The Journal of Systems and Software 122 (2016) 16–39

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software architectures for robotic systems: A systematic mapping study

Aakash Ahmad[a], Muhammad Ali Babar[a,b]

[a] CREST - Centre for Research on Engineering Software Technologies, Software and Systems Section, IT University of Copenhagen, Denmark
[b] CREST - Centre for Research on Engineering Software Technologies, School of Computer Science, University of Adelaide, Australia

ARTICLE INFO

Article history:
Received 4 May 2015
Revised 5 August 2016
Accepted 8 August 2016
Available online 21 August 2016

Keywords:
Software architecture
Robotic systems
Evidence-based software engineering
Software architecture for robotics
Systematic mapping study

ABSTRACT

Context: Several research efforts have been targeted to support architecture centric development and evolution of software for robotic systems for the last two decades.
Objective: We aimed to systematically *identify* and *classify* the existing solutions, research progress and directions that influence architecture-driven modeling, development and evolution of robotic software.
Research Method: We have used Systematic Mapping Study (SMS) method for identifying and analyzing 56 peer-reviewed papers. Our review has (i) taxonomically classified the existing research and (ii) systematically mapped the solutions, frameworks, notations and evaluation methods to highlight the role of software architecture in robotic systems.
Results and Conclusions: We have identified eight themes that support architectural solutions to enable (i) *operations*, (ii) *evolution* and (iii) *development* specific activities of robotic software. The research in this area has progressed from *object-oriented* to *component-based* and now to *service-driven* robotics representing different architectural models that emerged overtime. An emerging solution is *cloud robotics* that exploits the foundations of service-driven architectures to support an interconnected web of robots. The results of this SMS facilitate knowledge transfer – benefiting researchers and practitioners – focused on exploiting software architecture to model, develop and evolve robotic systems.

1. Introduction

Robotic systems are increasingly being integrated in various aspects of everyday life. The robotic applications range from mission critical (Parker, 1998) to infotainment and home service tasks (Mäenpää et al., 2004, Kwak et al., 2006). Robotic systems are expected to assist or replace their human counterparts for efficient and effective performance of all sorts of tasks such as industrial operations (Angerer et al., 2009) or surgical procedures (Kazanzides et al., 1992, Buzurovic et al., 2010). A robotic system is a combination of various components – hardware for system assembling and software for system operations – that must be seamlessly integrated to enable a robotic system's function as expected. To support the vision of a robotic-driven world[1], academic research (Parker, 1998, Hu et al., 2012, Brugali et al., 2007), industrial (Jackson and Coll, 2008, Katz and Some, 2003) and open source solutions (Garage, 2011, Bonarini et al., 2014) are striving to provide cost-effective and efficient solutions for the development, evolution and operations of robotics systems. Researchers (Brugali et al., 2007) and practitioners (Jackson and Coll, 2008) are increasingly focusing on exploiting software engineering methodologies to abstract complexities and enhance efficiency for modeling, developing, maintaining and evolving robotic systems cost-effectively (Oliveira et al., 2013, Zhi et al., 2013, Elkady and Sobh, 2012).

Software Architecture (SA) represents the global view of software systems by abstracting out the complexities of low-level design and implementation details (Wohlin, 2014). SA plays a vital role in ensuring the fulfillment of functional and non-functional requirements (Schmerl and Garlan, 2004). It is considered that architecture-centric software development helps increase quality, modularity and reusability (Brugali et al., 2007) through the use of patterns (Gomaa, 2000) and decrease complexity by applying model-driven development (Boyatzis, 1998). Researchers from different communities (such as robotics, software engineering, industrial engineering, and artificial intelligence) have exploited architectural models to design, reason about, and engineer robotic software. Architecture-centric robotics research and practice can be

E-mail addresses: aahm@itu.dk (A. Ahmad), ali.babar@adelaide.edu.au (M.A. Babar).

[1] EUROP, the European Robotics Technology Platform is an industry-driven platform compromising the main stakeholders in robotics. EUROP aims at enabling research and practices through *Robotic Visions to 2020 and Beyond - The Strategic Research Agenda for Robotics in Europe.*

http://dx.doi.org/10.1016/j.jss.2016.08.039
0164-1212/

characterized by various architectural models that emerged overtime such as: (i) object-oriented robotics (OO-R) enabling modularity (Miller and Lennox, 1991), (ii) component-based robotics (CB-R) supporting reusability (Jung et al., 2010), and (iii) service-driven robotics (SD-R) exploiting dynamic composition of software (Cepeda et al., 2011).

Since the early 90s, there has been a continuous stream of reported research on software architectures for robotic systems. It is a timely effort to analyze the collective impact of existing research on architectural solutions for robotic software. We decided to conduct a mapping study by following the guidelines reported in (Petersen et al., 2008) to investigate the state-of-the-art that promotes architectural solutions to model, develop and evolve robotic software. The objective of this mapping study is to: '*systematically identify, analyze, and classify the software architectural solutions for robotic systems and provide a mapping of these solutions to highlight their potential, limitations along with emerging and future dimensions of research*'. This study was motivated by a number of research questions, whose answers are expected to disseminate a systematized knowledge among researchers and practitioners who are interested in the role of software architecture for robotic systems. The key contributions of this study are:

- Systematic selection and analysis of the collective impact of the existing research on software architecture related aspects of robotic systems for identifying (i) predominant *research themes*, (ii) architectural *solutions* for each of the themes, (iii) *framework* support along with (iii) *modeling notations, evaluation methodologies* and *application domain* of architectural solutions.
- Reflections on the (i) *progression* and maturation of research overtime along with (ii) existing and emerging software architectural solutions for robotic systems.

The results of this study suggest that ***architectural solutions*** support (i) *operations* enabling information and resource sharing (ii) *evolution* with runtime adaptation and design-time re-engineering, along with (iii) development such as modeling, designing and programming of robotic software. A number of ***architectural frameworks*** - open source, academic and industrial solutions - are available for development; UML or its derivative notations are predominantly used for ***architectural modeling***. A majority of the architectural solutions support home service, mission critical and navigation robots. Architectural solutions in general have been evaluated using controlled experiments and simulation based techniques. However, there is a lack of reporting on ***architecture specific evaluations*** against functional and quality requirements (Orebäck and Christensen, 2003). An incremental progress of research from OO-R (Miller and Lennox, 1991) to CB-R (Jung et al., 2010) and more recently SD-R (Cepeda et al., 2011) (***architectural generations***) have resulted in an emergence of cloud robotics (Hu et al., 2012); while model-driven robotics is also gaining momentum (Boyatzis, 1998). The results of this SMS benefit:

- Researchers who are interested in knowing the state-of-the-art of software architecture for robotics systems. The systematic classification of the existing research provides a body of knowledge for deriving new hypotheses to be tested and identifying the areas of future research.
- Practitioners who may be interested in understanding the reported solutions in terms of frameworks, modeling notations, tools and validation techniques for architectural development and evolution of robotic systems.

The rest of this paper is organized as follows. Section 2 briefly introduces robotic systems, software architecture and related studies. Section 3 describes the research methodology used. Section 4 reports the years, classification and mapping of the research. Based on the classification, various architectural solutions and frameworks for robotic software are presented in Section 5. Modeling notations, validation techniques and application domains that complement architectural solutions are presented in Section 6. We discuss the progression, future dimensions, and emergence of various architectural models in Section 7. Validity threats are presented in Section 8. Section 9 presents key conclusions from this study.

2. Background and related studies

In this section, we briefly introduce robotic systems (in Section 2.1) and software architecture (in Section 2.2). We also discuss some existing secondary studies (in Section 2.3) that are related to our SMS.

2.1. An overview of robotic software systems

A robotic system is a combination of hardware and software components as two distinct layers that can be integrated to build a robot (Jackson and Coll, 2008, Yool et al., 2006). The ISO 8373:2012 standard provides a vocabulary of robots and various robotic devices that operate in industrial and non-industrial environments. The hardware components such as the sensors, robotic arms, navigation panels enable the assembly of a robot. Hardware components are controlled and manipulated by Control Layer that is essentially a collection of drivers (as system specific code) to interact with the hardware as in Fig. 1. For more complex functions of a robot, specialized software is provided for integration and coordination of hardware components to manipulate the robotic behavior. In Fig. 1, this refers to as Application Layer that utilizes the control layer to support robotic operations. For example, considering a home service robot (Schofield, 1999), the control layer provides a driver that enables access to a robotic arm. Depending on specific requirements, a software system at *application layer* must be provided. Such software system is expected to utilize drivers from *control layer* to enable the movement of arm for home service robot at certain degrees of precision and/or avoiding any obstacles.

Fig. 1 highlights that from a software perspective robotic systems are an example of a layered design, where each layer encapsulates a specific functionality and depends on the layer(s) above or below to complete a system's functionality. The focus of this study is to review the research state-of-the-art for architectural solutions supporting robotic software – *application layer*.

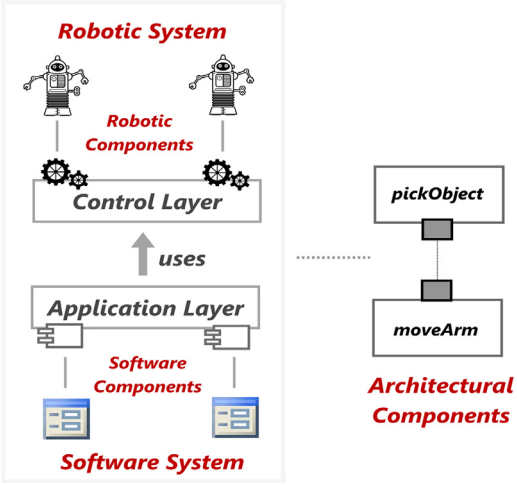

Fig. 1. A reference model for robotic software systems.

Table 1
A summary of secondary studies on software engineering for robotics.

Study reference	*Study focus*	Publication year	Studies reviewed
Systematic literature reviews			
Oliveira et al. (Oliveira et al., 2013)	*Service-oriented robotics*	2013	39
Pons et al. (Pons et al., 2012)	*Software engineering for robotics*	2012	67
Survey-based studies			
Zhi et al. (Zhi et al., 2013)	*Robotic coordination systems*	2013	No explicit mention
Elkady et al. (Elkady and Sobh, 2012)	*Robotic middleware*	2012	21
Kramer et al. (Kramer and Scheutz, 2007)	*Robotic development environments*	2007	09

2.2. Software architecture

The ISO/IEC/IEEE 42010 is a standard for architecture description of software systems that represents architecture as an aggregated or high-level view of software in terms of architectural components as computational elements and connectors that enable interconnections between components. This is also referred to as the component and connector (C&C) view of a system; where architectural components as computational elements represent an abstraction of executable code (Schmerl and Garlan, 2004) and communicate to each other using connectors. For example, considering Fig. 1, the classes or modules of code that supports movement of robotic arm can be packaged into *moveArm* component that communicates with *pickObject* component. With C&C view the complex and implementation specific details (i.e., modules and function calls) are abstracted to highlight that picking or moving an object depends on the movement of robotic arm.

Architecture models are beneficial for stakeholders in general and software designer and architects in particular to model and evolve software systems. An architectural model helps focus on higher level of abstractions, when implementation details have yet to be decided or are not considered relevant during reasoning about high level software and system decisions. For example, model-driven engineering (Boyatzis, 1998) allows requirements of a domain (e.g., movement of robotic arm) to be mapped to architecture models or components (e.g., *moveArm*) to generate executable code in a stepwise manner (Jung et al., 2010, Brugali et al., 2007).

2.3. Existing secondary studies on software for robotic systems

We found two types of secondary studies related to our study, presented in Table 1. We briefly discuss these studies in terms of their scope and contributions to justify the needs for our study. Table 1 presents a summary of these studies in terms of study reference, focus, publication year with individual details below.

2.3.1. Systematic literature reviews of software engineering for robotics

Oliveira et al. (Oliveira et al., 2013) have reported a systematic review of Service-Oriented Development of Robotic Systems based on 39 primary studies published from 2005 to 2011. Their review reports the solutions that support design, development and operation of robotic systems based on software services using service oriented approaches. Their review highlights the needs for improving reuse, productivity and quality of service-oriented robotic systems; thus providing a catalogue of solutions to develop service-driven robotics.

Pons et al. (Pons et al., 2012) have reported another systematic review of Software Engineering Approaches for Robotics based on 67 studies that were published from 1999 to 2011. They highlight the prominent trends of software engineering techniques for robotic software. The review highlights the application of (i) component based, (ii) service oriented as well as (iii) model driven development of robotics as the emerging research trends.

2.3.2. Survey-based studies of robotic software

Zhi et al. (Zhi et al., 2013) have reported a survey-based study of Multi-Robot Coordination Systems. As shown in Table 1, their study is aimed at identifying research challenges and problems including communication mechanisms, planning strategies and decision-making structures for robotics. These problems are discussed in the context of cooperative and competitive environments in which team of robots coordinate to accomplish specific missions. A systematic classification and comparison of the problems helps to identify the dimensions of future research such as energy efficiency, decision making, and heterogeneous mobile robots.

Elkady et al. (Elkady and Sobh, 2012) have reported a survey study of Robotics Middleware by reviewing over twenty solutions such as OPRoS, MRDS, CLARAty. The survey evaluates the strengths and limitations of robotic middleware and provides a set of guidelines to select the most appropriate middleware based on the requirements of robotic systems. The study highlights some important attributes of each middleware such as simulation environment, support for distributed environment, fault detection and recovery and behavior coordination capabilities that guide developers while selecting an appropriate middleware for robotics software.

Kramer et al. (Kramer and Scheutz, 2007) have reported another survey study of Robotic Development Environments (RDEs) to highlight the importance of development environments and frameworks for robotic systems or specifically mobile robots. The survey primarily aims at analyzing the usability and impact of the major RDEs (e.g., Player, Miro, and MARIE). The survey provides guidelines to analyze the strength and limitations that help researchers/practitioners to select an appropriate RDE. A systematic comparison and evaluation of these RDEs also highlights areas of future research on the development and application of RDEs.

The surveys reported in (Elkady and Sobh, 2012, Kramer and Scheutz, 2007) suggest that the terms *middleware* and *development environments* are complementary and often interchangeable. Elkady and Sobh (Elkady and Sobh, 2012) refer to frameworks like Player, Pyro, and Miro as middleware to abstract platform and hardware specific details. On the other hand, Karmer and Scheutz (Kramer and Scheutz, 2007) describe the above-mentioned frameworks as open source RDEs for architectural development of mobile robots. It is vital to mention about (Biggs and MacDonald, 2003) that presents the state-of-the-art in *robot programming systems* with a distinction between manual and automatic programming. The survey suggests that software architectures are important to both of these programming systems, as architecture provides underlying support to model and develop robotic software. We assert that our study complements the existing body of research on architecture-centric development of robotic software. Our study specifically addresses various architectural aspects, challenges, and their

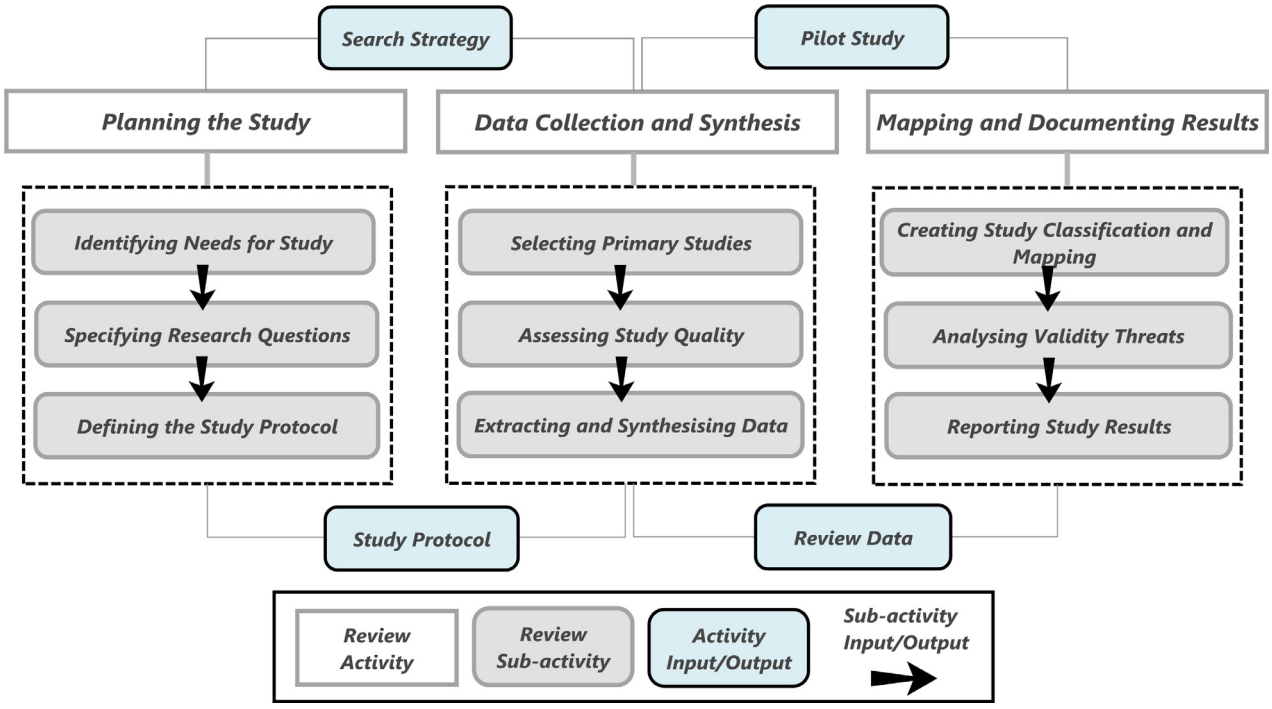

Fig. 2. An overview of the research methodology for mapping study.

solutions for robotic systems by investigating the state-of-the-art that has progressed over more than two decades (1991–2015).

3. Research methodology

We used Systematic Mapping Study (SMS) method (Petersen et al., 2008) that involves a three step process as illustrated in Fig. 2. The methodology for conducting SMS comprises of: (i) planning a study, (ii) data collection and synthesis, and (iii) mapping and documenting the results. A systematic approach for a review reduces bias in identifying, selecting, synthesizing the data and reporting results. Following sub-sections provide the details of research methodology guided by Fig. 2. The extended details of the research methodology in terms of the *needs for SMS, literature search strategies, data extraction, qualitative analysis, definition and evaluation of the SMS protocol* are provided in (Ahmad and Babar, 2016).

3.1. Specifying the research questions

To conduct the mapping study, we have formulated research questions whose answers support an objective investigation of the research. Specifically, the questions aim to investigate the following aspects.

A. **Research Themes, Architectural Solutions and Frameworks**

RQ 1: What is the publication frequency and fora of research on software architecture for robotics?
RQ 2: What are the existing research themes and how are they classified?
RQ 3: What architectural solutions are provided to support robotics software?
RQ 4: What architectural frameworks exist to support the solutions?

In these questions, first we investigate the frequency; focus and fora of published research to reflect its temporal progression. Afterwards, the classification of existing research themes helps us to illustrate and compare various software architectural solutions and frameworks for robotic systems.

B. **Modeling, Validation and Application of Architectural Solutions**

RQ 5: What architectural notations have been used to represent the solutions?
RQ 6: What validation methods have been employed to evaluate architectural solutions?
RQ 7: What were the application domains for architectural solutions?

We aim to analyse three important aspects of the architectural solutions in terms of notations to represent solutions, methods to validate these solutions and the domains (e.g., medical assistance (Kazanzides et al., 1992), home service (Mäenpää et al., 2004, Kwak et al., 2006) where architectural solutions are applied.

C. **Past, Present and Emerging Architectural Solutions**

RQ 8: What are the past and present types of architectures models for robotic software?
RQ 9: What emerging solutions can be observed as an indication of future research?

We discuss the solutions that have emerged in the past and their contribution to the development of existing solutions or active research. Finally, by analyzing the past and existing solutions, we aim to identify the emerging problems and innovative solutions as the possible dimensions of emerging and futuristic research on architecting robotic software.

3.2. Searching the relevant literature – primary studies

After specifying the RQs, we followed the steps to collect and synthesize the data from Fig. 2. In the protocol for mapping study (Ahmad and Babar, 2016), we document the steps taken for (i) *selection of primary studies*, (ii) *screening and assessment of the studies*, and (iii) *data extraction for synthesis*. Fig. 3 shows the search process used for this study.

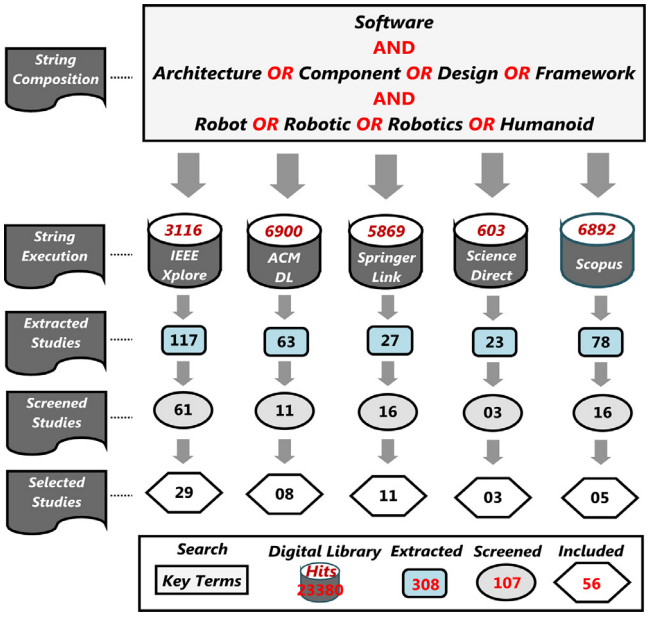

Fig. 3. Summary of the literature search process with search string.

The above-mentioned research questions helped us to identify a set of keywords that were used to build a search string that was applied to five databases as shown in Fig. 3. We limited our search to the peer-reviewed literature from years 1991 to 2015. The year 1991 was chosen as the initial search found no earlier results related to any of the research questions with 23,380 hits. In the primary search process (Ahmad and Babar, 2016), we focused on title and abstract, therefore, it resulted in a high number of studies that were not relevant, which we refined with secondary search process - limiting the extracted studies to 308 in total. In order to identify and select the primary studies to be reviewed, the search string was customized for each of the searched databases for effective search (Brereton et al., 2007) (details in (Ahmad and Babar, 2016)). The search string in Fig. 3 aims to identify the primary studies that focus on methods and techniques for architecture-driven robotics software. Based on the screening and qualitative assessment of the extracted studies, we selected 56 out of 107 studies for inclusion in this review. The studies included in the review are listed in Appendix A.

4. A taxonomical classification and mapping of the research

To analyze the state-of-the-research, we answer a number of questions (RQ 1 – RQ 4). First, we highlight the years and types of the published research (Section 4.1) as the publication frequency and fora reflects the temporal progression of the research over the years (Wohlin, 2014, Gomaa, 2000). We have identified and analyzed the predominant research themes using a well-known qualitative data analysis approach called thematic analysis (Boyatzis, 1998). This analysis has enabled us to taxonomically classify the main themes of the existing research. Fig. 5 shows the taxonomy of the existing research (Section 4.2) and guides the discussion on the results for this mapping study (Section 4.3).

4.1. Years and types of publications

Fig. 4 shows the years and types of publications reviewed to answer **RQ 1**. Fig. 4 shows the relation between the total numbers of studies (y-axis) published during individual years (x-axis) since 1991 to 2015. Each bar of Fig. 4 shows a relative distribution of the different types of publications (*Book Chapters, Journal, Conference, Workshop and Symposium Papers*). For example, the bar relative to year 2007 represents a total of 06 studies published (publication distribution as: ***Conference Papers: 03, Journal Paper: 01*** and ***Book Chapters: 02***) in that year. The initial studies (such as [S10] and [S13]) were focused on exploiting object-oriented design and framework to enable modular and reusable programming of robotic systems. Fig. 4 indicates that, no study from 1993 to 1997 has been included in our review because: (i) our literature search did not produce any relevant results, or (ii) the studies did not pass the qualitative evaluation for inclusion from those years.

From 1998 to 2003, only three studies [S8, S2, S28] were published that moved the research beyond the robotics programming to focus more on component-driven development. Since 2004–2015, there has been a noticeable increase in the number of studies on architectural solutions for robotic systems. There were 50 studies (90%) published from 2004 to 2015 that represents the research progression in the last decade.

The solutions reported in these studies mainly focus on component-based (Brugali et al., 2007) and service-driven (Oliveira et al., 2013) robotics. The research on service-driven robotic systems [S20, S27] has been mainly published from 2012 to 2014. The number of conference papers represents approximately 54% of all the publication types; the percentage of journal, symposium and workshop papers is 30% and 12% respectively. We also reviewed two book chapters [S45, S46] published in 2007. The book itself (Brugali et al., 2007) provides interesting insights about addressing various software engineering challenges for robotic systems. The analysis of frequency and growth of published research helps us to taxonomically classify the state-of-research below for further analysis and investigation of architectural solutions.

4.2. A taxonomy of the research themes

The taxonomy in Fig. 5 provides a systematic *identification, naming* and *classification* of various research themes based on the similarity or distinctions of their relative contributions to answer **RQ 2**. By analyzing some relevant studies (e.g., (Medvidovic and Taylor, 2000, Babar et al., 2004)) and following some of the guidelines from *ACM Computing Classification System*[2] and *Computing Research Repository*[3] we derived the following categories:

1. ***Generic Classification*** that highlights the role of software architecture to support the operations, evolution (post-deployment) and development (pre-deployment) phases of robotic systems. The generic classification is used to organize the results (reviewed studies) into three distinct areas. Specifically, in the context of Fig. 5 the literature is generally classified into approaches for i) operational ii) evolution specific and iii) development related issues of robotics.
2. ***Thematic Classification*** extends the generic classification by adding details based on the primary focus of research in a collection of related studies to identify and represent the recurring research themes using thematic analysis (Boyatzis, 1998). We identified three predominant themes as i) coordination (11 studies, i.e., 20% approx. of the reviewed studies), ii) adaptation and reengineering (13 studies, i.e., 23%), and iii) modeling, design and programming (32 studies, i.e., 57%) of robotics in Fig. 5.

Sub-thematic classification provides a fine-grained refinement of above-mentioned three themes with eight distinct sub-themes. Specifically, the research on architectural support to enable robotic

[2] The ACM Computing Classification System [1998 Version]: http://www.acm.org/about/class/1998/.

[3] Computing Research Repository (CoRR): http://arxiv.org/corr/home.

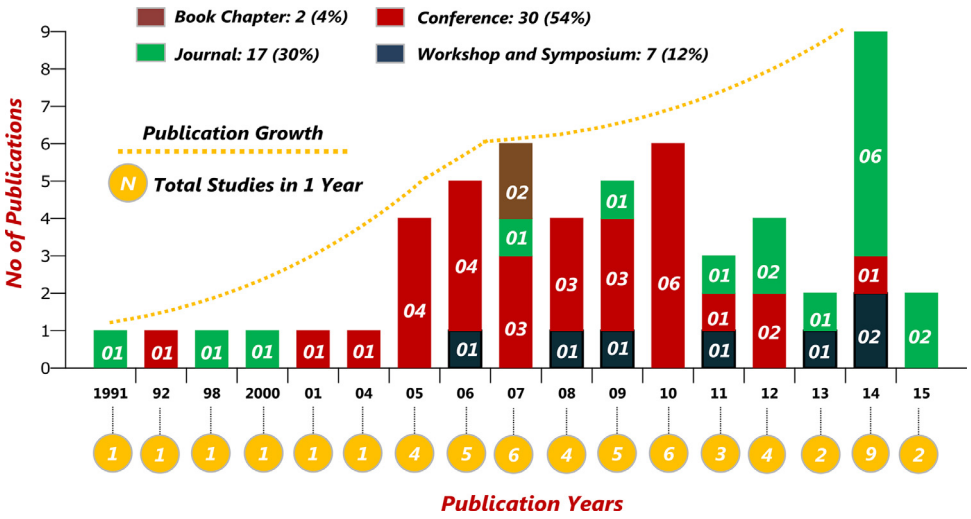

Fig. 4. Overview of the publication years and types.

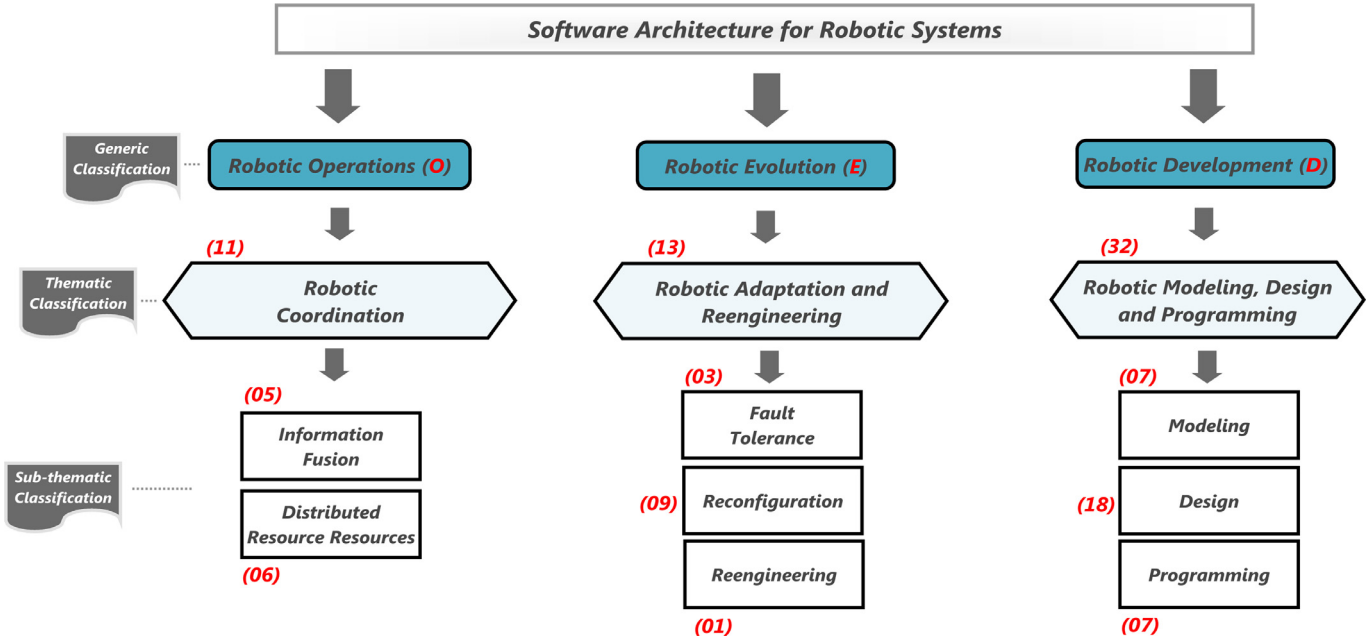

Fig. 5. A taxonomy of research themes on software architectures for robotics.

coordination can be classified into two distinct themes (i) *information fusion* (05 studies, i.e., 09% approx. of total reviewed studies) and *distributed resource access* (06 studies, i.e., 11% approx.). Robotic adaptation and reengineering solutions are decomposed into three types namely (i) fault tolerance (03 studies, i.e., 05%), ii) reconfiguration (09 studies, i.e., 16%), and iii) reengineering (01 studies, i.e., 02%). The studies on robotic development are sub-classified to support architecture-driven (i) modeling (07 studies, i.e., 12.5%), (ii) design (18 studies, i.e., 32%), and (iii) programming (07 studies, i.e., 12.5%) of robotic software. For example, in the classification scheme; the generic classification called *Evolution* (E) has a specific research theme named *Adaptation and Re-engineering* that has three sub-themes *Fault Tolerance* [S4, S8], *Reconfiguration* [S5, S7, S23][4] and *Reengineering* [S14].In contrast, supporting the fusion of information (a sub-theme) is an operational challenge (*O*) for mission critical robots that require coordination to enabling the robots for collaborative completion of a task such as search and rescues mission [S9].

Overlapping themes: In the taxonomy above, some of the studies could be classified into more than one theme or sub-theme, provided that their contributions were relevant to multiple (sub-) themes. We referred to such cases as thematic and sub-thematic overlaps. For example, the studies [S39, S42] could be classified under the themes robotic reconfiguration (i.e., evolution) or model-driven robotics (i.e., modeling for development). In all such cases, we decided to classify the overlapping studies based on their provided solutions (e.g., robotic reconfiguration) rather than the adopted technique (e.g., model-driven development) that achieves the solutions. Specifically, the studies [S39, S42] have been classified under the sub-theme reconfiguration; as the primary contribution is not to establish model-driven techniques but to exploit them to address the challenges of robotic reconfiguration.

Moreover, under the robotic adaptation and reengineering theme in Fig. 5, a number of studies had ***sub-thematic overlaps***.

[4] The notation [Sn] (n is a number) represents a reference to studies included in the review, which are listed in the **Appendix A**. The notation also maintains a distinction between the bibliography and selected literature for this study.

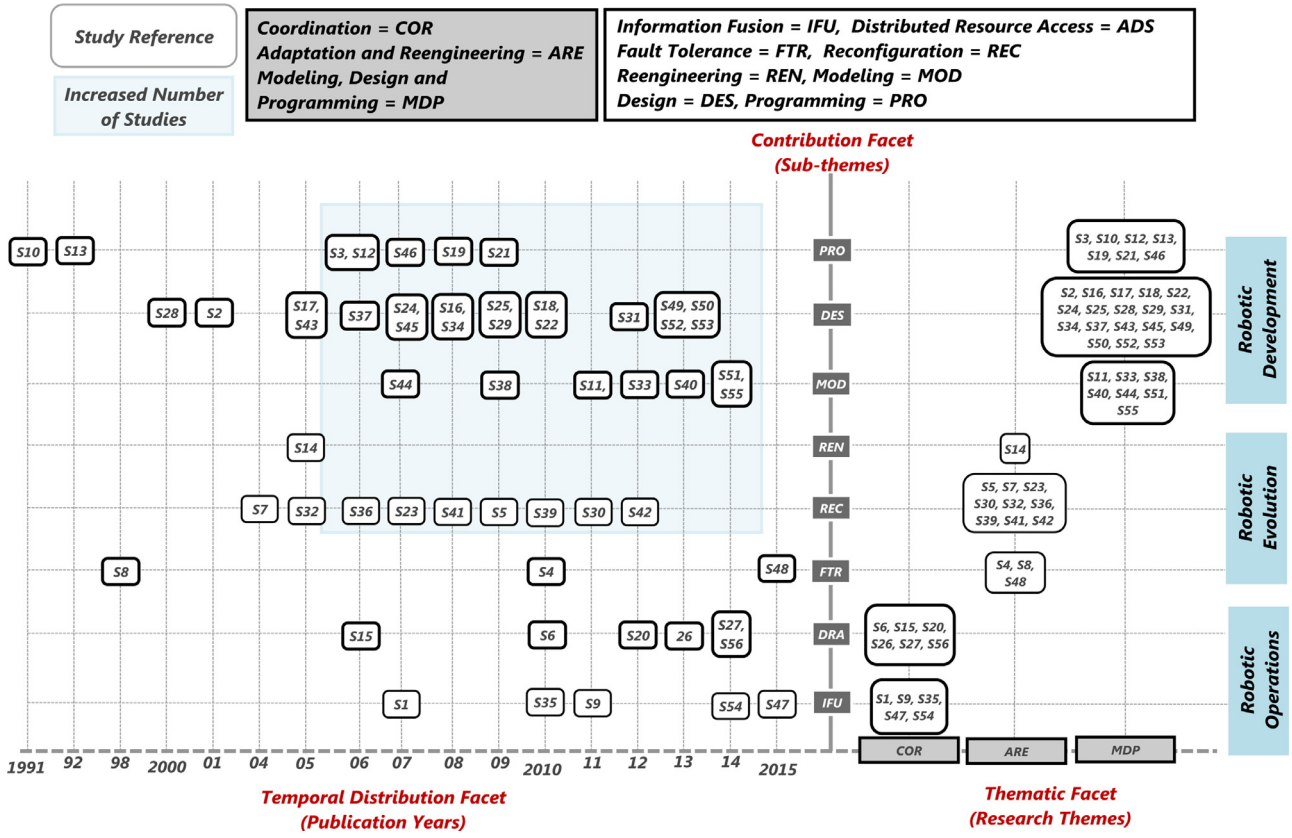

Fig. 6. A mapping of research themes, sub-themes and yearly distribution of studies.

For example, the studies (Chaimowicz et al., 2003, Malavolta et al., 2013) could be classified under the sub-themes of robotic reconfiguration or fault tolerance. However, we have classified both of the studies under fault tolerance (proposed solution) that utilizes reconfiguration strategies and policies (adopted techniques) to support self-adaptive and fault tolerant robots. Architecture-driven reengineering of robotic was also identified in only one study [S14]. We decided to classify it under robotic adaptation and reengineering to support design time evolution of a legacy robotic system.

4.3. A mapping of research themes and corresponding evidence

While the taxonomy (cf. Fig. 5) provided a broader classification of the existing research, we draw a map of the identified themes, sub-themes and their corresponding evidence (relevant studies) to improve the understanding of the results. Fig. 6 shows a three dimensional map whose details are given below.

1. Three distinct research themes (adopted from Fig. 5 – Thematic Classification) on x-axis (right). We refer to this as thematic facet of the mapping in Fig. 6.
2. Yearly distribution of the relevant studies (as temporal distribution of evidence) on x-axis (left). We refer to this as temporal distribution facet of the published studies in Fig. 6.
3. We map eight distinct sub-themes (from Fig. 5 – Sub-themes) that present the specific research contributions on y-axis. We refer to this as contribution facet of the mapping in Fig. 6.

The white rectangles on x-axis indicate that: i) the right side highlights the relevant studies as the identified evidence corresponding to individual themes and their sub-themes, ii) the left side presents the temporal distribution or publication frequency of the evidence. The interpretation of the thematic mappings in Fig. 6 is based on locating a given research theme (x-axis) that allows the identification and mapping of its sub-themes (y-axis) along with the identification of each of the relevant studies in the corresponding circle. For example, the representation of [S5] highlights that '*a specific contribution of this research (*Publication Year 2009*) is to support architecture-driven reconfiguration of robotic systems as the identified evidence about the role of software architecture to support dynamic adaptation (*i.e., *runtime evolution) of robotic systems*'.

Interpretations and usage of classification and mapping schemes - The mapping diagram in Fig. 6 can serve multiple purposes. For example, the yearly distribution highlights that '*in order to support evolution; reconfiguration of robotics is the most researched area among others with a total of 9 studies published between 2004 to 2012, while reengineering got little attention with only 1 study in 2005*. In this context, following two points need to be raised and answered.

- *What research themes have received the most and the least attention in the last 3 years*? From Fig. 6, based on a combination of the publications years and sub-themes since 2012, most of the studies have focused on enabling *distributed resource access* [S20, S26, S27, S56] for supporting robotic coordination and architectural *design* of robotic development [S33, S40, S51, S55]. There is no evidence for the research on *reconfiguration, reengineering* and *programming* for robotics.
- *What years have emerged as the most progressive based on the number of research publications and what was the focus of research during those years?* By analyzing the temporal distribution facet (x-axis left), we can identify 2006–2014 as the most progressive years in terms of total publications (i.e., 42 studies, approx. 75%). During these years, the studies focused on all of the identified themes and their sub-themes except

for the architecture-based reengineering of robotics software. The major focus of the research during these years was on architecture-driven *reconfiguration* [S32, S36], *design* [S17, S43] and *modeling* [S11, S33] of robotics software.

5. Architectural solutions and frameworks for robotic software

Based on a taxonomical classification of the existing research, we now focus on answering **RQ 3** in Section 5.1 and **RQ 4** in Section 5.2.

5.1. Problems and architectural solutions for robotic software — an overview

Table 2 serves as a catalogue of the references to problem-solution mapping by highlighting the (i) recurring *problems* along with *architectural requirements*, and (ii) *solutions* representing architecture-centric approaches to address the problems. The architecturally significant requirements (ASRs) or simply architectural requirements are highlighted as part of problem, mainly because they exhibit implicit or explicit impacts/constraints on architectures (Chen et al., 2013). We have identified the architectural requirements in Table 2. Related studies complement the view by highlighting the evidence to support the problem-solution mapping.

In Table 2, we only focus on problems with architectural requirements and architectural solutions that were identified in at least three or more studies (recurring themes), except for [S14] that represents a special case as detailed earlier. However, other related aspects such as the frameworks, modeling notations, evaluations and application of architectural solutions are discussed in the dedicated sections. The thematic classification from Fig. 5 organizes similar or distinct problems, the sub-themes represent the specific type of problems. For example, in the context of *Robotic Adaptation and Reengineering* Table 2 highlights that:

- **Problem view***: how to recon*Fig. *a robot's components (and behavior) such that it can adapt itself to operate optimally with available resources and evolving requirements of its environment*?
- **Architectural requirements**: are (i) system maintainability, and (ii) dynamic composition of software architecture to develop systems that are easily maintainable and reconfigurable at runtime.
- **Solution view**: the proposed architectural approaches to address this problem are model-driven and policy-based adaptation of robotic software.
- **Related studies***:* the related studies supporting the above mentioned solutions are [S5, S7, S23, S30, S32, S36, S39, S41, S42].

5.2. Framework support for architecture-driven solutions

We report the identified architectural frameworks and their support for robotic software to answer **RQ 4**. An architectural[5] framework facilitates a diverse set of architecting activities (e.g., modeling to development and documentation) to promote architecture-centric solutions. Based on the analysis of architectural solution (RQ 3), we present the architectural frameworks for robotic systems that appeared in at-least two studies. In Table 3, we do not include any solution specific or once-off frameworks (such as SAW - Surgical Assistance Workbench for developing surgical robots [S23]). We present the key aspect of four architectural frameworks in Table 3, while the discussion is guided based on frameworks overview in Fig. 7.

We only highlight the core features of the frameworks and their impact on robotic systems. The extended details about individual frameworks can be found in the relevant studies (highlighted in Table 3). The frameworks in Table 3 act as middleware or an abstraction layer to hide the platform and implementation specific complexities to provide an environment for the development and execution of architectural components for robots. Table 3 presents i) primary intent of the frameworks, ii) the types of activities a framework supports (based on *Generic Classification* of the identified research themes, Fig. 4), iii) source type of the framework as either open or closed source project, iv) type of systems they support and v) the related studies that have utilized these frameworks. In recent years, an increasing distribution and utilization of open source software or specifically frameworks for robotic systems [S1, S4, S31, S32] is progressing the research and practices to a model in which code is distributed, repeatedly executed and built upon. The open-source robotics software accelerates robotics development as community-wide efforts by developing, sharing and optimizing robust algorithms, algorithm comparison, and collaborations between research groups and robotic software developers (Pantofaru et al., 2013).

Fig. 7 is a collection of diagrams adopted from their respective sources [S5, S23, S31, S32] and has two distinct purposes while answering RQ 4. Specifically, Fig. 7 provides i) a visual representation of each framework in terms of individual elements and their relations that constitute a framework, and ii) illustrative examples of the operations and application of the framework. For example, in Fig. 7A) an overview of OPRoS framework highlights that framework provides an environment (acting as a container) for composition and execution of various architectural components (A, B, C). Fig. 7 and Table 3 are complementary, Table 3 highlights the key attributes and Fig. 7 explains the frameworks.

1. **Orca,** an open-source framework, implements a component model to support component-based robotics. As illustrated in Fig. 7a), the components of Orca framework are deployed across two different robots (Robot X and Robot Y). The modularity of system is maintained by encapsulating specific functionality in an appropriate component that can be utilized and reused through components coordination. Fig. 7a) illustrates that by exploiting components, Orca implements i) interfaces (component's provided and required functionality), and ii) infrastructure to enable interface binding (component messaging). The benefits of Orca components distribution are demonstrated in [S1] to support coordination between a team of robots (i.e., Fig. 5 – Taxonomical Classification).
2. **OPRoS** is also an open-source and component-based framework to develop robotic systems. The framework consists of multiple components and acts as a process in an operating system (e.g., Microsoft Windows and Linux) for component-based development and operations of a robot as in Fig. 7b). First, the composer composes a number of components (having ports and connectors) that allows the execution engine to start executing the components to support robotic operations such as sensing and navigation. As illustrated in Fig. 7b), the framework acts as a container and provides the execution, lifecycle management, configuration, and communication of various components. In the review, the studies [S4] and [S31] have utilized OPRoS that supports evolution with fault tolerant robotic [S4] as well as the development with component-based robotic [S31]. It is vital to mention that OPRoS also provides an environment for robotic simulation by utilizing i) a physical engine that simulates the robot and ii) a graphic engine to simulate the objects and obstacles for the robot. The simulation feature offered by OPRoS

[5] SEBoK Architecture Framework (glossary) "An architecture framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community. Examples of architecture frameworks: MODAF, TOGAF, Kruchten's 4+1 View Model, RM-ODP. (ISO/IEC/IEEE 2007)".

Table 2
Problem solution mapping with regards to generic and thematic classification of research.

Classification: **robotic coordination**	
Research theme: **operations of robotic software**	
Problem–solution view (sub-themes of research)	
Problem	Solution
Type: **Information Fusion**	*Approach*: **Client-Server Model**
Challenge: How to support the collection, processing and sharing of information that is fused into a team of robots to support their operations?	*Architectural Solution*: Intra-robot interaction is enabled by collecting contextual and mission-specific information from individual robots (*mission agents*) and sharing it through a central node (*mission server*).
Architectural Requirements: - ***Component scalability***: architecture must support the growing number of components (addition of robotic system or its part) in information fusion process/system.	*Requirement satisfaction*: - components as agents, through publically exposed interfaces are composed and (statically or dynamically) added to the central server.
- ***System performance***: the architectural components must enable an efficient performance in system wide communication and coordination among robots.	- communication and coordination among individual architectural components (agents) is constrained/restricted and supported only by means of a central coordinator (server)
Related studies [S1, S9, S35, S47, S54]	
Type: **Distributed Resources Access**	*Approach*: **Service Oriented Architecture**
Challenge: How to enable the access and utilization of distributed resources (hardware components, that are often virtualized) to assemble/operate a robot?	*Solution*: Exploit distributed and loosely coupled services that can be dynamically discovered and invoked. A collection of software services can be mapped to one or more hardware resources.
Architectural Requirements: ***System interoperability:*** architectural components need to facilitate the communication between modules or components that consist of heterogeneous and potentially distributed robotic sensors and actuators.	*Requirement Satisfaction*: Software services to manipulate hardware components are published and requested though a brokered architecture that enables binding between providers (software functionality) and requesters (robotic components)
Related studies [S6, S15, S20, S26, S27, S56]	
Classification: Robotic adaptation and reengineering ***Research theme: Evolution of robotic software*** **Problem**	**Solution**
Type: **Reconfiguration**	*Approach*: **Model-driven and Policy-based Adaptation**
Challenge: How to reconfigure a robot's components and behavior such that it can adapt itself to operate optimally with available resources and evolving requirements?	*Solutions* - *Model-driven adaptation*, architecture model is transformed dynamically to support transformation-driven adaptation (models@runtime) - *Policy-based approaches* directly or indirectly rely on the IBM framework for Monitoring, Planning, Analyzing and Executing (MAPE loop) to support adaptation tasks.
Architectural Requirements: - ***System maintainability:*** architectural specifications must be able to dynamically change and evolve as per the operational context of robotic system. - ***Dynamic composition***: architectural components and their connectors needs to be dynamically composed to support runtime system reconfiguration.	*Requirement Satisfaction*: - System structure, behavior and evolving requirements are expressed as a model that is transformed at runtime to support dynamically maintainable system states. - Exploiting the notion of models at runtime, architectural composition policies guide and execute that runtime composition of system.
Related Studies [S5, S7, S23, S30, S32, S36, S39, S41, S42]	
Type: **Fault Tolerance**	*Approach*: **Fault Tolerant Middleware**
Challenge: How to enable a robot to continue with its operations despite the presence of failure?	*Solutions*: A middleware layer is provided that supports a minimum of three fundamental operations as i) context monitoring, ii) fault identification, along with fault ii) recovery and minimization.
Architectural Requirements: - ***System recoverability:*** to exploit system maintainability to support architecture components that are fault-tolerant and recoverable from unexpected scenarios and failures.	*Requirement Satisfaction*: the middleware is embedded between hardware level components and the control software that continuously monitors system states and provide adaptation policies for a system to recover and restore from faults.
Related Studies [S4, S8, S48]	
Type: **Reengineering**	*Approach*: **Architectural Refactoring**
Challenge: How to support re-engineering of an existing robot to an evolved robot that better satisfy new requirements?	*Solutions*: A conventional 3-step process that requires i) recovering, ii) refactoring, and iii) optimizing the legacy architecture towards an evolved architecture
Architectural Requirements: - ***System maintainability:*** the system must be easily and efficiently maintained to accommodate the new operational requirements into existing system	*Requirement Satisfaction*: Architecture models and specifications needs to be reverse engineered (deriving models from implementation) to enable architecture level refactoring and change management.
Related Studies [S14]	
Classification: Robotic modeling, design and development ***Research theme: Development of robotic software*** **Problem**	**Solution**
Type: **Modelling**	Approach: **Model-driven Development**
Challenge: How to exploit high-level models that take a step from code-based to model-based development of robotic systems?	*Solution*: Exploits the high-level models to capture the domain requirements that are used to generate the code. Architectural components bridge the gap between the requirements and executable source code.

(*continued on next page*)

Table 2 (*continued*)

Architectural Requirements: **- *Model to Code Generation:*** the system structure (and behavior) needs to be represented as a high-level design model that must be transformed into implementation model.	*Requirement Satisfaction*: Platform Independent Models (PIMs) are developed with necessary tools and infrastructure that must exploit model transformation to achieve Platform Specific Models (PSMs). System development, refinement and evolution is achieved by means of model transformation.
Related Studies [S11, S33, S38, S40, S44, S51, S55]	
Type: **Design**	*Approach*: **Component-based Robotics**
Challenge: How to abstract the modules of code and their interconnections as architectural components and connectors to support the notion of component-based robotics?	*Solution:* architectural components are developed or reused to abstract the source-code level complexities for component-based development. These are also referred to as *COTS (Component Off-The Shelf) based robotics*.
Architectural Requirements: **- *Component integration:*** the system must be developed by means of integrating various architectural components provided by different vendors. **- *Architectural reuse:*** system needs to exploit reusable component and artifacts to support architectural components that can be reused across multiple systems	*Requirement Satisfaction*: Architectural components are developed using open specifications and cross-platform execution that enable a flexible integration and reusability of component during system development, deployment and integration.
Related Studies [S2, S16, S17, S18, S22, S24, S25, S28, S29, S31, S34, S37, S43, S45, S49, S50, S52, S53]	
Type: **Programming**	*Approach*: **Object and Module-Oriented**
Challenge: How to develop and extend the frameworks that enable reuse and modularity for robotic programming?	*Solution*: The solution provides frameworks to abstract design notation (typically object-oriented modeling) to support reusable and modular programming.
Architectural Requirements: ***Modularity:*** needs to be achieved with modules of code that must enable flexible and reusable implementation of system	*Requirement Satisfaction*: by means of modular codes and objects that could be reused across libraries of source code.
Related Studies [S3, S10, S12, S13, S18, S19, S21]	

Table 3
A summary of the identified architectural framework for robotic software.

Frameworks				
Intent	Support activities	Source type	Support for systems	Studies
Component for robotics (Orca) - Component distribution - Component reusability - System modularity	Development	Open	Robotics	[S1, S32]
Open Platform for Robotic Service (OPRoS) - Component composition - Component execution - Component life-cycle management	Development	Open	Robotics	[S4, S31]
Programming in Small and Many (PRISM) - Distributed component execution - Resource efficiency - Dynamic reconfigurability	Evolution	Closed	- Hand-held devices - Embedded systems	[S5, S30, S39]
Self-Healing, Adaptive, and Growing SoftwarE (SHAGE) - Self-healing - Acquisition and application of adaptation knowledge	Evolution	Closed	- Robotics - Self-adaptive software	[S23, S24, S36]

helps robotic software developers to not only develop but also evaluate the solution by means of simulations before deployment [S31].

3. **SHAGE** as a framework consist of two core elements i) *modules* (inner or white area in Fig. 7c)) as a collection of components supporting various functionalities and ii) *repositories* (outer or blue area in Fig. 7c)) for data acquisition and utilization. The framework supports a dynamic adaptation of architecture. The modules support various tasks such as component brokerage, reconfiguration and decision making to address the issues of *what to adapt*? In Fig. 7c), the two circles represent external elements as *Environment* and *Human* that are outside the framework but have an impact on the adaptation process. For example, the environment (e.g., navigation path) on which a robot operates imposes certain adaptation constraints (i.e., obstacle avoidance) that must be accommodated by the framework. Therefore, SHAGE also enables a communication with the robotic environment and user elements to decide on *what and when to adapt*? In the review, the studies [S23, S24, S36] exploit the SHAGE framework. Two studies [S23, S36] are specifically focused on reconfiguration; while one study [S36] supports the development of an adaptive robot.
4. **PRISM** or sometimes also referred as Prism-MW (Prism-Middleware - implemented in Java and C++) is a framework that provides middleware support for an efficient implementation, deployment, and execution of architectural elements. A layered overview of PRISM is presented in Fig. 7d) that abstracts the hardware. We discuss the architectural middleware and advanced services (top two) layers that are more relevant to robotic software than the underlying layers. Fig. 7d) presents the view where architectural middleware layer abstracts the operating system and virtual machine details to provide an environment for the utilization of architectural elements including components, connectors, events, and ports. Advanced services layers support the deployment, monitoring, adaptation and other functionalities for architectural elements. Unlike Orca and OPRoS that are robot specific platform, PRISM enables the development of hand-held and embedded systems thus making

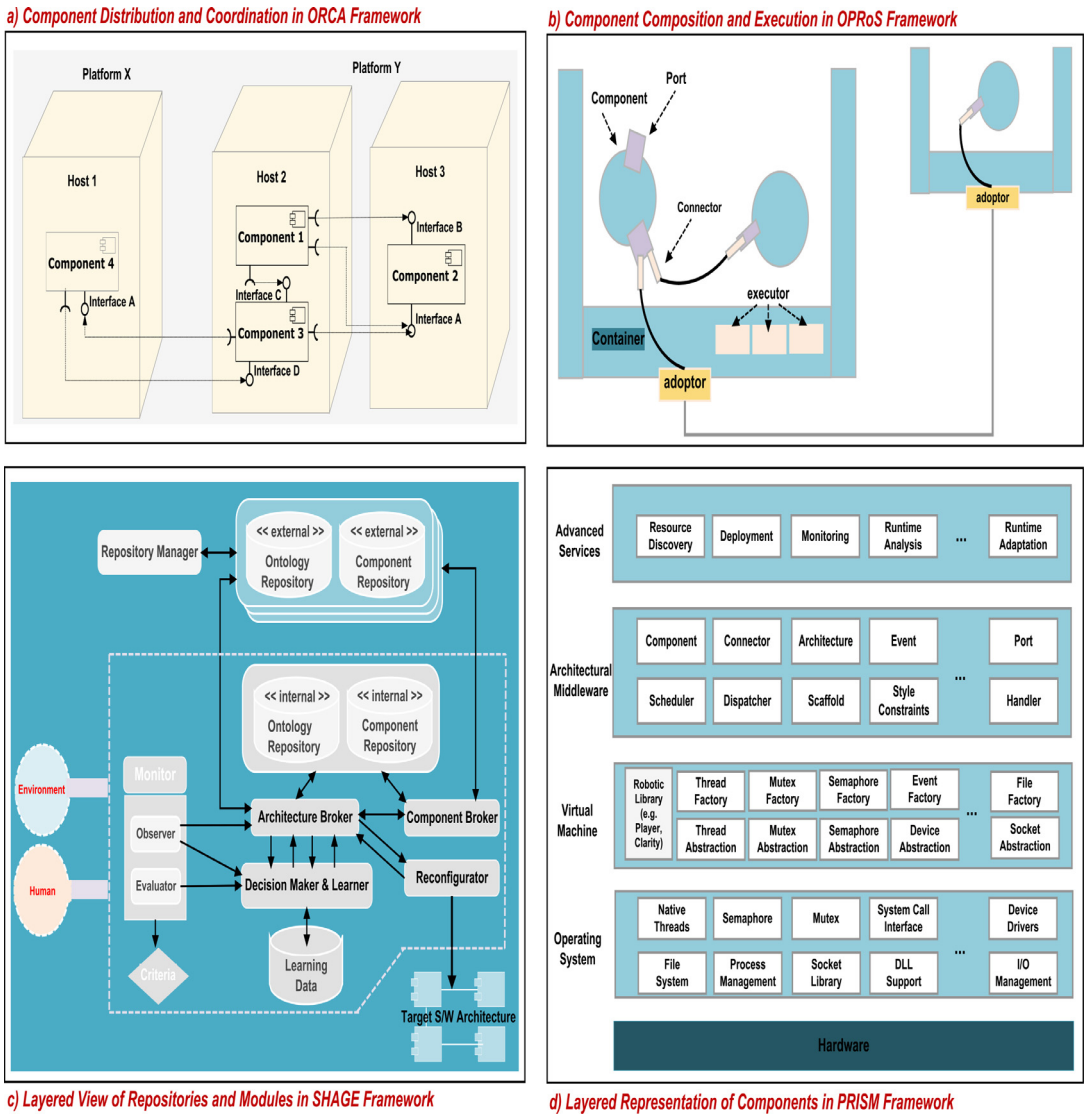

Fig. 7. An overview of the various architectural frameworks for robotic systems.

robotics as one specific domain that can utilize this framework. We identified three studies [S5, S30, S39] that utilize PRISM to support runtime evolution by means of dynamic reconfiguration of a robot. Two studies [S30, S39] utilize PLASMA [S39] framework that offers a plan-based adaptation of architectures but PLASMA itself extends PRISM to support adaptation plans on architectural component and connectors.

It is important to mention a couple of frameworks that are not included in Table 3 but were found in the reviewed studies. These frameworks are CLARAty, MRDS and ROS. CLARAty, Coupled Layer Architecture for Robotic Autonomy (CLARAty) [S2] has been developed at Jet Propulsion Lab for improving the modularity of software by NASA to support robotic autonomy for space exploration. Microsoft's Robotics Developer Studio (MRDS) is a platform-dependent environment for robotic control and simulation. One study [S9] utilizes MRDS's runtime support for services orchestration for developing service-driven robots.

Robot Operating System (ROS) [S27] is not an operating system in a traditional sense; rather it is an open source platform/framework that provides a structured communications layer/middleware above the operating systems to support the development and operations of robotic systems. In contrast to CLARAty [S2] and MRDS [S9], ROS represents the recent - community-driven efforts – for open-source and collaborative development of robotic software (Quigley et al., 2009, Pantofaru et al., 2013).

6. Notations, validations and domains of architectural solutions

After classification of architectural solutions, we now discuss the various aspects including *architectural notations*, *validation methods* and *application domains* of the solutions. We aim to investigate **RQ 5** in Section 6.1, **RQ 6** in Section 6.2 and **RQ 7** in Section 6.3.

6.1. Architectural notations for robotic systems

One of the primary benefits of architecture-centric development of a robotic system is related to abstracting the complex implementation specific details with architecture-driven modeling of software (Oliveira et al., 2013, Brooks et al., 2006). Architectural modeling provides a global view of software (robotic components

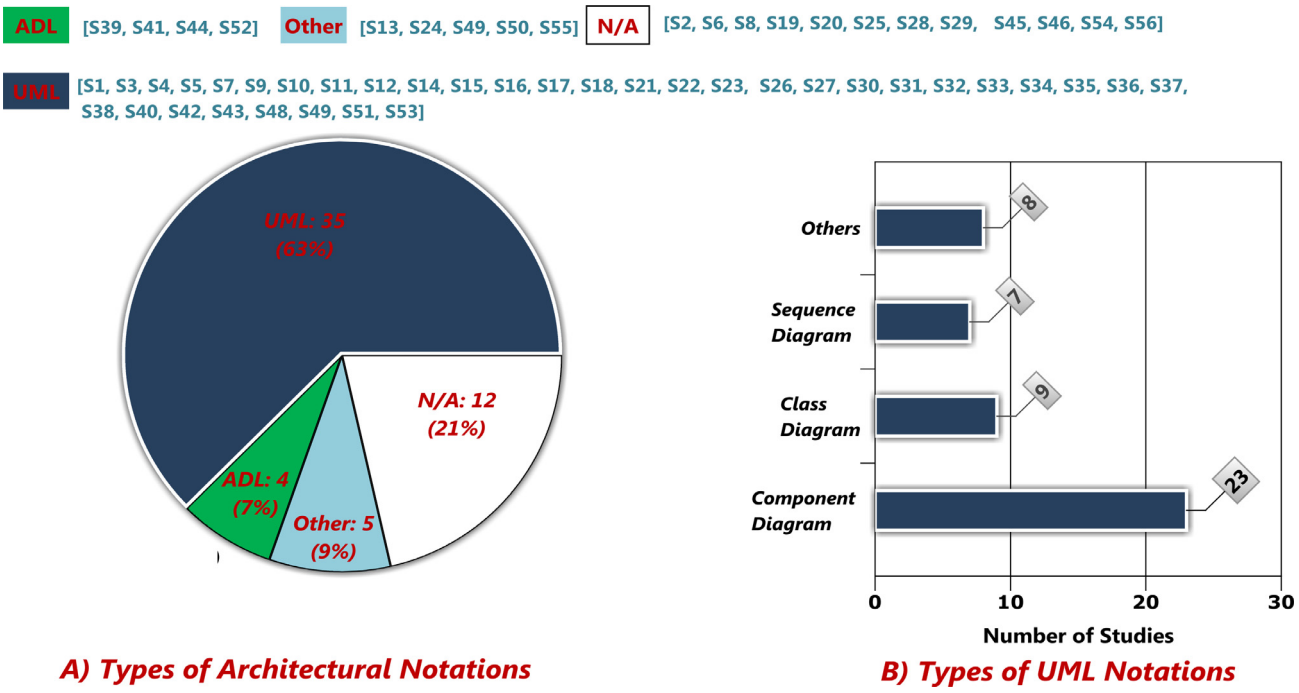

Fig. 8. Overview of architectural notations for modelling robotic software.

and their connectors) (Brugali et al., 2007) by abstracting lower-level details (such as manipulator and controller level code) (Miller and Lennox, 1991). We briefly discuss the reported architectural notations utilized for robotic software to answer **RQ 5**. The term notation refers to models/descriptions for architectural representation such as Unified Modelling Language (UML) [S23], Architecture Description Language (ADL) [S41] or similar models such as ontology based representation [S24, S55] as illustrated by Fig. 8. For example, Fig. 8A) [S41] indicates the application of an ADL for architecture representation. Fig. 8 also provides two-level details: (A) an overview of all the identified architectural notations and (B) UML-specific notations as they represent an overwhelming majority with 63% of the entire notation (identified in 35 studies).

UML-based notations have been used in the majority of the research solutions reviewed in this study. UML is an intuitive and widely adopted graphical notation for architecture modelling that might have been found easy to understand and use by robotic researchers. Another advantage of UML being industry standard for robotic development (Gomaa, 2000). In this context, the study [S24] demonstrates the feasibility of UML modeling to develop a home-service robot named T-Rot. We provide an overview of the specific UML notation (identified in at-least 3 or more studies) in Fig. 8B). Fig. 8A) also shows that some of the studies like [S34, S35, S37] utilized multiple UML notations (e.g., state-chart, use case and sequence diagram) and each notation from respective study is counted once to derive Fig. 8B). For example, the study [S37] utilizes a number of UML notations to model both the structure and behavior of the robots including requirements modeling (*use case diagram*), component interaction (*collaboration diagram*) and their execution (*sequence diagrams*). The component diagrams (in 23 studies) and class diagrams (in 9 studies) are prominent choices and fundamental notations for modelling robotic components that represent an overall structure of a robot system. In comparison, for behavioral modelling - component interactions such message passing between actuators and sensors [S35, S36], sequence diagrams are utilized (in 7 studies). Moreover, the categorization named "others" represents occasional notations such as activity (Groover, 2007), state [S7] and deployment diagrams [S9] used in specific and solution dependent scenarios for robotics software.

ADL-based notations have been used in 4 (7%) of the reviewed studies that exploit ADLs for dynamic adaptation [S39, S41] and aspect-oriented development of robotics [S44]. One study [S39] specifically extends C2SADEL with event-based architecture style for plan-based adaptation of robotics using PLASMA framework. Another study [S41] extends the xADL 2.0 schema for a policy-based adaptation of robotics. An interesting discussion about the practical implications and limitations of plan versus policy-based adaptation of robotics is provided in [S39]. The intent of both studies [S39, S41] is similar, i.e., to enable architecture-level adaptation (runtime evolution) of robotics. However, the solution specific needs (*plan* versus *policy* based adaptation) determine the needs for selection of an appropriate ADL. The review suggests the preference for UML models compared with ADLs is determined by its ease of use both in the context of academic research and industrial solutions (Gomaa, 2000). A recent industrial survey on architectural languages (Malavolta et al., 2013) also highlights that academic ADLs seem not to fulfill the industrial requirements. However, the survey also suggest that academic research and development of ADLs can provide some inspirations to develop and enhance industry specific ADLs.

Other notations are used in 5 (9% of) studies for a variety of purposes and solution-specific needs. For example, [S13] utilizes the module diagram to represent an object-oriented architecture for a surgical robot, named ROBODOC. The preference for modular representation is determined by a mapping of (design elements) modules and their interactions to (source code elements) objects and their relations. UML was not available at the time as the study was published in 1991. Park et al. [S24] reported an ontology-based modeling of architecture for the actions of a robot. The ontology-based modeling facilitated the structural and semantic matching and searching of appropriate actions on architecture models of robotic software.

No explicit notations have been presented in 12 (21% of) studies as illustrated in Fig. 8A). For example, one study [S6] provides

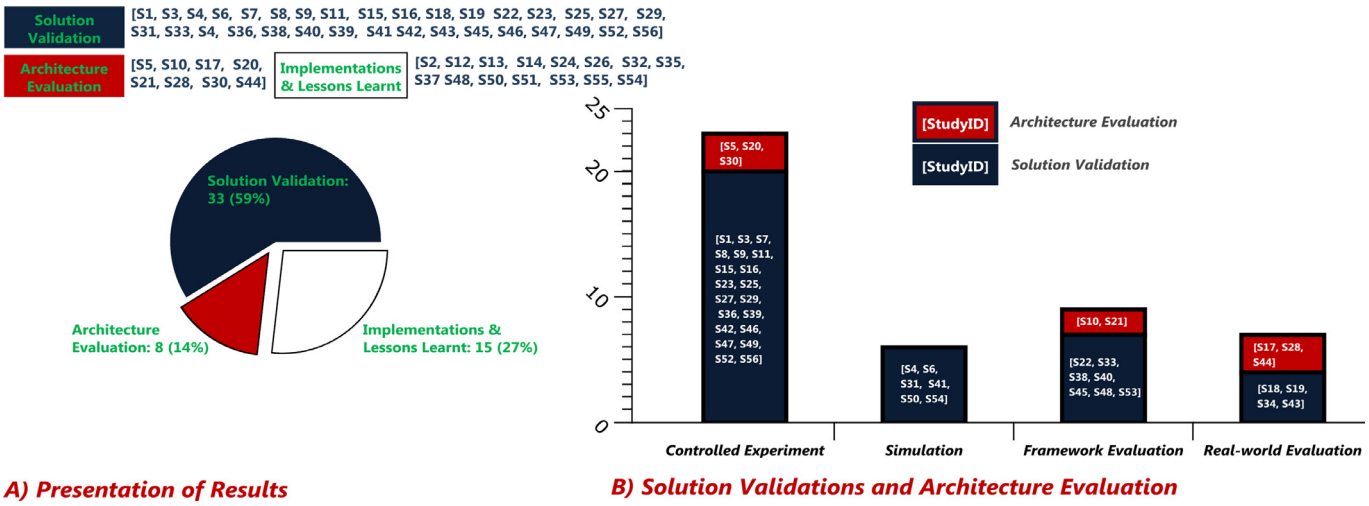

Fig. 9. Overview of solution validation and architectural evaluations.

a generic representation of SOA style (based on service provider, publisher and requester); however, no explicit details about modeling individual services have been provided. Another study [S19] utilizes the design patterns (black-board and message-broker) for architectural design of mobile robots, however, the study uses only generic boxes and arrows for architectural descriptions, rather than a specific notation such as the component diagrams.

These findings indicate that robotic research on architectural solutions heavily relies on UML models. The 4 studies that used ADLs have been reported by software architecture researchers, who are expected to have more knowledge about ADLs. We could not find any explicit combination of different notations such as UML and ADL to represent the architecture. In two of the studies [S39, S44], architecture is modeled using the architecture description languages; however, for illustrative reasons the components specified in the ADL are modeled using the UML component diagram. A possible reason for a lack of combined notations could be that non-software engineering researchers are more inclined towards easy to use and widely practical UML models. Other notations such as ADLs in comparison to UML are less popular and difficult to understand and utilize. In this context, a study (Malavolta et al., 2013) suggests that ADLs should have features to support communication among different types of stakeholders about individual activities of architecting process. A lack of flexibility and customization of the ADLs originated from academic research hinders other modeling notations to be combined or incorporated with ADLs. UML models despite their simplicity offer diverse modeling notations (structural, behavioral, and interaction diagrams) to address (i) different activities such as structural development, behavioral adaptation and (ii) accommodate more stakeholders such as robotics/artificial intelligence/industrial engineering and automation researchers, developers and other practitioners.

6.2. Solution validation and architectural evaluation techniques

We discuss how the architectural solutions have been evaluated to highlight the relative strength and validity of the evidence to answer **RQ 6**. We distinguish between solution validation and architecture evaluation as validating the overall solution aims to analyze the ability of a robotic (software) system to address the identified problems (ISO 2011, Adrion et al., 1982). In contrast, *architectural evaluation specifically focuses on the ability of an architecture (as a part of overall solution) to satisfy functional and non-functional requirements* (Babar et al., 2004). In Fig. 9A), first we highlight how the validation results have been presented that follows a discussion of the solution validation and architectural evaluations in Fig. 9B). Fig. 9A) illustrates three distinct types as: (i) validation of the overall solutions (59%), (ii) architecture-specific evaluation (14%), and (iii) implementations and lessons learnt (27%). Implementations and lessons learnt refers to implementing a preliminary solution (prototype or proof-of-concepts) and report the lessons learnt to guide future developments. These have been represented in a total of 15 studies as illustrated in Fig. 9A). For example, a study [S25] presents the lessons learnt in the development and preliminary evaluation of Service Oriented Robotic Architecture (SORA) for space exploration mission. The study highlights the benefits (e.g., component distribution and reusability) and their associated challenges (middleware requirement) of exploiting the concepts of service-orientation or service-oriented robotics (Oliveira et al., 2013, Cepeda et al., 2011) for mission critical tasks. Other lessons include the feasibility of programming models and components to support development, or evolution (reengineering [S14] and adaptation [S24]) of robotic systems.

We have identified four distinct validation methods that explicitly discuss architecture-specific evaluations. Fig. 9A) provides a percentage comparison of *solution validations* versus *architectural* evaluations. Fig. 9B) focuses on solution validation versus architectural evaluations in the reviewed studies.

Controlled experimentation *methods are focused on laboratory-testing to validate or experiment with the (partial) solutions in controlled environments*. Controlled experiments have been used in a majority of the studies (23 in total, 41% approx.). For example, one study [S39] has used controlled experiment to validate the adaptation of navigational robots (e.g., single robot [S1] and team of robots [S5]) on a predefined or fixed path. Early experimentation is fundamental to establish benchmarks and to seek preliminary feedback for solution refinements or further commercialization of robotic solutions (Thrun, 2010, Wurman et al., 2007).

Architecture-specific evaluations have been reported in a limited number of studies. One study [S5] evaluates the overall solution to support robotic navigation and then also compares and evaluates the proposed PRISM-MW with two reference architectures (i) PBAAM [S26] and (ii) layered adaptive architecture (Sykes et al., 2008). The studies [S20, S30] use empirical evaluation of the architectural solutions. Specifically, the study [S20] evaluates a novel cloud-based architecture by analyzing the energy

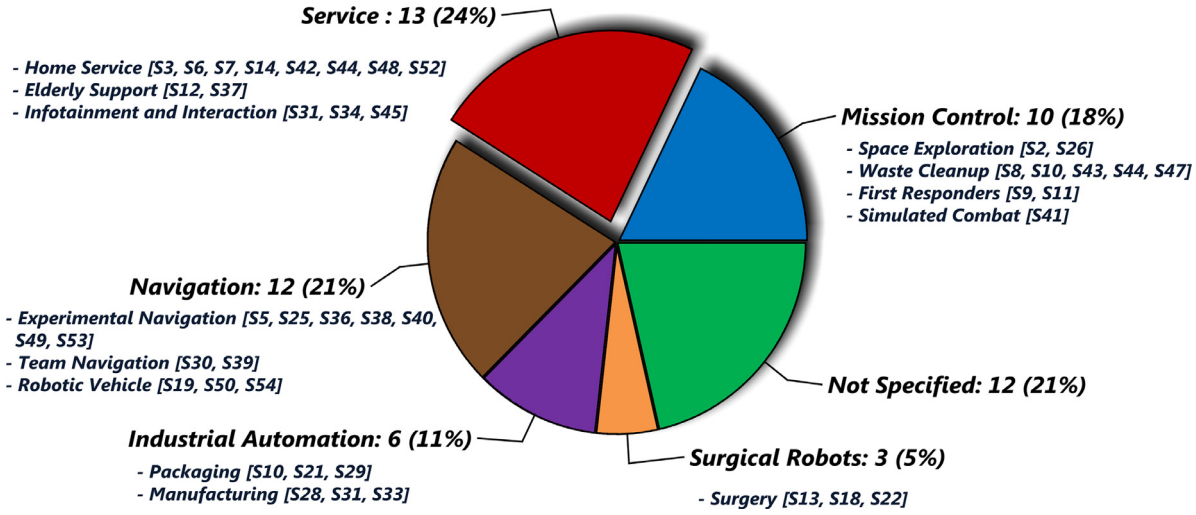

Fig. 10. Study distribution by application domain of robotics.

consumption and computational off-loading of robotics architecture by highlighting the feasibility of cloud robotics in resource constrained environment. In comparison, [S30] utilizes a conventional metric called Constructive Cost Model (COCOMO) for analyzing architectural support to develop solution with reduced efforts from months to weeks (approximate reduction from 4 – 9 months to 4 weeks).

Simulation *refers to establishing and exploiting a virtual environment (as closely mapped to real world context, as possible) to evaluate the functionality of the solution.* Simulations do not reflect all the possible scenarios that can be encountered in a real world, however, it projects many of the same characteristics as multiple sources of input to decide which actions to perform for robots (6 studies, 11% approx. of studies in Fig. 9B)). The simulation-based validations are conducted with two of the well-known frameworks OPRoS (Open Platform for Robotic Services also highlighted in Section 5.2, cf. RQ 4) and ROBOCODE that simulate robotics navigation. The OPRoS consists of a physical engine to simulate the robot and a graphics engine to simulate the objects and obstacles in the solution [S4]. In comparison, a specific example of ROBOCODE is the simulation of the battle-field to analyze the adaptation of robotics [S41]. We did not find any evidence of simulation-based methods to specifically evaluate the architecture of robotics software.

Framework evaluation *aims at evaluating the capability or effectiveness of newly developed or already existing frameworks that support development of robotic systems.* These frameworks are mostly evaluated to support the reusability and modularity (e.g., through programming [S10], componentization [S22] or modeling [S33]) to support architecture-based development of robotic systems. These represent a total of 9 (16%) studies. Architecture-specific Evaluations are supported such as evaluating Robot Independent Programming Language (RIPL) [S10] to support development and reusability of components for programming robotic systems. The study reported in [S21] evaluates the application of patterns to support component-based robotic systems (Jung et al., 2010, Brugali et al., 2007).

Real-world validations evaluate the solutions in a real world with realistic scenarios and environments. The validations are in the context of commercial robots that support industrial automation [S17, S28], surgical procedures [S18], and autonomous vehicle [S19, S43, S44]. Architecture-specific evaluations are only reflected in a limited number of studies focused on the degree of reusability of various architectural components [S28, S44] and evaluating the real-time execution of the components [S17].

A majority (33, 59%) of the studies focus on validating the overall software solution, while neglecting architectural evaluation. A lack of architecture-specific evaluations (14% studies) can be interpreted as a weakness in the evidence for the effectiveness of architectural solutions for robotic systems. In the context of Fig. 9B), a lack of architecture specific evaluations hinders the validation of non-functional or quality requirements (Babar et al., 2004). The work in (Orebäck and Christensen, 2003) can be viewed as an inspiration for the future research that aims to evaluate the architectures of mobile robots against the attributes such as portability, ease of use, software characteristics, programming and run-time efficiency to qualitatively validate a robotic system.

6.3. Application domains

We also identified the domains to which the architectural solutions had been applied (Schofield, 1999, Jackson and Coll, 2008) to answer **RQ 7**. The domain or application context of a robotic system defines*: the environment (real world or a simulations) in which a system operates under some specified conditions (to satisfy some requirements)* (Brugali et al., 2007, Redwine and Riddle, 1985). A typical example of such a domain is mission control in which robot(s) must support mission critical operations such as space exploration [S2, S26] or resolving some emergency scenarios [S9]. Fig. 10 presents a relative distribution of the types of domains to which architectural solutions have been applied.

Mission control robotics *represent a type of mission critical systems* (Cheng et al., 2009) *that are developed to accomplish a specific mission identified in a total of 10 studies (18%).* For example, the study [S9] represents (first responder emergency scenario) as type of a safety critical system, while [S13] (surgical robot) operate in a domain with health and safety criticality. The reviewed studies [S2, S26] suggest as early initiatives by NASA to develop robotic systems for space exploration. The space exploration robots represent earlier (prototype-oriented) solutions that require an appropriate human intervention or supervision to carry out their mission. We interpret Fig. 10 as: mission critical robots are discussed in 10 (19% of studies), and mission/system criticality (Hinchey and Coyle, 2010) are classified as: i) *space exploration* [S2, S26], ii) *hazardous waste cleanup* [S8, S10, S43, S44, S47], iii) *first responders* [S9, S11], and iv) *simulated combat* [S41] robotics. Early ex-

perimentation and lessons learnt [S26] provide the guidelines and architectural needs to develop futuristic robots that assist NASA with their space exploration program[6] (Katz and Some, 2003).

Service robotics *are the type of robots that replace humans' services (such as cleaning [S42] and home service [S14]) for common domestic tasks.* These robotic systems represent a relative majority of the reviewed literature (13 studies, 24%). Recently, several domestic service robots are increasingly considered as consumer products with the primary aim to support and increase the quality of life in many areas (Forlizzi and DiSalvo, 2006).

This has also prompted a number of renowned consumer product companies (Sony (Mäenpää et al., 2004), Honda (Parker, 1998), and Samsung [S14]) to invest in research and development of service robots. The home service robots range from experimental solutions (academic research) with path navigation and object pickup capabilities [S3, S6] to commercial solutions offering drink serving [S7, S42], clean-up [S42] and home surveillance solutions by Samsung robotics [S14].

Navigation robotics *also known as mobile robots that represent a class of robots that move autonomously from its current (source location) position towards a goal position (target location), while avoiding situations such as collisions and obstacles.* Most of the identified studies have focused on experimental navigation with a robot tracking certain objects or patterns of unknown paths [S25, S38]. A concrete implementation of robotic navigation is in home service robots where navigation is fundamental requirement to provide the services such as home clean-up [S42]. Another experimental case is leader-follower navigation by robots [S5, S30, S39] where instead of a single robot navigation, the follower must keep track of their leader and navigate accordingly to collect and share the environmental and contextual information. A specific example of navigational robotic is the autonomous vehicle [S19, S50] that can drive itself on a given path by following traffic lights and obstacle avoidance. This concept has gained a lot of attention and finding its application in various dimensions, specifically for the construction of military and commercial automobiles [35, S50, S54].

Industrial automation robotics *or industrial robotics (based on ISO 8373 standardization) aim to automate complex industrial tasks such as assembling, painting, testing and packaging of industrial products effectively and efficiently* (*Groover, 2007*). The studies [S21, S29] represent experimental solutions. A study in [S10] represents a real-world robotic application for holding/un-holding and packaging the bulk of industrial material. The studies [S28, S23] automate the assembling of the industrial products.

Surgical or medical robots *support the notion of minimally invasive surgery (MIS) where a robot plays the role of a surgeon or assists a surgeon to carry out the surgical procedures* (Taylor and Stoianovici, 2003). We found three studies with architectural support for surgical robots. Two studies [S13, S18] represent a robotic system to perform the planning, execution of surgery with manual takeover if required. One solution [S22] allows integration of various components (sensors and imaging devices) that help a surgical system to function.

Our findings enable us to conclude that the growing complexity of modern and practical robots increase the significance and importance of good software architecture design for reliably and efficiently developing and evolving commercial robotic systems (Thrun, 2010). The state-of-the-research represents early solutions or prototypes that may have a significant impact in the near future on robotic driven world such as robotic surgeons [S13] or unmanned [S54] and self-driving vehicles [S50]. Moreover, the mission critical robotics are emerging as human alternate to execute the trivial and complex tasks such as hazardous waste cleanup [S8], rescue mission [S9] and space exploration [S2].

[6] Robotics | NASA: http://www.nasa.gov/education/robotics/#.U8O_9_mSzyl.

7. Discussion of research progression and emerging solutions

Based on the taxonomical classification and solution information from the reviewed research, we present our analysis of the research progression, types of architectures models and emerging solutions. We specifically aim to answer the sub-questions **RQ 8** in Section 7.1 and 7.2 and answer **RQ 9** in Section 7.3.

7.1. Research progression for software architecture-driven robotics

The prominent solutions of architecture-driven robotics research during (1999 – 2015) are shown in Fig. 11. The research progress has been measured in terms of the advancement of solutions that have emerged overtime. Specifically, the prominent solutions represent frequent research in terms of identified research theme(s) (cf. Section 4, Fig. 5). For example, Fig. 11 highlights that *programming models* or *frameworks* are solutions to enable the *modeling, development and programming* of robotics using *OO-R* techniques. Fig. 11 illustrates a transition of research in the context of OO-R, CB-R and SD-R that are referred to as three distinct architectural models or architectural generations for robotic software that have emerged and matured with more than two decades of research. For example, in Fig. 11 OO-R architecture model represents a progression of research that enabled object-orientation support for robotics development with studies published between 1991 to 2010.

In Fig. 11 an interesting observation is that the findings of the mapping study - progress and maturation of architectural solutions for robotic software - are consistent with the results of software technology maturation (Redwine and Riddle, 1985). Specifically, in (Redwine and Riddle, 1985) a review of the growth and propagation of software technologies reveals that since its inception a software technology typically takes 15 to 20 years for its wide-spread adoption and popularization. Fig. 11 shows that early research on architectural solutions started in 1990s, however, the maturation and concentration of the state-of-the-research is only evident in the last decade (approximately 20 years later). We do not refer to or discuss the details of individual studies/solutions (as they represent specific cases). Instead we focus on higher-level themes. Based on the publication types and years (cf. RQ 1, Fig. 4), we have identified three phases of the research progress. Considering Fig. 11, no study from 1992 to 1998 has been included in our review and the reasons have been stated earlier (cf. Section 4). In Fig. 11, a total of 45 studies could be classified under the OO-R, CB-R and SD-R types of architecture models. Moreover, [S14] is considered as an exception that is not associated to any of the three architecture types but it represents architecture-based reengineering of robotic software. The remaining studies could not be associated with any of three architecture models. For example, [S20] discusses cloud-based robots (only implicitly referencing to software services for developing cloud-robots).

– ***Foundation phase (1991–2004)*** refers to the years of the earliest research on utilizing architectural models that primarily included the object-oriented modelling to enable the programming for robotic software in the context of OO-R. As illustrated in Fig. 11, although there is less progress during this phase (number of studies and research themes), however, during this era of 14 years the concepts of architecture-driven robotics have been established. Specifically, architectural or design modeling is considered as the driver to develop reusable and modular programs. We have only identified 6 relevant studies (i.e., 11% approximately of the reviewed papers). After almost a

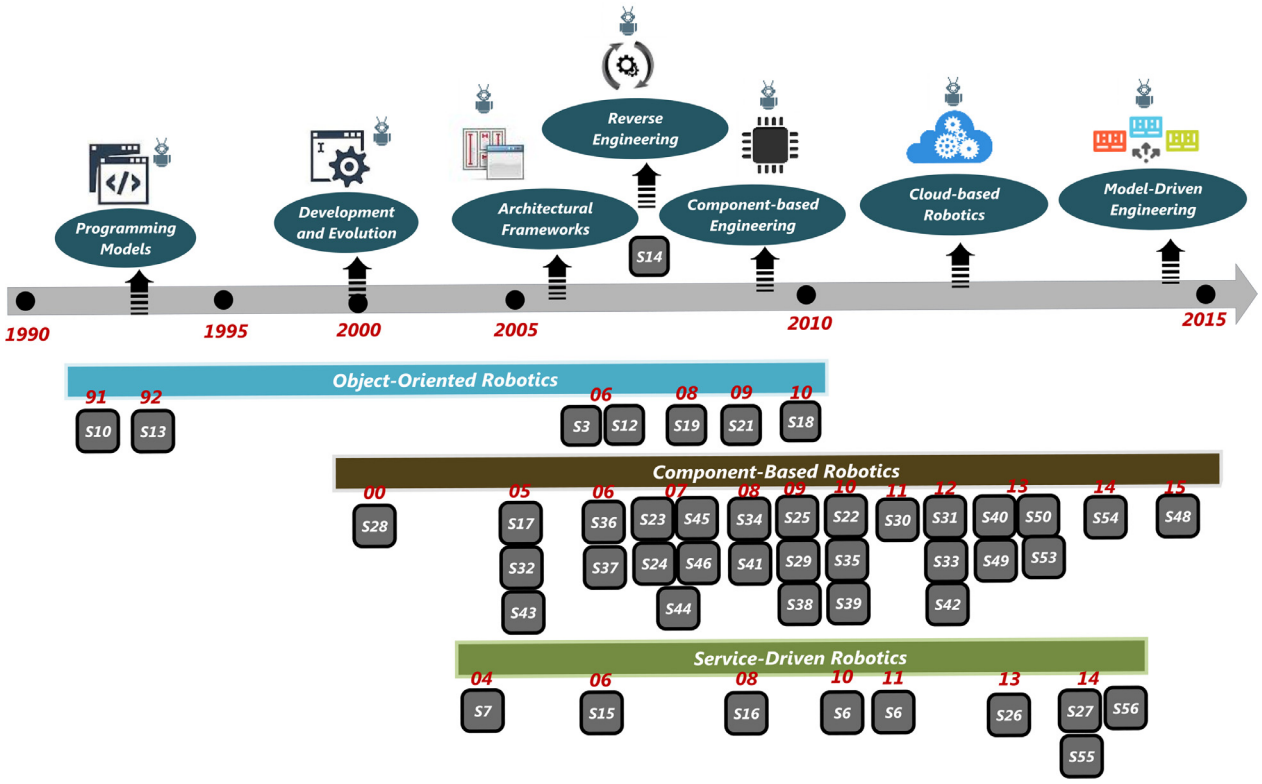

Fig. 11. Overview of research progression and types of architecture models.

decade through this phase during the early 2000, the research started to move beyond code-driven techniques to utilize the notion of component-based software engineering and specifically component-based robots to develop and evolve robotic software.

- ***Maturation phase (2005–2010)*** represents the consolidation and maturation of earlier solutions and the research progress aimed at establishing software architecture as an important artifact of robotic software development cycle. During this phase, a total of 30 studies (53%) have been identified that proposed innovative theory and solutions (such as modeling notations, and architectural frameworks) to extend or leverage the existing methods (robotics programming). One of the key advancements in this phase can be attributed to the application of component-based engineering techniques and specifically CB-R as a solution for reusability with the notion of Commercially off-the-shelf (COTS) components. During this phase architectural modelling notations such as UML and ADLs (cf. Fig. 8) became popular. The researchers also proposed or utilized various architectural frameworks (Fig. 7) to support architecture-based development and evolution of robotic software.
- ***Active phase (2011–2015)*** represents the recent progress in the area mostly based on CB-R and SD-R solutions. During this phase, the concept of service-orientation for robotic systems (Oliveira et al., 2013) has been extended to put forward the notion of cloud robotics [S20]. During this phase, a total of 20 studies (36%) have been published. The component-based techniques have been utilized to enable model-driven robotics. These research phases have also been discussed later in detail as the possible dimensions of future research on architecting robotic systems.

7.2. Types of architectural models for robotic systems

The types of architectural models namely OO-R, CB-R and SD-R have emerged and matured overtime, and they refer to different architectural representations and principle that guide architectural solutions. The discussion on this topic has been influenced by the findings about the present, past and future of software architecture (Kruchten et al., 2006). We specifically highlight various architectural models for robotics software based on the research progress in the context of each of the three models. In the context of architectural solution (cf. Section 5), we discuss architectural models for robotics based on the commonality or distinction of the problems addressed and the solutions provided. We have identified three distinct models: Object-oriented, Component-based and Service-driven robotics as presented in Table 4. Table 4 highlights the primary *challenges* these models aim to address, the *activity* they support, summary of the *solutions* they offered and the *relevant studies* associated to them.

A. Object-Oriented Robotics (OO-R) Architecture Model

Overview - OO-R is the first of the architecture models that exploits the principle of object-orientation to provide a modular and reusable objects and frameworks to support the programming of robotics – i.e., support for development activities. OO-R represents some of the earliest evidence (published in early 90s, 1991 [S10] and 1992 [S13]) on design and architectural support for robotics development. In terms of representation, the view of the OO-R has been provided in Fig. 12A). The view represents a generic and consolidated representation based on the analysis of architectural solutions earlier, where individual solutions may have more solution-specific details. The overall representation of the OO-R model consists of three distinct elements namely: *Core, Generic* and *Specific* along with two types of interconnections of elements named: *uses* and *inherits* (common OO relations). The hierarchy specifies that the generic components must have core components, while specific components must inherit from generic components. The hierarchal structure for OO-R cannot be altered, for example, specific components must rely on a mediator (generic components) to access the core. An upside view of Fig. 12A), OO-R is similar to the layered architecture (Avgeriou and Zdun, 2005), where each layer

Table 4
Overview of the architectural models for robotic systems.

Challenges	Model		
	Objected-oriented robotics (OO-R)	Component-based robotics (CB-R)	Service-driven robotics (SD-R)
Reduced complexity	√	√	√
Increased modularity	√	√	√
Software reusability	√	√	√
Model-based development		√	√
Software distribution			√
Software composition		√	√
activity			
Robotics development	√	√	√
Robotics operations		√	√
Robotics evolution		√	
Solution	*Objects to model and encapsulate reusable source-code for robotics.*	*Off-the-shelf architectural components to model and develop robotics.*	*Loosely-coupled, dynamically comoposable services for robotics.*
Studies	S3, S10, S12, S13, S18, S19, S21	S17, S22, S23, S24, S25, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S45, S46, S48, S49, S50, S53, S54	S6, S7, S9, S15, S16, S26, S27, S55, S56

The studies [S14, S20] could not be associated with any of three trends as: [S14] is an exception as a single study on architecture-based re-engineering that does not relate to any of the trends.
[S20] discusses cloud-based robots (with only implicit reference to software services for developing cloud-robots).

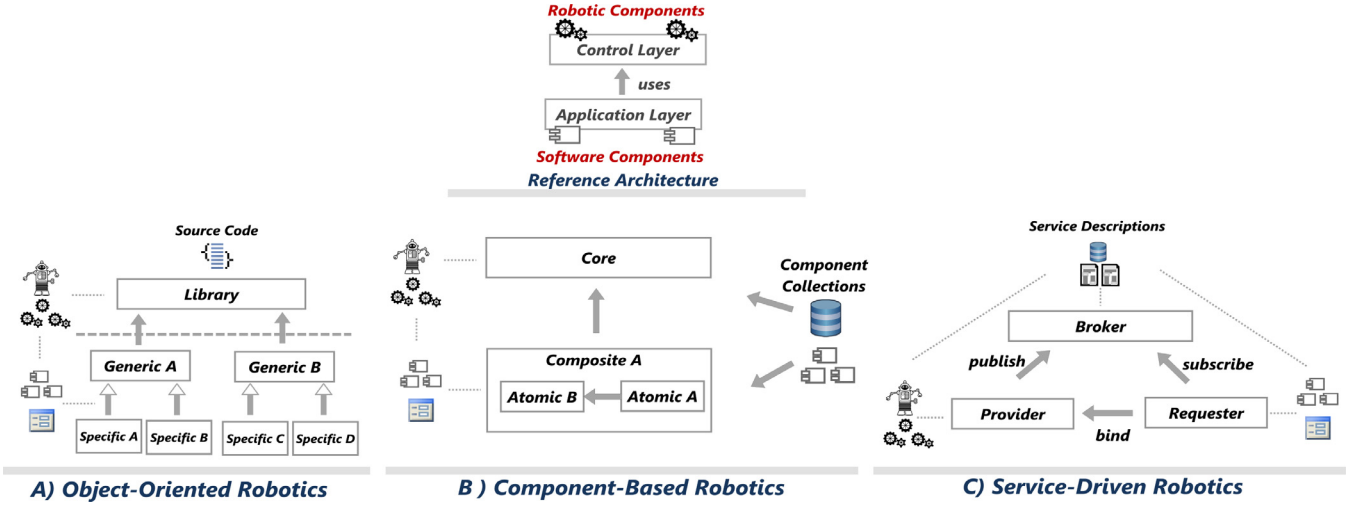

Fig. 12. An overview of various architectural trends.

encapsulates specific/partial functionality and relies on a layer directly below or above it.

Support for robotics – *Based* on OO-R representation in Fig. 12A), to support the development of a robot; the core component encapsulates the low-level control and manipulation logic of a robot in terms of the hardware drivers and configurations that enable software interaction(s) with the hardware. Instead of developing same logic each time for different systems, the generic components must utilize the core to provide the general functionality. A specific component can inherit the functionality from generic ones to support a fully functional robot with customized functionality. For example, in case of a robotic arm in [S13]:

Step I - Core pro*vides the necessary control logic to access robotic arm through hardware driver,*
Step II - Generic layer can reuse the control logic to enable the movement of robotic arm, and finally
Step III - Specific layer extends the generic functionality to rotate the arm at a specific angle and precision.

The studies incorporating OO-R [S10, S13] represent one of the earliest efforts in exploiting design and architectural specification to abstract the low-level programming details (through Core *component*) with modular and reusable components (i.e., Generic and Specific). These approaches led to the development of framework(s) and language(s) such as RIPL (Robot Independent Programming Language) [S13] and Robot Markup Language (RML) [S12] to support OO-R systems [S15, S20]. *Architectural notations* for OO-R are modeled with UML diagrams such as *class* and *sequence* diagrams (cf. Fig. 8).

B. Component-based Robotics (CB-R) Architecture Model

Overview - CB-R follows the component-based software engineering techniques. It utilizes architectural components to support the development, evolution and operations of robotics (Chaudron et al., 2008). CB-R also represents some of the earlier and most recent solutions (covering studies from 1998 [S8] to 2015 [S48]) on exploiting architectural component for robotic systems.

The structural view of CB-R in Fig. 12 B) is based on component composition and component interaction (Chaudron et al., 2008). CB-R utilized three distinct elements namely: Provider, Re-

quester and Component Repository. The Provider and Requester (as a composite) elements are connected using request and provide interconnections. The requester element requests the functionality provided by the provider, while the component repository provides a collection of reusable and off-the-shelf components (Jung et al., 2010, Brugali et al., 2007). The CB-R, aims to abstract the complexity of programming by means of reusable and off-the-shelf components. There are some specialized versions of component-based robotics that are developed by utilizing some well-known design/architectural patterns and techniques such as client-server [S3, S31, S22], layered [S5, S23, S24] and model-driven [S33, S38, S40] architecture.

Support for robotics: Based on CB-R representation in Fig. 12 B), to support robotics the control component abstracts the low-level control and manipulation logic of a robot. The requester component is a composite element based on its child components each of which can be responsible for a specific functionality to complete the functionality of its parents. For example, the requester element represents navigation (composite functionality) supported by combining partial functionalities such as localization, mapping and movement (atomic functionality). The control component acts as a provider, providing the necessary drivers and information to customize the robotics navigation. The requester component can utilize this functionality with its child component to support the robotic navigation in a leader-follower scenario.

The component repository plays an important role in a way that it provides a collection of reusable components, also called off-the-shelf components that can be used for development. As presented in Fig. 12 to highlight a difference between OO-R and CB-R, OO-R supports object reusability (at source code level) with inheritance type relationship in Fig. 12 A). In contrast, component reusability (at architecture level) is supported with component composition in Fig. 12 B). CB-R can rely on model-driven engineering techniques for an automatic generation of code by higher-level components. The CB-R techniques mostly utilized the UML models as well as ADLs for architectural representation and description.

C. **Service-Driven Robotics (SD-R) Architecture Model**

Overview - The SD-R is a representation of service-driven software or specifically Service Oriented Architecture (SOA) that relies on loosely coupled and dynamically composed software services for robotics. SD-R represents the recent and emerging solutions (published in years: 2012 [S20], 2013 [S26], 2014 [S55, S56]). Architectural view of the SD-R has been provided in Fig. 12C) that is based on the classical Service-Oriented Architecture (SOA) framework or other well-known patterns (mediator, publish-subscribe (Gamma et al., 1993)). The architectural representation for SD-R consists of elements named: *Broker, Publisher* and *Subscriber* of services along with their interconnections namely *provide, request* and *bind*. The constraints on the structure or the role of each component are specified based on the order of their interactions, such that (Step I) *Publisher* provides a description of service to the *Broker*, (Step II) *Subscriber* subscribes to the *Broker* for a notification of available services, and finally (Step III) *Subscriber* binds to *Subscriber* for required services - (Step I) and (Step II) above can be interchanged.

Support for robotics - Based on SD-R view in Fig. 12C), the publisher is a component responsible for providing software services that enable the interaction with hardware robotics. For example, the software services could include the control or manipulation logic of the robotics hardware. There are two scenarios, where these services (i) are provided along with hardware components [S27] or (ii) developed separately [S15], in both cases their descriptions (not implementation) need to be published to the Broker. The description can include the name, description of the provided services in terms of service interface using Web Service Description Language. Broker is simply a service description repository or directory that mediates between publisher and subscriber. When a description matches the requirements of the subscriber the publisher can be bound to offer services to the subscriber. The studies incorporating SD-R such as [S27, S55, S56] are more recent trends to exploit services for robotics. The service-oriented robots have found their application in home service [S6, S7] and mission-critical robots [S9, S26]. An interesting study [S27] has focused on developing robot as a service as a notion of cloud-based robotics. The prominent architecture notations for SD-R are UML *state-chart* [S6, S7], *deployment* [S9], *component* (Hu et al., 2012, Garlan et al., 2004, Chaudron et al., 2008) and *class diagram* [S16].

7.3. Emerging solutions for software architecture-driven robotics

Based on the overview of the research progress (Fig. 11) and the details of various research phases, we now discuss the emerging solutions in term of model-driven engineering and cloud-based robotics. The discussion of these solutions highlights dimensions of current and future research to answer **RQ 9**.

7.3.1. Model-driven engineering of robotic software

It aims at developing models by utilizing the model transformation techniques for supporting robotic software. Component-based software engineering (Brugali et al., 2007) or specifically CB-R (Chaimowicz et al., 2003, Jung et al., 2010) aims to minimize the inherent complexity associated with programming and enhance the reusability with architectural components for robotic software. Despite the benefits of CB-R (Section 7.2), architectural components are an abstraction of implementation-specific details and they must be refined to derive the executable code. In this context, model-driven engineering [S38] provides a solution by capturing the overall system model (with components) that is refined and transformed into executable model (with source code). Fig. 11 indicates that in recent years there is a growing interest in applying model-driven solutions to robotic software[7]. The model-driven engineering for robotics provides a methodology – exploiting languages, patterns, tools and infrastructures – to support the development, evolution and operation of robotic applications. It can be argued that CB-R has paved the way for model-driven robotics as architectural components bridge the gap between the requirements of a specific domain (navigation (Sykes et al., 2008), automation (Forlizzi and DiSalvo, 2006)) and satisfying those requirements with executable code. Model-driven robotics relies on three levels of abstraction as below.

- *Meta-model level capturing of domain assumptions and requirements*: first of all, the requirements or the assumptions of the domain are captured and represented as a meta-model that drives model-driven development. For example, the domain requirements may enforce team communication among robots using a central coordinator (mediator) rather than a robot-to-robot interaction (peer to peer).
- *Platform independent level representation of architectural components*: once the requirements have been captured, they are translated into components for architectural modeling independent of any (execution or deployment) platform. For example, based on the requirement specified above a broker component needs to be integrated between two robots for their coordination.

[7] We observed such research interests in some of most recent studies and efforts such as the *Workshop on Model-Driven Robot Software Engineering* to organize a community for promoting research and practices specific to the application of model-driven solutions to robotic systems.

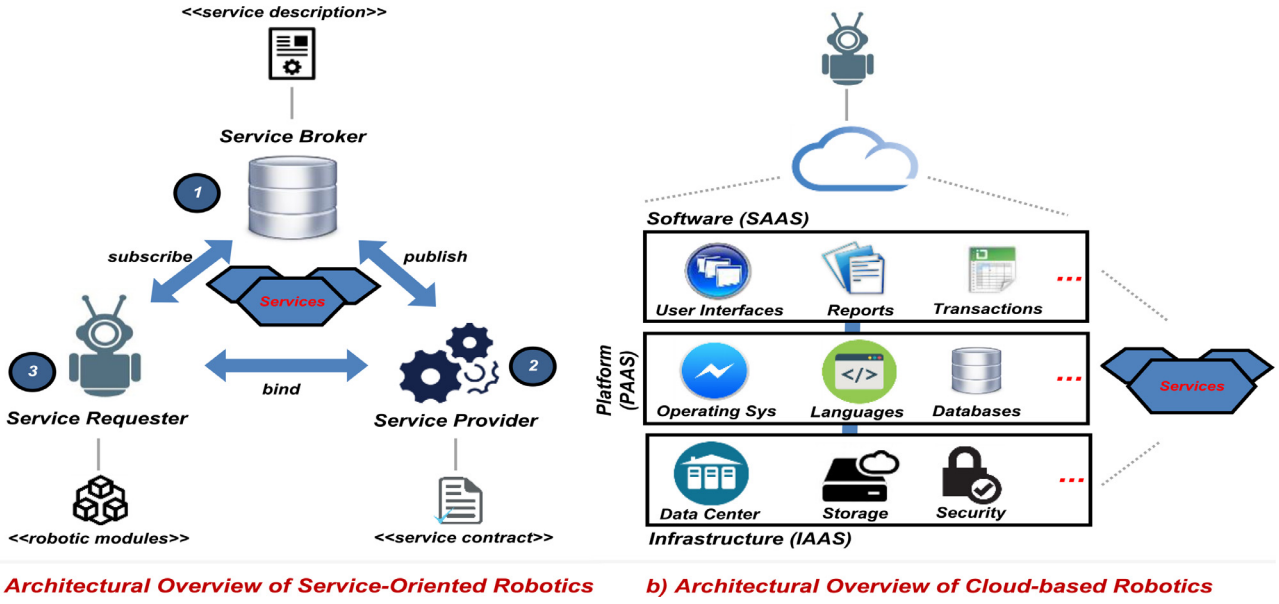

Fig. 13. An illustrative overview of service and cloud-based architectures for robotic software.

- *Platform specific level generation of executable code*: finally, the generic architectural components are transformed into executable source code for a specific execution platform.

The studies exploited model-driven techniques for the development and runtime evolution of robotic systems. In contrast to OO-R and CB-R, with model-driven robotics there is a greater emphasis and efforts on creating models. However, once models are established they help with minimizing the programming complexity (from OO-R) and support reusability across various platforms (compared to the conventional CB-R).

7.3.2. Cloud robotics

Cloud-based robotics appears to be the emerging and futuristic solutions for robotic systems, which aim to exploit the principle of service-orientation to develop and provide robotic software services. Such software services can be dynamically discovered, composed and executed to support SD-R [18, S55, S56] (cf. Section 7.2). The emergence of cloud robotics is a result of service-orientation and more specifically the cloud computing model (56). In a cloud computing model, instead of an upfront acquisition or deployment of computing resources, cloud resources can be utilized as pay-per-use services for hardware and software resources that are publically available (Armbrust et al., 2010). Cloud robotics enables the assembly or operations of a robot by utilizing the powerful computation, virtually unlimited storage as well as communication resources available from cloud-based infrastructures. This means that a robot can discover and utilize distributed hardware and software components on the Internet.

Cloud computing is specifically beneficial for developing mobile robots (Kramer and Scheutz, 2007) in which on-board computation can be offloaded to processors and data stores residing in a cloud. RobotEarth[8] is a relevant project that is focused on the concept to create a World Wide Web for robots where robots can upload/download, and share the knowledge about their operational environment. This creates a web of co-operative robots that are autonomous individually as well as they can also work as a team for an efficient completion of designated tasks. In the following, we also clarify the similarities and distinctions among *service* and *cloud-based* architectures for robotic software that also highlights the role SD-R architecture model(s) could play in developing cloud robotics.

7.3.3. Cloud vs service-based architectures for robotic software

In the context of software architecture, the terms *service* and *cloud* computing are complementary, both denoting the architectural style(s) and enabling technologies to develop and deploy architectural components as dynamically composed software services (Yool et al., 2006, Hu et al., 2012). However, we must distinguish between the concept of service-driven (Cepeda et al., 2011) and cloud-based robotics (Hu et al., 2012) while also acknowledging the fact that (software) services remain fundamental to both types. To maintain the distinction, we first look at the established definitions of both the service and cloud-based architectures and then exemplify them each with the help of Fig. 13. According to the definition by Open Group a *service-driven* or *service-oriented architecture* (SOA) is an architectural style that exploits loosely coupled - heterogeneous and dynamically composed - *software services* that can be *published* (by service provider) via a *broker, discovered* and *subscribed* (by service requester) to develop and deploy distributed system as illustrated in Fig. 13a). The service broker acts as a mediator between the service providers and requesters. For example, a team of first responder robots that coordinate and navigate together (Cepeda et al., 2011) rely on dynamically discovered and composed software services to accomplish their mission. Each of the robotic functionality such as obstacle avoidance, team-lead following or searching the target is supported in form of software services provided by the mission control server. In contrast to the built-in software functionality such as pre-packaged modules/components for traditional robots, the service-driven robot as service requester can communicate with the central server (service provider via broker) to acquire the required functionality by exploiting software services, whenever required.

NIST defines *cloud computing* as a combination of *software, platform* and *infrastructure* services that enable the entities or organizations to leverage distributed and interoperable services to develop and deploy their software systems over available resources (such as public or private servers), represented in Fig. 13 b). In

[8] RoboEarth: http://roboearth.org/cloud_robotics/.

comparison to the service provisioning as in SOA, the cloud computing model also offers platform and necessary infrastructure to service requesters for off-loading the computation and memory-intensive tasks. For example, in a robot that relies on Simultaneous Localization And Mapping (SLAM) operations, the computationally expensive map optimization and storage tasks are off-loaded to a Cloud [S56], whereas tasks like camera tracking is performed by the robot itself. In this scenario, a robot's onboard computers are freed from most of computation, while the only extra requirement is network connectivity.

We conclude that software services or specifically SOA is fundamental to cloud-based architectures. SOA focuses on a collection of services to bind the service providers and requesters in a distributed architecture. A cloud-based architecture goes beyond software services (e.g., user interfaces and transaction processing) to offer platforms (e.g., operating systems and databases) and infrastructure (e.g., storage center and security servers). In comparison to SOA that relies on traditional means of software quality such as modularity, heterogeneity, extensibility, a cloud-based architecture must also satisfy quality attributes like elasticity, multi-tenancy and resource virtualization that are specific to service-driven clouds (Hu et al., 2012, Armbrust et al., 2010).

Considering the context of cloud robotics, it is vital to distinguish between **cloud-native** and **cloud-enabled** robotics (Armbrust et al., 2010). Cloud-native are the types of robots that are specifically built to operate in cloud-based environment [S20, S27, S56]. In contrast, the cloud-enabled robots refer to the existing or legacy robots that need to be evolved or customized to enable them exploit the cloud computing model. Therefore, evolving a legacy robot towards a cloud-enabled robot requires additional efforts; however, such an evolution can enable a robot system to exploit cloud-based resources. There is a lot of progress on research that enables the migration or evolution of the legacy software systems to cloud-enabled software (Jamshidi et al., 2013) and the existing research can be influential to support robotics evolution. In this context, the study [S14] highlights a solution on architectural reengineering to support the evolution of legacy architecture to a new architecture. Architectural reengineering solutions and specifically the models of legacy migration towards cloud can facilitate *architecture-driven migration of legacy robots to cloud-enabled robotics* (Jamshidi et al., 2013).

8. Validity threats

This study provides a classification to map the reported solutions by reviewing and analyzing peer-reviewed literature. We followed the guidelines for conducting systematic mapping studies reported in (Brereton et al., 2007, Petersen et al., 2008, Cruzes and Dybå, 2010) based on a defined and internally and externally evaluated protocol for SMS (Ahmad and Babar, 2016). Like any other empirical study, systematic mapping studies can also have limitations that must be considered for analyzing the potential impact of the validity threats to the findings of SMS (Cruzes and Dybå, 2010). We discuss three types of validity threats associated with different activities of this SMS.

- ***Threats to identification of primary studies.*** In the literature search strategy (cf. Section 3.2), we aimed to retrieve as many relevant studies as possible to avoid any possible literature selection bias and to accommodate all the available evidence. We faced a challenge in determining the scope of our study as the notion of architecture means different things to different research communities including software engineering, robotics, artificial intelligence and others a discussed in (Ahmad and Babar, 2016). Therefore, to cover them all and avoid any bias, we searched the literature based on relevant terms (cf. Table 2) and combined them in our search string (cf. Fig. 3). While this search strategy and search string composition significantly increases the search work (Wohlin, 2014), however, it enabled us to find a comprehensive set of the relevant study. We also developed and evaluated a review protocol. The protocol provides a replicable blue-print to derive the search strategies, literature identification and selection.
- ***Threats to quality of studies and data extraction consistency.*** The results of this study and their quality are based on the quality of the studies that have been reviewed. This means that if the quality of the primary studies is low, the claims and their supporting evidence derived from these studies are unlikely to be strong and reliable. Therefore, it is vital to (i) minimize the threats regarding the quality of selected studies and to ensure (ii) a consistent representation of data extracted from these studies. It is of central importance to qualitatively analyze and synthesize of the data extracted from the selected studies (Cruzes and Dybå, 2010). As described in section 3, we followed a multi-step process and explicitly assessed the quality of each individual study to ensure that a lack of quality results in an exclusion of the study. The ideal scenarios may strictly adhere to the guidelines in (Brereton et al., 2007, Cruzes and Dybå, 2010), however, the quality metric can be subjective based on the objectives of SMS and the consensus among researchers as discussed in the protocol (Ahmad and Babar, 2016).
  Moreover, we derived a *structured* template to ensure consistency in data extraction and capturing as per the needs of the study's RQs. For a fine-grained representation of the extracted data, we have defined a generic and mapping specific attributes to capture data for detailed synthesis.
  - ***Threats to data synthesis and results reporting.*** The final type of threat corresponds to the bias or a lack of systematic approach to synthesize and report the results. We tried to mitigate this threat by conducting a pilot study. A limited number of researchers and their expertise (software engineering and software architecture) may have an internal bias on the style and reporting of results. We tried to minimize this by (i) classifying the studies (cf. Fig. 5) by following the ACM classification scheme for computing research, guidelines from existing studies (Medvidovic and Taylor, 2000), our experience from (Babar et al., 2004), (ii) mapping the results (cf. Fig. 6) to better organize the overlapping and disjoint findings and (iii) an external evaluation of the review protocol.
    The threat to the reliability of data synthesis and reporting has been mitigated based on discussion and peer review of the extracted data by the researchers, having a structured template for data synthesis, and several steps where the scheme and process were refined and evaluated. Whilst we followed the guidelines from (Petersen et al., 2008, Cruzes and Dybå, 2010) to conduct the study, we had deviations from the ideal approaches based on the requirements of this study. We believe that the validity of the study is high, given the use of a systematic and recommended procedure, the extensive discussions and evaluation of the protocol and a pilot study to refine the scope of review.

9. Conclusions

The goal of this SMS was to systematically identify, select, and analyze the published research that reports architecture related solutions for developing and evolving robotic systems. This SMS has identified and discussed various architecture-related challenges and solutions reported for robotic systems. The results of the classification and mapping of the existing research have been provided in terms of structured tables and illustrative figures with the aim

to systematized and disseminate the knowledge about architectural challenges and solutions for robotic systems. The taxonomical classification (cf. Fig. 5, Section 4) provides a holistic overview of the overlapping and distinct research themes and their sub-themes that emerged and progressed over time (early 90s – 2015). The study complements the existing research in terms of the systematic reviews (Oliveira et al., 2013, Pons et al., 2012) and surveys-based studies (Zhi et al., 2013, Elkady and Sobh, 2012, Kramer and Scheutz, 2007) on the application of various software engineering techniques to robotic systems. The key contribution of this research is to specifically investigate the state-of-the-research on the role of software architecture in robotic systems.

The classification and mapping along with their accompanying templates provides 10 data items as a moderate amount of extracted information from each of the reviewed studies. We systematically reviewed 56 studies using 10 data items that resulted in a collection of $56 \times 10 = 560$ potentially unique elements of investigation. Based on this, considering an example case [S39], a reader can analyze the information as:

- **Subject**: analyze *Architecture-driven Adaptation (problem space)*
- **Object:** using *Model-driven Engineering (solution space)*
- **Implications:** for *mission critical robots*.

Architecture-based engineering for robotic software Our study aims to support the emerging efforts focused on synergizing the current and future research on Robotics Systems and Software Engineering (IEEE Robotics and Automation Society, 2015 bib32, Brugali, 2010). Specifically, the mapping study is expected to highlight the role of software architecture for the development and evolution of large-scale robotic systems. The outcomes of our study highlight some innovative and frequently used architectural solutions to address various challenges of developing and evolving robotic systems. The study findings also provide inspiration and ideas for future research that is expected to contribute to the development of the next generation of robotic systems as part of multi-disciplinary research and development efforts. The results from this mapping study can have several implications for software architecture and robotic systems researchers and practitioners:

– *Researchers* who need to quickly access a body of knowledge based on relevant literature and solutions supporting the role of software architecture for robotics. This study also highlights features of existing and emerging dimensions of the research. Moreover, a mapping (cf. Fig. 6, Section 4) of the research state-of-the-art and its historical progression can help identify aspects like: (i) available evidence of research progression, (ii) temporal distribution and frequency of various research themes, (iii) the focus and contributions of most and the least frequent themes. Table 2 can be interpreted as a structured catalogue – mapping challenges and solutions – to help us quickly identify various architectural solutions and the role of software architecture in robotic systems.
– *Practitioners* interested in understanding the academic research in terms of existing solutions and frameworks (academic, industrial, and open-source efforts) along with modeling notations, evaluation methodologies to analyze possibilities where academic approaches can be leveraged for industrial solutions. The discussion of architectural solutions has been complemented with presentation of some relevant frameworks (cf. Fig. 7, Section 5) that support these solutions. The mapping study also highlights the needs for gathering and analyzing practitioners' observations and experiences of the role of software architecture for robotic systems.

Whilst the findings from this study can have several interpretations and uses depending upon the purpose, our goal was to provide a foundation to consolidate, analyze and map out the existing research on architecture-driven solutions for robotic systems. We conclude the reporting of this SMS by highlighting two core findings of this study:

– A systematic analysis of more than two decades of research reveals three prominent architectural models or generations that emerged and referred to as OO-R, CB-R and SD-R in order of their emergence. In the last decade, there has been a noticeable growth of research with various innovative solutions, however, model-driven and cloud-based robotics appear to be the most prominent and emerging solutions. Model-driven robotics bridges the gap between domain requirements and executable specifications, whereas cloud robotics exploits the concepts of SD-R to construct a web of distributed and connected robots.
– The conventional concepts of architectural frameworks and architectural notations are well integrated into the robotics domain. UML-based modeling is influential and utilized by researchers and practitioners to model, develop and evolve software architecture for robotics. Researchers have used various validation methods to evaluate the reported solutions; however, there is a lack of focus on architecture specific evaluation for validating the quality-oriented or architecturally significant requirements for robotic software.

Appendix A

Study ID	Author(s), title, channel of publication	Publication year	Quality score
[S1]	T. Kaupp, A. Brooks, B. Upcroft and A. Makarenko. ***Building a Software Architecture for a Human-Robot Team Using the Orca Framework***. In IEEE International Conference on Robotics and Automation	2007	3.5
[S2]	R. Volpe, I.A.D. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das. ***The CLARAty Architecture for Robotic Autonomy***. In IEEE Aerospace Conference	2001	2.5
[S3]	T. Baier, M. Hiiser, D. Westhoff, J. Zhang. ***A Flexible Software Architecture for Multi-modal Service Robots***. In Multiconference on Computational Engineering in Systems Applications	2006	2.5
[S4]	H. Ahn, D-S. Lee, S. C. Ahn. ***A Hierarchical Fault Tolerant Architecture for Component-based Service Robots***. In IEEE International Conference on Industrial Informatics.	2010	2.5
[S5]	G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme. ***Architecture-driven Self-adaptation and Self-management in Robotics Systems***. In ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems.	2009	3.0
[S6]	X. Ma, K. Qian, X. Dai, F. Fang, Y. Xing. ***Framework Design for Distributed Service Robotic Systems***. 5th IEEE Conference on Industrial Electronics and Applications.	2010	2.5
[S7]	T. Maenad, A. Tikanmäki, J. Riekki, J. Röning***. A Distributed Architecture for Executing Complex Tasks with Multiple Robots*** In IEEE International Conference on Robotics and Automation	2004	3.0
[S8]	L. E. Parker. ***ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation***. In IEEE Transactions on Robotics and Automation	1998	4.5
[S9]	J. S. Cepeda, R. Soto, J. L. Gordillo, L. Chaimowicz. ***Towards a Service-Oriented Architecture for Teams of Heterogeneous Autonomous Robots***. In 10th Mexican International Conference on Artificial Intelligence	2011	3.0
[S10]	D. J. Miller, R. C. Lennox. ***An Object-oriented Environment for Robot System Architectures***. In IEEE Control Systems	1991	4.0
[S11]	J. F. Inglés-Romero, C. Vicente-Chicote, B. Morin, O. Barais. ***Towards the Automatic Generation of Self-Adaptive Robotics Software: An Experience Report***. In 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.	2011	2.5
[S12]	J. Kwak, J. Y. Yoon, and R. H. Shinn. ***An Intelligent Robot Architecture based on Robot Mark-up Languages***. In IEEE International Conference on Engineering of Intelligent Systems.	2006	2.5
[S13]	P. Kazanzides, J. Zuhars, B. Mittelstadt, B. Williamson, P. Cain, F. Smith, L. Rose, B. Musits. ***Architecture of a Surgical Robot***. In IEEE International Conference on Systems, Man and Cybernetics.	1992	3.0
[S14]	M. Kim, J. Lee, K. C. Kang, Y. Hong, S. Bang. ***Re-engineering Software Architecture of Home Service Robots: A Case Study***. In 27th International Conference on Software Engineering.	2005	4.0
[S15]	J. Yool, S. Kim, and S. Hong. ***The Robot Software Communications Architecture (RSCA): QoS-Aware Middleware for Networked Service Robots***. In SICE-ICASE International Joint Conference	2006	3.0
[S16]	W. Hongxing, L. Shiyi, Z. Ying, Y. Liang, W. Tianmiao. ***A Middleware Based Control Architecture for Modular Robot Systems***. In IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications.	2008	2.5
[S17]	N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku and W. Yoon. ***Composite Component Framework for RT-Middleware (Robot Technology Middleware)***. In IEEE/ASME International Conference on Advanced Intelligent Mechatronics.	2005	3.0
[S18]	I. Buzurovic, T. K. Podder, L. Fu, and Y. Yu. ***Modular Software Design for Brachytherapy Image-Guided Robotic Systems***. In 2010 IEEE International Conference on Bioinformatics and Bioengineering.	2010	3.0
[S19]	S. Limsoonthrakul, M. N. Dailey, M. Srisupundit. ***A Modular System Architecture for Autonomous Robots Based on Blackboard and Publish-Subscribe Mechanisms***. In International Conference on Robotics and Biomimetics.	2008	2.5
[S20]	G. Hu, W. P. Tay, and Y. Wen. ***Cloud Robotics: Architecture, Challenges and Applications***. In IEEE Network.	2012	3.5
[S21]	A. Angerer, A. Hoffmann, F. Ortmeier, M. Vistein and W. Reif. ***Object-Centric Programming: A New Modeling Paradigm for Robotic Applications***. In International Conference on Automation and Logistics	2009	2.5
[S22]	M. Y. Jung, A. Deguet, and P. Kazanzides. ***A Component-based Architecture for Flexible Integration of Robotic Systems***. In IEEE/RSJ International Conference on Intelligent Robots and Systems.	2010	2.5
[S23]	D. Kim and S. Park. ***Designing Dynamic Software Architecture for Home Service Robot Software***. In International Conference on Embedded and Ubiquitous Computing.	2007	4.0
[S24]	Y. Park, I. Ko, and S. Park. ***A Task-based Approach to Generate Optimal Software-Architecture for Intelligent Service Robots***. 16th IEEE International Conference on Robot & Human Interactive Communication.	2007	2.5
[S25]	W. Hongxing, D. Xinming, L. Shiyi, T. Guofeng, W. Tianmiao. ***A Component Based Design Framework for Robot Software Architecture***. IEEE/RSJ International Conference on Intelligent Robots and Systems.	2009	2.5
[S26]	Lorenzo Flückiger, Hans Utz***, Service Oriented Robotic Architecture for Space Robotics: Design, Testing, and Lessons Learned.*** Journal of Field Robotics	2013	4.0
[S27]	Anis Koubaa***, A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds.*** 27th International Conference on Architecture of Computing Systems	2014	3.0
[S28]	James E. Beck, Michael Reagin, Thomas E. Sweeny, Ronald L. Anderson, Timothy D. Garner. ***Applying a Component-Based Software Architecture to Robotic Workcell Applications.*** In IEEE Transactions on Robotics and Automation.	2000	4.0
[S29]	Javier Gamez García, Juan Gómez Ortega, Alejandro Sánchez García, and Silvia Satorres Martínez***. Robotic Software Architecture for Multisensor Fusion System.*** IEEE Transactions on Industrial Electronics	2009	3.5
[S30]	Nenad Medvidovic, Hossein Tajalli, Joshua Garcia, and Ivo Krka, Yuriy Brun, George Edwards. ***Engineering Heterogeneous Robotics Systems: A Software Architecture-Based Approach.*** IEEE Computer.	2011	4.0
[S31]	Soohee Han, Mi-sook Kim, and Hong Seong Park. ***Development of an Open Software Platform for Robotic Services.*** IEEE Transactions on Automation Science and Engineering	2012	4.0
[S32]	Alex Brooks, Tobias Kaupp, Alexei Makarenko and Stefan Williams, Anders Oreback***. Towards Component-based Robotics.*** IEEE/RSJ International Conference on Intelligent Robots and Systems	2005	2.5
[S33]	Li Hsien Yoong, Zeeshan E. Bhatti, and Partha S. Roop***. Combining IEC 61,499 Model-Based Design with Component-Based Architecture for Robotics.*** The Third International Conference on Simulation, Modeling, and Programming for Autonomous Robots	2012	2.5

(*continued on next page*)

Study ID	Author(s), title, channel of publication	Publication year	Quality score
[S34]	Jonghoon Kim, Mun-Taek Choi, Munsang Kim, Suntae Kim, Minseong Kim, Sooyong Park, Jaeho Lee, ByungKook Kim. ***Intelligent Robot Software Architecture.*** 13th International Conference on Advanced Robotics.	2008	2.5
[S35]	G. Biggs, N. Ando, and T. Kotoku. ***Coordinating Software Components in a Component-Based Architecture for Robotics.*** In 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots	2010	3.0
[S36]	Dongsun Kim, Sooyong Park, Youngkyun Jin, Hyeongsoo Chang, Yu-Sik Park, In-Young Ko, Kwanwoo Lee, Junhee Lee, Yeon-Chool Park, Sukhan Lee. ***SHAGE: A Framework for self-Managed Robot Software.*** In 2006 International Workshop on Self-adaptation and Self-managing Systems	2006	2.5
[S37]	Minseong Kim, Suntae Kim, Sooyong Park, Mun-Taek Choi, Munsang Kim, Hassan Gomaa. ***UML-based Service Robot Software Development: A Case Study.*** 28th International Conference on Software Engineering.	2006	3.0
[S38]	Christian Schlegel, Thomas Haßler, Alex Lotz and Andreas Steck. ***Robotic Software Systems: From Code-Driven to Model-Driven Designs.*** International Conference on Advanced Robotics	2009	3.0
[S39]	Hossein Tajalli, Joshua Garcia, George Edwards and Nenad Medvidovic. ***PLASMA: A Plan-based Layered Architecture for Software Model-driven Adaptation.*** IEEE/ACM International Conference on Automated Software Engineering	2010	3.5
[S40]	Markus Klotzbücher, Nico Hochgeschwender, Luca Gherardi, Herman Bruyninckx, Gerhard Kraetzschmar, Davide Brugali. ***The BRICS Component Model: A Model-based Development Paradigm for Complex Robotics Software Systems.*** 28th Annual ACM Symposium on Applied Computing	2013	2.5
[S41]	John C. Georgas and Richard N. Taylor. ***Policy-Based Self-Adaptive Architectures: A Feasibility Study in the Robotics Domain.*** International Workshop on Software Engineering for Adaptive and Self-managing systems	2008	3.0
[S42]	Andreas Steck, Alex Lotz and Christian Schlegel. ***Model-driven Engineering and Run-time Model-usage in Service Robotics.*** 10th ACM International Conference on Generative Programming and Component Engineering	2012	3.5
[S43]	Francisco Ortiz, Diego Alonso, Bárbara Álvarez, Juan A. Pastor. ***A Reference Control Architecture for Service Robots Implemented on a Climbing Vehicle.*** 10th Ada-Europe International Conference on Reliable Software Technologies.	2005	2.5
[S44]	Jennifer Pérez, Nour Ali, Jose A. Carsı, Isidro Ramos, Bárbara Álvarez, Pedro Sanchez, Juan A. Pastor . ***Integrating Aspects in Software Architectures: PRISMA Applied to Robotic tele-operated Systems.*** Information and Software Technology.	2007	3.5
[S45]	Carle Côté, Dominic Létourneau, Clément Raïevsky, Yannick Brosseau, François Michaud. ***Using MARIE for Mobile Robot Software Development and Integration.*** Software Engineering for Experimental Robotics	2007	3.5
[S46]	Antonio C. Domínguez-Brito, Daniel Hernández-Sosa, José Isern-González, Jorge Cabrera-Gámez. ***CoolBOT: A Component Model and Software Infrastructure for Robotics.*** Software Engineering for Experimental Robotics	2007	2.5
[S47]	José Baca, Prithvi Pagala, Claudio Rossi, Manuel Ferre. ***Modular Robot Systems Towards the Execution of Cooperative Tasks in Large Facilities***. Robotics and Autonomous Systems	2015	3.5
[S48]	Didier Crestani, Karen Godary-Dejean, Lionel Lapierre. ***Enhancing Fault Tolerance of Autonomous Mobile Robots***. Robotics and Autonomous Systems	2015	3.5
[S49]	Andrea Bonarini, Matteo Matteucci, Martino Migliavacca, Davide Rizzi. ***R2P: An Open Source Hardware and Software Modular Approach to Robot Prototyping.*** Robotics and Autonomous Systems	2014	4.0
[S50]	Christian Berger. ***From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation.*** Journal of Software Engineering for Robotics	2014	3.5
[S51]	Jan Oliver Ringert, Alexander Roth, Bernhard Rumpe, Andreas Wortmann. ***Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems.*** International Workshop on Model-Driven Robot Software Engineering.	2014	3.0
[S52]	Peihua Chen, Qixin Cao. ***A Middleware-Based Simulation and Control Framework for Mobile Service Robots.*** Journal of Intelligent & Robotic Systems	2014	3.5
[S53]	Manja Lohse, Frederic Siepmann, Sven Wachsmuth. ***A Modeling Framework for User-Driven Iterative Design of Autonomous Systems.*** International Journal of Social Robotics	2014	3.0
[S54]	Gianluca Antonelli, Khelifa Baizid, Fabrizio Caccavale, Gerardo Giglio, Giuseppe Muscio, Francesco Pierri. ***Control Software Architecture for Cooperative Multiple Unmanned Aerial Vehicle-Manipulator Systems***. Journal of Software Engineering for Robotics	2014	3.5
[S55]	Ion Mircea Diaconescu, Gerd Wagner. ***Towards a General Framework for Modeling, Simulating and Building Sensor/Actuator Systems and Robots for the Web of Things***. International Workshop on Model-Driven Robot Software Engineering	2014	2.5
[S56]	L. Riazuelo· Javier Civera, J.M.M. Montiel. ***C²TAM: A Cloud Framework for Cooperative Tracking and Mapping***. Robotics and Autonomous Systems	2014	4.0

References

Adrion, W.R., Branstad, M.A., Cherniavsky, J.C., 1982. Validation, verification, and testing of computer software. ACM Comput. Surv. (CSUR), ACM 14 (2), 159–192.

A. Ahmad, M.A. Babar. Research protocol of software architectures for robotics systems: a systematic mapping study. TR-2016-22, Software and Systems Section, IT University of Copenhagen, 2016. [Online] http://media.wix.com/ugd/396772_cfaf0368a9374b99921c2e1c26691add.pdf, Accessed 04/20/2016.

Angerer, A., Hoffmann, A., Ortmeier, F., Vistein, M., Reif, W., 2009. Object-Centric Programming: A New Modeling Paradigm for Robotic Applications. In: International Conference on Automation and Logistics (ICAL 2009). IEEE, pp. 18–23.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. Commun. ACM (CACM) 53 (4), 50–58 ACM.

Avgeriou, P., Zdun, U., 2005. Architectural patterns revisited, a pattern language. In: 10th European Conference on Pattern Languages of Programs (EuroPlop 2005, pp. 1–39 .

Babar, M.A., Zhu, L., Jeffery, R., 2004. A framework for classifying and comparing software architecture evaluation methods. In: 2004 Australian Software Engineering Conference (ASWEC 2004). IEEE, pp. 309–318.

Biggs, G., MacDonald, B., 2003. A survey of robot programming systems. In: Australasian Conference on Robotics and Automation (CSIRO 2013), 4, pp. 3479–3484.

Bonarini, A., Matteucci, M., Migliavacca, M., Rizzi, D., 2014. R2P: an open source hardware and software modular approach to robot prototyping. Robot. Autonom. Syst. 62 (7), 1073–1084 Elsevier.

Boyatzis, R.E., 1998. Transforming Qualitative Information: Thematic Analysis and Code Development. Sage Publications, USA.

Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. (JSS) 80 (4), 571–583 Elsevier.

Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Oreback, A., 2006. Towards component-based robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 163–168.

Brugali, D., 2010. From the editor-in-chief: a new research community, a new journal. J. Softw. Eng. Robot. (JOSER) 1 (1), 01–02 ISSN: 2035-3928.

Brugali, D., Brooks, A., Cowley, A., Côté, C., Domínguez-Brito, A., Létourneau, D., Michaud, F., Schlegel, C., 2007. Trends in component-based robotics. In: Software Engineering for Experimental Robotics. Springer Tracts in Advanced Robotics, 30. Springer, pp. 135–142.

Buzurovic, I., Podder, T.K., Fu, L., Yu, Y., 2010. Modular software design for brachytherapy image-guided robotic systems. In: 2010 IEEE International Conference on Bioinformatics and Bioengineering (BIBE 2010). IEEE, pp. 203–208.

Cepeda, J.S., Soto, R., Gordillo, J.L., Chaimowicz, L., 2011. Towards a service-oriented architecture for teams of heterogeneous autonomous robots. In: 10th Mexican International Conference on Artificial Intelligence (MICAI 2011). IEEE, pp. 102–108.

Chaimowicz, L., Cowley, A., Sabella, V., Taylor, C.J., 2003. ROCI: a distributed framework for multi-robot perception and control. In: 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 266–271.

Chaudron, M.R.V., Szyperski, C., Reussner, R., 2008. 11th International Symposium on Component-Based Software Engineering Springer LNCS 5282, ISBN 978-3-540-87890-2.

Chen, L., Babar, M.A., Nuseibeh, B., 2013. Characterizing Architecturally Significant Requirements. IEEE Softw. 30 (2), 38–45 IEEE.

Cheng, B.H.C, et al., 2009. Software engineering for self-adaptive systems: a research road map. In: Software Engineering for Self-Adaptive Systems, 5525. Springer LNCS, pp. 1–26.

Cruzes, D.S., Dybå, T., 2010. Synthesizing evidence in software engineering research. In: Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM 2010). ACM.

Elkady, A., Sobh, T., 2012. Robotics middleware: a comprehensive literature survey and attribute-based bibliography. J. Robot..

Forlizzi, J., DiSalvo, C., 2006. Service robots in the domestic environment: a study of the roomba vacuum in the home. In: 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction (HRI 2006). ACM, pp. 258–265.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1993. Design patterns: abstraction and reuse of object-oriented design. European Conference on Object Oriented Programming (ECOOP 1993) 707, 406–431.

Garage, W., 2011. ROS: Robot Operating System [Online:] http://www.ros.org/, [Access:] 24/06/2014.

Garlan, D., Poladian, V., Schmerl, B., Sousa, J.P., 2004. Task-based self-adaptation. In: 1st ACM SIGSOFT workshop on Self-managed Systems (WOSS 2004). ACM, pp. 54–57.

Gomaa, H., 2000. Designing Concurrent, Distributed, and Real-Time Application with UML. Addison-Wesley Object Technology Series.

Groover, M.P., 2007. Automation, Production Systems, and Computer-integrated Manufacturing, 3rd Edition Prentice Hall Press, USA.

Hinchey, M., Coyle, L., 2010. Evolving critical systems: a research agenda for computer-based systems. In: Proceedings of the 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS 2010). IEEE, pp. 430–435.

Hu, G., Tay, W.P., Wen, Y., 2012. Cloud robotics: architecture, challenges and applications. IEEE Netw. 26 (3), 21–28 IEEE.

IEEE RobSotics and Automation Society: Technical Committee on Software Engineering for Robotics and Automation (TC-SOFT) [Access:] 08/03/2015,[Online:] http://robotics.unibg.it/tcsoft/.

ISO. *ISO/IEC 25010:2011 Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models*, 2011. [Online:] http://webstore.iec.ch/preview/info_isoiec25010%7Bed1.0%7Den.pdf, [Access:] 14/07/2014.

Jackson, J., Coll, C.H.M., 2008. Microsoft robotics studio: a technical introduction. Robot. Autom. Mag. 14 (4), 82–87 IEEE.

Jamshidi, P., Ahmad, A., Pahl, C., 2013. Cloud migration research: a systematic review. IEEE Trans. Cloud Comput. (TCC) 1 (2), 142–157 IEEE.

Jung, M.Y., Deguet, A., Kazanzides, P., 2010. A component-based architecture for flexible integration of robotic systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010). IEEE, pp. 6107–6112.

Katz, D.S., Some, R.R., 2003. NASA advances robotic exploration. IEEE Comput., IEEE 36 (1), 52–61.

Kazanzides, P., Zuhars, J., Mittelstadt, B., Williamson, B., Cain, P., Smith, F., Rose, L., Musits, B., 1992. Architecture of a surgical robot. In: IEEE International Conference on Systems, Man and Cybernetics (ICSMC 1992), 2. IEEE, pp. 1624–1629.

Kramer, J., Scheutz, M., 2007. Development environments for autonomous mobile robots: a survey. In: Autonomous Robots, 22. Springer, pp. 101–132.

Kruchten, P., Obbink, H., Stafford, J., 2006. The past, present, and future for software architecture. IEEE Softw. 23 (2), 22–30 IEEE.

Kwak, J., Yoon, J.Y., Shinn, R.H., 2006. An intelligent robot architecture based on robot mark-up languages. In: IEEE International Conference on Engineering of Intelligent Systems (ICEIS 2006). IEEE, pp. 1–6.

Mäenpää, T., Tekkamaki, A., Riekki, J., Röning, J., 2004. A distributed architecture for executing complex tasks with multiple robots. In: IEEE International Conference on Robotics and Automation (ICRA 2004), 4. IEEE, pp. 3449–3455.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A., 2013. What industry needs from architectural languages: a survey. IEEE Trans. Software Eng. (TSE) 39 (6), 869–891 IEEE.

Medvidovic, N., Taylor, R.N., 2000. A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. (TSE) 26 (1), 70–93 IEEE.

Miller, D.J., Lennox, R.C., 1991. An object-oriented environment for robot system architectures. IEEE Cont. Syst. 11 (2), 14–23 IEEE.

Oliveira, L.B.R., Osorio, F., Nakagawa, E.Y., 2013. An investigation into the development of service-oriented robotic systems. In: 28th Annual ACM Symposium on Applied Computing (SAC 2013). ACM, pp. 223–228.

Orebäck, A., Christensen, H.I., 2003. Evaluation of architectures for mobile robotics. Autonom. Robots 14 (1), 33–49 Springer.

Pantofaru, C., Chitta, S., Gerkey, B., Rusu, R., Smart, W.D., Vaughan, R., 2013. Special issue on open source software-supported robotics research. Autonom. Robots 34 (3), 129–131 Springer.

Parker, L.E., 1998. ALLIANCE: *an architecture for fault tolerant multirobot cooperation*. IEEE Trans. Robot. Autom. 14 (2), 220–240 IEEE.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008). ACM, pp. 68–77.

Pons, C., Giandini, R., Arévalo, G., 2012. A systematic review of applying modern software engineering techniques to developing robotic systems. Revista Ingeniería e Investigación 32 (1), 58–63.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A., 2009. ROS: an open-source robot operating system. ICRA Workshop on Open Source Software.

Redwine Jr., SamuelT., Riddle, WilliamE., 1985. Software technology maturation. In: Eighth International Conference on Software Engineering (ICSE 1985). ACM, pp. 189–200.

Schmerl, B., Garlan, D., 2004. AcmeStudio: *supporting style-centered architecture development*. In: 26th International Conference on Software Engineering (ICSE 2003). ACM, pp. 704–705.

Schofield, M., 1999. "Neither master nor slave...". a practical case study in the development and employment of cleaning robots.. In: 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 1999), 2. IEEE, pp. 1427–1434.

Sykes, D., Heaven, W., Magee, J., Kramer, J., Jeff, 2008. From goals to components: a combined approach to self-management. In: 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2008). ACM, pp. 1–8.

Taylor, R.H, Stoianovici, D., 2003. Medical robotics in computer-integrated surgery. IEEE Trans. Robot. Autom. 19 (5), 765–781 IEEE.

Thrun, S., 2010. Toward robotic cars. Commun. ACM (CACM) 53 (4), 99–106 ACM.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: 18th international Conference on Evaluation and Assessment in Software Engineering (EASE 2014). ACM, pp. 1–10.

Wurman, P., D'Andrea, R., Mountz, M., 2007. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In: 19th National Conference on Innovative Applications of Artificial Intelligence (IAAI 2007), 2. ACM, pp. 1752–1759.

Yool, J., Kim, S., Hong, S., 2006. The robot software communications architecture (rsca): qos-aware middleware for networked service robots. In: SICE-ICASE International Joint Conference (SICE 2006). IEEE, pp. 330–335.

Zhi, Y., Jouandeau, N., Cherif, A.A., 2013. A survey and analysis of multi-robot coordination. Int. J. Adv. Robotic Syst. 10, 399.