. Verifying multi-object invariants with relationships. In *European Conference on Object-Oriented Programming.* Springer, 358–382.

[5] Stephanie Balzer, Thomas R Gross, and Patrick Eugster. 2007. A relational model of object collaborations and its use in reasoning about relationships. In *European Conference on Object-Oriented Programming.* Springer, 323–346.

[6] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. 2009. Evaluating the effectiveness of the rainbow self-adaptive system. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems.* IEEE, 132–141.

[7] IBM Corporation. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31, 2006 (2006), 1–6.

[8] Rogério de Lemos, Holger Giese, Hausi A Müller, and Mary Shaw. 2013. *Software Engineering for Self-Adaptive Systems: International Seminar Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers.* Vol. 7475. Springer.

[9] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54.

[10] Valerio Genovese. 2007. A meta-model for roles: Introducing sessions. In *Proceedings of the 2nd Workshop on Roles and Relationships in Object Oriented Programming, Multiagent Systems, and Ontologies.* 27–38.

[11] Karl M Göschka, Lorenz Froihofer, and Schahram Dustdar. 2008. What SOA can do for software dependability. *Supplementary Volume of DSN* 8 (2008).

[12] Terry Halpin. 2005. ORM 2. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems".* Springer, 676–687.

[13] Chengwan He, Zhijie Nie, Bifeng Li, Lianlian Cao, and Keqing He. 2006. Rava: Designing a java extension with dynamic object roles. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06).* IEEE, 7–pp.

[14] Rolf Hennicker and Annabelle Klarl. 2014. Foundations for ensemble modeling–the Helena approach. In *Specification, Algebra, and Software.* Springer, 359–381.

[15] Stephan Herrmann. 2007. Programming with Roles in ObjectTeams/Java. *Applied Ontology* 2 (01 2007), 181–207.

[16] Markus C Huebscher and Julie A McCann. 2008. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)* 40, 3 (2008), 7.

[17] Tobias Jäkel, Thomas Kühn, Stefan Hinkel, Hannes Voigt, and Wolfgang Lehner. 2015. Relationships for dynamic data types in RSQL. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)* (2015).

[18] Tetsuo Kamina and Tetsuo Tamai. 2009. Towards safe and flexible object adaptation. In *International Workshop on Context-Oriented Programming.* ACM, 4.

[19] Jeff Kramer and Jeff Magee. 1990. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on software engineering* 16, 11 (1990), 1293–1306.

[20] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing* 17 (2015), 184–206.

[21] Thomas Kühn, Stephan Böhme, Sebastian Götz, and Uwe Aßmann. 2015. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering.* ACM, 113–124.

[22] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. 2014. A metamodel family for role-based modeling and programming languages. In *International Conference on Software Language Engineering.* Springer, 141–160.

[23] Max Leuthäuser. 2015. SCROLL-A Scala-based library for Roles at Runtime. In *Proceedings of the 3rd Workshop on Domain-Specific Language Design and Implementation (DSLDI 2015).* 7–8.

[24] Michael Pradel and Martin Odersky. 2008. Scala roles: Reusable object collaborations in a library. In *International Conference on Software and Data Technologies.* Springer, 23–36.

[25] Trygve Reenskaug and James O Coplien. 2009. The DCI architecture: A new vision of object-oriented programming. *An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html* (2009).

[26] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)* 4, 2 (2009), 14.

[27] Friedrich Steimann. 2000. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering* 35, 1 (2000), 83–106.

[28] Nguonly Taing. 2017. Run-time Variability with Roles. (2017).

[29] Nguonly Taing, Thomas Springer, Nicolás Cardozo, and Alexander Schill. 2016. A Dynamic Instance Binding Mechanism Supporting Run-Time Variability of Role-Based Software Systems *(MODULARITY Companion 2016).* Association for Computing Machinery, New York, NY, USA, 137–142. https://doi.org/10.1145/2892664.2892687

[30] Norha M Villegas, Hausi A Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. 2011. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems.* 80–89.

[31] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. 2011. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* ACM, 202–207.

[32] M Sc Martin Weißbach. 2018. Run-time Adaptation of Role-based Software Systems. (2018).

[33] Danny Weyns. 2017. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering* (2017).

[34] Danny Weyns and Radu Calinescu. 2015. Tele assistance: a self-adaptive service-based system examplar. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* IEEE Press, 88–92.

[35] Danny Weyns, Sam Malek, and Jesper Andersson. 2012. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7, 1 (2012), 8.

[36] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M Göschka. 2013. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II.* Springer, 76–107.

[37] Jochen Wuttke, Yuriy Brun, Alessandra Gorla, and Jonathan Ramaswamy. 2012. Traffic routing for evaluating self-adaptation. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* IEEE Press, 27–32.