Coordination and Explanation of Reconfigurations in Self-Adaptive High-Performance Systems

Sarah Stieß and Steffen Becker
{sarah.stiess|steffen.becker}@iste.uni-stuttgart.de
Institute of Software Engineering, University of Stuttgart
Stuttgart, Baden-Württemberg, Germany

Florian Ege, Stefan Höppner and Matthias Tichy
{florian.ege|stefan.hoeppner|matthias.tichy}@uni-ulm.de
Institute of Software Engineering and Programming Languages, Ulm University
Ulm, Baden-Württemberg, Germany

ABSTRACT

Self-adaptive systems that modify their architecture to achieve different Service Level Objectives (SLOs) are becoming well-established parts of a company's portfolio. Current systems however do a poor job of coordinating the underlying rules and explaining the system's behavior to developers. This negatively impacts a system's performance due to behaving in unintended and hard-to-comprehend ways.

In this vision paper, we introduce an approach that intends to alleviate these problems, for the use-case of high-performance cloud clusters, through the use of a cyber-cyber digital twin based on the Palladio ecosystem. The conceived digital twin allows for several functions that are unexplored in literature so far: the combination of design time analysis with explainability, use of state-space prediction bounded by likelihood, and explanation of non-applicability. This is achieved through a digital twin performing three main activities:

1) The predicting activity predicts system performance along several potential paths.
2) The planning activity analyzes reconfiguration rules and predicted system behavior to produce reconfiguration plans that try to optimally utilize available resources and achieve all SLOs without unintended interactions between the execution of rules.
3) The explaining activity processes all artifacts produced during system simulation time or runtime to provide comprehensive visualizations to developers.

We explain how each activity is performed and what models it processes and produces based on a running example.

KEYWORDS

Self-adaptive system, digital twin, cyber-cyber digital twin, Palladio, explainability, reconfiguration coordination

ACM Reference Format:
Sarah Stieß and Steffen Becker and Florian Ege, Stefan Höppner and Matthias Tichy. 2022. Coordination and Explanation of Reconfigurations in Self-Adaptive High-Performance Systems . In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3550356.3561555

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10.
https://doi.org/10.1145/3550356.3561555

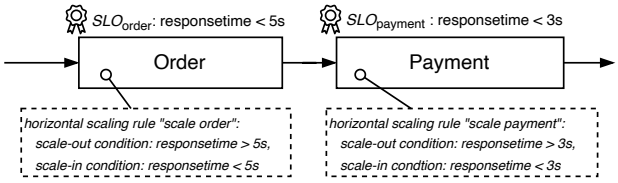

Figure 1: Running Example.

1 INTRODUCTION

Today, software engineers are developing systems with the ability to autonomously make decisions about their structure and behavior at runtime – so-called autonomic or self-adaptive systems [7, 13]. Such decisions might be scaling in and out service instances in a cloud-native system to maintain Service Level Objectives (SLOs) [5], or selecting collision-free flight trajectories of autonomous drone swarms performing complex joint tasks [17].

We use the first case as a running example throughout the paper. We consider a small excerpt of a cloud-native application for a web shop (as depicted in Figure 1). The excerpt comprises two self-adaptive services, each with a performance Service Level Objective (SLO) and reconfiguration rules to scale in and out, if their respective SLOs get violated. The *Order* service processes incoming order requests and calls a *Payment* service to handle the payment.

At some point in time, the number of incoming order requests increases, SLO_{order} is violated and *Order* scales out. As a consequence, it reestablishes its SLO.

The key ingredient of self-adaptive systems is the use of a digital twin to reason about the system's current and future states to plan autonomous decisions. The conceptual framework of Eramo et al. [9] structures the digital twin into a data part and a models part; the former covers the system's state and its history; the latter uses this data to drive the decision making. In our example, the system tracks measurements on request rates, response time, etc. of each service as the data model and maintains a structural model of the current system state (e.g. no of service instances) as a model at runtime. As the main parts of our actual system are software components, our digital twin is a cyber-cyber digital twin [1]. Though, one could argue that the hardware elements of a high-performance cloud system are also highly relevant and, thus, in our case, the line between a cyber-cyber and a cyber-physical digital twin is blurry.

Specifically, we focus on self-adaptive systems in our work that adapt their behavior by reconfiguration, i.e., adapting the system structure or assigning new parameter values to the architectural components [15]. These reconfigurations are specified as a set of

rules. The planning component of the self-adaption mechanism selects rules that it predicts will enable the system to achieve its aims.

Most current approaches [3, 11, 14, 19], consider adaptions independently of each other. In our running example this means that both services make scaling decisions independent of the other service. This may lead to contradictory or non-optimal behavior, e.g., oscillations between rule applications where one rule reverses the effect of another one [19]. The opaqueness of the decision-making process in the digital twin is a major impediment during development as well as in the operation of self-adaptive systems. In both situations, humans have to understand the behavior of the system to correct it successfully.

Continuing with our running example, the additional service instances of the *Order* service can now handle the incoming requests and forward more requests to the *Payment* service. However, now the *Payment* service cannot handle the increasing number of requests and *Order*'s SLO remains violated, until *Payment* is scaled too. In our vision, both services would have scaled together and explained the outlined interdependency to the developer on demand.

In this paper, we sketch an approach that addresses this vision by (1) an architecture instantiating the reference architecture of Eramo et al. [9] that includes prediction and planning, and simulation components to ensure the coordinated application of different rules and (2) an explainability approach that reuses the information from other components to aid the understanding about system behavior. We aim at extending prior work on SimuLizar [4] by implementing the presented digital twin in a more modern simulator called Slingshot [12]. Both SimuLizar and Slingshot focused so far on the design time analysis of uncoordinated reconfiguration rules. In this vision, we plan to extend it to be used at runtime to derive coordinated, explainable decisions.

2 THE DIGITAL TWIN AND ITS ACTIVITIES

The setup of our digital twin, its models and activities follow the conceptual framework for digital twins from Eramo et al. [9]. Our concrete implementation of the framework for our digital twin is shown in Figure 2. We extend the framework with data representing a systems SLOs, an explanation model for explanations and explicit representation of the engineer involved in developing and maintaining the Self Adaptive System (SAS). Our implementation realizes the MAPE-K loop [2].

The *digital shadow* represents raw monitoring, architecture and emergent context information such as request process rate over time in a cloud computing system. It also contains all reconfiguration rule data.

The *SLO data* represents information on the SLOs described for the actual system. SLOs are defined by the engineer during development and can be modified during maintenance.

The *descriptive model* is represented by Palladio Component Model (PCM) models containing all relevant data that describe those system aspects that are processed during prediction, and planning. This includes data on the current system setup, the monitoring data, relevant for rule selection and prediction, and the reconfiguration rules themselves.

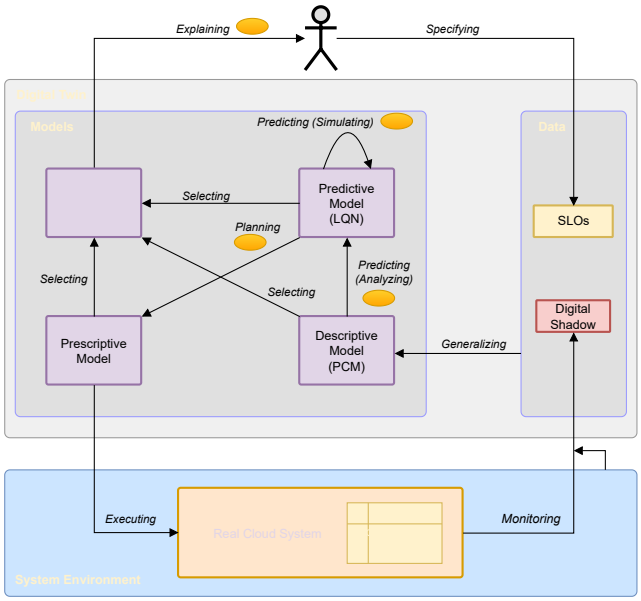

Figure 2: Aritfacts and activities in our digital twin. The ovals reference the sections where these activities are discussed.

The *predictive* model represents predictions on how the system will behave in the future based on simulations by Slingshot.

The *prescriptive model* represents a reconfiguration plan designed by the digital twin. It spans a set of possible future states and reconfigurations to be performed on the system based on conditions on the actual system state.

Lastly, the *explanation model* represents all data used to generate explanations for the developer. The data is selected from all other models and persisted over the course of the system's run-time.

In the following, we describe the three main activities that make up our conceptualized digital twin and explain how and which models are processed and produced in each activity.

2.1 Predicting

In our running example (see Figure 1) we model the incoming order requests as a stream with Poisson distribution. We then utilize a rapid Layered Queueing (LQN) solver [10] to predict potential future developments of the system. This results in a state space graph, which is a directed, acyclic graph, starting from the current system state and fanning out to various predicted future states of the system's digital twin during periods in time stretching into the future up to a certain limit.

The leaf nodes, i.e., the nodes at the front of the currently predicted state graph form the *prediction horizon*. To further expand the prediction (depending on the available time/memory budget), horizon nodes must be selected to progress further into the future from those states on. Fundamentally, a prediction of the future evolution of a system's state can be performed statically at design time (offline analysis) or dynamically at runtime (online analysis). Simulating the branching alternatives for possible future states of a complex system tends to be very time-consuming, as the system state space grows fast with the timespan that is considered. To make

predictions feasible with regard to time and memory, especially in the case of online prediction, it is necessary to guide the prediction expansion towards the most likely and relevant subset of the state space, e.g., by choosing horizon nodes for further expansion of predictions based on the predicted probability of reaching this state. If it is considered to be extremely improbable that the system will reach this state, expanding this part seems less important. Conversely, by evaluating states according to compliance with SLOs, we also consider potentially critical states: even given that an evolution of the system in this direction is deemed unlikely, if it would be some extremely desirable or undesirable state, this part of the state space should still be explored. With this information, we plan to adopt a Monte Carlo tree search-based approach for the expansion to make good use of the available budget.

Another technique that allows for a more comprehensive exploration of the future state space is to raise the level of abstraction as the prediction expansion progresses: The further into the future system states are stretching, the fewer details are considered, allowing for further, albeit coarser, predictions.

2.2 Planning

The planning activity processes both descriptive and predictive models to produce a coordinated reconfiguration plan. For this purpose, it analyzes the monitoring and system configuration as well as reconfiguration rules and their conditions contained within the current descriptive model, and all predicted future system developments from the *predictive model*. Based on these artifacts, a prescriptive model is produced that spans a set of possible future states and reconfigurations to be performed on the system based on the actual state that is reached. This timeline constitutes the reconfiguration plan. The system follows this plan until its behavior significantly deviates from the plan. At this point, both prediction and planning are triggered anew.

Planning is achieved through three levels that build on each other and allow higher and higher levels of *sophistication* in planning and coordinating rule execution. The advantage of this tiered coordination is that planning can be implemented and enhanced step-by-step.

The first level aims to evaluate the application condition of all rules together instead of separately and to find simple dependencies between the pre- and post-conditions of rules. Based on application conditions and dependencies rules are then scheduled to be executed consecutively. The second level additionally analyzes whether it is sufficient to apply only a subset of selected rules, based on cost-benefit analysis. Lastly, the third level utilizes more information from the predictive model to plan for multiple potential system behaviors thus creating a multi-set of reconfiguration plans.

In the following, we detail how each level can be realized and how this would look like on our running example.

2.2.1 Level 1: Coordinated evaluation and planned execution. Realizing level 1 implies several technical steps.It requires separation of rules application condition and their execution within the system. All conditions are evaluated and rules are marked with a label according to their applicability. Furthermore, it is necessary to obtain rule dependencies based on their pre- and post-conditions. This is achieved either through information provided by developers or by analyzing the state-space within the predictive model. Based on rules labels and detected dependencies a reconfiguration plan is written that projects the execution of all applicable rules and their dependent rules.

In terms of our running example, a SAS implementing level 1 will evaluate the response time of both services, label those scaling rules as applicable where the condition is met and schedule them for execution. In the case that a scale-out of the Order component is selected for execution, the scale-out for payment is also selected because it is detected that a higher number of Order components results in a higher load on existing Payment components and thus reduced response-time of the components.

2.2.2 Level 2: Cost-benefit filtering. Realizing level 2 sophistication implies implementing additional analysis capabilities. For this, functionality that filters the set of rules labeled as *applicable* in the prescriptive model in level 1 is implemented. Filtering is done based on cost-functions evaluating the cost-benefit of execution of each rule from the viewpoint of efficiency to achieve all SLOs. Cost functions can consider predefined resource allocation costs as well as prediction data from the *prediction model* for this purpose.

In terms of our running example, a SAS implementing level 2 can decide not to execute the *scale-out* rule for the *Payment* service if the prediction data shows a high likelihood of its response time dropping shortly anyway.

2.2.3 Level 3: Reconfiguration tree. Level 3 entails an even stronger entanglement of planning and predicting activities. Planning utilizes all predicted system behaviors to develop a reconfiguration plan in form of a tree. The reconfiguration tree starts with the current system state as its root and spans over all predicted futures. Each edge within the tree corresponds to either a reconfiguration that should be executed or a predicted change in the descriptive model triggered by changes in the system environment. The system tracks its path through the reconfiguration tree within the prescriptive model and executes the appropriate reconfiguration rules if it reaches a state where the next edge represents a reconfiguration. The system can be further enhanced with functionality that allows it to react to unexpected behavior if the resulting end state still corresponds to a state within the reconfiguration tree. In such a case the system can start to follow the path within the prescriptive model starting from this state.

In terms of our running example, a SAS implementing level 3 will plan both scaling-out and scaling-in of *Payment* and *Order* services over several seconds to minutes for several potential futures and execute them accordingly. If the system reaches an unexpected state or the horizon of the reconfiguration tree, planning and prediction are triggered anew.

2.3 Explaining

Based on the *explanation model*, our approach builds human-understandable explanations and visualizes the results. Initially, we focus on generating explanations for reconfiguration developers to understand why the system behaved a certain way at a specific time. The approach can be expanded to allow identification of relevant system components and reconfiguration rules which in turn allow to determine developers responsible for fixing misbehavior. Our

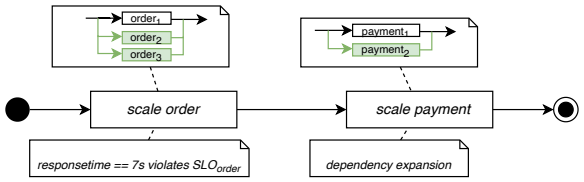

Figure 3: Explanations of two-rules-plan.

approach generates explanations on demand. If the system behaves suspiciously a developer can request an explanation. It also helps with problem identification.

For the example in Figure 1, a developer witnesses, that an additional instance of *payment* starts without any apparent reason, such as an SLO violation. For now, we assume that the developer already identified this point of interest. Instead we focus on how to connect the information provided by the digital twin to create the requested explanation.

To generate an explanation, we select data from all models of the twin into the explanation model, as depicted in Figure 2. We keep historical data, in case the developer inquires about past behavior and because past behavior might be part of the explanation of more recent behavior. For example, the twin prescribes a certain adaptation, because of the effects of a previously prescribed adaptation. In addition we keep rejected predictions, in case a developer asks for *why not* explanations. However, we cannot keep the entire history, nor all possible futures, thus we prioritize recent over older data and more likely over less likely futures.

From the explanation model, we generate explanations and present them to the developer. For a scenario as described above, we imagine a explanation visualized similar to the one depicted in Figure 3.

Rectangles represent the reconfiguration rules, for our above example these are *scale order* and *scale payment*. Sticky notes provide the textual reason for selecting a rule. Henshin diagrams visualize each rule's effect on the system configuration and the arrows between the rules represent their order of execution. In the depicted case, the developer asked about the new instances of *payment*. Because *scale payment* was planned in coordination with *scale order*, we visualize that rule as well. The developer can then recognize the witnessed effect of an additional *payment* instance and understand that this was caused by a violation of SLO_{order}.

In case the developer inquires about behavior that did not happen, we must include rules that were selected during planning, but not applied to the actual system, into the explanation as well. We visualize these with dotted outline, as depicted in Figure 4. In Figure 4 (a), two SLOs were violated, but only *scale order* was applied. The rule *scale payment* was applicable, but rejected because it was too expensive. In Figure 4 (b), *scale payment* is not executed, because the monitored data deviated from the predictions and the initial plan was discarded. In this case, the explanation must also represent the difference between the predicted and the actual system behavior, such that the developer understands the reason for replanning. In Figure 4 we omit the visualization of the rules' effects. Once the developer understands the system behavior, they might modify the SLOs, as depicted in Figure 2.

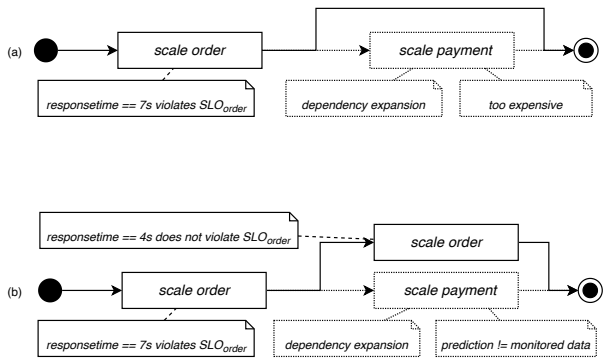

Figure 4: Explanations for plans with unexecuted rules.

3 RELATED WORK

We structure our discussion of related work based on the activities shown on the arrows in Figure 2. We sort them based on the number of activities supported and picked the most representative examples where needed. We begin with those works supporting the largest number. All works we found do not cover all activities needed but only subsets and none of them looks at the problem from the perspective of a cyber-cyber twin.

Klös et al. propose a rule-based self-learning and self-adaptive system [14]. They collect monitoring data into a knowledge base and use the knowledge to adjust the system's adaptation rules or create new rules if none of the existing rules is applicable. The approach covers most of the digital twin's activities, however, it lacks predictions and explanations in the decision-making process.

In another publication, Greenyer et al. argue, that the knowledge base is also used to construct human-understandable explanations of the applied and learned rules [11]. However, they consider the application of single, uncoordinated rules only, thus lacking the need and techniques to construct complex explanations for coordinated reconfigurations.

Bauer et al. [3] and Urgaonkar et al. [20] focus on reactive and predictive techniques to coordinate adaptations. They predict future workload and make coordinated scaling decisions based on queuing models. Unlike our approach, their adaptations are not rule-based and they provide no planing and corresponding explainability activity.

Regarding explainability, many works arose from the domain of explainable artificial intelligence (XAI) but without focus on digital twins [18]. This is helpful regarding concepts on *when*, *how* or *why* a user desires an explanation. In contrast to such approaches, however, we have an influence on which artifacts are generated and used during the decision-making process in the digital twin. This allows our explanations to work with less opaque relations.

Blumreiter et al. [6] and Chiyah Garcia et al. [8] propose a decision tree based technique to generate explanations about the decisions made in cyber-physical systems, for driver assistance systems in cars and for autonomous underwater vehicles, respectively. The root of the tree is the observed event and the other nodes are possible reasons for the event's occurrence. However, these trees

are limited, as they are created upfront and can only explain predicted decisions. This contradicts our desire to explain emergent states as well. In addition, Chiyah Garcia et al. propose natural language chatbots to navigate the explanations [8]. While intriguing for their domain, we assume that natural language processing is too imprecise to reason about the decision-making process in self-adaptive systems.

4 CONCLUSION

The envisioned usage of a digital twin in high-performance systems enables the self-adaptive system to systematically predict at runtime how potential reconfigurations (and sequences of reconfigurations) affect the satisfaction of service level objectives – also considering potential environmental changes. Furthermore, a digital twin enables explaining the chosen (and not-chosen) reconfigurations to engineers. We sketched in this paper how the concept of such a digital twin can be operationalized based on the conceptual framework by Eramo et al. [9] in terms of models and activities.

We believe that such a digital twin is not only relevant at runtime when operating high-performance systems. Additionally, it can already be used during development to shift left quality assurance activities w.r.t. to the service level objectives. Here, the actual system is replaced by simulation models.

We are currently developing a prototype of the presented digital twin for a simple demo shop system based on Palladio [16] and the Slingshot simulator [12].

ACKNOWLEDGMENTS

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 453895475.

REFERENCES

[1] John Ahlgren, Kinga Bojarczuk, Sophia Drossopoulou, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Simon M. M. Lucas, Erik Meijer, Steve Omohundro, Rubmary Rojas, Silvia Sapora, and Norm Zhou. 2021. Facebook's Cyber–Cyber and Cyber–Physical Digital Twins *(EASE'21)*. https://doi.org/10.1145/3463274.3463275

[2] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 13–23. https://doi.org/10.1109/SEAMS.2015.10

[3] André Bauer, Veronika Lesch, Laurens Versluis, Alexey Ilyushkin, Nikolas Herbst, and Samuel Kounev. 2019. Chamulteon: Coordinated Auto-Scaling of Micro-Services *(ICDCS'19)*. https://doi.org/10.1109/ICDCS.2019.00199

[4] Matthias Becker, Steffen Becker, and Joachim Meyer. 2013. SimuLizar: Design-Time Modeling and Performance Analysis of Self-Adaptive Systems *(SE'13)*.

[5] Steffen Becker, Gunnar Brataas, and Sebastian Lehrig. 2017. Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications: The CloudScale Method.

[6] Mathias Blumreiter, Joel Greenyer, Francisco Javier Chiyah Garcia, Verena Klös, Maike Schwammberger, Christoph Sommer, Andreas Vogelsang, and Andreas Wortmann. 2019. Towards Self-Explainable Cyber-Physical Systems *(MODELS-C'19)*. https://doi.org/10.1109/MODELS-C.2019.00084

[7] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*. Springer. https://doi.org/10.1007/978-3-642-02161-9_1

[8] Francisco Javier Chiyah Garcia, David A. Robb, Xingkun Liu, Atanas Laskov, Pedro Patron, and Helen Hastie. 2018. Explainable Autonomy: A Study of Explanation Styles for Building Clear Mental Models *(INLG'18)*. https://doi.org/10.18653/v1/W18-6511

[9] Romina Eramo, Francis Bordeleau, Benoît Combemale, Mark van den Brand, Manuel Wimmer, and Andreas Wortmann. 2022. Conceptualizing Digital Twins. *IEEE Softw.* (2022). https://doi.org/10.1109/MS.2021.3130755

[10] Greg Franks, Alex Hubbard, Shikharesh Majumdar, John Neilson, Dorina Petriu, Jerome Rolia, and Murray Woodside. 1995. A toolset for performance engineering and software design of client-server systems. *Performance Evaluation* 24, 1 (1995), 117–136. https://doi.org/10.1016/0166-5316(95)96869-T Performance Modeling Tools.

[11] Joel Greenyer, Malte Lochau, and Thomas Vogel. 2019. Explainable Software for Cyber-Physical Systems (ES4CPS): Report from the GI Dagstuhl Seminar 19023, January 06-11 2019, Schloss Dagstuhl. *CoRR* (2019). http://arxiv.org/abs/1904.11851

[12] Julijan Katic, Floriment Klinaku, and Steffen Becker. 2021. The Slingshot Simulator: An Extensible Event-Driven PCM Simulator (Poster) *(SSP'21)*. http://ceur-ws.org/Vol-3043/poster7.pdf

[13] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *IEEE Computer* (2003). https://doi.org/10.1109/MC.2003.1160055

[14] Verena Klös, Thomas Göthel, and Sabine Glesner. 2018. Comprehensible and dependable self-learning self-adaptive systems. *J. Sys. Arc.* (2018). https://doi.org/10.1016/j.sysarc.2018.03.004

[15] Sara Mahdavi-Hezavehi, Vinicius H. S. Durelli, Danny Weyns, and Paris Avgeriou. 2017. A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems. *Inf. Softw. Technol.* (2017). https://doi.org/10.1016/j.infsof.2017.03.013

[16] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann. 2016. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press.

[17] Gabriel S. Rodrigues, Ricardo Caldas, Gabriel Araujo, Vicente de Moraes, Genaína Nunes Rodrigues, and Patrizio Pelliccione. 2022. An architecture for mission coordination of heterogeneous robots. *J. Syst. Softw.* (2022). https://doi.org/10.1016/j.jss.2022.111363

[18] Erico Tjoa and Cuntai Guan. 2021. A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI. *IEEE TNNLS* (2021). https://doi.org/10.1109/TNNLS.2020.3027314

[19] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. 2005. An Analytical Model for Multi-Tier Internet Services and Its Applications. *SIGMETRICS Perform. Eval. Rev.* (2005). https://doi.org/10.1145/1071690.1064252

[20] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. 2005. Dynamic Provisioning of Multi-tier Internet Applications. In *ICAC'05*. https://doi.org/10.1109/ICAC.2005.27