An Architecture for Decentralized, Collaborative, and Autonomous Robots

Sergio García*, Claudio Menghi*, Patrizio Pelliccione*, Thorsten Berger*, Rebekka Wohlrab*†
*Chalmers | University of Gothenburg, Gothenburg (Sweden)
† Systemite AB, Gothenburg (Sweden)
Email: [sergio.garcia, claudio.menghi, patrizio.pelliccione, thorsten.berger]@gu.se, wohlrab@chalmers.se

***Abstract*—Robotic applications are typically realized using ad hoc and domain-specific solutions, which challenges the engineering and cross-project reuse of such applications. Especially in complex scenarios, where self-adaptive robots collaborate among themselves or with humans, the effective and systematic engineering of such applications is becoming increasingly important. Such scenarios require decentralized software architectures that foster fault-tolerant ways of managing large teams of (possibly) heterogeneous robots. To the best of our knowledge, no existing architecture for robot applications supports decentralized and self-adaptive collaboration. To address this gap, we conducted a design science study with 21 practitioners and experts in the field of robotics to develop an architecture fulfilling these requirements through several iterations. We present SERA, an architecture for robot applications that supports human-robot collaboration, as well as adaptation and coordination of single- and multi-robot systems in a decentralized fashion. SERA is based on layers that contain components that manage the adaptation at different levels of abstraction and communicate through well-defined interfaces. We successfully validated SERA by considering a set of real scenarios, by both using simulators and real robots, by involving robotic experts, and by benchmarking it with state-of-the-art solutions.**

1. Introduction

Robotic systems are increasingly integrated in our lives. Robots are used in a huge variety of contexts [1], such as service, industrial, military, or space robots, each of which target specific tasks. Such robots are used for replacing humans in repetitive, laborious or potentially dangerous activities. The World Robotic Survey [2] estimated 35 million indoor service robots to be sold by 2018, accumulating a sales value of $12 billion since 2015. The global sales of household and personal robots is expected to grow by 23.5% per year [3]. This increase is accompanied with huge progress in robot technology, especially in image processing, planning, control, and collaboration.

A robot typically performs specialized tasks; however, some tasks are highly complex and require a team of robots, whose capabilities (e.g., perception, manipulation, and actuation) are coordinated and supervised, potentially in direct interaction with humans. Such tasks are typically defined in terms of a high-level *mission*—a description of the goals that the robots shall achieve.

Such robot collaboration scenarios substantially increase the complexity of robot control software and demand appropriate software architectures. While many architectures, including layered [4], [5], component- or service-based architectures have been proposed [6], to the best of our knowledge, no architecture has explicitly focused on such multi-robot and robot-human decentralized collaboration. The closest approach to this result is the work of Kaupp et al. [7] who developed an architecture for performing an information-gathering task with human-robot teams. However, their work is limited to such tasks where robots and human do not physically interact.

In general, creating architectures for robotic applications is challenging. As we will show—through our literature study, experience, and expert opinions collected from robotic experts—the current architectural challenges faced by engineers can be classified into three main problems:

P1: Robotic framework. The lack of architectural models and methods in the production of software for robotic systems. Robotic systems development—instead of relying on established engineering processes—requires a craftsmanship for assembling and developing robotic software [8]. This also negatively impacts modularity and reusability of robotic applications and their components.

P2: Collaboration of heterogeneous teams. The absence of a common approach or strategy that might allow vendors to produce their own robots and deploy them within a team. These teams may also be formed by heterogeneous robots produced by different companies. The lack of a standard, decentralized architecture for robotic applications that is able to abstract the hardware of each robot prevents the usage of robotic applications developed by different vendors. Furthermore, it impedes the management, coordination, and collaboration of a robotic team (or a human-robot team) based on communication and information fusion.

P3: Self-adaptation. A lack of systematic support for adaptations of robot teams. Often, adaptations performed by a single robot are not sufficient to accomplish a specified mission. Tasks may need to be reassigned to other robots, or whole teams need to be reconfigured. Causes can be failures or changes in the mission or operational environment. Even though substantial research has been conducted on self-adaptive systems, including reference architectures for

self-adaptive robotic applications [4]–[6], [9], systematic engineering support for adaptation mechanisms for collaborating robots is needed [9].

To address these problems, we present SERA, a Self-adaptive dEcentralized Robotic Architecture. SERA is defined as a three-layered and distributed architecture defining core components (e.g., adaptation manager, mission manager), their relationships, and interfaces needed to control collaborating and potentially heterogeneous robots. As such, SERA also promotes engineering and integration of robotic applications developed by different vendors. Decentralization (i) allows managing large teams of robots, (ii) facilitates—through loose coupling—the addition, substitution, and removal of robots, (iii) minimizes the cost of changes since the responsibility of every action is within single robot that observes its environment and acts on events autonomously, (iv) facilitates data-centric workflows since data are passed directly to where they are required, at the next robot in the workflow, and (v) increases robustness and decreases system vulnerability (e.g., reducing single points of failure) [10].

Finally, SERA promotes the development of applications that embed self-* features—that is, "systems which are capable of self-configuration, self-adaptation and self-healing, self-monitoring and self-tuning" [5]. SERA has been proposed according with the Kramer et. al. [5] reference architecture to facilitate the development of self-* features. Existing self-adaptive architectures do neither directly support teams of robots nor decentralized adaptation techniques and algorithms. Our architecture will be able to self-adapt at two levels: at a local level (by individual robots) and at a global level (by a team of robots in a collaborative way).

To conceive SERA, we conducted a literature study of existing architectures (extending an existing mapping study [6]), held four focus groups with four industrial and 17 academic experts in robotics. We showed the feasibility of implementing SERA by realizing a prototype building upon the middleware ROS [11]. We evaluated SERA both in a simulation and a real-world scenario where a human guides a mobile robot to carry a payload. The evaluation is conducted with industrial partners participating in an EU H2020 project[1] on software systems engineering for *teams of robots* that collaborate to achieve complex missions. We discuss how SERA addresses the challenges identified in our literature survey. We provide an online appendix with the architecture, its documentation, and example implementations in [**?**]. Additional videos are available on the project website.

We proceed by describing our literature survey in Sec. 2 and our research method in Sec. 3. We then present SERA in detail in Sec. 4, together with its implementation in Sec. 5. We discuss its validation in Sec. 6 and conclude in Sec. 7.

2. Literature Review

To evaluate the state of the art of software architectures for robot applications and their suitability to address the three main problems, we systematically study related literature. We used and extended the mapping study of Ahmad et al. [6], who identified and analyzed 56 peer-reviewed papers on software architectures for robotic systems.

Architectural Challenges. Ahmad et al. [6] provide a brief discussion of eight challenges that architectures for robot applications need to address and guidelines based on the literature for tackling the challenges. Interestingly, their literature evaluation does not show whether existing architectures provide specific support for decentralized, collaborating robots, especially fault-tolerant interactions between multiple robots and between robots and humans or not. To this end, we extended their list of architectural challenges. First, *multi-robot* collaboration (Ch7, see below) must be addressed, which refers to an architecture's ability to manage teams of robots. A single robot can perform tasks, but some of them may be too complex for it to perform [10]—e.g., spatially separate tasks. Second, *decentralized* operations (Ch8, see below), which indicates architectural support for decentralized planning and collaboration, must be supported (as explained in Sec. 1). Third, *human-robot interaction* (Ch9) is an architectural challenge, which is pivotal since the presence of robots in environments shared with humans will be increased in the next years [2]. Consequently, SERA needs to address the following challenges, organized into our main problems P1–P3 from above (Sec. 1):

P1 - Robotic framework

Ch1 - Distributed Resources Access: How to enable access and utilization of distributed resources (hardware components—often virtualized) to develop robots? [6]

Ch2 - Re-engineering: How to support re-engineering or evolution of existing robots to satisfy new requirements? [6]

Ch3 - Modelling: How to exploit high-level models that raise abstraction from code to models in robotic systems? [6]

Ch4 - Design: How to abstract code components and their interconnections as architectural components to support the notion of component-based robotics? [6]

Ch5 - Programming: How to engineer a framework that enables reuse and modularity for robotic programming? [6]

P2 - Collaboration of heterogeneous teams

Ch6 - Information Fusion: How to support the collection, processing, and sharing of information that is fused into a team of robots to support their operations? [6]

Ch7 - Multi-robot: How to support the communication and coordination for managing a team of robots?

Ch8 - Decentralized: How to exploit decentralized architectures in order to manage the communication between robots without the need of a coordinating central control unit?

Ch9 - Human-robot collaboration: How to support the cooperation between robots and humans for realizing complex missions?

P3 - Self-adaptation

Ch10 - Reconfiguration: How to equip robots with self-reconfiguration facilities to allow them to operate optimally with available resources and evolving requirements? [6]

Ch11 - Fault Tolerance: How to enable robots to continue operations despite the presence of failures? [6]

Review Methodology. We evaluated the most significant works according to these challenges. From Ahmad et al.'s

1. http://www.co4robots.eu/

mapping study [6] we selected eight papers out of the 56 that are studied. We selected them based on our inclusion and exclusion criteria, which are listed in Table 1.

Since the time range of Ahmad et al.'s work ends in 2016, we extended the mapping study searching and identifying works that where accepted between 2016 and 2017. To perform the extension we followed Ahmad et al.'s approach using the following search string: *Software AND (Architecture OR Component OR Design OR Framework) AND (Robot OR Robotic OR Robotics OR Humanoid)*. We searched for suitable publications in the following digital libraries: *Google scholar*, *IEEE Xplore*, *ACM DL*, *Springer Link*, *Science Direct*, and *Scopus*. We screened many publications, but only six fulfilled the criteria and were evaluated by us [12]–[18].

Review Results. Table 2 describes the list of the identified architectures and which of the challenges each addresses.

The mapping study differentiates between four types of architecture models, namely: *Object-Oriented Robotics (OO-R)*, *Component-Based Robotics (CB-R)*, *Service-Driven Robotics (SD-R)*, and *Cloud robotics*. We evaluated the work of Baier et al. [19], that focuses on the robotics development activity and is part of the OO-R model. Their approach relies in a OO-R architectural model, which, contrary to the other models, is not able to tackle most of the challenges. Most of the extracted papers belong to the CB-R category, since this architectural model has been broadly used. As shown in the table, most of these works are able to tackle the problems of Design (Ch4), Programming (Ch5), and Reconfiguration (Ch10). Nevertheless, these architectures neither rely on decentralized approaches (Ch8) nor support a distributed access to the resources (Ch1). For example, Tajalli et al. [22] proposed a three-layered architecture intended for self-adaptation. It was implemented by Medvidovic et al. citemedvidovic, and put to use to orchestrate a heterogeneous team of robots. However, their implementation has the same limitations that PLASMA [22] has. On the contrary, SERA is able to deal with decentralization and collaboration between robots and humans. Another example is AEROSTACK [16], a novel software architecture for swarms of aerial robots that only fails at solving the modeling challenge (Ch3) and not supporting a decentralized way of managing robotic teams (Ch8). However, this architecture and its framework are only focused in the usage of aerial robots. Within the SD-R category, SORA [23] is able to tackle most of the challenges described above, but it lacks the ability of controlling a team of robots in a decentralized fashion. Unlike SERA, SORA heavily relies on a specific middleware. In contrast, we strive to make SERA supporting a wide variety of middleware. The work of Hu et al. [24] falls in the category of cloud computing since this work allows the most computing expensive tasks to be performed in the cloud instead of locally in each robot. Their architecture supports a decentralized way of communication between robots, except, indeed the communication with the cloud. Moreover, the control is decentralized.

TABLE 1. INCLUSION AND EXCLUSION CRITERIA.

	Inclusion criteria
1.	Primary studies
2.	Studies (i.e. papers) that present new reference software architectures for robotic applications.
3.	Studies that present instantiations of already existing reference software architectures within the field of robotics.
	Exclusion criteria
1.	Studies written in any language other than the English language.
2.	Short publications and posters ($<$ 3 pages).
3.	Publications that were unavailable through the search engine.
4.	Studies that focus on software frameworks, toolchains or platforms.

TABLE 2. SOFTWARE ARCHITECTURES PRE-STUDY

	P1					**P2**				**P3**	
	Ch1	**Ch2**	**Ch3**	**Ch4**	**Ch5**	**Ch6**	**Ch7**	**Ch8**	**Ch9**	**Ch10**	**Ch11**
OO-R											
Roblet [19]		•		•		•	•		•		•
CB-R											
BRICS [20]		•	•	•	•					•	
SAW [21]		•		•	•	•	•			•	
PLASMA [22]		•		•	•	•	•			•	•
SAFECASS [13]				•	•				•	•	•
AEROSTACK [16]	•	•		•	•	•	•		•	•	•
ArchGenTool [17]				•	•				•	•	•
MORPH [12]		•	•	•		•				•	•
SD-R											
SORA [23]	•	•	•	•	•	•	•		•	•	•
NUclear [15]	•	•			•	•	•				•
Cloud robotics											
Hu et al. [24]	•				•	•	•	*		•	•
RAPP [18]	•	•		•	•	•	•		•		•
SERA	*	•		•	•	•	•	•	•	•	•

Study Conclusion. In summary, while a large body of works exists on software architecture for robotic applications, most of these works do neither cope with decentralization nor with human-robot collaboration. Especially, none of them solves both challenges at the same time.

3. Research Method

Method. We followed the Design Science research approach [25] to conceive and implement SERA. The research method consists of three phases for iteratively creating an artifact (which is SERA): *knowledge base*, *development*, and *justification and evaluation*. The *environment* represents the problem space of our research field. It comprises people, organizations, and their technology. The research is addressed through *development* and *justification and evaluation* in an iterative way. Based on the input from the *environment* and the *knowledge base*, we developed our architecture and evaluated it. We realized this phase through meetings, both internal and external, with the rest of the project consortium, collecting feedback from them and improving SERA based on it. Finally, the *knowledge base* consists of the raw materials and already

TABLE 3. MEETINGS SPECIFICATION

	Expertise	Role	Experience	Participation
1	RTP	PL	23 years	4
2	RTP	AP	10 years	1
3-5	RTP	PDR	5-9 years	4
6	RTP	PhD	2 years	5
7	RMC	FP	30 years	2
8-9	RMC	PDR	5-9 years	4
10	RMC	PhD	2 years	4
11	RP	FP	39 years	2
12-13	RP	PDR	5-9 years	2
14	SE	AP	19 years	4
15	SE	AP	10 years	1
16	SE	PDR	9 years	4
17	SE	PhD	2 years	5
18	B	SM	10 years	2
19	B	SD	5 years	4
20	P	CEO	10 years	2
21	P	SD	15 years	2

Roles: Project Leader (PL); Full Professor (FP); Associate Professor (AP); Post-doc researcher (PDR); PhD student (PhD); Company's chief executive officer (CEO); Senior Manager (SM); and Software developer (SD).
Expertise: Real-time planning (RTP); Robotic motion and control (RMC); Robotic perception (RP); Software Engin. (SE);
Industrial partners: Bosch (B) and PAL Robotics (P). The field of expertise of participants 18 and 19 (B) is Artificial Intelligence. Number 20's (P) fields of expertise are Event Planning and Artificial Intelligence. Partner number 21 (P) is specialized in Computer Vision, Robotics, and Artificial Intelligence.

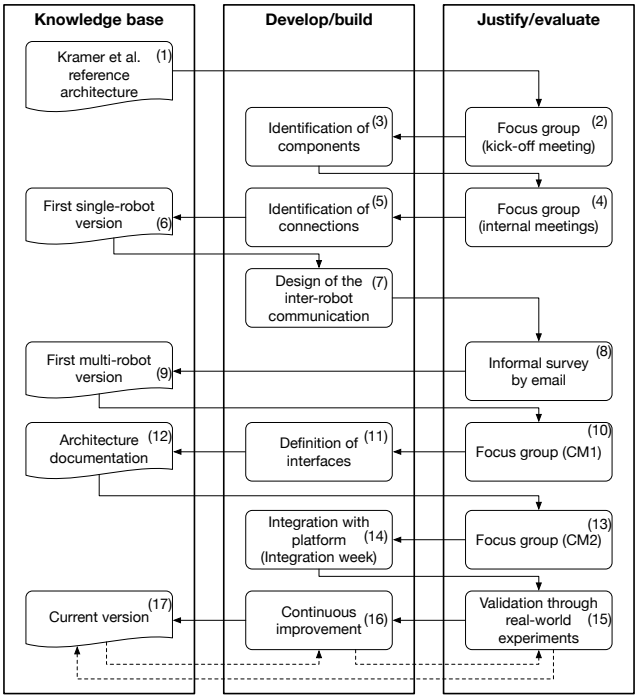

Figure 1. Process followed for defining SERA.

known research. We contribute both to the *environment* and to the *knowledge base* with our research outcome (SERA).

A central part of our research method is *focus groups* [26]. For this reason, we held several focus-group meetings with our partners while developing SERA. Specifically, in the meetings, we gave a presentation of the current work, discussed open questions and the next steps. Each discussion helped refining the main architectural decisions. We discussed, validated, and implemented/declined each of the points collected from every focus group, depending their agreed priority (i.e., whether requirement or suggestion).

Participants. Table 3 shows our focus-group participants along with their affiliations, roles, experience, and participation quantified in the number of meetings they attended. Regarding our industrial partners, Bosch[2] is a multinational, large-size company that mainly works in the field of engineering and electronics. Apart from the integration process within the project, they provide their own *mobile robotic platforms* and their facilities for real-world experiments. PAL Robotics[3] is a multinational, medium-size company that manufactures humanoid robots. Within the project they also provide a *mobile manipulator*, a *stationary manipulator*, and their facilities for real-world experiments. Also, both companies provided models of their robot and facilities for simulating purposes. The rest of the stakeholders have the role of *developers* and are in charge of developing component code for implementing SERA. The heterogeneity of our participants and the differences between the academic and the industrial partners allow the project to have different scopes and points of view when planning and tackling problems.

Process. The process followed for developing our architecture is depicted in Fig. 1; this figure represents how we performed the three phases of the Design Science research approach framework. Our starting point is the well-known three-layer reference architecture described by Kramer et al. [5] (1) and others [4]. At the beginning of the project, we identified our stakeholders' expectations for SERA in a first focus group (2). For example, a discussion was held to define what are the possible solutions for efficiently and effectively managing a robotic team in a decentralized way. This way, we identified the main functionalities that our architecture was required to support. Then, we defined them in the form of components (3). The refinement was conducted in eight internal focus groups (4) where only subjects from the SE group (i.e., the authors) participated. Each meeting consisted of a short presentation of the already identified functionalities and suggestions of new components. When the components were defined we identified the connections between them (5), achieving the first version of SERA (6). At this point it was only able to support a single-robot application, but it was used as a first release to be extended and refined. Then, we improved SERA and upgraded its version by adding the multi-robot functionality (7), i.e., enabling the inter-robot communication. We validated this functionality by getting feedback from stakeholders and practitioners of the project (8) through a set of informal surveys (one of them specific for industrial ones). In the surveys we inquired technical questions regarding the interfaces that each partner planned to provide and request (https://goo.gl/oPCHxV). The new

2. https://www.bosch.com/
3. http://www.pal-robotics.com/en/home/

multi-robot version of SERA was shared in a collaborative workspace where our partners could give us feedback in a more agile way (9).

The project held two consortium meetings (CM in the following), which had the same purpose: present the current state of our work and receive appropriate feedback from the rest of the consortium. During the first CM (CM1), participants were asked about the interfaces they require from, or provide to, other partners during the developing and integration processes (10). These questions and further discussion were continued afterwards via email to clearly define the interfaces (11). After documenting SERA (12), the second CM (CM2) was held (13) in order to discuss our progress before testing the software in a real robot. After CM2, the next steps were the integration within the software platform and testing. To test the work we had so far, the project also organized an integration week (IW), where all developed packages were first simulated and then deployed within a robot (14). In this context, we demonstrated through experiments in real-world scenarios that SERA can support the performance of a robot achieving complex missions (15)—i.e., collaborative transportation and autonomous navigation in a dynamic environment. After the validation, we performed minor improvements (16) to arrive at the current version of our architecture (17). Our architecture achieved a stable status; however, its design facilitates SERA's maintenance and continuous improvement in the future.

4. Architectural Design of SERA

SERA is inspired by and extends concepts of existing proposals for robot software architectures from the literature, as identified in the survey (cf. Sec. 2). We now present SERA and an example instantiation aimed at four real-world scenarios (explained shortly) by discussing the main design decisions, describing SERA's three layers, main components, and the functionalities they encapsulate. Our instantiation is depicted in Fig. 2, which guides our description.

The main stakeholders of SERA are: *developers* who instantiate SERA by implementing its components or reusing the already developed ones stored in a components repository, and *non-technical end-users* (without expertise in robotic and/or ICT), who will use the instance to specify missions for multi-robots using the so-called central station. The central station decomposes the global mission into local (robot-specific) missions *before* mission execution (adaptations during execution are performed in a decentralized way).

Our example instantiation of SERA (Fig. 2) is inspired by the following four scenarios of robot collaboration, whereas the first is used in our evaluation with real robots.

Case A, Physical Guidance: A human physically guides a robot to carry an object;

Case B, Collaborative Grasping: A mobile manipulator and a stationary manipulator collaboratively grasp and manipulate an object;

Case C, Collaborative Loading/Unloading: A mobile platform and a stationary manipulator collaborate to load and unload objects onto a mobile robotic platform; and

Case D, Information Exchange: Information exchange between a human giving orders and a robotic agent.

4.1. Main Architectural Units

The components of SERA are defined and allocated into two main units, the *central station* and the *robot*.

Central station. The central station interprets high-level mission specifications, comprising a hierarchy of three components. On top, the *High-level specification manager* provides an interface (a graphical one in our SERA instance) to specify high-level missions of the application in the component. The global mission—i.e., the mission to be achieved by the whole team in a collaborative manner—is processed by the *Global mission manager*, which checks the feasibility of the mission and forwards it to the *Global mission decomposer*. Global missions can only be altered by the user manually. The latter splits the global mission into local missions to be accomplished by each single robot. Decomposing temporal logic formulas into a set of formulas is a hard problem and cannot be solved in general. Therefore, our strategy is to specify missions through a Domain Specific Language (DSL) that has been conceived to enable non-technical users to specify missions for multiple robots. The decomposition of a global mission into local missions is based on the semantics of the operators of the DSL. Further information will be be found in Deliverable 4.1, available on the webpage of the project.

For example, the collaborative mission to be achieved in case C is split into "load the item on top of the mobile platform" and "receive the item". These local missions are transferred to the mobile platform and the stationary manipulator, respectively. Once the local missions are transferred (*before* execution), the central station ends its role and the mission execution and possible adaptations and reconfigurations are performed in a decentralized way.

Robot. This unit represents a framework targeting a single, deployed robot, aiming at controlling its run-time. It contains a set of components to achieve the local mission received from the central station. In SERA, each robot is realized as a service that communicates with other robots, loosely following the Service-Oriented Architecture (SOA) paradigm.

4.2. Architectural Layers in the Robots

The architecture of the robots follows the architectural guidelines of Kramer et al. [5] for self-adaptive systems and organizes the components in a three-layer architecture with the layers *Mission management*, *Change management*, and *Component control*.

Mission management layer. The local mission specifications, which typically incorporate real-time constraints, are provided to each robot of the team by means of timed temporal logic formulae. The formulae are built upon a discrete symbolic model (or abstraction) of the robots' motion capabilities, which is compliant with their dynamic behavior, and enables the synthesis of high-level plans.

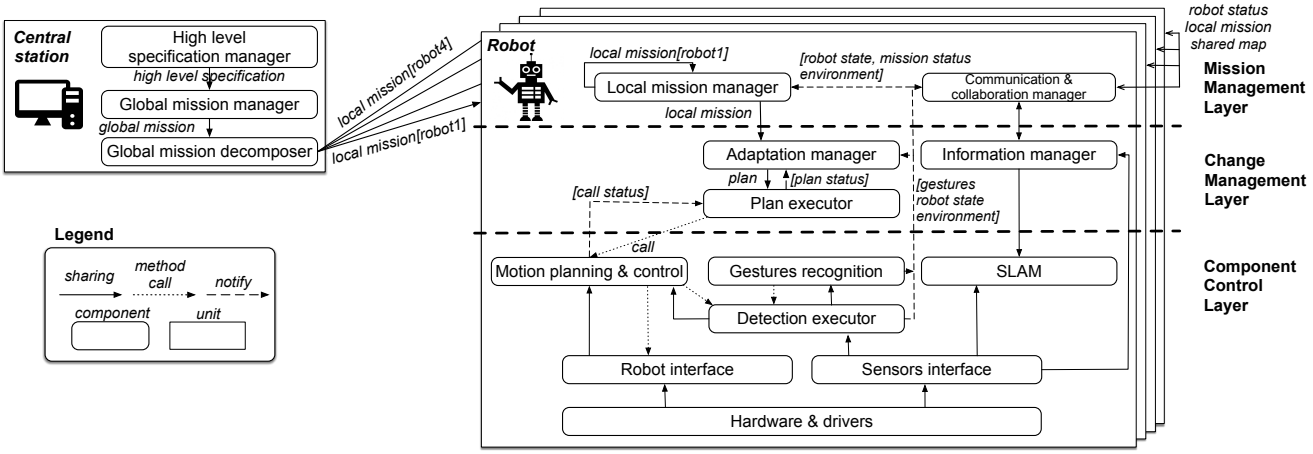

Figure 2. Instance of the SERA architecture

This layer manages the local missions of each robot and is in charge of communicating and synchronizing with the other robots in order to perform the collective adaptation. For example, in case C, in the event that a mobile robot transporting a load communicates a failure to the other robots, each robot synchronizes with other robots and computes a new set of local missions that allow achieving the global mission. The *Communication and collaboration manager* is the component in charge of establishing this communication between robots. This component shares required information from the local robot (info from the sensors, its position, etc. received from the middle and lower layers) and also reads the feedback provided by the rest of the team so that the adaptation, synchronization, and collaboration can be achieved. The *Local mission manager* checks the feasibility of the mission to be achieved by the robot. If accepted, it forwards the mission to the *Adaptation Manager*.

Change management layer. This layer manages the adaptation performed or requested by the upper and lower layers. This adaptation can be triggered by changes in the context and/or by mission infeasibility. It is responsible for the employment of reconfiguration strategies at a local level and the refinement of the discrete abstractions. When a satisfying plan cannot be found with the initial discretization of the dynamics, the symbolic models are refined accordingly with specific discrete paths. This permits to obtain finer discretizations and increases the possibility of finding a satisfying plan. The *Adaptation manager* receives the local mission, and produces a plan that specifies how the robot performs that mission. For example, in case C, the mission "load an item on top of the mobile platform" is converted into a plan including tasks such as "grasp the target item". If there is an unexpected event (e.g., an obstacle in the environment) this component adapts to it (e.g., replanning the path to follow). If adaptation at local level is not feasible (e.g., the gripper of a robot results broken), it is forwarded to the upper level to perform the global adaptation (e.g., replace the robot with an idle one). Then, the computed plan is forwarded to the *Plan executor*, which exploits the lower-level component *Motion Planning and Control*. The *Information manager* is defined for managing the communication between the upper and lower layers; it performs an asynchronous message passing to decouple the layers.

Component control layer. This layer focuses on the design of the control actions that are required for the implementation of the discrete paths that are generated by a high-level planner (performed by the *Adaptation manager*). This layer receives calls from the *Plan executor* and notifies the *Adaptation manager* about changes that are detected in the robot mission, environment, or current state.

The component *Motion Planning and Control* contains all the necessary packages needed for the correct performance of the robot's motion. It is responsible for the autonomous motion of the robots while simultaneously performing obstacle and collision avoidance (i.e., low-level planning). It receives information from the components *SLAM* (Simultaneous Localization and Mapping) and *Detection executor* to perform the motion planning. *Motion Planning and Control* receives calls from the *Plan executor*. It also maintains a bidirectional connection with the *Robot Interface* for communicating with the robot's actuators. Moreover, the *Gestures recognition* reads the information given by the algorithms from the *Detection executor* and recognizes different gestures from a human user. Then, it sends the collected information to the *Adaptation manager* in the change management layer above. Local missions can be changed dynamically through specific gestures from a human (e.g., stop the mission, follow me, etc.). The *Detection executor* is the component that perceives other agents (which could be humans or other robots) and objects of interest in the environment. It uses the feed from the visual sensors. It also publishes information regarding the state of other agents, as other robots or humans. The component *SLAM* receives the data from the *Sensors interface* (it is depicted as one interface, but the communication represents several interfaces from several robotic sensors) and also from the rest of the robots that compose the team. The SLAM algorithm then computes the current position of the robot in the environment and builds

the map of that environment; then, it shares both with the middle and upper layers. The interface for the robot, which represents the structures of each robot (we implemented the three robots mentioned in Sec. 3 in the current instantiation), is represented as the *Robot interface* component. In the same way, the *Sensors interface* component contains the interface for all the available sensors existing in the robots. Finally, the component *Hardware and Drivers* actually represents several components, one for each of the drivers implemented to control the real robot hardware.

5. Implementation

SERA can be implemented within a broad range of projects (can be realized using different middlewares and component frameworks) related with robotics. The instance we implemented [**?**] using Python and C++ relies on the middleware ROS [11], which provides a communication layer upon the Linux operating system, supporting the execution of components in a distributed system. ROS offers a message-based peer-to-peer communication infrastructure supporting the integration of independently developed software components called ROS nodes. To realize SOA-like communication among components in SERA, we developed a connector that treats every robot as a REST service and provides communication features. We developed the connector following the guidelines of previous examples, as ROStful[4]. Then, an instantiation of ROS with the *Robot* unit framework explained in Sec. 4 is deployed within each robot. The interface between REST and ROS is realized in SERA via the *Communication and collaboration manager*.

TABLE 4. ADAPTATION MANAGER INTERFACES

Interface name	I/O	Message type
/map	I	nav_msgs/OccupancyGrid
/action/result	I	roseus/StringStamped
/move_base/result	I	move_base_msgs/MoveBaseActionResult
/tracked_humans	I	Humans
/tracked_robots	I	Robots
/robot_pose	I	geometry_msgs/PoseWithCovarianceStamped
/robot_status	I	roseus/StringStamped
/mission_update	I	roseus/StringStamped
/local_mission	I	roseus/StringStamped
/action/id	O	roseus/StringStamped
/move_base/goal	O	roseus/StringStamped

As stated in the sections above, SERA is developed with the intention of facilitating the process of developing and deploying software into a robot. For this reason, we defined clear components that compose the architecture and the interfaces between them. Each instance of our architecture implements the components, whose interfaces are defined by SERA. We also created a framework that contains an abstract class for each of the SERA's components. Each of these classes encapsulates all the information regarding the communication code in ROS. It can be understood as a template method that can be filled in by developers.

4. https://goo.gl/1RVW54

On one hand, the template method approach takes some flexibility from the developer, disallowing developers to create or modify component interfaces. On the other hand, it simplifies the development or modification of the components for SERA. So, we let developers focus on component logic, alleviating the need to care about inter-component communication, such as defining message names that are provided or requested by components, as would be necessary with plain ROS. Consequently, instantiating a new component amounts to creating a new class that calls the abstract method contained in the abstract class belonging to the component in our framework. This decouples inter-component communication (i.e., message passing) from the component logic and reduces the complexity of the latter.

In the following, we exemplify how we implemented an instantiation of the *Adaptation manager* in SERA. Most of the components implemented in the current instance of SERA were developed (or migrated) by our partners, except the *High level specification manager* and the *Communication & collaboration manager*, which were developed by us. Due to space limitations, we detail only one component. A white paper with detailed explanation, diagrams, and an interfaces map is provided in the project's webpage (Deliverables 5.1, 5.2). The version of the *Adaptation manager* component built outside of the framework has 227 lines of code (LOC), while its version that is compliant to the framework has only 58 lines; therefore the reduction is valuable.

```
1 class LTLAdaptationManager(AdaptationManager):
2     def plan(self, currentmap, mission, robot_pose,
      robot_status, tracked_humans):
3         [robot_motion, init_pose, robot_action,
      robot_mission] = robot_model
4         full_model = MotActModel(robot_motion,
      robot_action)
5         planner = ltl_planner(full_model,'true',mission)
6         planner.optimal(10)
7         movement_plan=MovementPlan(planner.run.pre_plan,
      planner.run.suf_plan)
8         retplan=Plan('PLAN', '', movement_plan)
9         return retplan
```

Listing 1. Adaptation manager instantiation

The instantiation of the *Adaptation manager* is shown in Listing 1, contained in `ltladaptationmanager.py` in the repository. The concrete class extends the abstract class AdaptationManager (Line 1), which contains the logic of the component and calls to the abstract method (Listing 3), as shown in Listing 2. Listing 2 and Listing 3 represent pieces of code from the component, contained in `adaptationmanager.py` in the repository. Our goal with the template method is to allow the instantiation of a component by just defining some features defined by the abstract method (see Listing 1, line 2).

```
1 retplan=self.plan(self.status.map, task, self.status.
      robot_pose, self.status.robot_status, self.status.
      tracked_humans)
```

Listing 2. Call to the abstract class

```
1 @abstractmethod
2 def plan(self, currentmap, task, robot_pose, robot_status
      , tracked_humans):
```

Listing 3. Adaptation manager abstract class

Table 4 presents information about all the interfaces that are provided and requested in the *Adaptation manager*. For the current implementation of SERA, we use ROS as the middleware according to which the interfaces are build. For this reason, our interfaces are realized as ROS topics (named buses over which nodes exchange messages), which are specified in the Interface name column. The I/O column expresses whether an interface provides or requires information. The last column presents the message type (data description of each message) for each of our interfaces.

6. Validation

The validation of SERA was performed with respect to the challenges discussed in Sec. 2. This permitted us to benchmark SERA and its implementation with state-of-the-art solutions. In Sec. 6.1, we introduce the strategy we defined and followed in order to validate SERA before describing the validation results in Sec. 6.2.

6.1. Validation Strategy

The validation is performed through three main means:
Human experts. We performed the validation with the help of experts and practitioners in the field of robotics. Refer to Sec. 3 for a description of the stakeholders involved in this validation. This validation with human experts included three steps: (i) internal validation performed through focus groups involving only SE experts; (ii) presentation of preliminary and final results to all stakeholders and collection of feedback by means of focus groups; and (iii) further discussion via email and informal surveys.
Simulation environment. We use Gazebo (http://gazebosim.org/) as a simulator, a tool that allows us to easily simulate robots in various scenarios and that is well-integrated with ROS. Both of our industrial partners provided repositories (https://github.com/pal-robotics) that they used for internal testing in their everyday work. Specifically, in this study we used the models of the robots and of the company's facilities provided by the companies.
Real robots. We validated SERA with a real robot in a real-world scenario that has been conceived in order to validate the code and artifacts developed in this study, i.e., the case A in Sec. 4. Concretely, we tested SERA with an experiment that involved a mobile manipulator from PAL Robotics interacting and working together with a human being in PAL Robotics' facilities. Refer to the website of the project for demonstration videos. The experiment was performed during the IW, in which we focused on the realization of an experiment for case A. The scenario involves a human that has to guide a robot while performing collaborative transportation. In this case, a robot must achieve several tasks: (i) It should be able to autonomously navigate in a known but changing and dynamic environment shared with humans; (ii) When the robot reaches a certain predefined location the robot looks for a human being, the user; (iii) Once detected, the user points to the target object to be transported so the robot can detect and track it; (iv) Then, the *collaborative transportation* starts, where both human and robot transport an object in a collaborative manner; the human acts as the master and the robot as the slave; (v) The transportation goes on until the user indicates the end of the action via gestures.

6.2. Validation Results

We now discuss the results of our evaluation of SERA with respect to the challenges identified in Sec. 2. The validation made use of the validation means described in the previous section. The results of our evaluation are depicted in Table 2, where SERA is benchmarked with the other approaches discussed in this paper.
P1 Robotic Framework.
Ch1 - Distributed Resources Access: This challenge is addressed by exploiting loosely coupled services that can be dynamically discovered and invoked. Within SERA each robot is considered a service with limited knowledge of the configuration and status of other robots. This is scalable and enables an easy substitution of robots or addition of new ones. SERA implements both the SOA-like and the publish-subscribe paradigms for the communication among and within robots, respectively.
Ch2 - Re-engineering: For validating Ch2 we made use of two metrics: flexibility and efficiency. Regarding flexibility, the metric checks the number of architectural changes needed to deploy new software components and to manage different types of robots. This has been validated in the simulation environment and with real robots. Efficiency is defined in terms of amount and type of errors occurred when deploying new instantiations of components. Thanks to the well-defined interfaces of the architecture, different instantiations of software components (e.g., different high-level planners were used depending of the environment where the robot operates) were used during the IW without struggles.
Ch3 - Modeling: Although SERA has been defined to support this challenge, it has not been directly addressed yet. In fact, we plan to automatically generate the skeleton code that implements the architecture and that provides integration and communication facilities, as explained in Sec. 5.
Ch4 - Design: SERA encapsulates functionalities of the robotic system in reusable, modular, and exchangeable components, so each each component of the architecture could be developed in isolation and in parallel. We validated Ch4 by measuring the changes needed to integrate each component, and the number and type of errors. In the IW, the components integration (twelve components implemented) was straightforward, since interfaces could be defined beforehand in the iterative and interactive design process. No errors occurred during deployment, even for critical components such as the *Adaptation manager*, which shares many interfaces with other components.
Ch5 - Programming: The simulated team of robots (formed by three robots) is deployed in two different simulated environments: the facilities of both companies. The real-world scenario consisted of one robot performing different missions in the facilities of PAL Robotics. The instantiations

of each of SERA's components are defined by abstract classes that assure a correct implementation of communication between components. Recall that we also counted the number of lines of code necessary to implement an instantiation of a component with and without our approach (cf. Sec. 5). For instance, the lines of code for the implementation of the component Adaptation manager reduced from 227 to 58.

P2 Multi and heterogeneous collaborating robots.

Ch6 - Information Fusion: Information fusion is considered at two different levels: Information fusion of (i) data coming from different sensors of the same robot; and (ii) data coming from other sources that are external to the robot (e.g., other robots). The first level of information fusion is performed by the SLAM component. It fuses the raw feed from different sensors (e.g., laser, RGB-D camera, IMU). We validated the first level via the three main means. The second level is performed by means of the component Communication & collaboration manager. This level has been validated through the *Human experts* means. We performed an initial validation through simulation in order to validate the interaction of the component with the remaining part of the system.

Ch7 - Multi-robot: SERA uses the *Communication and collaboration manager* as an interface between the REST pattern and the ROS internal communication protocol for enabling inter-robot communication. The approach and components to be implemented in order to support the multi-robot feature were validated through the mean *Human experts*. Then, the ability of a team of robots for performing a mission in a collaborative way were tested through the *Simulation environment*. We performed 12 experiments with a team of three robots with different combinations of the three robots models provided by Bosch and PAL Robotics. The experiments were conducted in the two main scenarios (facilities of both companies).

Ch8 - Decentralized: As explained in Sec. 4, SERA works in a decentralized way during the mission execution. Ch8 was validated through the *Human experts* means during the CMs (cf. Sec. 3). Specifically, we used the means *Simulation environment*, where a robotic team was able to achieve a mission without receiving instructions from a central unit at run-time. The experiments simulated a robotic team of three robotic agents (different combinations of robot models provided by both companies) in the two main scenarios (facilities of both companies).

Ch9 - Human-robot collaboration: Robotic applications developed with SERA are able to interact with humans, which is validated in three steps: (i) strategies and functionalities to be encapsulated in components were discussed and defined through the *Human experts* mean; (ii) specific tasks related with this challenge were simulated (even though, we did not simulate human being models but human-shaped objects), e.g., grasping an object that is held by a human; and (iii) the experiment realized with a real robot in a real-world scenario involved a mobile manipulator transporting an item in a collaborative way with a human (cf. Sec. 6.1).

P3 Reconfigurability and adaptability.

Ch10 - Reconfiguration: The validation of the reconfigurability is performed through the three means, but our robots are not yet completely self-adaptive since we have not been able to validate the entire MAPE-K loop [27]. For illustrating it, we refer to a case that happened during the IW, where there was a human in the loop that triggered the activation and deactivation of specific features that were too computationally expensive—e.g., the gestures recognition. Thus, the loop lacks the Monitor step. For this reason, a desirable feature for robots is to be able to automatically enable and disable components at run-time under certain conditions. Nevertheless, SERA tackles Ch10 by other means, as is the case of the re-planning and the collision avoidance and is validated through the three means. Re-planning from a high-level planner is performed by the *Adaptation manager*. A low-level planning allows the avoidance of (dynamic) obstacles by means of the *Motion planning & control*. Gazebo (the *Simulated environment*) allows the user to dynamically manipulate the environment, so the robot has to adapt its path and/or mission accordingly. The validation through real robots consists in the execution of two missions (i.e. autonomous navigation and collaborative transportation) in a changing environment with the presence of humans.

Ch11 - Fault Tolerance: To classify and evaluate different categories of faults, we use Steinbauer's taxonomy [28], who classifies fault types into: (i) interaction, i.e. problems perceiving and acting with with other agents as robots or human beings; (ii) algorithms, i.e. badly implemented algorithms and shortcomings of the algorithm itself; (iii) software, i.e. faults related with design and implementation of robot's software systems; and (iv) hardware, i.e physical faults in robot's equipment. We evaluated Ch11 in terms of these categories and based on the three main means. The validation through the *Simulation environment* allowed us to evaluate the algorithms, software, hardware, and collaboration between robots (not humans, since we did not simulate them) types of faults. The validation through *Real robots* allowed us to evaluate various types of faults. For instance, we evaluated (i) cases in which the robot is not able to perceive other agents due to the lack of onboard computational capabilities; (ii) cases in which the design of the components and their interfaces was not correct and therefore the integration was not straight-forward; or (iii) cases in which the gripper of the robot did not work properly and then the mission was not achievable.

6.3. Threats to Validity

Internal validity. A potential threat to the internal validity of our work is that the experts who contributed and were interviewed are part of the same research project, which might have biased results. Furthermore, only two companies were part of the contributing group of experts. Yet, the project has further stakeholders with a heterogeneous background, who were informed and commented on SERA. Collaborating with more companies, such as via interviews and surveys, is valuable future work.

External validity. Our experiments include only small groups of three robots, which is not a sizable number regarding multi-robot coordination. These teams of robots just include

three robotic models, while we aim to support a heterogeneous team of robots. However, the three types of robots we considered differ substantially in their functionalities. Moreover, some of the experiments were only validated by simulation. Performing experiments with other real robots apart from the ones provided by our industrial partners is subject to our future work.

7. Conclusion

In this paper, we presented SERA, a decentralized architecture for teams of self-adaptive robots. We motivated it with four case studies. In a design-science approach to develop SERA, we collaborated with 21 experts and practitioners with heterogeneous background. This allowed us to conceive, improve, and validate our architecture based on expert robotics knowledge. We validated SERA through three different means, including experiments involving a real robot in a real-word scenario.

SERA supports robot-application developers, who instantiate it to their needs. They can use defined interfaces to implement specific components or reuse implemented components from our framework. Inter-component communication is provided by the framework—low-level message-passing (from ROS or another middleware) is abstracted.

As future work, we plan various extensions and refinements of SERA. We plan to automatize the instantiation of the architecture considering different contexts and to automate the code generation for the framework that supports the architecture. Furthermore, we will investigate an extension of the components' interfaces to enable a description of the behavior of each component; specifically, the extended interfaces will specify when and how a component can be used (assumptions on its environment) and what the component ensures when operating in such an environment (guarantees). SERA should also consider behavioral concerns and not just (syntactic) compliance to APIs—e.g., to detect whether new components can be effectively plugged into the system or a component can be substituted.

References

[1] H. H. Poole, *Fundamentals of robotics engineering*. Springer, 2012.

[2] IFR, "World Robotic Survey," https://ifr.org/ifr-press-releases/news/world-robotics-survey-service-robots-are-conquering-the-world-, 2016.

[3] C. Sheng, "Global dom. serv. robots market - size and trends to 2020," https://www.alliedmarketresearch.com/robotics-technology-market, 2016.

[4] E. Gat *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[5] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering*, 2007.

[6] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journ. of Syst. and Soft.*, 2016.

[7] T. Kaupp, A. Brooks, B. Upcroft, and A. Makarenko, "Building a software architecture for a human-robot team using the orca framework," in *Int. Conference on Robotics and Automation*, 2007.

[8] EU, "Robotics 2020 Multi-Annual Roadmap For Robotic in Europe," http://sparc-robotics.eu/wp-content/uploads/2014/05/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf-, 2016.

[9] R. de Lemos, H. Giese *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg, 2013, pp. 1–32.

[10] Z. Yan, N. Jouandeau, and A. Cherif, "A survey and analysis of multi-robot coordination," *Journal of Advanced Robotic Systems*, 2013.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Mg, "ROS: an open-source robot operating system," 2009.

[12] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel, "Morph: A reference architecture for configuration and behaviour self-adaptation," in *CTSE*, 2015.

[13] M. Y. Jung and P. Kazanzides, "An architectural approach to safety of component-based robotic systems," *ICRA*, pp. 3360–3366, 2016.

[14] S. Yang, X. Mao, S. Yang, and Z. Liu, "Towards a hybrid software architecture and multi-agent approach for autonomous robot software," *International Journal of Advanced Robotic Systems*, 2017.

[15] T. Houliston, J. Fountain, Y. Lin, A. Mendes, M. Metcalfe, J. Walker, and S. K. Chalup, "Nuclear: A loosely coupled software architecture for humanoid robot systems," *Frontiers in Robotics and AI*, 2016.

[16] J. L. Sanchez-Lopez, R. Fernndez, H. Bavle, C. Sampedro, M. Molina, J. Pestana, and P. Campoy, "Aerostack: An architecture and open-source software framework for aerial robotics," in *ICUAS*, 2016.

[17] E. Ruffaldi, I. Kostavelis, D. Giakoumis, and D. Tzovaras, "Archgen-tool: A system-independent collaborative tool for robotic architecture design," in *MMAR*, 2016.

[18] E. G. Tsardoulias and et al., "Towards an integrated robotics architecture for social inclusion – the rapp paradigm," *Cognitive Systems Research*, 2017.

[19] T. Baier, M. Huser, D. Westhoff, and J. Zhang, "A flexible software architecture for multi-modal service robots," in *Multiconference on Computational Engineering in Systems Applications*, 2006.

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

[21] M. Y. Jung, A. Deguet, and P. Kazanzides, "A component-based architecture for flexible integration of robotic systems," in *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010.

[22] H. Tajalli, J. Garcia, G. Edwards, and N. Medvidovic, "Plasma: A plan-based layered architecture for software model-driven adaptation," in *International Conference on Automated Software Engineering*, 2010.

[23] L. Fluckiger and H. Utz, "Service oriented robotic architecture for space robotics: Design, testing, and lessons learned," *Journal of Field Robotics*, 2014.

[24] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE Network*, 2012.

[25] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, 2004.

[26] J. Kitzinger, "Qualitative research: Introducing focus groups," *BMJ*, vol. 311, no. 7000, pp. 299–302, 1995.

[27] "An architectural blueprint for autonomic computing," IBM, Tech. Rep., 2005.

[28] G. Steinbauer, "A survey about faults of robots used in robocup," in *RoboCup 2012: Robot Soccer World Cup XVI*, 2013.