Development and Integration of Self-Adaptation Strategies for Robotics Software

Alberts, E.G.

DOI
10.1109/ICSA-C57050.2023.00038

Publication date
2023

Document Version
Final published version

Published in
Proceedings of the IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023

Citation (APA)
Alberts, E. G. (2023). Development and Integration of Self-Adaptation Strategies for Robotics Software. In *Proceedings of the IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023* (pp. 131-136). IEEE. https://doi.org/10.1109/ICSA-C57050.2023.00038

2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C) | 978-1-6654-6459-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSA-C57050.2023.00038

Development and Integration of Self-Adaptation Strategies for Robotics Software

Elvin Alberts
Vrije Universiteit Amsterdam & Technical University of Delft
Amsterdam & Delft, The Netherlands
e.g.alberts@vu.nl

***Abstract*—Robots are becoming more prevalent in industry and society as a whole. Alongside this growth their application domain is also broadening. Each application brings with it a host of potential uncertainties that the robots should be able to handle at runtime. To tackle this, the doctoral thesis outlined in this paper proposes to address three main problems. First, the current ad-hoc state of robotics software which impedes its evolution. Second, the inability to imagine every possible uncertainty at design time leading to unexpected scenarios at runtime. Third, unexpected scenarios resulting from the reality gap between the simulated environments in which robots are developed versus the real world. These unexpected scenarios may cause a system to violate its requirements, especially in our case non-functional requirements. In an attempt to solve these problems, we plan to implement a variety of self-adaptation strategies. These strategies allow systems to change their composition to handle the aforementioned unexpected events during operation autonomously. To accomplish this we will need to reason about how best to integrate these strategies into the software of existing robots, as well as how existing information available to designers regarding the robots can best be utilized to improve the strategies. Lastly, these strategies and the process through which they are integrated will be assessed in their impact across different robotic case studies. Preliminary results from the work towards the thesis are also presented, alongside a consideration of its potential industrial impact.**

***Index Terms*—robotics, runtime uncertainty, self-*, reinforcement learning, non-functional requirements**

I. Introduction

As the prevalence of robots increases in industry and society as a whole, so too does their application domain expand. For broader applications, expectations of robots, and by extension the software which drives them increase. To meet these expectations one dimension to improve the functioning of robotics software is endowing it with *self-adaptive* capabilities. Self-adaptivity here entails the ability to handle scenarios not accounted for at design time which may cause a violation of system requirements. This runtime uncertainty should be addressed inherently by the software controlling the robots to outright prevent or mitigate failures. More specifically, Weyns [1] defines a self-adaptive system as having two key principles, the external principle which is to handle changes and uncertainties autonomously, and the internal principle that this should comprise of a managed system responsible for domain concerns and a managing system responsible for adaptation concerns. The external principle is not new to robotics as software components exist which allow robots to handle uncertainties to accomplish some goal e.g. avoiding obstacles during navigation. The challenge we focus on is how to realize the internal principle in a reproducible and extensible way across different robots.

Particularly, the managing system requires architectural self-adaptation strategies to reason through for the system at runtime. By architectural self-adaptation we mean the runtime configuration of available components and their parameterization, in alignment with the work by Camara et al. [2]. Devising these strategies to choose between the alternative parameterizations and configurations and their implementation in a way non-specific to any one system and its application domain is necessary. Furthermore, of interest is the ability of these strategies to handle unpredictable uncertainties. This entails that the uncertainties are unanticipated and cannot be comprehensively accounted for at design time. Thus, self-adaptation strategies must be able to adapt to runtime uncertainty in an online fashion, determining which adaptations are most beneficial at execution. Through these mechanisms the possibility of a system's requirements being met over time is increased in turn contributing to the likelihood of success in a robot's mission. For example, by meeting a non-functional requirement of energy efficiency during execution, a robot may buy itself extra time to complete its mission if due to some external reason it is delayed during it despite a designer's expectation being that its large battery would be able to handle the mission while running at full capacity.

In this paper the foundation and planning for the author's doctoral thesis is described. In order to establish the state of the art the related work is discussed in Section II. Section III describes the problems that will be tackled and motivates them. Section IV builds on this by describing the research questions that will answered to address the problems. Preliminary work done within this plan on thus far is described in Section V, and the envisioned industrial impact of the work is described in Section VI. Lastly, we conclude the paper in Section VII.

II. Related Work

A. Robotics Software Architecture

Ultimately, the concern of this thesis is having software architectures which enable self-adaptation with the goal of meeting non-functional requirements at runtime despite potential unexpected scenarios. Related, Robot Quality Metrics (RoQME) which is an integrated technical project of

2768-4288/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSA-C57050.2023.00038

RobMoSys [3], provides a meta-model which models non-functional properties with contextual information and provides Quality of Service (QoS) metrics on the basis of these properties measured using generated components. This allows for self-adaptation as choices can be made based on non-functional properties via QoS metrics such as safety using decision-making algorithms e.g. reinforcement learning [4]. RoQME is promising but not fully realized as it has not been applied to real-world application of robotics systems as of yet. Software development of robotic systems is as a whole not a uniform process. However, one uniformity that is present is that robotic software frameworks tend to take the form of component-based systems. Prominent component-based frameworks are ROS [5], Orocos [6], OpenRTM [7], and OPRoS [8]. For this thesis we currently choose to focus primarily on ROS due to its high adoption [9]. Regarding robotics software architecture, Malavolta et al. [10] have done a survey to determine SoTA of architecting ROS-based systems and derive guidelines for doing so. They conclude that a focus on reusable architectures is lacking among the software projects they considered. Besides reusability, the Best Practice in Robotics (BRICS) [11] and its associated component model advocates for the use of model-driven engineering for the development of robotics software to achieve separation of concerns which is concluded to be absent when using frameworks such as ROS in isolation.

B. Self-adaptation Approaches

There has been research in the self-adaptive systems community on providing robotics systems with self-adaptation capabilities. We describe their adaptations loops and approaches to introduce self-adaptation to a robotic system. Cheng et al. authored a paper about the AC-ROS robotic system [12]. This system adapts to meet safety and energy non-functional requirements while it patrols. Assurance cases are used to implement a self-adaptation loop, utility functions are used to evaluate if goals are met, and if not adaptations are made to assure they are. Camara et al. [2] consider enabling self-adaptation in mobile service robots. In this scenario, the completion of a mission is traded off against energy consumption, safety, and speed. Their self-adaptation approach is formalized, available configurations of all components are considered and validated through probabilistic model checking, quantitative analysis is then used to determine which configuration to adapt to. Body Sensor Network [13] as the name suggests is different from the previous related work in that it concerns a network of sensors with an e-health applications powered by ROS. A managing system monitors the tradeoff between QoS metrics, and use control theory to keep these within desired thresholds. Aguado et al. [14] consider an underwater robot, this dealing with thruster failures. They accomplish self-adaptation using an ontological reasoner, based on a knowledge representation of the system and its requirements. The reasoner analyzes the system state and should this be invalid adapts to another configuration based on design-time estimations. The extension of this work features in the preliminary results of this thesis in Section V. From the works detailed above and other relevant ones [15]–[17] we note that there is a lack of reuse in the approaches for enabling self-adaptation for robotics. Therefore, new work risks unnecessarily re-inventing the wheel, with a potentially worse result.

C. Dealing with Unexpected Adaptation Scenarios

To handle unexpected adaptation scenarios, one approach is the use of online planning. This entails that online learning algorithms are used in the planning phase (deciding on a particular adaptation) of self-adaptation once the need for adaptation has been determined. There are numerous examples of the usage of online learning within self-adaptation, not only for planning [18]. A prominent approach is reinforcement learning, which allows an system to learn the benefit associated with particular adaptations at runtime. In our previous work [19], [20] as well as that of Porter et al. [21] a subset of reinforcement learning algorithms, multi-armed bandits, are used to accomplish this for a self-adaptive web application. This approach will be covered further in the preliminary results in Section V. Kim and Park [22] also use reinforcement learning, this time the popular algorithm Q-learning to create self-adaptive agents for a robot battle simulator. These agents uncover winning combat strategies at runtime, faced with unpredictable environments due to the acts of other autonomous agents. Therefore a system can explore choices and then exploit the best one when encountering an unexpected adaptation scenario. Outside of reinforcement learning, Kinneer et al. use genetic algorithms to enable the re-use of plans to handle unexpected adaptation scenarios [23].

III. Problem Statement and Motivation

The main problem is that there is an emerging need for robotics systems to deal with unanticipated problems/uncertainties, yet no consolidated approach for doing so exists. This is due to the different dimensions that need to be considered such as a specific robot's mission, its existing software, the functionalities it provides, the strengths and weaknesses of differing self-adaptation strategies, and how these interplay. In the problems to follow, we motivate how the uncertainties self-adaptation should handle come about in robotics systems, while also considering how the state of existing robotics software hinders the integration of extensions such as self-adaptation.

P1 Ad-hoc state of robotics software There is a disconnect between software engineering/architecture and robotics software. Existing robotics software is ad-hoc in its design and implementation [24]. This impedes its evolution which is necessary to accommodate self-adaptation. Software is not developed as a first-class concern for robotics but rather as a tool to facilitate the mission at hand. Instead there is a default of developing architectures to suit the mission at hand without extensibility or reusability being a priority [10]. There are some standards in robotics software that can be relied on. A prominent one is the Robot Operating System (ROS), ROS describes itself as "a set of software libraries and tools that

help you build robot applications" [5]. Practically speaking, ROS provides a framework of reusable software components to control robots. Architecturally, it consists of 'nodes' or components which communicate with one another based on variations of the client-server model. Whether this imposed architecture will prove limiting for adding self-adaptive capabilities remains to be seen. The absence of a unified process underlying development of robotics software hampers this thesis' aim of having a general solution for self-adaptation in robotic systems. Simply put, there is less potential for reuse of existing robotic software.

P2 Failure of Imagination at Design Time By virtue of the complex operating environments associated with robotics, dealing with unpredictable uncertainties becomes a must as applications broaden. While robotic systems tend to have missions and controllers defined in great detail, there are (at least) two issues which still see them 'lose' control. The first of which, a 'failure of imagination' at design time constitutes P2, while the second forms the basis for P3. When designing a system, developers have some idea, possibly formulated as requirements which provide the scope for its functionality. However, it is infeasible to imagine every single possible obstacle faced or requirement needed of a system at runtime, prior to runtime creating the potential for failures. A clear-cut example would be an unexpected environmental change, such as a rare weather event affecting the deployment conditions of a robot (e.g. rendering a robot's sensor unusable). It is somewhat inevitable then that even in a system performing unaffected by (unexpected) uncertainties, as will be covered in P3, the potential for unknown scenarios increases as its applications do. For any system in which it is desirable to maintain long-term and sustainable execution through its various applications, the ability to adapt to new scenarios increases its chances to fulfill this.

P3 Uncertainties crawling out of the Reality Gap There is a gap between robot simulation software and the real world known as the reality gap. It is clear from robotics research [25] that there are persistent issues when migrating robotics solutions from the simulated environments in which they are developed to reality. A form of 'overfitting', solutions are designed while (possibly unintentionally) relying on the idiosyncrasies of the simulation software used. Additionally, sources of uncertainty which are not or cannot be simulated affect performance, possibly without being measurable during operation. Take for example the effect of temperature on the robot's hardware in real life, which is not simulated during development. It is not realistic either to strive towards simulating every single facet of the real world. Even if one were to recreate every factor in a particular mission, this arduous task would need to be repeated for every application of a robotic system.

IV. Research Plan

To attempt to solve the problems as described in Section III we propose the following research question: (RQ) How to build robotics software that may meet its non-functional requirements despite the uncertainty present in complex runtime environments? To answer this broader research question, we divide it into four smaller prompts.

RQ1: Which architectural self-adaptation strategies can be used for runtime adaptation in unexpected scenarios in robotics software? We use the term self-adaptation strategy in accordance with our earlier work [26]. There it is defined as algorithms and mechanisms that allow self-adaptive software systems to "change their behavior and structure at runtime to accommodate changes in their operational environment and maintain or even improve their functionalities and qualities without human intervention". These changes stem from uncertainties, therefore this research question addresses P2 and P3 as these strategies allow a system to handle uncertainties. To deal with the changes the strategy plans out one or more adaptation tactics. Our definition of an adaptation tactic aligns with that of an architectural tactic as defined by Bass et al. [27] as a reusable design decision that influences the achievement of a quality attribute. Specifically, the strategies that will be researched need to be endowed with the ability to plan *online*. Planning here refers to the choice in adaptation tactic to execute in response to a need for adaptation. This planning being online implies that the system decides which tactics are an appropriate response to stimuli at runtime and *not* preemptively at design time. One clear approach to accomplish this is machine learning, as the algorithms 'learn' correct tactics at runtime.

We plan to apply such algorithms, specifically from Reinforcement Learning in building on our previous work [19], as well as ontological reasoning [14] which has already successfully been applied to robotics software. Both of these are detailed further in Section V. Additionally we will iteratively develop these strategies by both attempting to improve existing ones and implementing new solutions. This, with the aim of maximizing the ability of robotics systems to handle uncertainties, especially those which are harder to predict.

RQ2: How to integrate architectural self-adaptation strategies in the development and operation of robotics software? Building on RQ1, this research question addresses P2 and P3 but also P1 as it concerns the integration of the planned solutions from RQ1 into existing robotics systems. Realizing runtime adaptation in (existing) robotics software is not technically straightforward. This especially considering the traditional focus of robotics systems on control and certainties. From the related work in Section II it is clear that various solutions for self-adaptation in robotic systems exist. What these solutions lack is a common reusable approach for extending robotics systems with this capability with generality.

We plan to create a framework which clarifies what the SoTA is when it comes to adaptive software architectures for robotics systems. This framework is planned to be based on a systematic review of different architectures used, building on their work. This should provide the basis for the approach we adopt in this thesis to introduce self-adaptation to existing robotic systems. As we apply this approach iteratively to more existing robotic systems the hope is that it becomes

more refined and nuanced while prioritizing extensibility and reusability.

RQ3: How to incorporate models of the robotic runtime uncertainty in architectural self-adaptation strategies? The self-adaptation strategies from RQ1 are inextricably reliant on an abstract *knowledge* of the system at hand. It is necessary then to define models to represent this knowledge for use at runtime. This can be aided by design time specifications such as ontologies from knowledge representation, which provide the guidelines for incorporating potential runtime observations. To accomplish 'online' planning, it also becomes necessary to have a concrete representation of state to *associate* with tactics to apply. For example, measuring the density of obstacles in the immediate surroundings of a robot [28]. This knowledge can then be exploited by the adaptation strategy to reason which tactics best meet a non-functional requirement of safety. These tactics would manifest as different re-configuration of the (possibly component-based) architecture to one which has a higher level of safety e.g. using parameterizations or alternatives to components which cause slower movement when the density of obstacles is high. In the paper by Hernandez et al. [29] the authors describe there being differing levels of automation possible in robotics depending on the assumptions made. Each specific system a self-adaptation strategy is being applied to has different assumptions that can be relied upon.

For this thesis, we plan to always explicitly outline when particular solutions are reliant on assumptions, especially those specific to system. By making their role clear, the allow for more reusability as potential incompatibilities for new applications are clarified. What this means practically is that the models used are informed as much as possible at runtime with system-specific details rather than being hardcoded at design-time. This allows broader usage of the same representations with the different self-adaptation strategies stemming from RQ1.

RQ4: How to assess the impact of the proposed development process and the self-adaptation strategies in different robotic cases studies? To combine the efforts RQ1-3, it is ultimately planned to evaluate through comparison both the strategies devised in RQ1 making use of RQ3, and the process to implement them stemming from RQ2. For the former, direct comparison can be done between adaptation strategies in how well they meet non-functional requirements during controlled execution scenarios. For the latter, the introduction of facets of the development process from RQ2 is measured against their absence and alternatives. As it is possible that choices made in the development process will differ in impact on a case-by-case basis it is necessary to isolate their effects on different systems through careful evaluation. This ultimately serves to evaluate the applicability of the development process and by extension the self-adaptation strategies themselves. We plan to consider at least two robotic case studies, with more being added should there be time.

To summarize the aims and expected outcomes detailed in each research questions, Figure 1 illustrates the cyclical steps that are to be taken in fulfilment of the thesis. It shows that while not dependent on the respective previous' completion, the research questions are sequential. The self-adaptation strategies of RQ1 are developed, then integrated in RQ2 while making use of the models of RQ3, and then finally evaluated. For the second iteration and onwards, a comparative analysis of self-adaptation strategies and their impact becomes possible within RQ4 as well.

V. Preliminary Results

In this section we describe the topics that are explored thus far as part of fulfilling the thesis. This mainly revolves around beginning to answer RQ1, leading into less work on RQ2 and RQ3, followed by some potentially useful developments for RQ4, in accordance with Figure 1.

A. Multi-Armed Bandits

As a potential adaptation strategy, this subsection addresses RQ1. Multi-Armed Bandits or MABs are a subcategory of Reinforcement Learning (RL) algorithms. These algorithms or policies learn which action should be taken based on a reward which is determined through a Monte Carlo method of exploring an action, sampling its reward, and committing this to knowledge for future exploitation. MABs differ from other RL algorithms in that they consider a singular state when formalized as a finite Markov decision process. This entails that actions taken do not directly change the state of the agent/system. Within MABs, contextual bandits exist which learn to associate this state and its actions with a *context* or side information. This extends the MDP formalism by adding states or contexts to which the transitions are triggered due to external causes. For example a website backend might face different contexts of user load which imply different rewards for having a number of servers active depending on the load. If there are few users visiting then an MAB may learn that having eight servers running is undesirable, but if there is a high load then the 'reward' for having eight servers would be much higher. In previous work [19] we have implemented a library[1] of MABs for use with self-adaptive systems. For the domain of robotics we anticipate contextual bandits to be necessary to associate learned adaptation actions with the context a robot is in. This allows specific tactics to be applied for those contexts rather than generally throughout a mission. For example, adjusting a robots configuration to reduce energy consumption by turning off a particular sensor may be worthwhile while it is in the context of navigating, but not in a context of scanning (using that sensor).

To learn to account for various contexts without prior knowledge of them (as to make fewer assumptions) challenges need to be overcome in deciding on the right parameterization of MABs. We have published a paper [20] which looks at this problem. We introduce the a metric called 'convergence inertia' which quantifies the difficulty both contextual and non-contextual yet flexible MABs have in dealing with a change in context. There is a so-called inertia in overcoming previous

[1]https://github.com/EGAlberts/MASCed_bandits

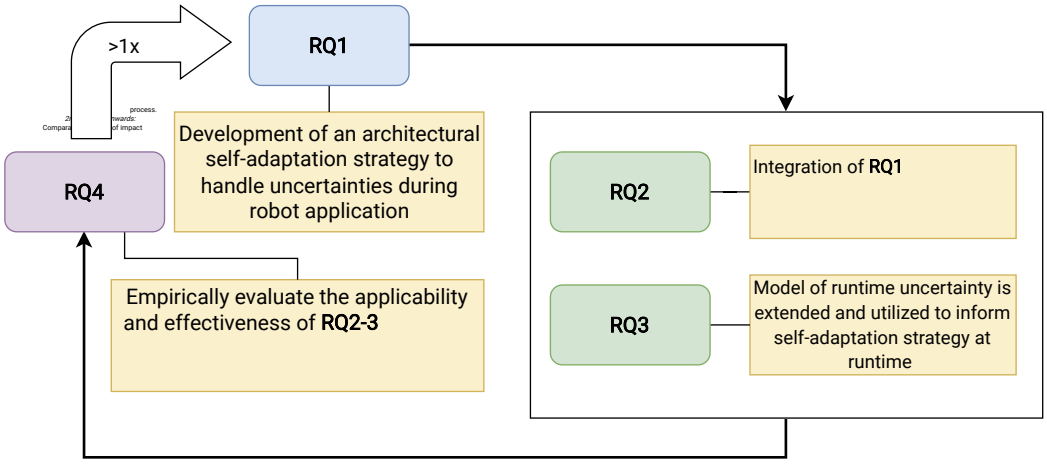

Fig. 1. Planned Research Process

knowledge to learn about the new context. This metric makes it possible to compare solutions depending on the application. For one application it may be more desirable to maintain knowledge of every context encountered and attempt to realize the re-occurrence of a context, while for others it may be desirable to forget previous knowledge and quickly commit to new contexts with a short memory.

B. Metacontrol

As a potential adaptation strategy, this subsection addresses RQ1; additionally, it is being integrated into a robotic system using knowledge representation and so addressed RQ2 and RQ3 as well. Metacontrol concerns using an ontology written in OWL to analyze the state of a self-adaptive robotic system. Through this ontology it can be reasoned whether the current state is untenable with regards to the non-functional requirements (NFRs) to be met, and an adaptation may be necessary. A selection is then made between adaptation tactics and applied to resolve the invalid state based on their known potential to meet the non-functional requirement. For example, if a robot is using too much energy to meet an energy-conscious non-functional requirement then the reasoning triggers an adaptation of re-parameterizating the components of the robot to use less energy. Currently we are working on a self-adaptive exemplar surrounding applying the metacontrol to an underwater robot. The underwater robot needs to make adaptations due to uncertainties that arise from a mission to discover an underwater pipeline. Adaptations include choosing different search patterns, as well as dealing with potential thruster failures and communicating these to the autopilot system onboard at runtime.

C. Evaluating Self-Adaptive Systems

With regards to RQ4, we have published a paper [30] which introduces a Python library to comparatively evaluate adaptation strategies without the need to invoke the real system. This allows evaluations to be done quicker and therefore with more intricacy. Additionally should a profile be detailed enough new scenarios can be extrapolated without any involvement of the real system. Profiles of the system in question are devised. Such a profile specifies the potential adaptation tactics of a system and how the context of a system may affect these. Adaptation strategies such as the aforementioned MABs may then be used with a simulation of these system conditions. Different paramerization of MABs can then be compared with the exact same conditions. The conditions can also be extrapolated (although in limited fashion) to consider system states otherwise more difficult to recreate in a real deployment. Besides comparing instances of strategies in the same group, the hope is that comparison can be done across strategies e.g. MABs vs. another form of reinforcement learning.

VI. Industrial Impact

The rate of installation of robots in the world is at an all-time high[2]. These robots to varying degrees require software to operate, as mentioned in the introduction. We intend to participate in events such as the ROS-Industrial conference to discuss the work done as a part of this PhD as well as be informed of developments in the robotics industry. The work in this PhD is planned to see us collaborate with the Horizon Europe CoreSense[3] project. Work done for the CoreSense project and other parts of this thesis will be released open-source, where relevant through the ROS suite or otherwise. The different systems that will be selected in answering RQ4 will be chosen with realistic use cases as a priority. At the time of writing these include an underwater remote vehicle, a mobile manipulator, and a terrestrial robot. These applications of these systems will likely mirror those found in industry, as is also often the case with self-adaptive exemplars.

[2]https://ifr.org/ifr-press-releases/news/wr-report-all-time-high-with-half-a-million-robots-installed

[3]http://www.coresense.eu

VII. Conclusion

This paper has outlined the foundation and planned work for the doctoral thesis to be completed by the author. After introducing the topic, we outline the SoTA in the related work, describing the landscape of robotics software architecture, different approaches to self-adaptation and research tackling unpredictable uncertainties. We then state and motivate the core problems which the thesis will try to address. These entail the presence of unpredictable uncertainties at runtime for robotic systems, and the ad-hoc nature of existing robotics software. To address these problems we proposed a series of research questions and outlined the planned process by which these will be answered. This will see the development and integration of self-adaptation strategies aided by runtime representative models followed by evaluation as to the impact in meeting non-functional requirements despite the uncertainties. Lastly, we have outlined the preliminary work which has been done towards the thesis thus far, and considered its potential industrial impact.

References

[1] D. Weyns, *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley & Sons, 2020.

[2] J. Cámara, B. Schmerl, and D. Garlan, "Software architecture and task plan co-adaptation for mobile service robots," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 125–136.

[3] J. F. Inglés-Romero, J. Martínez, D. Stampfer, A. Lotz, M. Lutz, and C. Schlegel, "A component-based and model-driven approach to deal with non-functional properties through global qos metrics," in *MODELS Workshops*, 2018.

[4] J. F. Inglés-Romero, J. M. Espín, R. Jiménez-Andreu, R. Font, and C. Vicente-Chicote, "Towards the use of quality-of-service metrics in reinforcement learning: A robotics example." in *MoDELS (Workshops)*, 2018, pp. 465–474.

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[6] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the orocos project," in *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)*. IEEE, 2003.

[7] N. Ando, T. Suehiro, and T. Kotoku, "A software platform for component based rt-system development: Openrtm-aist," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2008, pp. 87–98.

[8] B. Song, S. Jung, C. Jang, and S. Kim, "An introduction to robot component model for opros (open platform for robotic services)," in *Proceedings of the International Conference Simulation, Modeling Programming for Autonomous Robots Workshop*, 2008, pp. 592–603.

[9] B. Wire, "The rise of ros," May 2019. [Online]. Available: https://www.businesswire.com/news/home/20190516005135/en/

[10] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, and D. Garlan, "How do you architect your robots? state of the practice and guidelines for ros-based systems," in *2020 IEEE/ACM 42nd international conference on software engineering: software engineering in practice (ICSE-SEIP)*. IEEE, 2020, pp. 31–40.

[11] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The brics component model: a model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[12] B. H. Cheng, R. J. Clark, J. E. Fleck, M. A. Langford, and P. K. McKinley, "Ac-ros: assurance case driven adaptation for the robot operating system," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 102–113.

[13] E. B. Gil, R. Caldas, A. Rodrigues, G. L. G. da Silva, G. N. Rodrigues, and P. Pelliccione, "Body sensor network: A self-adaptive system exemplar in the healthcare domain," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021, pp. 224–230.

[14] E. Aguado, Z. Milosevic, C. Hernández, R. Sanz, M. Garzon, D. Bozhinoski, and C. Rossi, "Functional self-awareness and metacontrol for underwater robot autonomy," *Sensors*, vol. 21, no. 4, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1210

[15] D. De Leng and F. Heintz, "Dyknow: A dynamically reconfigurable stream reasoning framework as an extension to the robot operating system," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, 2016, pp. 55–60.

[16] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus, "Architecture-driven self-adaptation and self-management in robotics systems," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009, pp. 142–151.

[17] S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns, "Undersea: an exemplar for engineering self-adaptive unmanned underwater vehicles," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 83–89.

[18] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 3, 2021.

[19] E. G. Alberts, "Adapting with regret: Using multi-armed bandits with self-adaptive systems," Master's thesis, University of Amsterdam, 2022. [Online]. Available: https://scripties.uba.uva.nl/search?id=727497

[20] E. Alberts and I. Gerostathopoulos, "Measuring convergence inertia: Online learning in self-adaptive systems with context shifts," in *Proceedings of the 2022 International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, to appear*, Oct. 2022.

[21] B. Porter and R. Rodrigues Filho, "Distributed emergent software: Assembling, perceiving and learning systems at scale," in *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2019, pp. 127–136.

[22] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2009, pp. 76–85.

[23] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. L. Goues, "Managing uncertainty in self-adaptive systems with plan reuse and stochastic search," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, 2018, pp. 40–50.

[24] F. Ciccozzi, D. Di Ruscio, I. Malavolta, P. Pelliccione, and J. Tumova, "Engineering the software of robotic systems," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 507–508.

[25] J.-B. Mouret and K. Chatzilygeroudis, "20 years of reality gap: a few thoughts about simulators in evolutionary robotics," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 1121–1124.

[26] I. Gerostathopoulos, C. Raibulet, and E. Alberts, "Assessing self-adaptation strategies using cost-benefit analysis," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2022, pp. 92–95.

[27] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[28] D. Bozhinoski and J. Wijkhuizen, "Context-based navigation for ground mobile robot in semi-structured indoor environment," in *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2021, pp. 82–86.

[29] C. H. Corbato, M. Bharatheesha, J. Van Egmond, J. Ju, and M. Wisse, "Integrating different levels of automation: Lessons from winning the amazon robotics challenge 2016," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, 2018.

[30] E. Alberts, I. Gerostathopoulos, and T. Bures, "Mocksas: Facilitating the evaluation of bandit algorithms in self-adaptive systems," in *Companion Proceedings of the 2022 European Conference on Software Architecture, to appear*, Sep. 2022.