Software reconfiguration in robotics

Empirical Software Engineering (2025) 30:94
https://doi.org/10.1007/s10664-024-10596-9

Software reconfiguration in robotics

Sven Peldszus[1] · Davide Brugali[2] · Daniel Strüber[3,4,5] · Patrizio Pelliccione[6,7] · Thorsten Berger[1,3,4]

Accepted: 19 November 2024

Abstract
Robots often need to be reconfigurable—to customize, calibrate, or optimize robots operating in varying environments with different hardware. A particular challenge in robotics is the automated and dynamic reconfiguration to load and unload software components, as well as parameterizing them. Over the last decades, a large variety of software reconfiguration techniques has been presented in the literature, many specifically for robotics systems. Also many robotics frameworks support reconfiguration. Unfortunately, there is a lack of empirical data on the actual use of reconfiguration techniques in real robotics projects and on their realization in robotics frameworks. To advance reconfiguration techniques and support their adoption, we need to improve our empirical understanding of them in practice.
We present a study of automated reconfiguration at runtime in the robotics domain. We determine the state-of-the art by reviewing 78 relevant publications on reconfiguration. We determine the state-of-practice by analyzing how four major robotics frameworks support reconfiguration, and how reconfiguration is realized in 48 robotics (sub-)systems. We contribute a detailed analysis of the design space of reconfiguration techniques. We identify trends and research gaps. Our results show a significant discrepancy between the state-of-the-art and the state-of-practice. While the scientific community focuses on complex structural reconfiguration, only parameter reconfiguration is widely used in practice. Our results support practitioners to realize reconfiguration in robotics systems, as well as they support researchers and tool builders to create more effective reconfiguration techniques that are adopted in practice.

Keywords Software reconfiguration · Robotics · State of the art · State of practice

1 Introduction

Robots are increasingly deployed in our lives. Being multi-purpose, they can operate in a variety of safety-critical environments, such as private homes, hospitals, restaurants, factories, and museums. Such robots have a variety of mobility and manipulation devices, as well as redundant sensors brought together in a robotic control system—"*an interconnection of components forming a system configuration that will provide a desired system response*" (Dorf and Bishop 2011).

Communicated by: Hélène Waeselynck

Extended author information available on the last page of the article

Published online: 08 April 2025

Robots are concurrent and distributed software systems (Aarsten et al. 1996) composed of many components. Configuring and assembling these components—many with alternative implementations—is essential to customize robotic systems towards different hardware, runtime environments, or non-functional properties (e.g., performance or energy consumption). In fact, the initial release of the Robot Operating System (ROS) (Cousins et al. 2010) in 2010 already contained hundreds of open-source components stored in 15 repositories, accounting for customization needs and enhancing robot versatility to changing application requirements. To this end, so-called configuration mechanisms enabling customization are considered essential by robot manufacturers and developers (García et al. 2020b; García et al. 2022).

Configuration determines which components of a system are present and active, and how they are connected. It also instantiates the component parameters. Configuration is often not static, but *reconfiguration* is typically needed to assure the correct and trustworthy operation of robotic systems. Reconfiguration (Kortenkamp et al., 2016) often involves the activation or deactivation of components, the modification of their connections, the replacement of control and functional algorithms, and changing control parameters at runtime. Also, reconfiguration needs to be triggered, which is usually provided via specific interfaces. The topic of reconfiguration is quite broad. This article focuses on automated software reconfiguration at runtime in robotics control systems. Static configuration and manual reconfiguration, but also related topics under the umbrella of self-adaptation, are beyond the scope of this article.

Self-adaptation is the ability of a system to dynamically adapt to unexpected environmental changes and failures. Self-adaptation subsumes reconfiguration. It allows a system to adapt its configuration when the conditions change, to deal with uncertainties that are difficult or impossible to anticipate before deployment (Calinescu et al. 2020). Uncertainty is a system state of incomplete or inconsistent knowledge caused, e.g., by unpredictable phenomena in the execution environment or incomplete and inconsistent information obtained by potentially imprecise, inaccurate, and unreliable sensors (Ramirez et al. 2012). At runtime, a self-adaptive system collects additional information to resolve the uncertainty and adapt itself (Calinescu et al. 2020). To this end it may use reconfiguration. For example, let us assume that a configuration parameter is the nominal maximum speed of a robot, which depends on the robot kinematics. Then, the actual maximum speed of the robot is adapted (i.e., reduced) by using reconfiguration for meeting safety requirements, taking also environment uncertainty (e.g., wet floor) into account. When the robot is out of safety-critical zones or conditions, the maximum speed is adapted (i.e., increased). Finally, the actual speed is continuously regulated according to the path geometry. As an alternative to reconfiguration, self-adaptation may also use other techniques, i.e., internal variables, as an alternative to configuration parameters to achieve the intended behavior.

To enable effective and safe reconfiguration as described above, specific implementation techniques are needed. To this end, the scientific community has conducted extensive research on software reconfiguration of robotic systems. At the same time, popular robotics frameworks, such as the Robotic Operation System (ROS) (Quigley et al. 2009), YARP (Metta et al. 2006; Fitzpatrick et al. 2008), or Smartsoft (Lotz et al. 2013b) provide mechanisms for implementing reconfiguration. However, surprisingly little is known about the adoption and characteristics of reconfiguration mechanisms in practice. While many secondary studies on the physical reconfiguration of modular robots and robot swarms (see Sec. 2) exist, to the best of our knowledge, there is no systematic overview of available technologies for designing reconfigurable robotic systems. To improve the situation, *the state of the art in robot software reconfiguration needs to be assessed*—the first motivation for our study.

Reconfiguration can be even more difficult in practice. Static configuration already adds substantial complexity to the design of a software system (Berger et al. 2013), where reusable software components are selected and integrated during deployment according to the specific requirements of the robot embodiment, task, and environment. Empowering the robot with the ability to dynamically self-reconfigure at runtime according to context changes requires to face additional challenges. For instance, developers need to (i) define triggers of reconfiguration, (ii) declare constraints among the configuration options, consistent with the domain knowledge and the actually implemented system, and (iii) assure that reconfiguration is not only timely, but also puts the system into the desired, and valid state of operation. To improve the situation, *the state of the practice in robot software reconfiguration needs to be assessed*—the second motivation for our study.

It can be argued that tool and framework builders should enhance the reconfiguration mechanisms to benefit practitioners, who could then build more reliable and safe reconfigurable robotics software (Dalal et al. 1993). However, to the best of our knowledge, there are no studies or at least experience reports that address the practical implementation of dynamic reconfiguration. This lack of empirical data impedes the development of effective reconfiguration methods and tools, as well as the scoping of research projects and the selection of relevant research directions.

To address these shortcomings, we present a comprehensive review of the literature on reconfiguration of robotic software systems (state-of-the-art) and an in-depth analysis of reconfiguration support in robotic frameworks, as well as the implementation of reconfiguration in robotic (sub-)systems (state-of-practice). Our research questions are:

RQ1: *What are the motivations for developing dynamically reconfigurable robotic software systems?*

We aim at understanding the needs of reconfiguration in robotics, e.g., for enhancing performance, robustness, and reusability. To this end, we analyze the literature on robotic software reconfiguration.

RQ2: *What software parts of a reconfigurable robot control system are reconfigured and at what granularity?*

When talking about reconfiguration in robotics, it is often not clear which parts of a robotic system are adapted to needs using reconfiguration or maybe other mechanisms. We aim at understanding what is commonly considered in reconfiguration and at which granularity, e.g., entire components or single software functionalities. To this end, we address this question from two perspectives, first by analyzing the literature on robotic software reconfiguration, and second by also reviewing robotic frameworks in terms of the granularity of reconfiguration that they support.

RQ3: *What mechanisms exist and how they are used for developing reconfigurable robotic software systems?*

We aim at understanding the mechanisms used for specifying and performing the reconfiguration, together with implementation practices. To this end, we analyze the literature on robotic software reconfiguration, we review robotic frameworks and their support for reconfiguration, and we analyze how researchers and practitioners implement reconfiguration in their robotic (sub-)systems to answer RQ3.

Based on the academic literature, robotic frameworks, and robotic (sub-)system implementations, we address reconfiguration from two perspectives: (i) an academic perspective based on our systematic literature review (SLR) and (ii) a practitioner's perspective based on how robotic frameworks address reconfiguration and how reconfiguration is implemented in real robotic (sub-)systems. We synthesize the individual perspectives to identify discrepancies and common perceptions of software reconfiguration in robotics, thereby providing a comprehensive view of reconfiguration. Additionally, we contrast the state of practice in reconfiguration of robotic systems with reconfiguration in other domains and derive implications for researchers and practitioners.

2 Background and related work

In this section, we first introduce background and related work on reconfiguration in robotics, and thereafter, an overview of six robotic frameworks.

2.1 Overview of reconfiguration in robotics

A first initial search for systematic literature reviews and surveys on reconfiguration of robotic systems (via ACM Digital Library, Scopus, and IEEE Explore) by March 2024 revealed that the terms *"reconfigurability"* or *"reconfigurable"* or *"reconfiguration"* have been used in the robotics literature with different meanings depending on the type of robotic system.

The common ground on which all studies agree is that reconfigurability implies an ease of modification and an absence of irreversible or rigid commitments in some aspects of the robotic system (mostly related to its embodiment) and the potential to assume different arrangements of the constituent elements.

In the last 30 years, surveys have been published on three classes of reconfigurable systems. Robots composed of multiple identical modules can be mechanically assembled to form systems with different physical shapes (e.g., a snake robot or a spider robot) and can autonomously reconfigure their shape for adapting to different environments (Dudek et al. 1993; Yim et al. 2007; Moubarak and Ben-Tzvi 2012; Ahmadzadeh and Masehian 2015; Chennareddy et al. 2017; Alattas 2018). Robots composed of modules with specialized functionalities can self-reconfigure to perform different tasks (Liu et al. 2016). Swarms of mobile robots (e.g., swarms of UAVs) that can reconfigure the swarm topology resulting from coalition formation during the execution of cooperative tasks (Abukhalil et al. 2015; Shlyakhov et al. 2017).

The Robotics 2020 Multi-Annual Roadmap (MAR) (SPARC 2016) provides a more general definition of configurability, as "*the ability of the robot to be configured to perform a task or reconfigured to perform different tasks.*" This may range from the ability to re-program software modules and components to being able to alter the configuration of sensing and other electronic systems and the mechanical structures of the system.

Moreover, the MAR distinguishes reconfiguration from the concept of adaptability, which is defined as "*the ability of the system to adapt itself to different work scenarios, different environments and conditions*" and implies that the system performs optimization against some performance criteria.

Interestingly, two secondary studies investigated the intriguing relationship between reconfigurability and adaptability and are specifically relevant for our investigation because they introduce the concept of dynamic reconfiguration. More specifically, Fornari and de Santiago Júnior (2019) define a Dynamic Reconfigurable System as "*a system whose subsystems can be modified or have their configurations changed during operation; dynamic reconfiguration enables real-time systems adaptation.*" The paper focuses on two specific aspects: the reconfiguration of the computing hardware (mostly FPGAs) and, although only marginally, the reconfiguration of the robot control system (e.g., the autonomous navigation system). Tan et al. (2020) define self-reconfigurable robots as "*machines that can change their morphologies as per prescribed requirement or are adaptable to the environments with provided level of autonomy.*" In turn, autonomous reconfigurability is described as "*the extent to which a self-reconfigurable robot can sense its environment, plan its configuration based on that environment, and act to transform into specific configurations upon that environment with the intent of achieving some goal.*"

In the context of this paper, we focus on automated runtime reconfiguration of the software control system of autonomous robots. In order to clarify the scope of the paper, it is useful to refer to concepts and definitions that we have found in some works included in our study.

Stewart et al. (1997) argue that the need for dynamic reconfiguration stems from the need to change control algorithms on the fly to support more intelligent control strategies.

Jamshidi et al. (2019) propose a technique to enable self-adaptation of robots operating in dynamic and uncertain environments, using configuration change as the main mechanism to enact adaptation. In their paper, robotics software is considered as a highly-configurable system, in which system characteristics (e.g., usage of sensors) are treated as configuration options.

Macdonald et al. (2004) explain that robot software reconfiguration requires specific mechanisms for component deployment. In some cases, state information (e.g., map data or sensor data history) must be transferred to a newly deployed software component during robot operation without service disruption.

Pham et al. (2000) discuss three forms of software adaptation in robotic systems: parametric fine tuning, algorithmic change, and task migration to remote computational resources in a distributed environment.

2.2 Robotic Frameworks

During the last twenty years, several research and development projects have produced a variety of component-based robotic frameworks (see the surveys of Brugali et al. 2007 and Elkady and Sobh 2012) that in many cases build on state-of-the-art middleware infrastructures (e.g., CORBA OMG 2012 and DDS OMG 2015). These frameworks provide domain-specific software abstractions that are amenable to robotics experts and hide the complexity of middleware mechanisms for real-time execution of concurrent control activities, synchronous and asynchronous communication between components, dynamic wiring of component interfaces, remote configuration of component properties, and runtime loading of plug-in functionality. In the following, we briefly introduce some of these frameworks.

GenoM (Fleury et al. 1997) is a component-oriented software framework developed by the LAAS CNRS robotics group. Components interface hardware devices or encapsulate common robotics algorithms. GenoM components are collections of control services, which manage incoming requests, implement specific algorithms, and execute services. The framework is at the basis of the LAAS Architecture (Alami et al. 1998), one of the most relevant examples of robot architectures that enforces software quality attributes, such as modularity, maintainability, and usability.

The Claraty (Volpe et al. 2001) framework has been developed with the specific goal of improving the modularity, interoperability, and reusability of the control software operating the large variety of NASA/JPL autonomous robots for planetary exploration. The framework provides mechanisms for encapsulating perception capabilities, robot action, and control loops that are encapsulated into software components, which can be activated by the decisional level.

OROCOS is one of the oldest open-source frameworks in robotics, under development since 2001. Professional industrial applications and products use it since 2005. Its focus has always been to provide a hard real-time component framework, the so-called Real-Time Toolkit (RTT), that is as independent as possible from any communication middleware and operating system.

YARP (Metta et al. 2006; Fitzpatrick et al. 2008) is a multiplatform and multiprotocol communication framework for robotic research. Available protocols are tcp, udp, multicast, shared memory, and protocols for interfacing with ROS. Typical YARP applications consist of several intercommunicating modules distributed on different machines that exchange messages according to a port-based publisher/subscriber protocol. YARP is the reference software platform for the iCub humanoid robot designed by the Italian Institute of Technology to help developing and testing embodied AI algorithms. The iCub robot is currently used by more than thirty research institutions worldwide.

RobMoSys (RobMoSys 2023) is a EU-funded research project that provides interfaces, methods, and tools for the model-driven development of robotic systems. It enables the management of interfaces between different robotics-related domains, roles, and levels of abstractions. Its interfaces can either be implemented by concrete frameworks or wrappers around low-level implementations mapping them to the RobMoSys APIs. For our study on software reconfiguration, we consider the SmartSoft framework (Schlegel and Worz 1999; Schlegel et al. 2013) as a reference implementation of RobMoSys. While the *RobMoSys* ecosystem, like many tools developed by academia, suffers the problem of sustainability after the completion of the project, the SmartSoft framework has reached the Technology Readiness Level (TRL) 6 and is used in industrial projects in collaboration with several companies, including Bosch, REC, and FESTO.

ROS (Open Robotics 2007) is a mature open-source robotics middleware that provides a framework for robotic software. It allows implementing modular robotic applications in multiple programming languages and provides services to realize the interaction of modules. Many state-of-the-art algorithms have been developed for ROS, and major robotic systems, such as the self-driving vehicle software Autoware.auto, are implemented using ROS. The transition from ROS to ROS 2 (Macenski et al. 2022) was accompanied by the definition of a more standard structure of ROS packages (e.g., the navigation stack), which use behavior trees to orchestrate the behavior of many concurrent functionality and their runtime reconfiguration. What is still missing are reference architectures that guide application developers in the integration of ROS packages in whole applications.

Malavolta et al. (2021) mined robotics software repositories to extract guidelines for building robotic systems. Although the extracted guidelines do not explicitly address reconfiguration, they cover aspects necessary for building reconfigurable systems. First, there are guidelines that focus on increasing component cohesion and reducing component coupling, which are essential for reconfiguration. Second, extracted design guidelines focus on components that provide interfaces to adjust their operation, such as changing the frequency at which sensors collect data. Third, guidelines focus on providing status and health information, which can be seen as potential triggers for reconfiguration. Finally, some guidelines focus on separating the monitoring of such health information from the functional implementations.

3 Methodology

Since we want to study how automated robot reconfiguration at runtime is addressed by the scientific community and contrast it with how it is implemented by practitioners, we decided to study three different types of artifacts to answer the research questions. As illustrated in Fig. 1, we studied the state-of-the-art and the state-of-practice on reconfiguration in the robotics domain in three phases. Thereby, we applied two research methodologies (systematic reviews and repository mining), as described by Ralph et al. (2023), depending on the investigated artifacts:

(i) **Literature:** *Systematic Review* of the literature on the automated reconfiguration of robotic software systems at runtime.
(ii) **Frameworks:** *Systematic Review* of robotic frameworks and their support for reconfiguration.
(iii) **Implementations:** *Repository Mining* to analyze how reconfiguration is implemented in open-source robotic (sub-)systems.

Specifically, we conducted the systematic literature review (SLR) to capture the scientific perspective on automated software reconfiguration at runtime in robotics and to extract theoretical foundations for the subsequent studies. We focused at identifying the reasons for reconfiguration, the expected lifespan of a configuration, and the granularity of reconfiguration, along with the means for specifying reconfiguration. Based on the robotic frameworks mentioned in the papers surveyed in the SLR and additional literature, we identified the most used frameworks for analysis in phase (ii) and which ones to investigate in phase (iii).

Taking into account the observations from the SLR, we synthesized a conceptual model as a basis for systematically analyzing robotic frameworks. Conceptual models are means for systematically capturing and communicating common knowledge about a domain (Gemino and Wand 2004), therefore, being a suitable means to build a common basis according to which we can classify and compare robotics frameworks. Based on the identified conceptual model, we then analyzed and classified robotics frameworks in phase (ii).

Our goal in phase (iii) is to determine what role automated reconfiguration plays in the software of robotic (sub-)systems and what practices are used to implement it. To enable this analysis of concrete implementations of reconfiguration in robotic (sub-)systems (iii), during

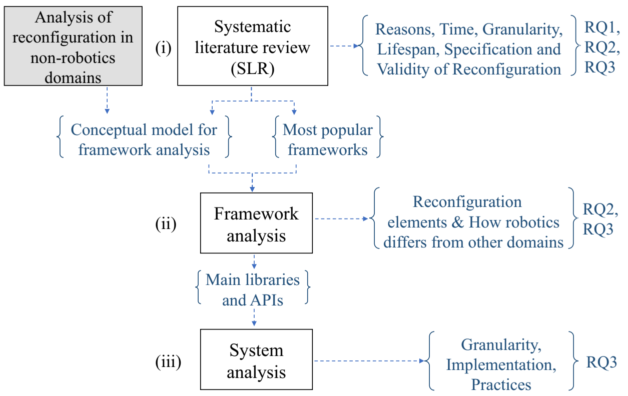
Analysis of reconfiguration in non-robotics domains
(i)
Systematic literature review (SLR)
Reasons, Time, Granularity, Lifespan, Specification and Validity of Reconfiguration
RQ1, RQ2, RQ3
Conceptual model for framework analysis
Most popular frameworks
(ii)
Framework analysis
Reconfiguration elements & How robotics differs from other domains
RQ2, RQ3
Main libraries and APIs
(iii)
System analysis
Granularity, Implementation, Practices
RQ3

Fig. 1 Overview on our research methodology

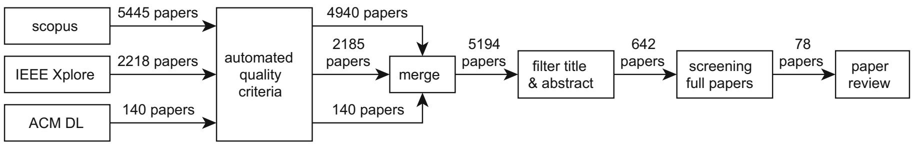

Fig. 2 Selection process of the primary studies

the analysis of robotic frameworks (ii), we also focused on identifying the main libraries used in robotics that can be used to implement reconfiguration.

Our replication package provides all raw data, scripts that have been used, and results (Peldszus et al. 2024a).[1]

3.1 Systematic Literature Review

The goal of our SLR is to capture the academic perspective on reconfiguration in robotics and what techniques exist to address reconfiguration. While we do not expect the academic perspective to perfectly match practice, we do expect it to at least cover the practical aspects, but also to provide a broader range of techniques and considerations that may not yet be practical.

As described in Sec. 2, most publications dealing with robot reconfiguration mainly focus on reconfigurable hardware whereas software aspects related to the reconfiguration of robots are not treated as main concerns. Nevertheless, the software configuration of a robot control system is highly affected by its mechanical structure, task, and operating environment (García et al. 2022). While some papers (e.g., Stueben et al. 2021) present case studies that motivate the runtime reconfiguration of the robotic systems, they do not give a systematic overview. For this reason, we defined a generic search string based on only three keywords:

```
("reconfigur*" OR "re-configur*") AND (robot*)
```

The search string was customized for the IEEE Xplore, Scopus, and ACM DL and applied to the title and abstract. Following the process shown in Fig. 2, the results were automatically filtered according to quality criteria to exclude studies not written in English, short papers ($<$ 3 pages), posters, and workshop summaries.

Even though the keyword search was limited to the fields “Title” and “Abstract,” it resulted in a high number of studies. The searches returned 2,185 results on IEEE Xplore, 4,940 results on Scopus, and 140 results on the ACM DL. We implemented a script to merge the results and remove the duplicates, which returned a total of 5,194 results.

We aimed at analyzing how the software reconfiguration of robots is implemented. Consequently, we selected those papers that fulfill the following inclusion criteria:

- I1: Studies focusing on the software of robotic systems;
- I2: Studies that present examples of robots that perform complex tasks, i.e., that require the integration and configuration of a variety of functionalities (e.g., perception, control, planning);
- I3: Studies that consider changing the configuration of robotic systems at runtime.

In addition, we applied the following exclusion criteria to exclude works that primarily focus on parts other than the software of a robotic system, or robotic systems that do not themselves include significantly complex robotic control software:

[1] Replication package at Zenodo: https://doi.org/10.5281/zenodo.14013818

E1: Studies focusing on single-purpose robotic demonstrators, microrobots, and robotic devices (e.g., a robot hand), since they controlled from the outside or do not run software on their own, and therefore, their software cannot be reconfigured directly;
E2: Studies that do not deal with robot control (e.g., papers solely focusing on reconfigurable computer architectures and communication networks, the mechanical design of metamorphic robotic systems);
E3: Studies that focus only on algorithms (e.g., for coalition formation of multi-robot systems or kinematic calibration of configurable robots) and do not include reconfiguration of the individual robots involved;
E4: The software part of reconfiguration plays only a minor role and other aspects such as hardware reconfiguration are the main focus.

A significant portion of the 5,194 papers that resulted from the high yield and low precision associated with our generic search string was filtered out by scanning the title and abstract. This step was performed by the second author with 20 years of experience in robotics research at a reading rate of three studies per minute. This step identified 642 full papers. Thereafter, we applied the inclusion and exclusion criteria based on the introduction and conclusion of the paper, resulting in 78 primary studies for analysis.

These papers were then randomly divided into three groups for review and classification of the full papers. Each group was then classified by a different author according to criteria relevant to reconfiguration. If the classification of a paper was in question regarding one of the criteria, it was analyzed by at least three authors. Disagreements were resolved by involving all authors, who discussed and reached agreement. We carried out this step over a period of 4 weeks, with a weekly meeting where we discussed between two and four concrete classifications per meeting.

During this full paper review, we excluded 46 of the primary studies based on the content of the full paper. This exclusion was because they were duplicates of other included papers, e.g., a paper is also included as an extended journal version (6 papers), or the content did not include a contribution to automated software reconfiguration at runtime, although we had first expected that based on the introduction and conclusion (42 papers).

Based on the primary studies, we performed then one iteration of backward snowballing, resulting in 1,040 papers cited by the primary studies. As for the primary studies, we again filtered these papers based on title and abstract using the exclusion criteria from above, resulting in 125 potentially relevant papers. These 125 potentially relevant papers were then filtered based on the full paper, resulting in an additional 21 papers for analysis. As before, in each step of the snowballing process, we randomly divided the papers into three groups, each of which was reviewed by a different author. Because of our experience with the classification of the primary studies, we did not need to have group discussions for the remaining papers.

Overall, the review process resulted in a total of 78 papers that were analyzed in detail to answer the research questions. Table 1 gives an overview of the papers identified in the initial search, and Table 2 of the papers identified via backward snowballing.

3.2 Robotic frameworks review

We selected and analyzed common frameworks used in robotics to identify at what granularity reconfiguration is technically supported (RQ2) and what interfaces frameworks provide to robotics software developers that can be used for implementing reconfiguration (RQ3). For all frameworks, we investigated how configurable software parts can be *specified* and

Table 1 Papers identified in the SLR's initial search

ID	Paper
P1	Berge-Cherfaoui and Vachon (1994)
P2	Hayes-Roth et al. (1995)
P3	Schneider et al. (1995)
P4	Stewart et al. (1997)
P5	Budenske and Gini (1997)
P6	Vos and Motazed (1998)
P7	Fayman et al. (1998)
P8	Boluda et al. (1999)
P9	Gafni (1999)
P10	Lindström et al. (2000)
P11	Karuppiah et al. (2001)
P12	Kubota et al. (2001)
P13	Zhang et al. (2001)
P14	Cobleigh et al. (2002)
P15	Kim et al. (2003)
P16	Bi et al. (2003)
P17	Inohira et al. (2003)
P18	Kim and Kim (2004)
P19	Roh et al. (2004)
P20	Lee et al. (2005)
P21	Lee and Kang (2006)
P22	Kim and Park (2006)
P23	Yu et al. (2006)
P24	Maeda (2006)
P25	Kim et al. (2006)
P26	Brandstötter et al. (2007)
P27	Scheutz and Kramer (2007)
P28	Braman et al. (2007)
P29	Morris (2007)
P30	de Cabrol et al. (2008)
P31	Lee et al. (2008b)
P32	Nilsson and Bengel (2008)
P33	Lee et al. (2008a)
P34	Santos et al. (2009)
P35	Xiao (2012)
P36	Benmoussa et al. (2013)
P37	Marques et al. (2013)
P38	Goldhoorn and Joyeux (2014)
P39	Scala et al. (2014)
P40	Szlenk et al. (2015)
P41	Frost et al. (2015)
P42	Shaukat et al. (2016)
P43	Mészáros and Dobrowiecki (2017)

Table 1 continued

ID	Paper
P44	Doose et al. (2017)
P45	Brugali et al. (2018)
P46	Jamshidi et al. (2019)
P47	Ramachandran et al. (2019)
P48	Murwantara (2020)
P49	Cardoso et al. (2019)
P50	de la Cruz et al. (2020)
P51	Brugali (2020)
P52	Cámara et al. (2020)
P53	Bozhinoski et al. (2021)
P54	Pane et al. (2021)
P55	Stueben et al. (2021)
P56	Kozov et al. (2021)
P57	Nordmann et al. (2021)

Table 2 Papers identified in the backward snowballing of the SLR

ID	Paper
P58	Ferrell (1994)
P59	Lee et al. (1994)
P60	Firby et al. (1995)
P61	Stewart and Khosla (1996)
P62	Pham et al. (2000)
P63	Wills et al. (2001)
P64	Macdonald et al. (2004)
P65	Kramer and Scheutz (2006)
P66	Parker and Tang (2006)
P67	Yoo et al. (2006)
P68	Calisi et al. (2008)
P69	Edwards et al. (2009)
P70	Tajalli et al. (2010)
P71	Inglés-Romero et al. (2012)
P72	Hartmann et al. (2013)
P73	Iftikhar and Weyns (2014)
P74	Gherardi and Hochgeschwender (2015)
P75	de Leng and Heintz (2016)
P76	Aguado et al. (2021)
P77	Nordmann et al. (2021)
P78	Bozhinoski et al. (2022)

encapsulated, how these can *interact* with each other, and how reconfiguration is *planned* and *executed*.

3.2.1 Framework selection

To increase the relevance of the frameworks selected for the review, we considered observations from the SLR in the selection process. In our SLR, 24 different frameworks were mentioned but only Chimera (a robot programming environment from the 90s Stewart et al. 1990), real-time CORBA (an OMG standard for real-time management of objects from the 2000s Schmidt and Kuhns 2000), and ROS (Open Robotics 2007) were mentioned more than once. The first two were not mentioned in papers after 2006. For this reason and for their age, we considered them as dated and, did not investigate them in depth. Consequently, besides ROS, we selected alternatively the most popular robotic frameworks based on existing surveys on software engineering practices in the service robotics domain (García et al. 2020a, b). ROS is used by 88.5% of the survey participants, followed by its successor ROS2 (22.4%) and OROCOS (18.6%). After a larger gap in popularity, YARP and SmartSoft follow with 4.5% and 3.8%. As SmartSoft is a collection of concepts and tools for RobMoSys (Lotz et al. 2013b), an abstract framework for the model-based development of robotic systems (RobMoSys 2023), we decided to consider SmartSoft as one example for a practical realization of RobMoSys.

In summary, we selected the following frameworks for an analysis: (i) ROS (and ROS2), (ii) OROCOS, (iii) YARP, and (iv) RobMoSys (incl. SmartSoft).

3.2.2 Conceptual model

The first step in actually investigating the robotics frameworks is to identify the concepts we will use to investigate the frameworks. To this end, to systematically derive the relevant concepts and ensure their completeness, we have created a conceptual model for software reconfiguration in robotics. This conceptual model will serve as a means to identify relevant concepts that frameworks must support to enable reconfiguration and to systematically analyze the frameworks concerning them. We proceeded in the creation of the conceptual model as follows.

We started the synthesis of the conceptual model from an existing schema (Berger et al. 2014), which was proposed in a study on variability mechanisms in software ecosystem platforms (e.g., Linux kernel, Eclipse plugins, and Android apps), and adapted it to the robotics domain based on insights from the SLR and further literature (Mens et al. 2003; Fritsch et al. 2008; Eddin 2013; Krupitzer et al. 2015; Mens et al. 2016; Tan et al. 2020). Therefore, we first compared this model with our observations on the scientific understanding of reconfiguration in robotics and excluded aspects that are not applicable from this initial model. Thereafter, we added further concepts that we observed in the SLR based on the additional literature.

3.2.3 Review process

Using the derived conceptual model, we classified the robotic frameworks based on their documentation and scientific publications about them. The first and second authors of this

Table 3 Data sources used during the frameworks review

	Wikis	Websites	Publications
ROS (ROS2)	wiki.ros.org	www.ros.org	Quigley et al. (2009); Estefo et al. (2019);
	docs.ros.org		Cousins et al. (2010); Gerkey (2014)
OROCOS	docs.orocos.org	orocos.org	Bruyninckx (2001)
YARP	wiki.icub.eu/wiki/YARP	yarp.it	Metta et al. (2006) Paikan et al. (2015)
RobMoSys	robmosys.eu/wiki	robmosys.eu	Lotz et al. (2013a)
(SmartSoft)	wiki.servicerobotik-ulm.de	smart-robotics.sourceforge.net	Schlegel and Worz (1999) Schlegel et al. (2021)

paper independently read the documentation provided for the robotic frameworks, mainly wikis, the websites of the robotic frameworks, and related publications, and recorded relevant references for each part of the conceptual model. Table 3 lists the sources investigated during the frameworks review. For the identification of the features of these frameworks that are relevant for our investigation, we also analyzed tutorials (e.g., by Brugali and Scandurra 2009 and Brugali and Shakhimardanov 2010), experience reports (e.g., by Brugali 2020), and secondary studies (e.g., by Ahmad and Babar 2016 and Albonico et al. 2023).

Thereafter, the two authors discussed and consolidated their findings. For aspects for which no agreement was achieved, the authors again independently investigated the available resources and then continued with the consolidation. Full agreement was achieved after four iterations. Based on the classifications, we then compared the robotic frameworks with each other and also with the findings from the SLR.

3.3 Investigation of robotic (sub-)systems

To identify what mechanisms are used in practice and how they are used (RQ3), we mined data from robotic software repositories. In doing so, we focused on robotics (sub-)systems that contain usages of the interfaces of robotic frameworks that we identified in the frameworks investigation as being suitable to implement reconfiguration although this might not be their main purpose. We realized that among the identified robotic frameworks, only ROS (Quigley et al. 2009) is mature enough to serve as a convincing basis for the in depth evaluation of robotic (sub-)system implementations. This is also supported by the findings from our SLR and the survey of García et al. (2020b).

To select suitable ROS (sub-)systems, we use a list of open-source ROS (sub-)systems repositories that has been created by Malavolta et al. in 2021 when studying general guidelines for developing robotic (sub-)systems (Malavolta et al. 2021). They define a ROS-based system as "*a system that contains robotic capabilities built using the ROS framework*." To identify suitable ROS-based systems, Malavolta et al. mined GitHub, GitLab, and Bitbucket and did extensive quality assessments to identify deployable ROS (sub-)systems that implement huge parts of their logic on their own and are not only wrappers for libraries or toy examples.

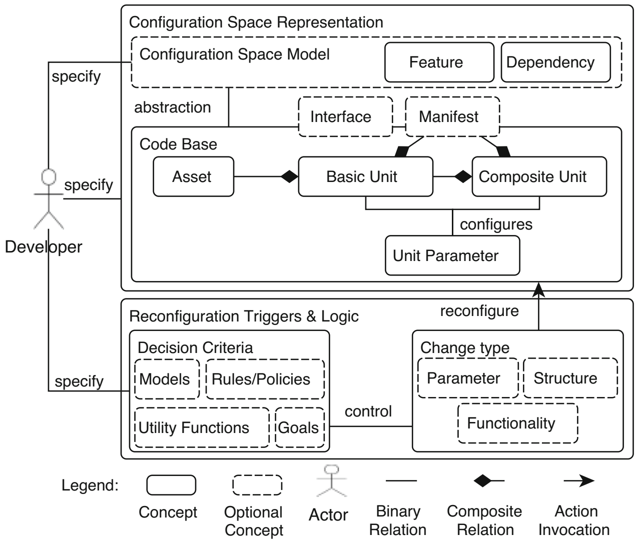

Fig. 3 Conceptual model of reconfiguration in robotics

Thereby, implementing the logic of the system does not prevent it from using libraries such as the ROS navigation stack[2] to realizing this logic.

To systematically investigate how reconfiguration is implemented, we use the APIs of ROS that we identified in the frameworks review as being suitable to implement reconfiguration as entry points for our software inspections. Starting from these entry points, we inspected how the API is used to understand how reconfiguration is implemented in the robotic (sub-)systems. Since, many APIs allowed to register callback functions, we followed the calls in both directions forward and backwards to visit all relevant code parts.

4 Conceptual model of reconfiguration

For investigating robotic frameworks, we created a conceptual model of reconfiguration in robotics according to which we classified the investigated robotic frameworks. Figure 3 shows the identified concepts as well as their relations. Although this conceptual model is partially based on findings from our SLR, we present it here prior to the detailed discussion of the findings of that SLR to allow for a discussion of findings from all three types of artifacts reviewed together.

We started to create the conceptual model for reconfiguration in robotics from an existing conceptual model proposed in a study on variability mechanisms in software ecosystem platforms (Berger et al. 2014). Then, based on the insights from the performed SLR, we excluded all aspects that do not apply to reconfiguration in robotic systems. For example, we conclude that in robotics, decisions to bind a feature as active or inactive are only temporal and made dynamically. For this reason, we excluded the decision lifecycle and binding-related aspects from our framework. As we focus on open-source robotic frameworks, we also dropped the aspect of platform openness.

[2] ROS Nav2: https://navigation.ros.org/

While the reduced conceptual model captures what can be reconfigured, how reconfigurable assets are specified, and how these can interact, it does not capture how reconfiguration is specified and executed, yet. In our SLR we have seen that decision making is an essential part of reconfiguration, and there are various ways of specifying it. In an investigation of self-adaptation in robotic systems (Krupitzer et al. 2015), such specifications have been categorized into four kinds of decision criteria for specifying reconfiguration triggers and actions (Models, Rules/Policies, Utility Functions, and Goals), introduced in detail in Sec. 4.5. To use the same naming as the existing literature, we added the decision criteria to our conceptual model. Also, we identified three fundamental change types realized by reconfiguration mechanisms in the literature (Fritsch et al. 2008; Krupitzer et al. 2015), namely changing parameter values, software structure, and functionality, which we introduce in detail in Sec. 4.5. Thereafter, we iteratively confirmed and adjusted our conceptual model based on additional related literature (Mens et al. 2003; Fritsch et al. 2008; Eddin 2013; Krupitzer et al. 2015; Mens et al. 2016; Tan et al. 2020). Finally, we tailored the definitions toward reconfiguration in robotic systems.

4.1 Configuration space (RQ2)

For answering what parts of the software of a robotic system can be reconfigured and at what granularity (RQ2), it is essential to identify what are the reconfigurable elements in the robotic frameworks, how these are specified, and what dependencies might exist (Mens et al. 2003; Eddin 2013). The *Code Base* refers to the structuring of these reconfigurable elements of the robotic system.

To allow consistent and domain-independent investigation of the supported granularity, in this work, we use the following three levels of granularity that are oriented on the naming from a prior study investigating variability in software ecosystem platforms (Berger et al. 2014) and the observations from the SLR: (i) A *Composite Unit* represents a larger software entity, which can function independently of other units and usually aggregates and controls further smaller units, (ii) reconfiguration takes place on the granularity of a single class or function that usually cannot function independently (*Basic Unit*), e.g., using a different perception mechanism provided by the same composite unit, and (iii) the value of a configuration attribute of a *Unit parameter*, e.g., a parameter of a method or an environment variable, is changed. *Manifests* provide information about these reconfigurable entities such as their identifier or parameters required at reconfiguration, e.g., to identify a composite unit that is suitable for a specific task.

4.2 Configuration space model (RQ2)

For safe reconfiguration, it is essential to know the space in which a reconfiguration of the code base can take place and what are valid configurations (Svahnberg et al. 2005). To this end, these reconfigurable elements are represented using the notation of *features* (Kang 1990) that serve as an abstraction of the *code base*, therefore, providing a conceptual view on what elements of a robotic system can be reconfigured (RQ2). Thereby, multiple *features* can have dependencies among each others, e.g., one feature requiring or excluding another feature, which can limit what can be reconfigured. Therefore, besides the reconfigurable elements, one has to know all possible combinations of these (Mens et al. 2016). *Features* and their *dependencies* are usually specified in a *Configuration space model* (Berger et al. 2013). We are particularly interested in the languages supported by the frameworks for specifying

Configuration space models, as they are the main artifact used by developers for specifying the configuration space and could be analyzed by tools.

4.3 Encapsulation (RQ2)

For reconfiguration in terms of turning features on or off, reconfigurable assets must be well encapsulated and allow one to access reconfigurable elements. To this end, different *Interface mechanisms* (Berger et al. 2014; Tan et al. 2020) can be used to interact with reconfigurable elements. These can range from well-defined APIs to system-wide messaging services. Besides knowing which mechanisms the different robotic frameworks provide to allow communication among the reconfigurable elements, we also have to know in which way the interfaces of reconfigurable elements are specified so that developers can use the *Interface specification*.

4.4 Interactions (RQ3)

To understand reconfiguration, particularly, in terms of the mechanisms that can be used for implementing reconfiguration (RQ3), the management of interactions is essential. To allow interaction at runtime, the statically defined interface usages have to be bound to the concrete running instances and have to be updated at each reconfiguration. For executing the bindings, a reconfigurable system needs a *Run-time Manager* that performs this binding (Berger et al. 2014). Consequently, we are interested in how the robotic frameworks implement interaction binding.

While the already considered interface mechanisms are a static view on specifying interaction, we also need to know the *Interaction mechanism* using which dynamic interaction with reconfigurable assets is realized. Interactions among basic units require *Interaction binding* for identifying and binding the concrete target, which can happen at different times ranging from static binding to dynamic binding (Fritsch et al. 2008).

4.5 Trigger & logic (RQ3)

Reconfiguration consists of three essential steps: planning, decision, and execution (Tan et al. 2020). These steps must be implemented using appropriate mechanisms (RQ3), which, in the best case, are provided by robotics frameworks.

A reconfiguration trigger is a set of conditions that, if true, activate one or more reconfiguration actions (Soria et al. 2009). We need to know what logic can be used to specify reconfiguration triggers (Mens et al. 2003; Krupitzer et al. 2015) and how the reconfiguration will be executed (Mens et al. 2003). We are interested in what support the robotics frameworks provide to developers for specifying *Decision Criteria* to define when and how to reconfigure a robotic system.

As also confirmed by our SLR, decision criteria can be defined in various ways (Krupitzer et al. 2015): (i) model-based specification of the reconfiguration, e.g., a statemachine, (ii) rule-based triggering of reconfiguration, (iii) goal-based reconfiguration, and (iv) frameworks, which offer utility functions that simplify the implementation of reconfiguration.

The reconfiguration itself can then be realized in different ways. In the literature (Mens et al. 2003; Fritsch et al. 2008; Krupitzer et al. 2015), we can find three different *Change types* of reconfiguration at runtime that correspond to a robots environment and the three software granularities in the asset base: (i) changing parameter values to permanently influence the internal control flow of units (*parameter*), (ii) changing a functionality while keeping its interface, and (iii) structural reconfiguration (*structure*), which changes the running system's structure, e.g., by loading or unloading a composite unit.

This conceptual model allows us to systematically capture how reconfigurable elements are specified, how these interact with each other, how the reconfiguration logic can be specified, and which reconfiguration support robotic frameworks provide.

5 Motivations for reconfiguration (RQ1)

First, we want to get a better understanding of what scientists see as the reasons that can be addressed by reconfiguring robotic systems. We assume that understanding their motivations, allows us to reason better on why they address reconfiguration as they did. To this end, we only investigated the academic literature on reconfiguration to answer RQ1.

5.1 Reasons for reconfiguration in the academic literature

Of the 78 papers, 27 papers (34.62%) mention more than just one reason for reconfiguring robotic systems. On average 1.45 reasons are mentioned per paper. Figure 4 shows the most popular reasons and their relative frequencies among all mentioned reasons. Only P4 and P67 mention no reason for reconfiguration.

Environmental changes & task execution: As expected, reconfiguration mainly takes place to allow operation in dynamic environments (43.59%) and executing tasks (42.31%). There seems to be a significant correlation between these two reasons, since 15 papers, which is by far the most common combination in our sample, mention both of them at the same time (P1, P3, P17, P22, P30, P33, P41, P51, P56, P59, P68–69, P73–74, and P77). Reconfiguration is needed (i) between tasks that require different hardware and software configurations of a robot and (ii) in a single task execution to successfully fulfill the task. An additional 19 papers need reconfiguration to react to environmental changes (P12, P14, P20–21, P25, P29, P31, P36–37, P39, P52–53, P57, P62–65, P72, and P78) and 16 papers for changes as part of task execution (P2, P5, P10, P15–16, P18, P24, P34–35, P38, P40, P43, P48, P50, P66, and P70).

Fault handling & resilience: Reacting to events such as a sensor returning constantly faulty data, is the third most frequently mentioned reason for reconfiguration (32.05%). In this case, the aim is usually to reconfigure the system so that it does not use the faulty sensor

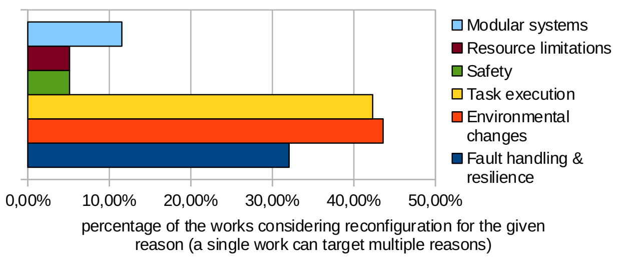

Fig. 4 Reasons for reconfiguring a robotic system

any longer by replacing it with a spare sensor, i.e., in software switching to a different sensor (P1, P6–7, P11, P13, P26–28, P42, P46–47, P49, P54, P57–62, P64, P69–70, P75–77).

Modular systems: More than 11% of all papers claim to need reconfiguration for developing modular systems (P3, P13, P19, P23, P30, P32–33, P43, and P70). To this end, they consider modularity at development time and hardware modularity at runtime, e.g., a robot that can change the actuators it uses, which triggers the reconfiguration of the running software. While we consider the second also as reconfiguration, in our understanding, building a software system at development time from different modules is not reconfiguration, but general design-time variability (Chen and Babar 2011; Apel et al. 2013; Berger et al. 2020). Related to this, papers P23 and P70 mention, besides other reasons, also maintenance or updates at runtime as a reason for reconfiguration, e.g., a human having to manually perform ad-hoc reconfigurations to allow a robot to complete its task.

Resource limitations: As one would expect due to the ever-increasing computational capabilities, it seems that limited resources, often considered as the main driver for reconfiguration, have become a much less relevant challenge in the last decade. Still, a significant number of papers mention limited resources as a reason for reconfiguring a robotic system, but it is mainly the older papers considered in our SLR that mention this reason. First, in the two oldest papers (P8 and P9), which are all from 1999, it is due to generally limited computing capacities. Later, in P24 from 2006 and P71 from 2012 the resource limitation is due to more sophisticated tasks, such as speech-based interaction with robots. The most recent paper of these (P47), discusses limited resources in terms of network capacity for teams of interacting robots.

Safety: Finally, responding to safety issues is mentioned as a reason for reconfiguration by four papers (P44–45, P55, and P71), which notably is the only mentioned reason in all papers except P71.

5.2 Discussion of reasons for reconfiguration

In summary, the main driver for reconfiguration in the literature is robots performing complex tasks in diverse environments, which includes both reconfiguration to respond to environmental changes during task performance, as well as adapting the configuration of the robot according to task needs (64.1% mention one of these two reasons). Orthogonal to this, reconfiguration is also needed to ensure proper operation by enabling fault handling and increasing the robustness (which is partly closely related to reconfiguration due to environmental changes) of the robots.

6 Reconfigurable elements of robotic systems (RQ2)

We answer RQ2 both from the academic perspective, as represented by the academic literature, and from the perspective of robotics frameworks. We first present our analysis of the academic literature with respect to RQ2, then our analysis of robotics frameworks, and finally discuss the observations of both together.

6.1 Academic literature on what parts of a robotic system can be reconfigured

Based on the discussions above, we identified three aspects relevant for discussing what software parts of a robotic system can be reconfigured.

1) Granularity at which a robot is reconfigured: Since reconfiguration can be thought of in completely different dimensions, ranging from changing the hardware a robot uses to small differences in how it will behave, we want to assess what granularity of reconfiguration is considered in the academic literature on automated reconfiguration of robotics software at runtime. Again, we expect to extract a baseline that can be used in a comparison with reconfiguration of robots in practice.

2) How long is the intended lifespan a configuration before reconfiguring the robotic system again: We assume that robotic systems are not only configured once but are reconfigured more or less frequently according to relevant circumstances. Therefore, we aim at assessing how often reconfiguration is expected to take place and how long a robotic system will stay in a configuration before it is reconfigured again.

We classified the investigated papers according to these aspects to capture the academic perspective on RQ2. In what follows, we present these classifications.

6.1.1 Granularity of reconfiguration

The investigated papers use many different terms for naming reconfigurable entities of robotic systems. On one side, there are domain-specific terms, such as *ROS node* due to the considered robotic framework. On the other side, commonly used names, such as *component*, are used differently among the papers. This resulted either in different terms being used for referring to the same granularity or one term referring to different granularities in multiple papers. To compare granularities across multiple papers, we assigned them to four categories based on the semantics described in each paper. We oriented the definition of these categories on definitions from the literature (Berger et al. 2014).

Figure 5 shows what percentage of the considered papers supports which level of granularity. Thereby, a single paper can support multiple levels. However, with an average of 1.18 levels per paper, the papers that consider multiple levels (P3, P36, P41, P51, P58, P60–62, P67–68, P73, and P78) are a minority. Overall, the literature seems to focus on reconfiguration at a coarse-grained level.

Composite unit: More than half of the papers (57.69%), support reconfiguration of larger software entities, which can function independently of other units, such as loading or unloading entire components of the system (P1–4, P7, P13–22, P25, P27, P31, P33, P36–38, P40–42, P44, P47–48, P51–53, P57, P61–67, P69–71, P75, and P77–78).

Basic unit: In more than a fourth of the papers (26,92%), reconfiguration takes place on the granularity of a single class or function, e.g., using a different perception mechanism provided by the same composite unit (P5, P8–9, P12, P23, P26, P49–51, P54–56, P58–62, P68, and P72–74).

Unit parameter: The fine-grained reconfiguration of concrete parameters, such as changing the value of a field that represents a state, is considered by 30.77% of the papers (P3, P6, P10–11, P24, P28–30, P32, P34–35, P39, P43, P45–46, P51, P58, P60, P62, P67–68, P73, P76, and P78).

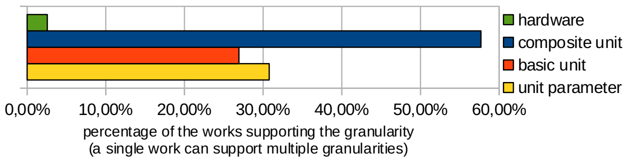

Fig. 5 Granularity at which a reconfiguration is supported by the techniques presented in the literature

6.1.2 Lifespan of configurations

In the literature, configurations are typically meant to be alive for a relatively long time. Still, reconfiguration is considered to happen rather often and is not seen only in exceptional cases. We identified six different lifespans of configurations, whose popularity is shown in Fig. 6.

Stable: The configurations are stable for a very long time, and reconfiguration is only performed due to exceptional events at runtime. These events are usually errors, such as sensors returning implausible values when hardware fails (P26, P28, P47). Also included in this category are reconfigurations that require a reboot of the robot (P4) or a manual trigger by the user (P4, P19, P32, and P47), for example by manually attaching or detaching some hardware. Further cases comprise the generation of reconfigurations that require the robot to be placed in a non-dynamic state, such as stopping movements before executing a reconfiguration (P4).

Long: The configuration is changed in the case of major environmental changes that are likely to occur, but do not occur frequently (P21, P25, P37, P63–65, and P78), or when a new configuration is required to perform a mission that may include multiple tasks (P2, P15, P16, P18, P24, P43, P18, and P66). Five papers (P17, P22, P30, P68, and P74) consider major environmental changes as well as new mission requirements. In summary, we assume that a configuration will be used for the duration of an entire mission. Failures that are likely to occur in practice, but not frequently, may also lead to reconfigurations whose target configuration will be alive for a long time (P11, P13, P42, P46, P49, P58, P64, and P75). Similarly, this lifespan also applies to some types of resource constraints (P8 and P24) and runtime maintenance (P23).

Medium: Already smaller environmental changes (P12, P14, P20, P29, P31, P39, P52–53, P57, and P72) are likely to be addressed by reconfiguration, or reconfiguration is needed within a mission to execute the tasks of which the mission consists (P5, P10, P35, P38, P40, P50, and P70). Both of these two variants are considered at the same time by eleven papers (P1, P3, P33, P41, P51, P56–57, P59, P91, P69, and P77). Three papers (P7, P27, and P54) consider faults that are likely to occur more often; and six papers (P57, P59, P61, P69–70, and P77) consider them in addition to environmental changes or mission requirements. Also, resource limitations (P9) or safety reasons (P44–45, and P55) are considered to trigger a reconfiguration that results in a configuration that is assumed to be alive for a medium time. Accordingly, we define the scope of a configuration with a medium lifespan to be of the lifespan of a single task.

Short: To execute a single task of a mission already multiple reconfigurations might be needed. Configurations have a relatively short lifespan and are reconfigured quite frequently. P60 reconfigures for each step of an executed task; in P71 every minor change will be addressed by reconfiguring the robot; and in P73 a model used in a MAPE-K feedback loop is frequently updated.

None: The configuration is constantly changing, and therefore, has no lifespan. In most cases, this is realized by continuously updating parameters (P6, P34, and P76). In P6, feedback in the form of parameter reconfiguration is added to a linear time-invariant system model;

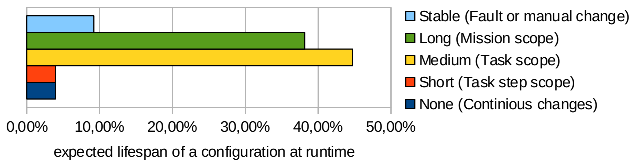

Fig. 6 Lifespan of configurations

in P34 the parameters of a time-division multiple access protocol are reconfigured; and in P76 the parameters of a force allocation matrix are reconfigured, which is used to define how the thruster configuration affects the dynamics of the UX-1 robot, an under water robot for exploration of flooded mine tunnels (Fernández et al. 2019).

Stable: The configurations are stable for a very long time, and reconfiguration is only performed due to exceptional events at runtime. These events are usually errors, such as sensors returning implausible values when hardware fails (P26, P28, P47). Also included in this category are reconfigurations that require a reboot of the robot (P4) or a manual trigger by the user (P4, P19, P32, and P47), for example by manually attaching or detaching some hardware. Further cases comprise the generation of reconfigurations that require the robot to be placed in a non-dynamic state, such as stopping movements before executing a reconfiguration (P4).

Long: The configuration is changed in the case of major environmental changes that are likely to occur, but do not occur frequently (P21, P25, P37, P63–65, and P78), or when a new configuration is required to perform a mission that may include multiple tasks (P2, P15, P16, P18, P24, P43, P18, and P66). Five papers (P17, P22, P30, P68, and P74) consider major environmental changes as well as new mission requirements. In summary, we assume that a configuration will be used for the duration of an entire mission. Failures that are likely to occur in practice, but not frequently, may also lead to reconfigurations whose target configuration will be alive for a long time (P11, P13, P42, P46, P49, P58, P64, and P75). Similarly, this lifespan also applies to some types of resource constraints (P8 and P24) and runtime maintenance (P23).

Medium: Already smaller environmental changes (P12, P14, P20, P29, P31, P39, P52–53, P57, and P72) are likely to be addressed by reconfiguration, or reconfiguration is needed within a mission to execute the tasks of which the mission consists (P5, P10, P35, P38, P40, P50, and P70). Both of these two variants are considered at the same time by eleven papers (P1, P3, P33, P41, P51, P56–57, P59, P91, P69, and P77). Three papers (P7, P27, and P54) consider faults that are likely to occur more often; and six papers (P57, P59, P61, P69–70, and P77) consider them in addition to environmental changes or mission requirements. Also, resource limitations (P9) or safety reasons (P44–45, and P55) are considered to trigger a reconfiguration that results in a configuration that is assumed to be alive for a medium time. Accordingly, we define the scope of a configuration with a medium lifespan to be of the lifespan of a single task.

Short: To execute a single task of a mission already multiple reconfigurations might be needed. Configurations have a relatively short lifespan and are reconfigured quite frequently. P60 reconfigures for each step of an executed task; in P71 every minor change will be addressed by reconfiguring the robot; and in P73 a model used in a MAPE-K feedback loop is frequently updated.

None: The configuration is constantly changing, and therefore, has no lifespan. In most cases, this is realized by continuously updating parameters (P6, P34, and P76). In P6, feedback in the form of parameter reconfiguration is added to a linear time-invariant system model; in P34 the parameters of a time-division multiple access protocol are reconfigured; and in P76 the parameters of a force allocation matrix are reconfigured, which is used to define how the thruster configuration affects the dynamics of the UX-1 robot, an under water robot for exploration of flooded mine tunnels (Fernández et al. 2019).

In summary, reconfiguration in the literature is considered to be executed quite frequently, but not continuously. Configurations are mostly considered to be alive for a long time, having an entire mission as scope (38.16%); or for a medium time with an entire task of a mission as scope (44.74%). Considering the most frequent granularity, which is the reconfiguration of composite units, the time needed to execute a reconfiguration could be a major reason that stands against more frequent reconfiguration. This reason was also mentioned in some of the papers, among others in detail in P4.

6.2 Reconfigurable elements in robotics frameworks

We investigated the robotics frameworks to determine which elements of a robotics system they support in terms of reconfiguration. We based this analysis on our conceptual model introduced in Sec. 4. Table 4 shows a summary of the four major robotic frameworks we analysed according to the introduced conceptual model.

6.2.1 Configuration space

All frameworks provide structures for specifying reconfigurable assets and realize the three different element kinds that can be part of the *Code Base*. Most frameworks provide multiple realizations of *basic units*. ROS, OROCOS, and YARP realize *basic units* as shared or dynamically loadable libraries. With *plugins* and *nodelets*, ROS implements two kinds of composite units. Plugins allow us to load additional functionalities into the executed methods. In contrast to this, *nodelets* can be executed in separate threads in parallel to the current

Table 4 Reconfiguration elements in robotic frameworks

		ROS/ROS2	OROCOS	YARP	RobMoSys (SmartSoft)
Configuration Space	**Asset Base**				
	Basic units	dynamic libraries (ROS plugin/nodelet)	dynamic libraries (plugin services)	dynamic libraries (YARP plugin services), static library (YARP module)	service
	Composite units	executable programs (ROS node)	dynamic libraries (component)	c++ executable programs (component)	component
	Unit parameters	ROS parameter	data flow port	YARP port	component parameter
	Configuration model				
	Features	node, plugin/nodelet	component, plugin	component, plugin	component
	Language	N/A	N/A	N/A	Variability Modeling Language (VML)
	Manifest (Schema)	XML-based DSL (launch file, plugin description file)	XML deployment file	XML-based DSL (YARP manager.ini, plugin manifest)	component model, system con figuration model
Encapsulation	**Interface mechanism**	IDL-based messages (node), base class API (nodelet/plugin), ROS parameters	TaskContext API, IDL-based component services	IDL-based messages (programs), base class API (RFModule)	provided/requires services
	Interface specification	Documented interfaces (message types and package descriptions) in public repositories	Documented interfaces in the Orocos Component Library	Documented interfaces	SmartMARS Metamodel (communication objects + communication patterns)
Interactions	**Run-time Manager**	ROS master, DDS	OROCOS DeploymentComponent	YARPserver, YARPmanager	SmartEventServer, SmartParameterMaster
	Characteristics				
	Mechanism	publish/subscribe, client/server, parameters	data- flow, client/server	data- flow, client/server	publish/subscribe, client/server
	Binding mode	dynamic	dynamic	dynamic	dynamic
Trigger & Logic	**Decision Criteria**				
	Models	state machine (ROS SMACH) behavior trees (py_tress_ros & BehaviorTree.CPP)	state machine	behavior trees	dynamic statecharts
	Rules/Policies	N/A	N/A	N/A	VML: ECA rules
	Goals	N/A	N/A	N/A	SmartTCL (Task Coordination Language)
	Utility function	ROS APIs	RTT:APIs	YARP APIs	N/A
	Change Type				
	Parameter	dynamic_recon figure	RTT::TaskContext (data flow port)	YARP::os::BufferedPort	SmartParameterMaster/ Client (parameter)
	Functionality	pluginlib (plugin)	N/A	N/A	N/A
	Structure	roslaunch (node), pluginlib + nodelet (plugin/nodelet)	RTT::Scripting (plugin)	YARP::dev::DriverCreator	Smart Task (component)

execution. In RobMoSys, *basic units* are the provided services, which are specified as models according to a metamodel of RobMoSys. Depending on the concrete implementation, e.g., SmartSoft, the code for a service can be generated and loaded dynamically.

The *composite units* are realized in ROS and YARP as executable programs that use the frameworks for inter-unit communication and for accessing basic units. In OROCOS, composite units provide the basic infrastructure to make a system out of pieces of code that can interact via data and events. RobMoSys services are gathered in components.

Unit parameters are realized differently across the frameworks. In ROS, composite units can have parameters that are managed globally and must be actively accessed by the composite units. OROCOS and YARP are based on input and output ports of composite units that are explicitly connected with each other, data directly flows between composite units according to their linking. In RobMoSys, components have a model-based parameter specification.

6.2.2 Configuration space model

In all frameworks, the composite units realize features that can be turned on or off as they are considered in the literature (Kang 1990). In ROS, OROCOS, and YARP, even the basic units are features.

All frameworks work with XML-based *Manifests (Schema)* for specifying the reconfigurable elements (*features*). Thereby, the *Manifests* can be distributed over multiple files, e.g., one per composite unit. However, only RobMoSys provides a detailed metamodel that defines the syntax of the *Manifest (Schema)*. Similarly, among the robotic frameworks, only RobMoSys provides an explicit model of the configuration space, despite its importance for safe reconfiguration. This model can be specified using the *Variability Modeling Language* (VML) (Schlegel et al. 2013). For ROS, there are at least third-party extensions that provide such capabilities, such as HyperFlex (Brugali and Gherardi 2016).

6.2.3 Encapsulation

All robotic frameworks provide suitable interface mechanisms to realize well-encapsulated reconfigurable assets. Except for RobMoSys, the only explicitly specified *Interface mechanism* is an abstract class that must be implemented by the reconfigurable assets. RobMoSys provides a metamodel for describing communication objects and communication patterns, e.g., publish/subscribe or client/server. However, while the interfaces are clearly defined, this does not necessarily imply that their semantic usage is equally simple. For example, interfacing with the ROS navigation stack can require non-trivial sequences of messages.

Except for RobMoSys, all frameworks rely on manual documentation of the interfaces. ROS and OROCOS encourage developers to provide such *Interface specification* by making it an essential part of public ROS repositories, and of the OROCOS component library. In practice, however, the documentation of packages is often outdated and not maintained. It is well-known that good documentation is needed for effectively working with software (Aghajani et al. 2019), and, therefore, this lack in up-to-date documentation is a significant challenge for the implementation of robotic systems.

6.3 Discussion of reconfigurable elements

Three distinct levels of granularity were identified at which robotic systems can be reconfigured. The first level comprises concrete configuration parameters that can be assigned values.

The second level encompasses basic units that provide concrete functionalities but are not independently executed. The third level encompasses composite units that provide larger software entities that can comprise multiple basic units and are executable on their own. While the majority of literature focuses on the structural reconfiguration of robotic systems, primarily at the granularity of composite units, robotic frameworks offer only limited support for this granularity concerning reconfiguration. Technically, the frameworks facilitate the structuring of robotic systems according to different relevant granularities for reconfiguration. However, they lack the means to explicitly specify configurable elements and possible constraints on them. This represents a significant challenge to their reconfiguration since possible interactions are not explicitly captured and, therefore, difficult to handle.

7 Reconfiguration mechanisms (RQ3)

To answer RQ3, we conducted an investigation of all three kinds of artifacts. First, we present the investigation of the academic literature that represents the state-of-the-art concerning RQ3. Then, we present the corresponding state-of-practice, in terms of robotic frameworks and robotic (sub-)systems. Finally, we answer RQ3 concerning all three artifacts.

7.1 Academic literature on reconfiguration mechanisms

We reviewed the academic literature on reconfiguration for the following three aspects of reconfiguration mechanisms needed to develop reconfigurable robotic systems.

1) Specification of the reconfiguration logic: Reconfiguration is a multi step process, before actually executing a reconfiguration, the robotic system must identify the need to do so and plan on how to react. The latter two steps must be implemented in some way in a robotic system and we are interested in identifying the techniques that have been proposed in the scientific literature.

2) Ensuring the validity of reconfigurations: To allow safe and secure operation of robotic systems, it is essential that they are always in a valid configuration. Invalid configurations can cause security issues (Peldszus et al. 2018) as well as safety issues. Unfortunately, it has been shown that already statically ensuring the validity of configurations is challenging (Yin et al. 2011). Therefore, we are interested how this issue can be addressed in the robotics domain.

3) Cost of reconfiguration: The different mechanisms for specifying, validating, and executing reconfigurations usually come with some cost for executing these tasks. However, in what form these cost arise, what dimensions of cost must be considered for reconfiguration, and their impact are unclear so far.

7.1.1 Specification of reconfiguration

In the investigated papers, we found seven different approaches to specifying the reconfiguration logic of a robotic system. Figure 7 shows the approaches and how often these were implemented in the investigated papers. Only 6 papers do not explicitly consider the specification of the reconfiguration logic (P33, P43, P51, P63–64, and P75), which is indicated by *n/a*.

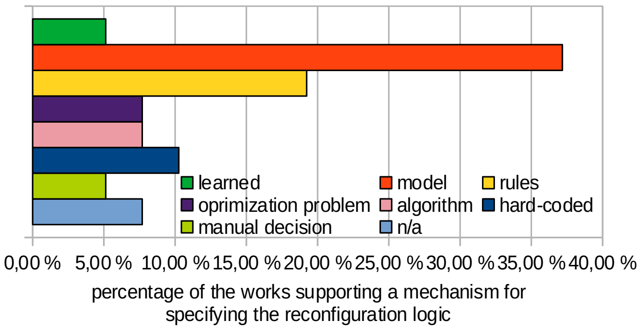

Fig. 7 Specification of reconfiguration logic

Learning: In 4 papers, no explicit specification of the reconfiguration logic is needed, since the papers focus on automatically learning when and how to reconfigure the robot (P41–42, P52, and P72).

Model: The reconfiguration space and task or mission requirements are explicitly modeled. There are techniques that rely on two connected models. First, a model that captures the possible configuration space, i.e., alternative configurations for various situations are explicitly defined or the configuration space is specified using feature models (Kang 1990). Second, a model of the tasks or missions to be executed that is connected to needed elements from the latter model via explicit references. Some central unit selects the best configuration based on the current task and situation (P2, P5, P8–10, P16–18, P20–22, P25, P27–29, P39, P50, P53, P56, P59, P61, P66, P70, P73, P76–78). P38 express the configuration logic as executable models, such as a state machine, avoiding the need for some central unit for selecting target configurations; and P49 uses a mathematical model.

Rule: The reconfiguration is explicitly specified in rules, whose application conditions are monitored at runtime and the respective rule is executed as soon as the condition applies (P15, P24, P31, P35–36, P40, P48, P54, P57–58, P60, P68–69, P71, and P74). The rule then changes to a specific configuration; or a specific reconfiguration action will be executed, such as replacing a composite unit. Rules are typically encapsulated in separate artifacts that can be maintained without touching the implementation of the robotic system. In addition, domain-specific languages are often used to facilitate the specification of these rules.

Optimization problem: Six papers treat the reconfiguration as an optimization problem of the kind of finding the best configuration that fulfills some given criteria (P1, P26, P30, P46–47, and P55). For example, in P26 the developers define a multi-dimensional constraint problem for which an optimal solution in terms of a target configuration must be identified. In contrast to the papers that are based on models of the robotic system, which are likely to select a new configuration also based on the outcome of some optimization problem, in these papers, the optimization problem is hard coded and is not based on the interpretation of a model.

Algorithm: In another six papers (P6–7, P11–12, P34, and P45), the reconfiguration logic is realized as an algorithm determining the conditions that trigger a reconfiguration. The algorithms considered in this category are typically fixed reconfiguration algorithms for specific reconfiguration tasks, e.g., error detection algorithms to detect a faulty component and reconfigure the system (usually by disabling the faulty component and enabling a backup).

These algorithms cannot be tailored to project-specific needs by developers but are intended to be used as they are for a specific reconfiguration task.

Hard-coded logic: In eight papers (P13–14, P19, P32, P37, P62, P65, and P67), the reconfiguration logic is a hard-coded part of the implementation of the robotic system and typically consists of if-then statements. Unlike fixed algorithms, the logic is tailored towards the specific robotic system and is maintained as part of the source code of the robotic system.

Manual selection of configuration: Finally, four papers (P3–4, P23, and P44) do not consider the specification of when and how to reconfigure. These papers focus on automatically reconfiguring into a target configuration. For example, manual triggers, such as attaching a tool (e.g., a screwdriver) to the robot serve as a trigger to perform a automated reconfiguration into a specific configuration (e.g., loading the software modules needed to work with the screwdriver). While the target configuration is given manually, the system is configured automatically, among others involving activities such as bringing the system into a safe state before changing its configuration (P4).

Altogether, basing the reconfiguration logic on models of the robotic systems is the most popular way of specifying the reconfiguration logic with 37.18% of the papers. The relatively similar, but much simpler specification of the reconfiguration logic in reconfiguration rules follows with 19.23% of the papers. The reconfiguration logic being hand-written in code is considered by 10.26% of the papers. All other specifications of the reconfiguration logic account for only between 5.13% and 7.69% of the papers and could be considered less relevant or at least less popular for the development of dynamically reconfigurable robotic systems.

Both rule- and model-based reconfiguration are built on user-defined specifications of the logic itself or of the system. For these papers, we found mostly *Domain-Specific Languages (DSLs)* (Wasowski and Berger 2023) for defining when and how to reconfigure. These DSLs range from providing simple mappings between tasks and composite units to languages that support complicated conditions that are solved to determine a suitable configuration within the robot's configuration space. Unfortunately, many papers only mention a type of specification that is not detailed in the paper. Usually these specifications are mappings between goals or tasks and reconfigurable assets of the robot that are suitable for achieving the goal or performing the task.

7.1.2 Validity of reconfigurations

While all of the 78 reviewed papers describe approaches that enable reconfiguration, only 28 of them explicitly consider ensuring that a reconfiguration will result in a valid configuration of the robot. Figure 8 shows the identified validation approaches and how often these appear in the investigated papers.

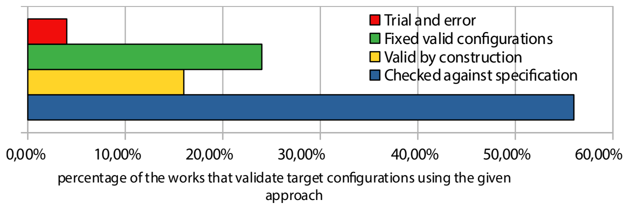

Fig. 8 Assurance of target configuration validity

Checking against specification: In 14 papers (P7, P45, P48, P51–52, P55, P61, P66, P69–70, P72–74, and P77), the validity of a reconfiguration is checked against a specification before executing the reconfiguration. In four of these papers (28.57% of the cases) this specification is a feature model (Kang, 1990) (P45, P48, P55, and P74), in another four cases it is an architecture specification (P52, P72, P74, and P77), in three cases the checking is performed against a state machine containing the reconfiguration logic (P51, P70, and P73), and in another three cases general well-formedness rules are used (P61, P66, and P69). In P7, the validity is only checked in regard to whether application-specific timing constraints are met, which differs from the other papers in this category.

Fixed valid configurations: Six papers (P46, P53, P63, P65, P76, and P78) use a fixed configuration space that only contains valid configurations. Either the reconfiguration algorithm can only chose from a predefined set of valid configurations (P46, P53, P76, and P78), or the reconfiguration space consists only of simple replacements that are always valid (P63 and P65).

Valid by construction: Four papers (P4, P56, and P59–60) ensure the validity of target configurations by the way they construct their reconfiguration mechanism. Compared to using a fixed number of valid configurations to choose from, in these papers, additional constraints to be fulfilled by possible target configurations must be considered, and the reconfiguration logic cannot just choose any valid configuration to fulfill the task. In P4, the reconfiguration of a robotic system realized based on port-based objects, which are composite units that have defined numbers of input and output ports using which they can be connected with each other. Validity is achieved when all input ports of port-based objects are connected to output ports. In P59 and P60, validity must be manually verified at development time, while in P56, validity is encoded in the optimization problem to be solved at runtime, resulting in only proposing valid reconfigurations.

Trial and error: P12 proposes a trial-and-error-based reconfiguration method. This method involves attempting various configurations until the robot has reconfigured into a configuration that allows the execution of an intended task.

Overall, since below a third of all investigated papers consider checking the validity reconfigurations, validity seems to be a minor concern in the robotics community so far, although it is essential for the huge configuration spaces proposed in many papers. Much work in this direction has already been done in the product line community, e.g., when focusing on finding optimal configurations (Henard et al. 2015; Guo and Shi 2018; Pereira et al. 2021). Nevertheless, the more recent papers considered in our SLR already seem to integrate such results (P45, P48, P51, P52, P55, P56, P70, P72–75, and P77), the oldest of which (P70) dates from 2010.

Besides this, nine additional papers consider the validity of reconfiguration, but these are practically infeasible for more complex reconfiguration scenarios. In particular, this comprises the papers that work with a fixed pool of valid configurations (P46, P53, P63, P65, P76, and P78), a constructive approach that limits the number of possible configurations (P4), or completely manual verification (P59 and P60). However, these papers were mainly published between 1997 and 2006 and might represent a dated view on the validity of reconfigurations. Still, four of these papers (P46, P53, P76, and P78) were recently published between 2019 and 2022.

Nevertheless, we see a trend towards systematic validity checks against the design-time specification of the configuration space and task requirements, as 56% of the papers dealing with target configuration validation (mainly the more recent ones) check the validity of reconfigurations against such specifications.

7.1.3 Cost of reconfigurations

Roughly, one third of all investigated papers (36.62%) consider the cost related to reconfiguration in some form. In 51.85% of the papers, this cost arises from additional computations necessary for performing computations. All other papers do not explicitly mention a source of cost. The papers capture the cost in two different dimensions, were only P39 considers both dimensions:

Time: With 18 papers, the majority considers the cost in terms of time needed for reconfiguration (P9, P17, P38–39, P42, P44, P47–50, P52, P56, P61, P64, P66, P69–70, and P78). Among others, taking too long for reconfiguration could lead to violating real-time constraints or having to interrupt the operation.

Resources: Three papers name the resource usage caused by a reconfiguration. P73 reports a static memory overhead due to running a reconfiguration engine. P27 and P39 generally mention that the usage of computing resources could impact the performance of a robotic system since they are partially blocked during reconfiguration.

Among the papers that consider the cost of reconfiguration, 11 papers do not further specify where these cost arise (P9, P20, P22–23, P27, P38, P44, P50, P61, and P75–76). The other papers identify two sources of cost:

Planning: The majority (13 papers) consider the cost of planning of how to reconfigure the robotic system (P17, P39, P46–49, P52, P56, P63, P66, P69–70, and P73). The cost of planning a reconfiguration depends on the configuration space and the number of possible reconfigurations as well as the way how possible configurations are computed. Calculating an optimal solution in a large configuration space might not scale (P46, P49, P56, P63, P66, P69, and P73). However, assuming a sufficiently small configuration space, an optimal solution is still feasible, since the authors measured for P52 on average 5.57s generating reconfigurations followed by another 5.94s for model checking them ro generate an optimal target configuration. Faster planning can be achieved by accepting non optimal target configurations or restricting the reconfiguration problem (P17, P47, P49, and P70) or incremental reconfiguration planning (P39). If the planning is designed well, the time for planning can be reduced to times below six seconds (P70) or even 0.66 seconds (P47).

Execution: Only three papers consider the costs of actually executing a planned reconfiguration (P42, P64, and P78). P42 and P78 actually measured the cost of executing a structural reconfiguration of composite units, with the latter measuring 0.47 seconds to change a configuration and the other 4.2 seconds. In summary, the cost of executing a reconfiguration is not negligible, but can be relatively low.

7.2 Reconfiguration mechanisms of robotics frameworks

To capture the state-of-practice concerning RQ3, we investigated what reconfiguration mechanisms robotics frameworks provide. Again, we based this analysis on our conceptual model introduced in Sec. 4, and Table 4 also contains the classifications of the robotics frameworks concerning reconfiguration mechanisms.

7.2.1 Interactions

All frameworks provide means to implement runtime interactions, e.g., data exchanges or calling functionality, among reconfigurable elements that are orchestrated by one or more

Runtime Manager. RobMoSys uses SmartEventServer for publish/subscribe communication and SmartParameterMaster for parameter-based communication. ROS has a centralized ROS master, while ROS2 uses a decentralized data distribution service (DDS). In YARP, every component periodically retrieves incoming messages from central framework entities.

The robotic frameworks provide different *Interaction mechanisms*, allowing reconfigurable assets to interact at runtime. They mainly use message-based communication; e.g., ROS nodes can publish and subscribe to messages based on topics, and composite units can subscribe to others in RobMoSys. Furthermore, ROS, YARP, and RobMoSys offer client/server communication. In contrast, the communication in OROCOS is purely parameter based and the entire control flow is handled by the framework. In all robotic frameworks, *Interaction binding* takes place when the source code statements implementing interactions are executed (late dynamic).

7.2.2 Trigger & logic

Only RobMoSys provides a wide variety of possibilities for specifying *Decision Criteria* in terms of reconfiguration triggers and reconfiguration logic. ROS, OROCOS, and YARP provide some kind of behavior model that is not explicitly intended to specify reconfiguration. Instead, reconfiguration is mainly implemented in the composite units using the framework's utility functions (equivalent to the hard-coded specification of reconfiguration triggers and logic that we found in the literature (see Sec. 7.1.1)). Consequently, developers are obliged to consider and address all potential side-effects, such as the time required for startup or shutdown of a component, and any intermediate states that may arise from this. In this regard, the frameworks offer minimal support in addressing these challenges.

Of the *Change types* in our conceptual model, all frameworks support reconfiguration of the *structure* of a robot's software, which is also the most commonly observed change type in the SLR, and *parameter* reconfiguration. Only ROS supports the *Change type* of *functionality* reconfiguration through its `pluginlib`.

In ROS, parameter reconfiguration is implemented in the *dynamic_reconfigure* library. While ROS does not provide explicit support for structural reconfiguration, launching and terminating composite units is implemented in the *roslaunch* library, which can also used for structural reconfiguration at runtime. Loading basic units is implemented in *pluginlib/nodelet* depending on the concrete realization of the basic unit that should be instantiated. OROCOS offers the API *TaskContext* for accessing and changing parameters and the API *Scripting* for structural reconfiguration in terms of loading plugins. A component called *DriverCreator* manages the loading of components in YARP. In the SmartSoft implementation of RobMoSys, parameter reconfiguration is managed by a *SmartParameterMaster* and *SmartParameterClient*, while components are organized by the service *SmartTask*. However, the intention of RobMoSys is to approach reconfiguration using model-based techniques, where reconfiguration is specified in models rather than implemented in code; models are then executed to realize the behavior and reconfiguration of the robotic system.

In conclusion, the specification of reconfiguration triggers and logic, as well as their execution in robotic systems, is where we see the biggest mismatch between what is proposed in the literature and what robotic frameworks provide. With the exception of RobMoSys, no framework provides the means to explicitly specify the reconfiguration space, nor to specify concrete reconfigurations. To some extent, behavioral models, e.g., using statecharts or behavior trees (Colledanchise and Ögren 2018), could be used to specify reconfiguration. However, their main purpose is to describe actual behaviors in terms of self-adaptation. As part of this, reconfigurations could be triggered, for example, in the implementation of an action node in a behavior tree. In this case, however, the actual reconfiguration must still be implemented using the reconfiguration-related APIs of the frameworks. In the end, the intended method is to hard-code the reconfiguration mechanisms using the reconfiguration-related APIs of the frameworks, e.g., for launching and terminating composite units from code. This pure focus on source-code APIs of the frameworks is somewhat contradictory to the significant focus on specifying reconfiguration using DSLs and reference architectures for reconfigurable robots observed in our SLR. However, one can clearly see that RobMoSys originates from an academic research project, as it provides exactly such specification formats for decision criteria. However, these libraries are not explicitly defined in any framework with a focus on reconfiguration, but are intended for launch-time configuration.

7.3 Reconfiguration in ROS (sub-)systems

To investigate how reconfiguration is actually implemented in practice and to derive best practices for implementing reconfiguration, we investigated the source code of open-source robotics (sub-)systems. To this end, we started the investigation with a list of 115 open-source ROS-based (sub-)systems of Malavolta et al. (2021). One of these repositories was no longer publicly accessible. We filtered the accessible repositories for all (sub-)systems that use one of the identified ROS libraries (see Sections 6.2 and 7.2) suitable to implement reconfiguration. Since the libraries have to be explicitly imported, this was easily done by name matching. We kept all (sub-)systems whose source files contain one of the following library names: (i) *roslaunch*, (ii) *nodelet*, (iii) *pluginlib*, and (iv) *dynamic_reconfigure*. This resulted in 48 (sub-)systems that potentially contain implementations of reconfiguration. Table 5 shows a list of the investigated (sub-)systems and which libraries they use. In some cases, one of the keywords was used in comments of the source code, e.g., in a description of how to manually launch the (sub-)system using *roslaunch*. We provide all details on the (sub-)systems and our findings in our replication package (Peldszus et al. 2024a).

To get a better overview of these (sub-)systems, we extracted their ROS versions. We identified 104 (sub-)systems that are based on ROS and 10 (sub-)systems based on ROS2 in the dataset of Malavolta et al. (2021). After filtering these (sub-systems) as described above, only *moveit2* remained as ROS2-based (sub-)systems that potentially implements reconfiguration.

We then studied the selected (sub-)systems in-depth, analyzing how reconfiguration is implemented (RQ3). For each match location, we first checked whether the match could be part of a reconfiguration or not, e.g., because the match was in a comment. Thereafter, we started our in-depth investigation from each code line in which one of the libraries is used. First, we inspected each usage in detail and identified the purpose for which the library is used at that location. Afterwards, we did both a forward navigation along the dependencies, i.e., method calls and field accesses, and looked at all the code that was referenced from that location, as well as a backward navigation. This allowed us to determine how the reconfiguration was triggered and executed.

7.3.1 Observations

Figure 9 summarizes how often we found which kind of reconfiguration in the investigated robotic (sub-)systems. Of the 48 (sub-)systems investigated in-depth, 17 systems contain no reconfiguration at all. The libraries are used purely for launching the (sub-)systems and sometimes for launch-time configuration. Table 5 shows in detail which (sub-)systems contain

Table 5 Investigated robotic (sub-)systems, the ROS libraries they use (✓used library, (✓) only text match in the source code, – not used), and the implemented reconfiguration (✓implemented reconfiguration, (✓) partially implemented reconfiguration, – not implemented)

		Used ROS Libraries				Reconfiguration of		
ID	(Sub-)System	dynamic_reconfigure	nodelet	pluginlib	roslaunch	Unit Parameters	Basic Units	Composite Units
1	ani	–	–	–	–	–	–	–
2	autorally	✓	✓	✓	–	✓	✓	–
3	avoidance	✓	✓	–	–	✓	–	–
4	bebop_autonomy	✓	✓	✓	–	✓	–	–
5	capabilities	–	✓	–	–	–	–	–
6	cob_command_tools	✓	–	–	–	–	–	–
7	cob_control	✓	–	✓	–	✓	–	–
8	cob_environment_perception	✓	✓	✓	–	✓	✓	–
9	cola2_core	✓	–	–	✓	✓	✓	–
10	ed	–	–	✓	–	–	–	–
11	elfin_robot	✓	–	✓	–	✓	–	–
12	evapi_ros	✓	–	–	–	✓	–	–
13	exotica	–	✓	✓	–	–	–	–
14	flexbe_behavior_engine	–	–	–	✓	–	–	–
15	franka_ros	–	–	✓	–	–	–	–
16	free_gait	–	–	✓	✓	–	✓	–
17	FusionAD	✓	✓	✓	–	✓	–	–
18	grvc-ual	–	–	–	(✓)	–	–	–
19	h4r_ev3_ctrl	–	–	✓	–	–	–	–
20	iop_core	–	–	✓	–	–	–	–
21	mas_domestic_robotics	✓	–	–	–	✓	–	–
22	mavros_controllers	✓	–	–	–	✓	–	–
23	micros_swarm_framework	–	–	✓	–	–	✓	–
24	motoman_project	–	–	✓	–	–	–	–
25	moveit	✓	✓	✓	✓	✓	✓	(✓)
26	moveit2	✓	✓	✓	✓	✓	✓	(✓)
27	multimaster_fkie	✓	✓	–	✓	✓	–	–
28	multi_tracker	✓	–	–	–	–	–	–
29	navigation	✓	–	✓	–	✓	✓	–
30	neonavigation	✓	–	–	–	✓	–	–

Table 5 *Continued*

		Used ROS Libraries				Reconfiguration of		
ID	(Sub-)System	dynamic_reconfigure	nodelet	pluginlib	roslaunch	Unit Parameters	Basic Units	Composite Units
31	opencv_apps	✓	✓	✓	–	✓	–	–
32	rapp-platform	✓	–	✓	✓	✓	–	–
33	roboracing-software	✓	✓	✓	(✓)	✓	(✓)	–
34	rocon_multimaster	–	–	–	✓	–	–	–
35	ros_control	–	–	✓	–	–	–	✓
36	ros_people_model	✓	–	–	–	✓	–	–
37	rosplane	✓	–	–	–	✓	–	–
38	ros_tms	–	✓	✓	✓	–	–	–
39	rtabmap_ros	–	✓	✓	–	✓	–	–
40	rtmros_common	✓	–	–	–	✓	–	–
41	SailBoatROS	–	–	–	✓	–	–	–
42	self-driving-golf-cart	✓	✓	✓	–	✓	–	–
43	spencer_people_tracking	✓	✓	✓	✓	–	–	–
44	sphero_swarm_ros	✓	–	–	–	✓	–	–
45	teb_local_planner	✓	–	✓	–	✓	–	–
46	tuw_multi_robot	✓	✓	✓	–	✓	–	–
47	utexas-art-ros-pkg	✓	–	–	–	✓	–	–
48	vigir_footstep_planning_core	–	–	✓	–	–	–	–

```
1  void CameraTrigger::onInit() {
2    ...
3    // set up dynamic_reconfigure server
4    dynamic_reconfigure::Server<camera_trigger_paramsConfig> ::CallbackType cb;
5    cb = boost::bind(&CameraTrigger::configCallback, this, _1, _2);
6    m_dynReconfigServer.setCallback(cb);
7    ...
8  }
9  ...
10 void CameraTrigger::configCallback(const camera_trigger_paramsConfig &config, uint32_t) {
11   m_triggerFPS = config.camera_trigger_frequency;
12
13   // send new FPS to arduino
14   m_port.lock();
15   m_port.writePort("#fps:" + std::to_string(m_triggerFPS) + "\r\n");
16   m_port.unlock();
17 }
18 ...
```

Listing 1 Parameter Reconfiguration in "*AutoRally*"

which kind of reconfiguration. In what follows, we describe implementations of reconfiguration we found.

The majority (28 robotics (sub-)systems) uses parameter reconfiguration, mostly to reconfigure low-level parameters close to the hardware. For example, Listing 1 shows a code excerpt of the "*AutoRally*" implementation in which parameter reconfiguration is used to update the frequency of a camera. In lines 2 to 5, the dynamic_reconfigure server is configured by registering a call back method at the server. This method is shown in lines 10 to 17 and simply changes the frequency of the camera and writes it to the hardware. We found no (sub-)system in which parameter values are changed from within the implementation, but these are provided for assignment by external entities. Surprisingly, only a few (sub-)systems check assigned parameter values, e.g., whether these are within a plausible range. While most (sub-)systems assign the updated parameters immediately to local variables, we still frequently observed locks or flags indicating changes to avoid changing ongoing executions. For example, in "*evapi_ros*," the configuration of controllers is only updated when the configuration values are changed. Listing 2 shows the implementation of this behavior. Whenever the callback method registered with dynamic_reconfigure is called, a flag is set in line 3. During the main execution loop in lines 9 through 17, this flag is checked and only if there are changes, the individual controller configurations are updated.

Nine (sub-)systems contain reconfiguration at the level of basic units. Functionalities are dynamically enabled, which is usually done by reacting to parameter reconfiguration or providing services. For example, as shown in Listing 3, "*micros_swarm_framework*" provides

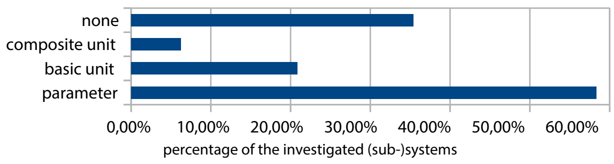

Fig. 9 Granularity of reconfiguration implemented in the investigated ROS (sub-)systems

```
// Callback function to change pid parameters at runtime.
void CallbackReconfigure(evarobot_controller::ParamsConfig &config, uint32_t level) {
 b_is_received_params = true;
 g_d_wheel_separation = config.wheelSeparation;
 ... // copying all remaining values of the config
}

int main(int argc, char **argv) {
 while ( ros::ok() ) {
  // If new parameters are set, ... and controller parameters are updated.
  if ( b_is_received_params ) {
   controller1.UpdateParams(g_d_p_1, g_d_i_1, g_d_d_1, g_d_w_1);
   controller2.UpdateParams(g_d_p_2, g_d_i_2, g_d_d_2, g_d_w_2);
   b_is_received_params = false;
  }
  ...
 }
 ...
}
```

Listing 2 Use of Update Flags to Indicate Changes during Parameter Reconfiguration in "*avapi_ros*"

```
1  AppManager::AppManager():app_loader_("micros_swarm", "micros_swarm::Application") {
2   ros::NodeHandle nh;
3   app_load_srv_ = nh.advertiseService("app_loader_load_app", &AppManager::loadService, this);
4   app_unload_srv_ = nh.advertiseService("app_loader_unload_app", &AppManager::unloadService, this);
5   ...
6  }
7
8  bool AppManager::loadService(app_loader::AppLoad::Request &req, app_loader::AppLoad::Response &resp) {
9   std::string app_name = req.name;
10  std::string app_type = req.type;
11
12  bool app_exist = recordExist(app_name);
13  if(app_exist) {
14   ...
15   return false;
16  }
17  else {
18   boost::shared_ptr<micros_swarm::Application> app;
19   try {
20    app = app_loader_.createInstance(app_type);
21   }
22   catch(pluginlib::PluginlibException& ex) {
23    ROS_ERROR(...);
24   }
25   ...
26   return true;
27  }
28 }
```

Listing 3 Reconfiguration of Basic Units in "*micros_swarm_framework*"

services for loading and unloading apps, which are tightly coupled to the framework and, therefore, are classified as basic units by us. In line 3, the method for loading services, which is defined in lines 8 to 28, is advertised as a ROS service. When this service is called, this method will load an app using *pluginlib* (line 20) if it is not already running. However, only two (sub-)systems dynamically load plugins or nodelets (the realization of basic units in ROS) as

```
1  void AutoRallyChassis::onInit() {
2   ...
3   runstopSub_ = nh.subscribe("/runstop", 5, &AutoRallyChassis::runstopCallback, this);
4   ...
5  }
6
7  void runstopCallback(const autorally_msgs::runstopConstPtr& msg) {
8   runstops_[msg−>sender] = ∗msg
9  }
10
11 void AutoRallyChassis::setChassisActuators(const ros::TimerEvent&) {
12  ...
13  //check if motion is enabled (all runstop messages = true)
14  if(runstops_.empty()) {
15    chassisState−>runstopMotionEnabled = false;
16  } else {
17    chassisState−>runstopMotionEnabled = true;
18    int validRunstopCount = 0;
19    for(auto& runstop : runstops_) {
20      ...
21    }
22  }
23 }
```

Listing 4 Reconfiguration of Basic Units in "*AutoRalley*" based on Variables to Enable or Discable Units

part of the reconfiguration. The other system (*cob_environment_perception*[3]) creates algorithm objects configured for their intended use case, but classloading takes place already at the responsible node's instantiation. The rest enables or disables basic units by using variables that are checked in conditions to decide whether the basic unit, e.g., a functionality, should be executed or not. For example, Listing 4 shows how "*AutoRalley*" allows us to write messages to the topic "*/runstop*" to enable motion. During initialization, the corresponding topic is subscribed (line 3) and a callback function registered, which writes all received messages into class-scope variable. During execution, this information is processed to decide whether the basic unit implementing movement is enabled or disabled (lines 8–19).

Only three (sub-)systems provide reconfiguration on the level of composite units to some limited extent. The supported reconfiguration is far behind what is considered in the literature and provided by frameworks such as RobMoSys. Further, two of these subsystems are ultimately the same subsystem (*moveit*), in its two implementations for ROS and ROS2, respectively. The other subsystem is *ros_control* that provides a service to load controllers using *pluginlib*. In principle, this allows structural reconfiguration, but the logic has to be implemented in another subsystem using *ros_control*. The corresponding code is essentially the same as the one shown in Listing 3, with the exception of independently executable composite units being loaded.

7.3.2 Best practices for reconfiguration

From our insights on the implementations of reconfiguration in robotics (sub-)systems, we derived best practices for implementing reconfiguration. Since only parameter recon-

[3] https://github.com/ipa320/cob_environment_perception/blob/indigo_dev/cob_3d_registration/ros/src/registration_nodelet.cpp

```
boost::recursive_mutex parameter_server_mutex_;

// The callback method registered with dynamic_reconfigure
void cbParameter(const SafetyLimiterConfig& config, const uint32_t) {
 boost::recursive_mutex::scoped_lock lock(parameter_server_mutex_);
 hz_ = config.freq;
 timeout_ = config.cloud_timeout;
 ... // setting of further config values
}
```

Listing 5 Lock-based Parameter Reconfiguration in "*neonavigation*"

figuration is widely implemented, we focus on best practices for implementing parameter reconfiguration.

Patterns for parameter reconfiguration Any robotic (sub-)systems that includes parameter reconfiguration must implement its reconfiguration logic. Depending on how many different parts of the implementation need to be reconfigured to react to a parameter reconfiguration, we observed two best practices for implementing the reconfiguration logic. To implement an easily comprehensible reconfiguration logic, it is essential to choose the identified practice that best suits the robotic system.

The first one is called *Reconfigure callback*, reconfiguration logic is implemented in a callback method that is registered at the ROS API *dynamic_reconfigure*. In particular, for simple reconfigurations that can be applied immediately to the robotic system, we observed many examples in which this is a simple, but perfectly suitable practice. The example in Listing 1 can be seen as one instance. However, for more complex systems, particularly, systems in which reconfiguration has to be applied in multiple parts, following this pattern does not scale.

The second pattern addresses the cases in which the first one does not scale. It is called *Message-based* and concerns letting single working parts of the implementation subscribe a topic and to broadcast the new values. It is used when many different and probably independently working parts are affected by a parameter reconfiguration. Using this practice, the logic specific to an individual part of the robotic system can be located in this part, avoiding overly complex implementations of the callbacks.

Processing of updated values It is essential to be able to update parameter values in the running system without having a negative impact on the current execution. We mostly observed two ways to implement the processing of updated parameter values: *Stateless execution* and *stateful execution*. Both serve as best practices for specific task characteristics as outlined in what follows.

In stateless execution, the reconfigured parameters are applied each time the reconfigured functionality is executed. Typically, the functionality is executed in a loop, and the parameter values are copied to variables that are in the scope of the loop at the beginning of each iteration. This way there is no interference during execution and the implementation is simple and easily comprehensible. However, the values are only updated at the beginning of an iteration, making this practice suitable for frequently executed, short-running tasks.

For tasks that need updating parameters also during task execution, another practice that considers the state of the running execution is needed. In stateful execution, the object whose functionality can be reconfigured using parameter reconfiguration, maintains an internal state besides the parameters. Thereby, the parameters can potentially interact with the internal state

```
class CameraWrapper : public UEventsHandler {
  // Callback registered at dynamic_reconfigure
  void callback(rtabmap_legacy::CameraConfig &config, uint32_t level) {}
  if(camera) {
    camera−>setParameters(config.device_id, config.frame_rate, config.video_or_images_path, config.pause);
  }
}

void setParameters(int deviceId, double frameRate, const std::string & path, bool pause) {
  if(cameraThread_) {
    rtabmap::CameraImages ∗ imagesCam = dynamic_cast<rtabmap::CameraImages ∗>(camera_);
    ...
    imagesCam−>setImageRate(frameRate);
    if(pause && !cameraThread_−>isPaused()) {
      cameraThread_−>join(true);
    } else if(!pause && cameraThread_−>isPaused()) {
      cameraThread_−>start();
    } else { ... }
    ...
  }
 }
}
```

Listing 6 Parameter Reconfiguration in "*rtabmap_ros*" Pausing or Resuming Execution of the Reconfigured Camera If Needed

of the object, requiring a mechanism to notify the object about changed parameters. In the (sub-)systems, we found three different realizations of this practice: (i) Change Flag – the object frequently checks a change flag and, if necessary, the reloading of parameter values is triggered, as shown in Listing 2; (ii) Lock – the object provides direct access to the internal configuration values, and a locking mechanism controls the execution while updating live parameters, such as *neonavigation* realizes it using the C++ boost library as shown in Listing 5; and (iii) Callback – a callback method is provided to stop the current execution and to trigger continuing the execution with the new parameters, such as the implementation of parameter reconfiguration of *rtabmap_ros* shown in Listing 6 that resumes the camera threat if it has been paused. Actually starting and stopping the camera is offered via two services. However, due to the complex execution logic, the latter mechanism is mainly suitable for continuous or long-running tasks.

Soundness check of new values To avoid faulty reconfigurations, it is essential to check new, potentially externally provided, values for validity. Although this is a well-known best practice (OWASP 2021), we only rarely found such checks in the parameter reconfiguration implemented in the robotic (sub-)systems we studied. Given the importance of sound parameter values for safe execution, this best practice is related to the validation of reconfigurations that we considered in the SLR. Since the frameworks do not provide support for checking the validity of reconfiguration, this must be implemented in the robotic (sub-)system itself. Here, we observed two aspects in the robotic (sub-)systems that need to be considered for properly implementing soundness checks concerning the individual characteristics of a robotic system, namely *Time of Check* and *Checked Properties*.

The new values assigned to unit parameters may be checked at different times, depending on the execution characteristics of the functionalities that use these parameters. We have observed two common practices: (i) immediate checking – new parameter values are checked immediately by simple runtime checks to prevent them from being inadvertently processed unchecked; and (ii) on-demand checking – if the new parameter values are not used immediately, the checks are performed on demand to avoid unnecessary checks. On-demand checking is particularly appropriate when there is only one entity that reads them. Depending on the likelihood that parameters will change again between reconfiguration and the next use of that parameter, immediate checks or on-demand checks may be more suitable. For unit parameters that are reconfigured more often than they are used, on-demand checks are more suitable, while frequently used but infrequently reconfigured unit parameters can be checked more efficiently with immediate checks.

One challenge in implementing soundness checks is to determine what to check. While the properties to check can be application-specific, we identified two practices that should be mostly applicable and that are frequently checked in the (sub-)systems implementing soundness checks of new values: (i) value range – it should be always checked if the new parameter values are within the expected value range; and (ii) consistency – when having multiple parameters, it is essential to check if these are consistent and expected relations among the unit parameters are fulfilled; the same might apply if the parameter value has to relate to the current system state.

7.4 Discussion of reconfiguration mechanisms

The majority of the literature focuses on approaches for specifying structural reconfiguration using domain-specific languages (DSLs) and executing them at runtime. During this execution, constraints on the validity of target configurations are usually checked or only valid configurations are computed by design. In contrast, robotic frameworks only provide low-level application programming interfaces (APIs) for loading or unloading assets and changing parameter values. Consequently, the necessary logic for implementing the structural reconfiguration considered in the literature must be handwritten by the developers of robotic systems. Academic robotic frameworks can provide wrappers around more low-level frameworks that allow specifying the reconfiguration logic using models. However, in robotic (sub-)systems, we did not observe such reconfiguration, but only the frequent use of parameter reconfiguration. A particular challenge arises from ensuring the validity of reconfigurations and avoiding invalid target configurations and or intermediate states. While the frameworks provide no support in this sense, we identified concrete practices for implementing reconfiguration safely. Further academic work, such as GenoM (Fleury et al. 1997), specifically focuses on generating code with corresponding validations. However, this work was not covered by our literature review and not observed in practice.

8 Discussion

We now discuss our results. In particular, we relate the outcomes of our three data sources, the SLR, the frameworks, and the robotic (sub-)systems. But, we also discuss our observations with respect to the state-of-the-art of reconfiguration in other domains.

8.1 State-of-the-art vs state-of-practice

From the analysis of the state-of-the-art as captured by our SLR, we identified several techniques to structural reconfiguration. As shown in Fig. 5, the most popular granularity for reconfiguration is composite unit, followed by unit parameter, and basic unit. Thereby, the configurations have primarily a lifespan at the scope of a task or mission (see Fig. 6). However, the frameworks do not provide advanced support in terms of reconfiguration triggers and logic for this kind of reconfiguration, only low-level APIs that allow manual implementation of structural reconfiguration. Probably related to this limitation, in robotics, ROS-based (sub-)systems, as shown in Fig. 9, structural reconfiguration is almost completely absent, and only parameter reconfiguration is used intensively. The low-level APIs provided by ROS are primarily used for implementing parameter reconfiguration. Thereby, the best practices concerning safety derived by us are only applied in few robotics (sub-)systems.

One explanation for this discrepancy could be that these reconfigurations are not necessary in real-world settings and in non-academic setups. Another explanation could be that the required advanced reconfiguration support is only now entering the robotics domain, e.g., in projects such as RobMoSys, but is not widely available. Also, current robotic systems have not yet reached the complexity envisioned by the scientific community. However, we expect them to do so in the near future, and this will trigger the need to provide sophisticated reconfiguration support to robotic system designers. We believe that robots will increasingly become multi-purpose, i.e., able to perform various tasks, with the consequent need of customization and reconfiguration capabilities. Moreover, they will be increasingly required to be used in uncontrolled environments, often shared with humans. This is visible also in the various competitions and challenges organized in robotics conferences and forums. The demand for autonomy, run-time reconfiguration, and deployment of software also after production, as well as the use of AI solutions, will often need to deal with requirements of compliance to security and safety standards. This is indeed another dimension of complexity that the community will need to face; the compliance to safety and security standards will become incremental and continuous as it is already happening in the automotive domain (Santilli et al. 2024). Finally, the programming and use of robots should become more accessible to end-users without knowledge in robotics or ICT. This will be a need both for industrial and service robots. In fact, according to our experience, small and medium enterprises are sometimes reluctant to use robotics solutions for automating their tasks because they miss the capabilities to program and configure robots according to their (evolving) needs.

In the related domain of autonomous driving, we already found such an example of composite unit-level reconfiguration in the open-source driving system Autoware.auto (The Autoware Foundation 2023). In this system, different motion planners are used based on the current driving scenario, such as lane following, lane changing, or parking. The reconfiguration mechanism is a custom implementation of this project, and they use behavior trees (Colledanchise and Ögren 2018) as an executable model for specifying the reconfiguration logic. Considering the assumed complexity of reconfiguration in the reviewed literature, this example is still relatively simple and is the only example of coarse-grained reconfiguration that we found in this system, but may indicate a future need for more complex reconfiguration.

Another driver for reconfiguration in robotics could be machine learning (ML). Today, many ML-enabled system, i.e., robotic systems, employ complex data processing pipelines involving multiple ML models, e.g., for multi-stage perception of the environment (Peldszus et al. 2024c). In this context, particularly due to the size of models and the numerous tasks implemented in robotic systems using ML, i.e., the number of ML models used. Particularly large language models (LLM), such as GPT, Llama, Claude, or Qwen, need multiple gigabytes of memory to be executed. For example, we measured that a Qwen2.5-72b LLM (Yang et al. 2024) with 3.5 bits per word and 24k tokens context length already needs 40GB of GPU memory. Due to this huge memory demands it is infeasible to execute all models in parallel but it may be necessary to reconfigure the robotic system, i.e., which model is loaded. This could be a main driver for structural reconfiguration at the granularity of composite units or even base units in practice. Also, this could lead to resource limitations, which we observed in our SLR to be not being a main reason for reconfiguration any longer, to become more relevant again.

One of the obstacles to the more widespread adoption of reconfiguration may be the additional complexity it introduces and the necessity to address it in all phases of development. Among other things, this added complexity is particularly challenging for testing. In a previous interview study we conducted to validate a reactive security monitor capable of reconfiguring the monitored system (Peldszus et al. 2024b), e.g., to put it into a safe mode in case of detected malicious actions, a security engineer from a large automotive supplier pointed out the testing challenges. Standards such as SOTIF, which can be relevant to any robotic system, require companies to test all possible operating modes for safety, which can be challenging in the context of reconfiguration, since all possible configurations must be tested in combination with all points in time at which reconfigurations can occur.

8.2 Reconfiguration in other domains

The reconfiguration mechanisms provided by robotic frameworks, especially the non-academic ones, are currently limited. However, state-of-the-art frameworks in many domains allow sophisticated reconfiguration, e.g., the Linux kernel, the Debian Linux distribution, the Eclipse IDE (OSGi), and the Android operating system (Berger et al. 2014). Others, such as the eCos operating system, only support compile-time configuration, but come with sophisticated DSLs and configuration tools that may be suitable for reconfiguration.

The investigated robotic frameworks have a well-defined asset base, but with the exception of RobMoSys, they lack a clear specification of the reconfiguration space. The Linux kernel and eCos can serve as inspiration with their feature-model-like DSLs to specify the configuration space. Robotic systems would benefit from explicit configuration space modeling also for the reconfiguration at runtime, e.g., to analyse possible configurations and potential conflicts that could occur at runtime (Franz et al. 2021) or to analyse security vulnerabilities (Peldszus et al. 2018). Positively, with the RobMoSys project, researchers have already been working on providing such methods and tools to the robotic domain.

Robotic frameworks provide more interfaces for interaction among the assets than state-of-the-art software ecosystems (Berger et al. 2014) that mainly rely on the programming language-specific interface specifications and focus on direct source code interactions among the assets. The focus on pure source code interfaces in the state-of-the-art software ecosystems allows mainly static and dynamic linking as interaction mechanisms. Only Eclipse and Android, which support sophisticated class or module loading, need an interaction manager at runtime. Such a manager or even multiple managers are provided by all robotic frameworks. In the end, the interaction management in the robotic frameworks is comparable to Eclipse's services and extension points.

When it comes to the reconfiguration mechanisms, the software ecosystems are comparable to the investigated robotic frameworks. In the Linux kernel, kernel modules are dynamically loaded when these are accessed. The same applies to plugins in Eclipse, which can be dynamically loaded by OSGi once any of the contained Java classes is accessed. Like most robotic frameworks, none of the software ecosystems provides techniques for explicitly specifying reconfiguration. However, unlike the robotic frameworks, reconfiguration is also not considered one of the core aspects in them.

9 Threats to validity

A threat to the validity of the SLR could arise from the fact that the initial paper selection depends primarily on the second author. To mitigate this threat, we tended to consider a paper as potentially relevant rather than discarding it. In addition, we performed a snowballing iteration that allowed important but initially erroneously discarded papers to be reconsidered. In fact, of the 5,089 initially excluded papers, 48 papers were reconsidered in this way, and we decided to look at 24 of them in more detail using multiple authors, but only included 6 papers for classification.

The focus on ROS (sub-)systems in the review of how reconfiguration is implemented in practice represents a threat to external validity. However, ROS could be considered representative because it is the only widely used robotic middleware. Although our investigation of robotic frameworks has revealed that ROS and even ROS2 do not provide advanced reconfiguration techniques, our investigation shows that reconfiguration is actually implemented in ROS systems. Therefore, ROS might not be the perfect example for investigating reconfiguration in practice but we also lack better practically adopted frameworks, which yet have to be developed.

Since ROS2-based systems are underrepresented in the sample of robotic systems studied, our results apply only to ROS-based systems. However, we did not find any major improvements in ROS2 regarding reconfiguration during our framework review, nor did we find any differences between *moveit* and *moveit2* regarding reconfiguration. This suggests that our findings may also apply to ROS2-based systems, but it remains for future work to confirm this hypothesis.

A further threat is that we only considered open-source systems and we did not consider bespoke or industrial frameworks. In a follow-up study, it would be interesting to investigate also these frameworks since they could have better reconfiguration mechanisms and/or make use of a custom middleware.

A possible source of bias arises from the backgrounds of the authors and might threaten internal validity. We mitigated this bias by including a diverse set of authors, including authors from the software engineering domain, and authors whose expertise is in robotics. Moreover, as a template basis for our comparison, we used the categorization from Berger et al. (2014). Two related threats are that this template might give mainly a software viewpoint, as opposed to a hardware viewpoint, and that it might be outdated. For mitigation, we adapted the template based on our SLR and secondary literature.

Finally, while we provided justification for our conceptual model based on the papers considered in our SLR, we did not perform additional evaluation, independent of the SLR, for it. Such an evaluation, e.g., based on expert opinions, could provide further valuable insight. However, we are confident that the conceptual model in its present form serves its purpose for this paper, to compare how available state-of-practice frameworks implement the reconfiguration of robots described in the state-of-the-art literature.

10 Conclusion

We determined the state-of-the-art and the state-of-practice of automated runtime software reconfiguration in robotics. We analyzed the former by surveying the literature on reconfiguration in robotic systems, inspecting 78 relevant papers in detail. We analyzed the latter

Table 6 Implications for researchers and practitioners

Implications
Academics: In the summary of RQ1 above, we highlight that reacting to safety issues is a not so common motivation for developing dynamically reconfigurable robotic software systems. However, since often robots need to work in collaboration with humans and/or in safety critical domains, there is the need of rigorous software enginering approaches in robotics. Due to the identified significant mismatch between the state-of-the-art and state-of-practice, it is essential to further collaborate with practitioners. On one side, this is necessary to guarantee that academic research is aligned with industrial needs, and, on the other side, to enable technology transfer from academia to industry.
Practitioners: Robots will be multi-purpose and need to deal with high levels of uncertainty, e.g., in the environment, reflected in sensor performance and reliability. This will probably lead to more complexity and variability in robots and, consequently, requires structural and more sophisticated reconfiguration techniques. It will become increasingly important to decouple the specification of reconfigurability from the application logic. Available DSLs we surveyed in the state-of-the-art could become relevant. Moreover, explicitly specifying the configuration space of robotic systems allows configuration tools and configuration editors, and increases the reliability of reconfigurable robotic systems. The need of structural and more sophisticated reconfiguration techniques may require to mitigate the identified discrepancies between the state-of-the-art and state-of-practice concerning reconfiguration. Robotic frameworks could be extended to (i) explicitly support context reconfiguration, (ii) provide support for reconfiguration triggers and logic, taking inspiration from academic frameworks that provide more sophisticated languages for specifying reconfiguration, and (iii) provide more advanced APIs than those low-level provided by ROS so to enable derived best practices concerning safety.

by reviewing how four major robotic frameworks support reconfiguration and how reconfiguration is realized in 48 real robotic (sub-)systems. Based on our analysis of robotic (sub-)systems, we derived best practices for implementing parameter reconfiguration. Table 6 provides an answer to the research questions and Table 7 draws take away messages for both academics and practitioners. We identified a significant mismatch between the state-of-the-art

Table 7 Summarized answers to our RQs

Answers to the research questions
RQ1: What are the motivations for developing dynamically reconfigurable robotic software systems?
Researchers consider reconfiguration of robotic systems mostly to deal with changes in dynamic environments and due to the execution of various tasks by a single robot. Further, the reconfiguration as measure for reacting to hardware or software faults is also a popular motivation. Less common motivations relate to limited resources and to react to safety issues. Finally, reconfiguration is frequently used as synonym for other concepts, such as, deployment-time variability.
RQ2: What aspects of a robotic system can be reconfigured?
While literature mainly focuses on the structural reconfiguration of robotic systems, robotic frameworks support this granularity only to a limited extent and require developers to implement the entire reconfiguration logic. Academic robotic frameworks can provide wrappers around more low-level frameworks that allow specifying the reconfiguration logic using models. So far, only parameter reconfiguration has been widely used in robotic (sub-)systems.
RQ3: What mechanisms are used for developing dynamically reconfigurable robotic software systems?
The literature mainly focuses on approaches for specifying structural reconfiguration using DSLs and executing them at runtime. Instead, robotic frameworks provide low-level APIs for loading or unloading assets and changing parameter values. The logic for implementing the structural reconfiguration considered in the literature has to be handwritten by the developers of robotic systems. However, in robotic (sub-)systems, we did not observe such reconfiguration, but only the frequent use of parameter reconfiguration.

and state-of-practice in reconfiguration of robotic systems. As future work, we plan to further investigate the discrepancies we identified to unleash their reasons and to identify research directions and strategies to fill this gap. This mainly concerns structural and more sophisticated reconfiguration approaches that we believe will be needed in robotics in the near future. Since the observed discrepancies might already stem from the motivation for reconfiguration, we will follow up on the motivations for reconfigurations for which until now we only captured the academic perspective by interviewing developers of robotic systems.

Acknowledgements We thank Sergio García for discussions on earlier versions of this paper.

Funding Open Access funding enabled and organized by Projekt DEAL. This work has been partially funded by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP: D13C21000430001. This work has been also partially funded by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem, grant PE0000020 - CHANGES - CUP: D53C22002560006 This work has been also partially funded by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem, grant PE00000014 - SERICS - CUP: D33C22001300002. Also, the authors acknowledge the support of the MUR (Italy) (i) Department of Excellence 2023 - 2027, (ii) PRIN PNRR 2022 project "RoboChor: Robot Choreography" (grant P2022RSW5W), and (iii) PRIN project "HALO: etHical-aware AdjustabLe autOnomous systems" (grant 2022JKA4SL).

Data Availability Our replication package provides all raw data, scripts that have been used, and results (Peldszus et al. 2024a).

References

Aarsten A, Brugali D, Menga G (1996) Designing Concurrent and Distributed Control Systems. Commun ACM 39(10):50–58. https://doi.org/10.1145/236156.236168

Abukhalil T, Sobh T, Patil M (2015) Survey on decentralized modular robots and control platforms. Lect Notes Electr Eng 313:165–175

Aghajani E, Nagy C, Vega-Márquez OL, Linares-Vásquez M, Moreno L, Bavota G, Lanza M (2019) Software Documentation Issues Unveiled. In: Atlee JM, Bultan T, Whittle J (Eds) International conference on software engineering (ICSE), IEEE / ACM, pp 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

Aguado E, Milosevic Z, Hernández C, Sanz R, Oviedo MAG, Bozhinoski D, Rossi C (2021) Functional Self-Awareness and Metacontrol for Underwater Robot Autonomy. Sensors 21(4):121. https://doi.org/10.3390/s21041210

Ahmad A, Babar MA (2016) Software architectures for robotic systems: A systematic mapping study. J Syst Softw (JSS) 122:16–39

Ahmadzadeh H, Masehian E (2015) Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization. Artif Intell 223:27–64. https://doi.org/10.1016/j.artint.2015.02.004

Alami R, Chatila R, Fleury S, Ghallab M, Ingrand F (1998) An architecture for autonomy. Int J Robot Res 17(4):315–337. https://doi.org/10.1177/027836499801700402

Alattas R (2018) Analyzing modular robotic systems. Lect Notes Netw Syst 22:1014–1028

Albonico M, Đorđević M, Hamer E, Malavolta I (2023) Software engineering research on the robot operating system: A systematic mapping study. J Syst Softw 197:11157. https://doi.org/10.1016/j.jss.2022.111574

Apel S, Batory D, Kästner C, Saake G (2013) Feature-Oriented Software Product Lines. Springer, Berlin Heidelberg

Benmoussa S, Loureiro R, Touati Y, Merzouki R (2013) Monitoring of Robot Path Tracking: Reconfiguration Strategy Design and Experimental Validation. In: Proceedings of the international conference on intelligent robots and systems, pp 5821–5826. https://doi.org/10.1109/IROS.2013.6697199

Berge-Cherfaoui V, Vachon B (1994) Dynamic Configuration of Mobile Robot Perceptual System. In: Proceedings of 1994 IEEE international conference on multisensor fusion and integration for intelligent systems (MFI), pp 707–714. https://doi.org/10.1109/MFI.1994.398385

Berger T, She S, Lotufo R, Wasowski A, Czarnecki K (2013) A study of variability models and languages in the systems software domain. IEEE Trans Softw Eng 39(12):1611–1640

Berger T, Pfeiffer RH, Tartler R, Dienst S, Czarnecki K, Wasowski A, She S (2014) Variability mechanisms in software ecosystems. Inf Softw Technol 56(11):1520–1535

Berger T, Steghöfer JP, Ziadi T, Robin J, Martinez J (2020) The state of adoption and the challenges of systematic variability management in industry. Empir Softw Eng 25:1755–1797

Bi ZM, Gruver WA, Zhang W (2003) Adaptability of Reconfigurable Robotic Systems. In: Proceedings of the international conference on robotics and automation (ICRA), pp 2317–2322. https://doi.org/10.1109/ROBOT.2003.1241939

Boluda JA, Pardo F, Blasco F, Pelechano J (1999) A Pipelined Reconfigurable Architecture for Visual-Based Navigation. In: Proceedings of the 25th conference on informatics: theory and practice for the New Millenium (EUROMICRO), pp 1071–1074. https://doi.org/10.1109/EURMIC.1999.794449

Bozhinoski D, Aguado E, Oviedo MG, Corbato CH, Sanz R, Wasowski A (2021) A Modeling Tool for Reconfigurable Skills in ROS. In: Proceedings of the 3rd international workshop on robotics software engineering (RoSE@ICSE), pp 25–28. https://doi.org/10.1109/RoSE52553.2021.00011

Bozhinoski D, Oviedo MG, Garcia NH, Deshpande H, van der Hoorn G, Tjerngren J, Wasowski A, Corbato CH (2022) MROS: Runtime Adaptation for Robot Control Architectures. Adv Robot 36(11):502–518. https://doi.org/10.1080/01691864.2022.2039761

Braman JMB, Murray RM, Wagner DA (2007) Safety Verification of a Fault Tolerant Reconfigurable Autonomous Goal-based Robotic Control System. In: Proceedings of the international conference on intelligent robots and systems, pp 853–858. https://doi.org/10.1109/IROS.2007.4399230

Brandstötter M, Hofbaur MW, Steinbauer G, Wotawa F (2007) Model-based Fault Diagnosis and Reconfiguration of Robot Drives. In: Proceedings of the international conference on intelligent robots and systems, pp 1203–1209. https://doi.org/10.1109/IROS.2007.4399092

Brugali D (2020) Runtime Reconfiguration of Robot Control Systems: A ROS-based Case Study. In: Proceedings of the international conference on robotic computing (IRC), pp 256–262. https://doi.org/10.1109/IRC.2020.00047

Brugali D, Gherardi L (2016) Hyperflex: A Model Driven Toolchain for Designing and Configuring Software Control Systems for Autonomous Robots, Springer, pp 509–534

Brugali D, Scandurra P (2009) Component-based robotic engineering (Part I) [Tutorial]. Robot Autom Mag, IEEE 16(4):84–96. https://doi.org/10.1109/mra.2009.934837

Brugali D, Shakhimardanov A (2010) Component-Based Robotic Engineering (Part II). Robot Autom Mag 17(1):100–112. https://doi.org/10.1109/mra.2010.935798

Brugali D, Broten G, Cisternino A, Colombo D, Fritsch J, Gerkey B, Kraetzschmar G, Vaughan R, Utz H (2007) Trends in robotic software frameworks. Springer Tracts Adv Robot 30:259–266

Brugali D, Capilla R, Mirandola R, Trubiani C (2018) Model-based Development of Qos-aware Reconfigurable Autonomous Robotic Systems. In: Proceedings of the international conference on robotic computing (IRC), pp 129–136

Bruyninckx H (2001) Open robot control software: the OROCOS project. In: Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, IEEE, vol 3, pp 2523–2528

Budenske J, Gini ML (1997) Sensor Explication: Knowledge-based Robotic Plan Execution through Logical Objects. IEEE Trans Syst Man Cybern Syst 27(4):611–625. https://doi.org/10.1109/3477.604104

de Cabrol A, Garcia-Fernandez T, Bonnin P, Chetto M (2008) A Concept of Dynamically Reconfigurable Real-Time Vision System for Autonomous Mobile Robotics. Int J Autom Comput 5(2):174–184. https://doi.org/10.1007/s11633-008-0174-0

Calinescu R, Mirandola R, Perez-Palacin D, Weyns D (2020) Understanding uncertainty in self-adaptive systems. In: 2020 IEEE international conference on autonomic computing and self-organizing systems (ACSOS), pp 242–251. https://doi.org/10.1109/ACSOS49614.2020.00047

Calisi D, Iocchi L, Nardi D, Scalzo CM, Ziparo VA (2008) Context-based Design of Robotic Systems. Robot Auton Syst 56(11):992–1003. https://doi.org/10.1016/j.robot.2008.08.008

Cámara J, Schmerl B, Garlan D (2020) Software architecture and task plan co-adaptation for mobile service robots. In: Proceedings of the 15th international symposium on software engineering for adaptive and self-managing systems, pp 125–136

Cardoso RC, Dennis LA, Fisher M (2019) Plan Library Reconfigurability in BDI Agents. In: Proceedings of the 7th international workshop on engineering multi-agent systems (EMAS), pp 195–212. https://doi.org/10.1007/978-3-030-51417-4_10

Chen L, Babar MA (2011) A Systematic Review of Evaluation of Variability Management Approaches in Software Product Lines. Inf Softw Technol 53(4):344–362

Chennareddy S, Agrawal A, Karuppiah A (2017) Modular self-reconfigurable robotic systems: A survey on hardware architectures. J Robot 2017

Cobleigh JM, Osterweil LJ, Wise AE, Lerner BS (2002) Containment Units: A Hierarchically Composable Architecture for Adaptive Systems. In: Proceedings of the 10th ACM SIGSOFT symposium on foundations of software engineering (FSE), pp 159–165. https://doi.org/10.1145/587051.587076

Colledanchise M, Ögren P (2018) Behavior trees in robotics and AI: An introduction. CRC Press

Cousins S, Gerkey B, Conley K, Garage W (2010) Sharing software with Ros [ros topics]. Robot Autom Mag 17(2):12–14. https://doi.org/10.1109/MRA.2010.936956

de la Cruz P, Piater JH, Saveriano M (2020) Reconfigurable Behavior Trees: Towards an Executive Framework Meeting High-level Decision Making and Control Layer Features. In: Proceedings of the international conference on systems, man, and cybernetics (SMC), pp 1915–1922. https://doi.org/10.1109/SMC42975.2020.9282817

Dalal S, Horgan J, Kettenring J (1993) Reliable software and communication: software quality, reliability, and safety. In: International conference on software engineering (ICSE), pp 425–435. https://doi.org/10.1109/ICSE.1993.346023

Doose D, Grand C, Lesire C (2017) MAUVE Runtime: A Component-Based Middleware to Reconfigure Software Architectures in Real-Time. In: Proceedings of the international conference on robotic computing (IRC), pp 208–211. https://doi.org/10.1109/IRC.2017.47

Dorf R, Bishop R (2011) Modern control systems. Person Education

Dudek G, Jenkin M, Milios E, Wilkes D (1993) Taxonomy for Swarm Robots. In: Proceedings of the international conference on intelligent robots and systems, pp 441 – 447

Eddin MC (2013) Towards a Taxonomy of Dynamic Reconfiguration Approaches. J Softw 8(9):2202–2207

Edwards G, Garcia J, Tajalli H, Popescu D, Medvidovic N, Sukhatme GS, Petrus B (2009) Architecture-Driven Self-Adaptation and Self-Management in Robotics Systems. In: Proceedings of the workshop on software engineering for adaptive and self-managing systems (SEAMS), pp 142–151. https://doi.org/10.1109/SEAMS.2009.5069083

Elkady AY (2012) Robotics middleware: A comprehensive literature survey and attribute-based bibliography. J Robot 959013:1-959013:15

Estefo P, Simmonds J, Robbes R, Fabry J (2019) The robot operating system: Package reuse and community dynamics. J Syst Softw 151:226–242

Fayman JA, Rivlin E, Mossé D (1998) FT-AVS: A Fault-tolerant Architecture for Real-Time Active Vision. Real Time Imaging 4(2):143–157. https://doi.org/10.1006/rtim.1997.0077

Fernández RAS, Grande D, Martins A, Bascetta L, Domínguez S, Rossi C (2019) Modeling and control of underwater mine explorer robot UX-1. IEEE Access 7:39432–3944. https://doi.org/10.1109/ACCESS.2019.2907193

Ferrell C (1994) Failure Recognition and Fault Tolerance of an Autonomous Robot. Adapt Behav 2(4):375–398. https://doi.org/10.1177/105971239400200403

Firby RJ, Slack MG, Drive C (1995) Task Execution: Interfacing to Reactive Skill Networks. In: Proceedings of the AAAI spring symposium

Fitzpatrick PM, Metta G, Natale L (2008) Towards long-lived robot genes. Robot Auton Syst 56(1):29–45. https://doi.org/10.1016/j.robot.2007.09.014

Fleury S, Herrb M, Chatila R (1997) G/sup en/om: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In: Proceedings of the international conference on intelligent robots and systems (IROS), vol 2, pp 842–849. https://doi.org/10.1109/IROS.1997.655108

Fornari G, de Santiago Júnior VA (2019) Dynamically Reconfigurable Systems: A Systematic Literature Review. J Intell Robot Syst 95(3–4):829–849. https://doi.org/10.1007/s10846-018-0921-6

Franz P, Berger T, Fayaz I, Nadi S, Groshev E (2021) Configfix: Interactive configuration conflict resolution for the linux kernel. In: Proceedings of the international conference on software engineering, software engineering in practice track (ICSE/SEIP)

Fritsch S, Senart A, Schmidt DC, Clarke S (2008) Time-bounded adaptation for automotive system software. In: International conference on software engineering (ICSE), pp 571–580. https://doi.org/10.1145/1368088.1368166

Frost J, Stechele W, Maehle E (2015) Self-Reconfigurable Control Architecture for Complex Mobile Robots. It - Inf Technol 57(2):122–129. https://doi.org/10.1515/itit-2014-1063

Gafni V (1999) Robots: A Real-Time Systems Architectural Style. In: Proceedings of the 7th european software engineering conference (ESEC/FSE), pp 57–74. https://doi.org/10.1007/3-540-48166-4_5

García S, Strüber D, Brugali D, Berger T, Pelliccione P (2020a) Accompanying Technical Report for An Empirical Assessment of Robotics Software Engineering

García S, Strüber D, Brugali D, Berger T, Pelliccione P (2020b) Robotics Software Engineering: A Perspective from the Service Robotics Domain. In: European software engineering conference and symposium on the foundations of software engineering

García S, Strüber D, Brugali D, Di Fava A, Pelliccione P, Berger T (2022) Software variability in service robotics. Empir Softw Eng (EMSE)

Gemino A, Wand Y (2004) A Framework for Empirical Evaluation of Conceptual Modeling Techniques. Requir Eng 9(4):248–260. https://doi.org/10.1007/S00766-004-0204-6

Gerkey B (2014) Why ROS 2? https://design.ros2.org/articles/why_ros2.html

Gherardi L, Hochgeschwender N (2015) RRA: Models and Tools for Robotics Run-Time Adaptation. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 1777–1784. https://doi.org/10.1109/IROS.2015.7353608

Goldhoorn M, Joyeux S (2014) Extension of a Plan-based Component Manager for Real Time Adaptation. In: Proceedings of the 41st international symposium on robotics (ISR), pp 1–6

Guo J, Shi K (2018) To preserve or not to preserve invalid solutions in search-based software engineering: A case study in software product lines. In: Proceedings of the 40th international conference on software engineering, pp 1027-1038. https://doi.org/10.1145/3180155.3180163

Hartmann J, Stechele W, Maehle E (2013) Self-reconfigurable Control Architecture for Complex Robots. In: Proceedings of the 43rd Jahrestagung der Gesellschaft für Informatik, Informatik angepasst an Mensch, Organisation und Umwelt (INFORMATIK), pp 2742–2748

Hayes-Roth B, Pfleger K, Lalanda P, Morignot P, Balabanovic M (1995) A Domain-Specific Software Architecture for Adaptive Intelligent Systems. IEEE Trans Softw Eng 21(4):288–301. https://doi.org/10.1109/32.385968

Henard C, Papadakis M, Harman M, Le Traon Y (2015) Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. Int Conf Softw Eng (ICSE) 1:517–528. https://doi.org/10.1109/ICSE.2015.69

Iftikhar MU, Weyns D (2014) ActivFORMS: Active Formal Models for Self-Adaptation. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 125–134. https://doi.org/10.1145/2593929.2593944

Inglés-Romero J, Lotz A, Chicote CV, Schlegel C (2012) Dealing with Run-time Variability in Service Robotics: Towards a Dsl for Non-functional Properties. In: Proceedings of the 3rd international workshop on domain-specific languages for robotic systems (DSLRob)

Inohira E, Konno A, Uchiyama M (2003) Layered Multi-Agent Architecture with Dynamic Reconfigurability. In: Proceedings of the international conference on robotics and automation (ICRA), pp 4060–4065. https://doi.org/10.1109/ROBOT.2003.1242221

Jamshidi P, Cámara J, Schmerl BR, Kästner C, Garlan D (2019) Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots. In: Proceedings of the 14th international symposium on software engineering for adaptive and self-managing systems (SEAMS)

Kang K (1990) Feature-Oriented Domain Analysis (FODA): Feasibility Study. Tech. rep, DTIC Document

Karuppiah DR, Zhu Z, Shenoy PJ, Riseman EM (2001) A Fault-Tolerant Distributed Vision System Architecture for Object Tracking in a Smart Room. In: Proceedings of the international conference on computer vision systems (ICVS), pp 201–219. https://doi.org/10.1007/3-540-48222-9_14

Kim D, Park S (2006) Designing Dynamic Software Architecture for Home Service Robot Software. In: Proceedings of the international conference on embedded and ubiquitous computing (EUC), pp 437–448. https://doi.org/10.1007/11802167_45

Kim D, Park S, Jin Y, Chang H, Park Y, Ko I, Lee K, Lee J, Park Y, Lee S (2006) SHAGE: A Framework for Self-Managed Robot Software. In: Proceedings of the international workshop on self-adaptation and self-managing systems (SEAMS), pp 79–85. https://doi.org/10.1145/1137677.1137693

Kim J, Im C, Shin H, Yi KY, Lee HG (2003) A New Task-based Control Architecture for Personal Robots. In: Proceedings of the international conference on intelligent robots and systems, pp 1481–1486. https://doi.org/10.1109/IROS.2003.1248853

Kim JH, Kim JO (2004) A Task Management Design for Task-based Control Architecture for Personal Robots. In: Proceedings of the 30th annual conference of the IEEE industrial electronics society (IECON), pp 152–156. https://doi.org/10.1109/IECON.2004.1433301

Kortenkamp D, Simmons R, Brugali D (2016) Robotic Systems Architectures and Programming. Springer Handbook of Robotics pp 283–06. https://doi.org/10.1007/978-3-319-32552-1_12

Kozov A, Volosatova TM, Tachkov A (2021) Method of the Dynamic Reconfiguration of a Group Control System for Mobile Robots. In: Proceedings of the international russian automation conference (RusAutoCon), pp 991–996

Kramer JF, Scheutz M (2006) ADE: A Framework for Robust Complex Robotic Architectures. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 4576–4581. https://doi.org/10.1109/IROS.2006.282162

Krupitzer C, Roth FM, VanSyckel S, Schiele G, Becker C (2015) A Survey on Engineering Approaches for Self-adaptive Systems. Pervasive Mob Comput 17:184–206. https://doi.org/10.1016/j.pmcj.2014.09.009

Kubota N, Nojima Y, Kojima F, Fukuda T (2001) Dual Learning for Perception and Behavior of Mobile Robots. In: Proceedings of the joint 9th IFSA world congress and 20th NAFIPS international conference, vol 3, pp 1401–1406. https://doi.org/10.1109/NAFIPS.2001.943754

Lee H, Choi H, Ko I (2005) A Semantically-Based Software Component Selection Mechanism for Intelligent Service Robots. In: Proceedings of the 4th mexican international conference on advances in artificial intelligence (MICAI), pp 1042–1051. https://doi.org/10.1007/11579427_106

Lee J, Kang KC (2006) A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In: Proceedings of the 10th international conference on software product lines (SPLC), pp 131–140. https://doi.org/10.1109/SPLINE.2006.1691585

Lee J, Huber MJ, Durfee EH, Kenny PG (1994) UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications. In: Proceedings of the conference on intelligent robotics in field, factory, service and space (CIRFFSS)

Lee J, Kim J, Lee B, Wu C (2008a) Utilizing Semantic Web 2.0 for Self-Reconfiguration of SOA based Agent Applications in Intelligent Service Robots. In: Proceedings of the 8th international conference on computer and information technology (CIT), pp 784–789. https://doi.org/10.1109/CIT.2008.4594774

Lee J, Kim J, Lee C, Lee B (2008b) Agent Based Dynamic Adaptation of Intelligent Robots Using Enterprise Service Bus. In: Proceedings of the international conference on information science and security (ICISS), pp 94–97. https://doi.org/10.1109/ICISS.2008.31

de Leng D, Heintz F (2016) DyKnow: A Dynamically Reconfigurable Stream Reasoning Framework as an Extension to the Robot Operating System. In: Proceedings of the international conference on simulation, modeling, and programming for autonomous robots (SIMPAR), pp 55–60. https://doi.org/10.1109/SIMPAR.2016.7862375

Lindström M, Orebäck A, Christensen HI (2000) BERRA: A Research Architecture for Service Robots. In: Proceedings of the international conference on robotics and automation (ICRA), pp 3278–3283. https://doi.org/10.1109/ROBOT.2000.845168

Liu J, Zhang X, Hao G (2016) Survey on research and development of reconfigurable modular robots. Adv Mech Eng 8(8):1–21

Lotz A, Inglés-Romero JF, Vicente-Chicote C, Schlegel C (2013) Managing run-time variability in robotics software by modeling functional and non-functional behavior. Enterprise. Springer, Business-Process and Information Systems Modeling, pp 441–455

Lotz A, Schlegel C, Lutz M, Stampfer D (2013b) The Smartsoft Project. http://smart-robotics.sourceforge.net/

Macdonald B, Hsieh B, Warren I (2004) Design for Dynamic Reconfiguration of Robot Software. In: Proceedings of the international conference on autonomous robots and agents

Macenski S, Foote T, Gerkey B, Lalancette C, Woodall W (2022) Robot operating system 2: Design, architecture, and uses in the wild. Sci Robot 7(66):eabm607. https://doi.org/10.1126/scirobotics.abm6074

Maeda T (2006) Reconfigurable system architecture for net-accessible pet-type robot system. In: Proceedings of the 15th international symposium on robot and human interactive communication (RO-MAN), pp 668–673. https://doi.org/10.1109/ROMAN.2006.314477

Malavolta I, Lewis GA, Schmerl BR, Lago P, Garlan D (2021) Mining Guidelines for Architecting Robotics Software. J Syst Softw (JSS) 178:11096. https://doi.org/10.1016/j.jss.2021.110969

Marques F, Santana PF, Guedes M, Pinto E, Lourenço A, Barata J (2013) Online Self-Reconfigurable Robot Navigation in Heterogeneous Environments. In: Proceedings of the 22nd international symposium on industrial electronics (ISIE), pp 1–6. https://doi.org/10.1109/ISIE.2013.6563831

Mens K, Capilla R, Cardozo N, Dumas B (2016) A taxonomy of context-aware software variability approaches. In: Proceedings of the international conference on modularity, pp 119-124. https://doi.org/10.1145/2892664.2892684

Mens T, Buckley J, Zenger M, Rashid A (2003) Towards a Taxonomy of Software Evolution. In: Proceedings of the international workshop on unanticipated software evolution

Mészáros T, Dobrowiecki TP (2017) Agent-based Reconfigurable Natural Language Interface to Robots - Human-Agent Interaction using Task-specific Controlled Natural Languages. In: Proceedings of the 9th

international conference on agents and artificial intelligence (ICAART), pp 632–639. https://doi.org/10.5220/0006205306320639
Metta G, Fitzpatrick P, Natale L (2006) Yarp: Yet another robot platform. Int J Adv Robot Syst 3(1):43–48
Morris A (2007) The Process Information Space: The Importance of Data Flow in Robotic Systems. In: Proceedings of the international conference on robotics and automation (ICRA), pp 293–298. https://doi.org/10.1109/ROBOT.2007.363802
Moubarak P, Ben-Tzvi P (2012) Modular and reconfigurable mobile robotics. Roboti Auton Syst 60(12):1648–1663
Murwantara IM (2020) An Initial Framework of Dynamic Software Product Line Engineering for Adaptive Service Robot. In: Proceedings of the international conference on computer science and its application in agriculture (ICOSICA), pp 1–6. https://doi.org/10.1109/ICOSICA49951.2020.9243199
Nilsson K, Bengel M (2008) Plug-and-Produce Technologies Real-time Aspects - Service Oriented Architectures for SME Robots and Plug-and-Produce. In: Proceedings of the international conference on informatics in control, automation and robotics, robotics and automation, pp 249–254
Nordmann A, Lange R, Rico FM (2021) System Modes - Digestible System (Re-)Configuration for Robotics. In: Proceedings of the 3rd international workshop on robotics software engineering (RoSE@ICSE), pp 19–24. https://doi.org/10.1109/RoSE52553.2021.00010
OMG (2012) Common Object Request Broker Architecture (CORBA). https://www.omg.org/spec/CORBA/About-CORBA/
OMG (2015) Data distribution service (dds). https://www.omg.org/spec/DDS/About-DDS/
Open Robotics (2007) Robot Operating System. http://www.ros.org
OWASP (2021) OWASP Cheat Sheet Series – Input Validation Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
Paikan A, Pattacini U, Domenichelli D, Randazzo M, Metta G, Natale L (2015) A Best-Effort Approach for Run-Time Channel Prioritization in Real-Time Robotic Application. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 1799–1805. https://doi.org/10.1109/IROS.2015.7353611
Pane YP, Mokhtari V, Aertbeliën E, Schutter JD, Decré W (2021) Autonomous Runtime Composition of Sensor-Based Skills Using Concurrent Task Planning. IEEE Robot Autom Lett 6(4):6481–6488. https://doi.org/10.1109/LRA.2021.3094498
Parker LE, Tang F (2006) Building Multirobot Coalitions Through Automated Task Solution Synthesis. Proc IEEE 94(7):1289–1305. https://doi.org/10.1109/JPROC.2006.876933
Peldszus S, Strüber D, Jürjens J (2018) Model-based security analysis of feature-oriented software product lines. In: Wyk EV, Rompf T (eds) Proceedings of the 17th ACM SIGPLAN international conference on generative programming: concepts and experiences, GPCE 2018, Boston, MA, USA, November 5-6, 2018, ACM, pp 93–106. https://doi.org/10.1145/3278122.3278126
Peldszus S, Brugali D, Strüber D, Pellicone P, Berger T (2024a) Software Reconfiguration in Robotics – Replication Package. https://doi.org/10.5281/zenodo.14013818
Peldszus S, Bürger J, Jürjens J (2024) Umlsecrt: Reactive security monitoring of java applications with round-trip engineering. IEEE Trans Softw Eng 50(1):16–47. https://doi.org/10.1109/TSE.2023.3326366
Peldszus S, Knopp H, Sens Y, Berger T (2024c) Towards ML-Integration and Training Patterns for AI-Enabled Systems. In: AISOLA'2023 Post-Proceedings
Pereira JA, Acher M, Martin H, Jézéquel JM, Botterweck G, Ventresque A (2021) Learning software configuration spaces: A systematic literature review. J Syst Softw 18. https://doi.org/10.1016/j.jss.2021.111044
Pham TQ, Dixon KR, Khosla PK (2000) Software Systems Facilitating Self-Adaptive Control Software. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 1094–1100. https://doi.org/10.1109/IROS.2000.893165
Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: An Open-Source Robot Operating System. In: Proceedings of the ICRA workshop on open source software
Ralph P, et al. (2023) SIGSOFT Empirical Standards. https://acmsigsoft.github.io/EmpiricalStandards/docs/
Ramachandran RK, Preiss JA, Sukhatme GS (2019) Resilience by Reconfiguration: Exploiting Heterogeneity in Robot Teams. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 6518–6525. https://doi.org/10.1109/IROS40897.2019.8968611
Ramirez AJ, Jensen AC, Cheng BHC (2012) A taxonomy of uncertainty for dynamically adaptive systems. In: 2012 7th International symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 99–108. https://doi.org/10.1109/SEAMS.2012.6224396
RobMoSys (2023) RobMoSys Project Page. https://robmosys.eu/

Roh S, Park KH, Yang K, Park JH, Kim H, Lee H, Choi H (2004) Development of Dynamically Reconfigurable Personal Robot. In: Proceedings of the international conference on robotics and automation (ICRA), pp 4023–4028. https://doi.org/10.1109/ROBOT.2004.1308900

Santilli T, Pelliccione P, Wohlrab R, Shahrokni A (2024) Continuous compliance in the automotive industry. IEEE Softw 01:1–10. https://doi.org/10.1109/MS.2023.3342974

Santos F, Almeida L, Pedreiras P, Lopes LS (2009) A Real-Time Distributed Software Infrastructure for Cooperating Mobile Autonomous Robots. In: Proceedings of the 14th international conference on advanced robotics (ICAR), pp 1–6

Scala E, Micalizio R, Torasso P (2014) ReCon: An Online Task ReConfiguration Approach for Robust Plan Execution. In: Proceedings of the 6th international conference on agents and artificial intelligence (ICAART), pp 262–279. https://doi.org/10.1007/978-3-319-25210-0_16

Scheutz M, Kramer JF (2007) Reflection and Reasoning Mechanisms for Failure Detection and Recovery in a Distributed Robotic Architecture for Complex Robots. In: Proceedings of the international conference on robotics and automation (ICRA), pp 3699–3704. https://doi.org/10.1109/ROBOT.2007.364045

Schlegel C, Worz R (1999) The software framework smartsoft for implementing sensorimotor systems. In: Proceedings of the international conference on intelligent robots and systems (IROS), IEEE, vol 3, pp 1610–1616

Schlegel C, Lutz M, Lotz A, Stampfer D, Inglés-Romero JF, Vicente-Chicote C (2013) Model-driven software systems engineering in robotics: Covering the complete life-cycle of a robot. Jahrestagung der Gesellschaft für Informatik. Informatik angepasst an Mensch, Organisation und Umwelt (INFORMATIK), pp 2780–2794

Schlegel C, Lotz A, Lutz M, Stampfer D (2021) Composition, Separation of Roles and Model-Driven Approaches as Enabler of a Robotics Software Ecosystem, Springer International Publishing, Cham, pp 53–108. https://doi.org/10.1007/978-3-030-66494-7_3

Schmidt DC, Kuhns F (2000) An overview of the real-time CORBA specification. Computer 33(6):56–6. https://doi.org/10.1109/2.846319

Schneider S, Chen V, Pardo-Castellote G (1995) The ControlShell Component-based Real-Time Programming System. In: Proceedings of the IEEE international conference on robotics and automation, pp 2381–2388. https://doi.org/10.1109/ROBOT.1995.525616

Shaukat A, Burroughes G, Gao Y (2016) Springer, chap Self-Reconfiguring Robotic Framework Using Fuzzy and Ontological Decision Making. Intell Syst Appl. https://doi.org/10.1007/978-3-319-33386-1_7

Shlyakhov N, Vatamaniuk I, Ronzhin A (2017) Survey of Methods and Algorithms of Robot Swarm Aggregation. J Phys Conf Ser 803(1)

Soria CC, Pérez J, Carsí JA (2009) Handling the dynamic reconfiguration of software architectures using aspects. In: European conference on software maintenance and reengineering (CSMR), pp 263–266. https://doi.org/10.1109/CSMR.2009.33

SPARC (2016) Robotics 2020 Multi-Annual Roadmap. https://eu-robotics.net/sparc/upload/about/files/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf

Stewart D, Schmitz D, Khosla P (1990) Implementing real-time robotic systems using chimera ii. In: International Conference on Robotics and Automation, vol 1 pp 598–603. https://doi.org/10.1109/ROBOT.1990.126047

Stewart DB, Khosla PK (1996) The Chimera Methodology: Designing Dynamically Reconfigurable and Reusable Real-Time Software Using Port-Based Objects. Int J Softw Eng Knowl Eng 6(2):249–277. https://doi.org/10.1142/S0218194096000120

Stewart DB, Volpe R, Khosla PK (1997) Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. Trans Softw Eng (TSE) 23(12):759–776. https://doi.org/10.1109/32.637390

Stueben M, Hoffmann A, Reif W (2021) Constraint-based Whole-Body-Control of Mobile Manipulators in Human-Centered Environments. In: Proceedings of the international conference on emerging technologies and factory automation (ETFA), pp 1–8

Svahnberg M, van Gurp J, Bosch J (2005) A taxonomy of variability realization techniques. Softw, Pract Exper 35(8):705–754

Szlenk M, Zielinski C, Figat M, Kornuta T (2015) Reconfigurable Agent Architecture for Robots Utilising Cloud Computing. In: Progress in automation, robotics and measuring techniques - Volume 2, Springer, pp 253–264. https://doi.org/10.1007/978-3-319-15847-1_25

Tajalli H, Garcia J, Edwards G, Medvidovic N (2010) PLASMA: A Plan-based Layered Architecture for Software Model-driven Adaptation. In: Proceedings of the international conference on automated software engineering (ASE), pp 467–476. https://doi.org/10.1145/1858996.1859092

Tan N, Hayat AA, Elara MR, Wood KL (2020) A Framework for Taxonomy and Evaluation of Self-Reconfigurable Robotic Systems. IEEE Access 8:13969–1398. https://doi.org/10.1109/ACCESS.2020.2965327

The Autoware Foundation (2023) Github reposiroty of autoware.auto. https://github.com/autowarefoundation/autoware

Volpe R, Nesnas I, Estlin T, Mutz D, Petras R, Das H (2001) The claraty architecture for robotic autonomy. In: Proceedings of the aerospace conference, vol 1, pp 1/121–1/132 . https://doi.org/10.1109/AERO.2001.931701

Vos D, Motazed B (1998) Fault Tolerant Control Design for Parameter Dependent Systems. In: Proceedings of the american control conference (ACC), vol 2, pp 679–680. https://doi.org/10.1109/ACC.1998.703491

Wasowski A, Berger T (2023) Domain-Specific Languages: Effective Modeling, Automation, and Reuse. Springer

Wills L, Kannan S, Sander S, Guler M, Heck B, Prasad J, Schrage D, Vachtsevanos G (2001) An Open Platform for Reconfigurable Control. IEEE Control Syst Mag 21(3):49–64. https://doi.org/10.1109/37.924797

Xiao L (2012) From Adaptive Software Bahaviour to Adaptive Robotic Bahaviour. In: Proceedings of the international conference on systems and informatics (ICSAI), pp 2448–2452. https://doi.org/10.1109/ICSAI.2012.6223549

Yang A, Yang B, Hui B, Zheng B, Yu B, Zhou C, Li C, Li C, Liu D, Huang F, Dong G, Wei H, Lin H, Tang J, Wang J, Yang J, Tu J, Zhang J, Ma J, Yang J, Xu J, Zhou J, Bai J, He J, Lin J, Dang K, Lu K, Chen K, Yang K, Li M, Xue M, Ni N, Zhang P, Wang P, Peng R, Men R, Gao R, Lin R, Wang S, Bai S, Tan S, Zhu T, Li T, Liu T, Ge W, Deng X, Zhou X, Ren X, Zhang X, Wei X, Ren X, Liu X, Fan Y, Yao Y, Zhang Y, Wan Y, Chu Y, Liu Y, Cui Z, Zhang Z, Guo Z, Fan Z (2024) Qwen2 Technical Report. https://doi.org/10.48550/ARXIV.2407.10671 arXiv:2407.10671

Yim M, Shen W, Salemi B, Rus D, Moll M, Lipson H, Klavins E, Chirikjian GS (2007) Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robot Autom Mag 14(1):43–52. https://doi.org/10.1109/MRA.2007.339623

Yin Z, Ma X, Zheng J, Zhou Y, Bairavasundaram LN, Pasupathy S (2011) An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In: Proceedings of the symposium on operating systems principles (SOSP), pp 159–172. https://doi.org/10.1145/2043556.2043572

Yoo J, Kim S, Hong S (2006) The Robot Software Communications Architecture (RSCA): QoS-Aware Middleware for Networked Service Robots. In: Proceedings of the SICE-ICASE international joint conference, pp 330–335. https://doi.org/10.1109/SICE.2006.315702

Yu Z, Warren I, MacDonald BA (2006) Dynamic Reconfiguration for Robot Software. In: Proceedings of the international conference on automation science and engineering, pp 292–297. https://doi.org/10.1109/COASE.2006.326896

Zhang Y, Roufas K, Yim M (2001) Software Architecture for Modular Self-reconfigurable Robots. In: Proceedings of the international conference on intelligent robots and systems (IROS), pp 2355–2360. https://doi.org/10.1109/IROS.2001.976422

Authors and Affiliations

Sven Peldszus[1] · Davide Brugali[2] · Daniel Strüber[3,4,5] · Patrizio Pelliccione[6,7] · Thorsten Berger[1,3,4]

✉ Sven Peldszus
sven.peldszus@rub.de

Davide Brugali
davide.brugali@unibg.it

Daniel Strüber
danstru@chalmers.se

Patrizio Pelliccione
patrizio.pelliccione@gssi.it

Thorsten Berger
thorsten.berger@rub.de

1 Ruhr University Bochum, Bochum, Germany

2 University of Bergamo, Bergamo, Italy

3 Chalmers University of Technology, Gothenburg, Sweden

4 University of Gothenburg, Gothenburg, Sweden

5 Radboud University, Nijmegen, The Netherlands

6 Gran Sasso Science Institute (GSSI), L'Aquila, Italy

7 Bergen University, Bergen, Norway