Architectural design of service-oriented robotic systems

Lucas Bueno Ruas de Oliveira

Lucas Bueno Ruas de Oliveira

Projeto arquitetural de sistemas robóticos orientados a serviços

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP e ao *Institut de Recherche en Informatique et Systèmes Aléatoires* - IRISA-UBS, como parte dos requisitos para obtenção dos títulos de Doutor em Ciências e *Docteur en Informatique* pelo convênio acadêmico internacional entre o Programa de Pós-graduação em Ciências de Computação e Matemática Computacional (USP) e o Programa Doutoral em Matemática e Ciências e Tecnologias de Informação e Comunicação (UBS). *VERSÃO REVISADA*

Áreas de Concentração: Ciências de Computação e Matemática Computacional / Matemática e Ciências e Tecnologias de Informação e Comunicação

Orientadora: Profa. Dra. Elisa Yumi Nakagawa (ICMC-USP, Brasil)
Orientador: Prof. Dr. Flavio Oquendo (IRISA-UBS, França)

ICMC-USP, Brasil / IRISA-UBS, França
Julho de 2015

Ficha catalográfica preparada pela Seção de Tratamento
da Informação da Biblioteca Prof. Achille Bassi – ICMC/USP

O48a

Oliveira, Lucas Bueno Ruas de
Architectural design of service-oriented robotic systems / Lucas Bueno Ruas de Oliveira ; orientadores Elisa Yumi Nakagawa e Flavio Oquendo. -- São Carlos, 2015.
187 p.

Tese (Doutorado Duplo - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional e Programa Doutoral em Matemática e Ciências e Tecnologias de Informação e Comunicação) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo; *Institut de Recherche en Informatique et Systèmes Aléatoires, Université de Bretagne-Sud (França)*, 2015.

1. Software architecture. 2. Service-oriented architecture. 3. Reference architecture. 4. Robotic system. 5. Robotics. I. Nakagawa, Elisa Yumi, orient. II. Oquendo, Flavio, orient. III. Título.

Lucas Bueno Ruas de Oliveira

Architectural design of service-oriented robotic systems

Doctoral dissertation submitted to the *Instituto de Ciências Matemáticas e de Computação* - ICMC-USP and the *Institut de Recherche en Informatique et Systèmes Aléatoires* - IRISA-UBS, in partial fulfillment of the requirements for the degrees of Doctorate in Science and *Docteur en Informatique* in the agreement between the Graduate Program in Computer Science and Computational Mathematics (USP) and the Doctoral Program in Mathematics and Information and Communication Science and Technology (UBS). *FINAL VERSION*

Concentration Areas: Computer Science and Computational Mathematics / Mathematics and Information and Communication Science and Technology

Advisor: Prof. Dr. Elisa Yumi Nakagawa
(ICMC-USP, Brazil)
Advisor: Prof. Dr. Flavio Oquendo
(IRISA-UBS, France)

ICMC-USP, Brazil / IRISA-UBS, France
July 2015

To my mother, Rosa Maria Bueno.
To my love, Juliana M. Menegussi.

À minha mãe, Rosa Maria Bueno.
Ao meu amor, Juliana M. Menegussi.

Acknowledgements

I would never have been able to finish my thesis without the guidance of my advisors, help from friends and colleagues, and support from my family. Particularly, I would like to thank:

Professor Elisa Yumi Nakagawa, for the knowledge, patience, and trustworthiness in advising this work. You are an example of kindness and commitment.

Professor Flavio Oquendo, from the Université de Bretagne-Sud (UBS), France, for his wisdom and precious advices during the conduction of this work, and also to Andressa for her cordial welcome and support.

My mother Rosa Maria Bueno, for her devotion to my education and constant love. This work is also yours.

My love, Juliana, for understanding and inspiring me everyday. This thesis is dedicated to you!

My family and family-in-law for their example, good wishes, and words of encouragement. In particular, Ivone, Michelle, Eduardo, Lourdes, Hideyo, Mario Menegussi, and Maria José Menegussi.

The colleagues of START and ArchWare teams, for the cooperation and enriching discussions about Software Architecture.

My friends of the Software Engineering Laboratory (ICMC/USP), for the partnership, talks, and coffee time. They are so many that I do not want to make the mistake of leaving somebody out. Therefore, I am not risking listing all of them here.

The Mobile Robotics Laboratory (ICMC/USP), in particular, my friend Professor Fernando Osório, who taught me about altruism and robotics.

The friends who have reviewed this thesis: Angela Giampedro, Pamela Carreño, Arthur Brenaut, Rafael Durelli, Draylson de Souza, Faimison Porto, André Oliveira, Daniel Soares, Brauner Oliveira, Lina Garcés Rodriguez, and Valdemar Neto.

The people that welcomed me with open arms in Vannes. In particular, Salma Hamza, Petra Bosilj, Pamela Carreño, Arthur Brenaut, Romain Liron, Alan Srey, Armel Esnault, Abdulkader Benchi, Ali Makke, and Jean-François Kamp.

My dear friends Gustavo, Guima, Eloy, and Marina.

The ICMC/USP and its employees for the efficiency and continuous support.

All teachers and professors that contributed to my education.

Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (grant 2011/06022-0) for financially supporting this doctoral project and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (grant 239981/2012-0) for funding the time I spent abroad.

Agradecimentos

Eu não seria capaz de concluir essa tese sem as direções dadas pelos meus orientadores, a ajuda de amigos e colegas, bem como o apoio da minha família. Em particular, eu gostaria de agradecer:

A professora Elisa Yumi Nakagawa, pelo conhecimento, paciência e confiança na orientação deste trabalho. Você é um exemplo de gentileza e comprometimento.

Ao professor Flavio Oquendo, da Université de Bretagne-Sud (UBS), França, pelo seu conhecimento e preciosas orientações durante a condução deste trabalho, bem como a Andressa pela recepção cordial em Vannes e o apoio durante minha visita.

A minha mãe, Rosa Maria Bueno, por sua devoção a minha educação e constante amor. Esse trabalho também é seu.

Ao meu amor, Juliana, pela compreensão e por me inspirar todos os dias. Essa tese é dedicada a você!

A minha família, pelo pensamento positivo e palavras de encorajamento. Em especial, Ivone, Michelle, Eduardo, Lourdes, Hideyo, Mario Menegussi e Maria José Menegussi.

Aos colegas dos times START e ArchWare, pela colaboração e discussões enriquecedoras sobre Arquitetura de Software.

Aos meus amigos do Laboratório de Engenharia de Software (LabES), pelo companheirismo, bate-papos e pela hora do café. São tantos que eu nem me arrisco a listar.

Ao Laboratório de Robótica Móvel (LRM), em especial, ao meu amigo e Professor Fernando Osório, que me ensinou muito sobre altruísmo e robótica.

Aos amigos que me ajudaram na revisão desta tese: Angela Giampedro, Pamela Carreño, Arthur Brenaut, Rafael Durelli, Draylson de Souza, Faimison Porto, André Oliveira, Daniel Soares, Brauner Oliveira, Lina Garcés Rodriguez e Valdemar Neto.

As pessoas que me receberam de braços abertos em Vannes. Em especial, Salma Hamza, Petra Bosilj, Pamela Carreño, Arthur Brenaut, Romain Liron, Alan Srey, Armel Esnault, Abdulkader Benchi, Ali Makke e Jean-François Kamp.

Aos queridos amigos Gustavo, Guima, Eloy e Marina.

A todos os professores que contribuíram na minha educação.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (processo N. 2011/06022-0) por apoiar financeiramente este projeto de doutorado e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (processo N. 239981/2012-0) pelo financiamento do estágio no exterior.

"There are men who struggle for a day and they are good.
There are men who struggle for a year and they are better.
There are men who struggle many years, and they are better still.
But there are those who struggle all their lives:
these are the indispensable ones."

Berthold Brecht

Abstract

Robotics has experienced an increasing evolution and interest from the society in recent years. Robots are no longer produced exclusively to perform repetitive tasks in factories, they have been designed to collaborate with humans in several important application domains. Robotic systems that control these robots are therefore becoming larger, more complex, and difficult to develop. In this scenario, Service-Oriented Architecture (SOA) has been investigated as a promising architectural style for the design of robotic systems in a flexible, reusable, and productive manner. Despite the existence of a considerable amount of Service-Oriented Robotic Systems (SORS), most of them have been developed in an ad hoc manner. The little attention and limited support devoted to the design of SORS software architectures may not only hamper the benefits of SOA adoption, but also reduce the overall quality of robotic systems, which are often used in safety-critical contexts. This thesis aims at improving the understanding and systematization of SORS architectural design. It describes a taxonomy of services for the robotics domain, as well as proposes a process and a reference architecture that systematize the design of SORS software architectures. Results achieved in the evaluation studies evidence that both process and reference architecture can positively impact on the quality of SORS software architectures and, as a consequence, contribute to the development of robotic systems.

Keywords: Software architecture; Service-oriented architecture; Reference architecture; Robotic system; Robotics.

Resumo

A robótica tem passado por uma notável evolução ao longo dos últimos anos, juntamente com um crescente interesse por parte da sociedade. Robôs não são mais exclusivamente produzidos para realizar atividades repetitivas em fábricas, eles têm sido projetados para apoiar humanos em diversos e importantes domínios de aplicação. Os sistemas robóticos utilizados para controlar tais robôs têm, portanto, se tornado maiores, mais complexos e difíceis de desenvolver. Nesse cenário, a Arquitetura Orientada a Serviços (do inglês, *Service-Oriented Architecture* - SOA) tem sido investigada como um promissor estilo arquitetural para o desenvolvimento de sistemas robóticos de forma mais flexível, reusável e produtiva. Embora um número considerável de Sistemas Robóticos Orientados a Serviços (do inglês, *Service-Oriented Robotic Systems* - SORS) já exista, grande parte deles têm sido desenvolvida de maneira *ad hoc*. A pouca atenção e o suporte limitado ao projeto das arquiteturas de software de SORS pode não só impedir a obtenção dos benefícios associados à adoção da SOA, mas também reduzir a qualidade dos sistemas robóticos que, frequentemente, são utilizados em contextos de segurança crítica. Essa tese tem por objetivo aprimorar o entendimento e a sistematização do projeto arquitetural de SORS. Para isso, é proposta uma taxonomia de serviços para o domínio de robótica, bem como um processo e uma arquitetura de referência para sistematizar o projeto das arquiteturas de software de SORS. Os resultados obtidos evidenciam que tanto o processo quanto a arquitetura de referência podem impactar positivamente na qualidade das arquiteturas de software de SORS e, consequentemente, contribuir para o desenvolvimento de sistemas robóticos.

Palavras-chave: Arquitetura de software; Arquitetura orientada a serviços; Arquitetura de referência; Sistemas robóticos; Robótica.

Résumé

La Robotique a connu une évolution remarquable au cours des dernières années, couplée à un intérêt croissant de la société pour ce domaine. Des robots ne sont plus fabriqués exclusivement pour effectuer des tâches répétitives dans les usines, mais ils sont aussi créés pour collaborer avec les humains dans plusieurs domaines d'application d'importance. Les systèmes robotiques qui contrôlent ces robots sont donc de plus en plus larges, complexes et difficiles à développer. Dans ce contexte, l'Architecture Orientée Services (SOA) a été identifiée comme un style d'architecture logiciel prometteur pour concevoir des systèmes robotiques de manière flexible, réutilisable, et productive. Cependant, malgré le nombre considérable de Systèmes Robotiques Orientées Services (SORS) existants aujourd'hui, la plupart d'entre eux ont été développés de manière ad hoc. Le peu d'attention et le soutien limité portés à la conception d'architectures logicielles SORS peuvent non seulement masquer les avantages de l'adoption de la SOA, mais aussi réduire la qualité globale des systèmes robotiques, qui sont souvent utilisés dans des contextes de sécurité critiques. Cette thèse vise à améliorer la compréhension et la systématisation de la conception architecturale SORS. Elle décrit une taxonomie des services pour le domaine de la robotique, puis propose une processus ainsi qu'une architecture de référence afin de systématiser la conception d'architectures logicielles SORS. Les résultats obtenus dans les études d'évaluation montrent qu'à la fois la processus et l'architecture de référence peuvent avoir un impact positif sur la qualité des architectures logicielles SORS et, par conséquent, contribuent à l'amélioration des systèmes robotiques.

Mots-clés: Architecture logiciel; Architecture orientée services; Architecture de référence; Systèmes robotiques; Robotique.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ADL	-	Architecture Description Language
AGV	-	Automated Guided Vehicle
BPMN	-	Business Process Model and Notation
CBSE	-	Component-Based Software Engineering
DAO	-	Data Access Object
ESB	-	Enterprise Service Bus
GPS	-	Global Positioning System
HAL	-	Hardware Abstraction Layer
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Hypertext Transfer Protocol Secure
MDE	-	Model-Driven Engineering
MVC	-	Model-View-Controller
OO	-	Object-Oriented
QoS	-	Quality of Service
REST	-	Representational State Transfer
RPC	-	Remote Procedure Call
SLA	-	Service Level Agreement
SLAM	-	Simultaneous Localization and Mapping
SMTP	-	Simple Mail Transfer Protocol
SOA	-	Service-Oriented Architecture
SORS	-	Service-Oriented Robotic Systems
SoS	-	Systems-of-Systems
SPL	-	Software Product Line
SPEM	-	Software & Systems Process Engineering Metamodel Specification
UAV	-	Unmanned Aerial Vehicle
UDDI	-	Universal, Discovery, Description and Integration
URI	-	Uniform Resource Identifier
WS-BPEL	-	Web Services Business Process Execution Language
WS-CDL	-	Web Services Choreography Description Language
WSDL	-	Web Service Description Language
XML	-	Extensible Markup Language

Chapter

1

Introduction

Over the past decades, robots have been evolving from assembly-line devices to machines able to cooperate with or even replace humans in several dangerous, tedious, and error-prone activities. Recent advancements on hardware and software technologies have enabled the development of robots that support daily activities inside hospitals (Swisslog, 2015; Takahashi et al., 2010), houses (Husqvarna, 2015; iRobots, 2015a,b), and on the streets (Fernandes et al., 2014; Thrun et al., 2006). The potential of robotics for improving quality of live and productivity has motivated both academia and industry in investing in robots of higher autonomy and decision making capacity. Consequently, the robotic systems that control these robots have grown in size and complexity, demanding more mature practices of development. In this perspective, the design of software architectures able to accommodate such complexity and, at the same time, improve the quality of robotic systems turned into a key concern of robotics (Fluckiger and Utz, 2014). Presently, the study of the architectural design of robotic systems is considered an important research field in robotics (Brugali and Scandurra, 2009; Fluckiger and Utz, 2014).

Software architecture forms the backbone of any successful system and plays a fundamental role in determining quality (Shaw and Clements, 2006). It represents the structure or structures of the system, which comprise software elements, the externally visible properties of these elements, and the relationships among them (Bass et al., 2012). A software architecture is a collection of explicit architectural design decisions made about the software system over time (Jansen and Bosch, 2005). Decisions made at the architectural level directly impact on the realization of functional and quality requirements of software systems (Shaw and Clements, 2006). Therefore, both functional and quality characteristics must be considered, designed, and evaluated at the architectural level (Bass et al., 2012; Clements et al., 2002). In this context, the use of systematic processes associated with reference models and reference architectures can facilitate and improve the design of software architectures.

Reference architectures are important assets that guide the design of a range software architectures of given application domain (Bass et al., 2012). They promote reuse of design expertise by encompassing best practices of development and knowledge on how to structure software elements that support the creation of systems in such a domain (Nakagawa and Oquendo, 2012). Reference architectures are recognized as architectural blueprints, since they foster standardization, improve communication among stakeholders, and also reduce costs of development (Arsanjani et al., 2007; Cloutier et al., 2010; Nakagawa and Oquendo, 2012). Many companies and research institutes have already designed and adopted reference architectures for different application domains (Arsanjani et al., 2007; Nakagawa et al., 2011b; Oliveira and Nakagawa, 2011; The Open Group, 2015). In particular, several examples of reference architectures are already available for the development of embedded systems (AUTOSAR, 2015; Eklund et al., 2005; Eklund and Bosch, 2014; UniversAAL Project, 2015) and robotic systems (Albus, 2002; Alvarez et al., 2001; Clark, 2005; Heisey et al., 2013; Nakagawa et al., 2014; Ortiz et al., 2005; Weyns and Holvoet, 2006).

Another important asset that impacts on the development of software architectures is the architectural style. Nowadays, several software systems are created by the composition of existing modules and subsystems implemented by different technologies. In this scenario, Service-Oriented Architecture (SOA) is an architectural style traditionally used in the industry for the integration of heterogeneous, distributed software capabilities provided by multiple organizations (Erl, 2005; Josuttis, 2007). Software capabilities are encapsulated as services, which are self-contained, well-defined modules capable of providing business functionalities independently of the state and context of other services (Erl, 2005; Papazoglou and Heuvel, 2007). Functionalities of a service are offered through auto-descriptive standard interfaces and can be published in third-party repositories to be discovered by service consumers. Therefore, SOA can foster reuse and improve productivity of software systems development (Papazoglou et al., 2008).

SOA has also been recently investigated as a solution to produce more flexible, reconfigurable, and scalable software for robotic systems. The use of SOA is supporting developers to overcome traditional problems of robotics design, including integration of off-the-shelf hardware devices and reuse of complex software modules (Berná-Martínez and Maciá-Pérez, 2010; Fluckiger and Utz, 2014; Veiga et al., 2009). The adoption of this architectural style also enables integration of robots as part of larger systems (Waibel et al., 2011; Yang and Lee, 2013b), adaptation of robot's behaviour at runtime (Tenorth et al., 2011), and use of additional sources of knowledge in robotics (e.g., the Internet) (Blake et al., 2011). Important companies and research institutes are investing on the creation of Service-Oriented Robotic Systems (SORS). For instance, Microsoft Robotics Developer Studio (MSDS) (Jackson, 2007) and Robot Operating System (ROS) (Straszheim et al., 2011) are consolidated environments that currently support the development of robotic systems based on SOA.

1.1 Problem Statement and Justification for the Research

The potential of SOA for robotics has motivated researchers to develop their robotic systems as collections of services. Nowadays, there is an increasing number of studies reporting the design of SORS (Berná-Martínez and Maciá-Pérez, 2010; Cepeda et al., 2011; Doriya et al., 2012b; Insaurralde and Petillot, 2015; Koubaa, 2014; Raffaeli et al., 2012). The development of such systems has produced hundreds of services for the control of hardware devices, robotic navigation, probabilistic localization, and other complex functionalities associated with the creation of robots[1]. These services represent an important support for the development of future projects and can contribute to reduce complexity and time-to-market of new robots.

However, the use of SOA in robotics is considerably recent if compared to other architectural styles. As a consequence, most of SORS are still developed in different manners, without a common understanding on how and which software modules should be provided as services. The lack of consensus and unified terminology hampers the discovery of services available for reuse and their integration into other projects, reducing the capacity of SOA in fostering productivity in the development of SORS. In this perspective, research focusing on the establishment of a well-accepted vocabulary and classification of services for SORS can contribute to consolidate SOA in the domain of robotics. Furthermore, such a conceptual basis may pave the way for the creation of mechanisms able to facilitate publication and discovery of services for SORS.

Important advancements in technology over the past years have enabled the creation of robotic systems by the composition of services. Nevertheless, little attention has been paid to the design of their software architectures. Results of a systematic literature review[2] (Oliveira et al., 2013b) revels that most of software architectures for SORS are currently

[1]http://www.ros.org/browse/list.php, last accessed in March 25th, 2015.
[2]More details can be found in Chapter 2, Section 2.4.5.

designed in *ad hoc* manner, without considering an adequate approach of development, which can hinder the construction, maintenance, and reuse of these systems. Improvement in the design of SORS software architectures is a key concern, as robots have been used in safe-critical domains, where failures may cause non-recoverable financial losses and serious damage to the human lives, environment, and expensive equipments.

Contributions that facilitate the design of software architectures are necessary to increase quality and productivity in the development of SORS. The adequate design of a software architecture is fundamental for the achievement of system goals, functional and quality requirements (Bass et al., 2012). Traditional approaches, as those for object-oriented software, do not adequately support the design of SOA-based systems (Arsanjani et al., 2008). On the other hand, approaches for the development of service-oriented systems do not consider particularities of robotics design, such as limited processing capability and real-time constraints. Therefore, a systematic process for designing SORS software architectures can positively impact on the overall quality of these systems. In addition, reference architectures encompassing the knowledge on how to design SORS would ease the application of this process and, consequently, produce systems of higher quality. However, as far as we know, no process or reference architecture is available in the literature to support the design of robotic systems based on SOA.

1.2 Objectives

According to the research gap characterized in the previous section, the general research question to be investigated in this thesis is whether or not the systematization of the architectural design can positively impact on the quality of SORS software architectures. Based on this general research question, we defined the following objectives:

- *Definition of a taxonomy of service for SORS:* we intent to investigate types of services used in the development of SORS, the organization of these services, and their contexts of application. Based on our observations, knowledge of experts, and guidelines available for robotics and SOA, we aim at defining a taxonomy of services for the development of SORS to improve understanding among researchers and practitioners of this domain;

- *Automation of the classification and discovery of services for SORS:* we aim at using the taxonomy of services to create a mechanism that automates publication, classification, and transparent discovery of services for the development of SORS. By creating this mechanism, we intent to facilitate the reuse of services and, therefore, improve productivity in the development of robotic systems based on SOA;

- *Definition of a process for the design of SORS software architectures:* our main objective in this thesis is to propose a process to support the systematic design of SORS software architectures. Through this process, we aim at providing prescriptive guidance to the architectural design of SORS in order to increase the quality of the resulting software architectures and, as a consequence, support the development of these systems; and

- *Establishment of a reference architecture for SORS:* we aim at establishing a reference architecture for supporting the design of SORS software architectures created by the application of the process proposed in this thesis. In the definition of such reference architecture, we plan to focus on the indoor grounded mobile SORS, as they represent the most common type of robotic system developed using SOA.

The next section addresses contributions of this thesis in accordance with the proposed objectives. It also presents the organization of this doctoral dissertation and the structure of its chapters.

1.3 Thesis Outline and Summary of Contributions

Chapter 2 brings an overview of the background information that supports the topics investigated in this thesis. Initially, terminology and key concepts related to software architecture are discussed. Then, theory associated with the SOA architectural style and the main technologies for developing service-oriented systems are addressed. After that, an introduction to robotics and main characteristics of the development of robotic systems are presented. The chapter also summarizes results of a systematic literature review study that characterizes the state-of-the-art on the development of SORS. This systematic review was published in (Oliveira et al., 2013b) and updated for the inclusion in this thesis.

Chapter 3 describes two complementary works on the classification of services for the development of SORS. The first one, published in (Oliveira et al., 2014c), reports the establishment of a taxonomy created to improve communication among developers and enable classification of services for SORS. Results of a survey applied to specialists in robotics indicate this taxonomy is comprehensive enough to classify services that can be used to develop SORS. The second work, published in (Oliveira et al., 2014b) and later extended in (Oliveira et al., 2015), presents the design and implementation of a tool that uses the taxonomy to enable publication, classification, and transparent discovery of services for SORS. Preliminary results of a case study suggest this tool facilitates the discovery and reuse of services at design time, which may foster productivity in the development of robotic systems.

Chapter 4 introduces ArchSORS (Architectural Design of Service-Oriented Robotic System), a process to support and systematize the design of SORS. ArchSORS encompasses a set of methods that describe key activities, roles, and document deliverables to the development of software architectures for robotic systems based on SOA. We conducted an experimental study to evaluate the influence of this process on the quality of SORS software architectures. Results show evidences that ArchSORS can produce architectures of higher modularity and cohesion, which are also less coupled and centred in coarse-grained services. Improvements in these characteristics suggest that ArchSORS can positively impact on the quality of SORS software architectures and, therefore, benefit the development of such systems. This chapter extends content and results reported in (Oliveira et al., 2014a).

Chapter 5 describes the establishment of RefSORS (Reference Architecture for Service-Oriented Robotic Systems), a reference architecture for indoor grounded mobile SORS. RefSORS encompasses the knowledge on how to model and compose services identified in the taxonomy (Chapter 3) to design software architectures for this particular type of robotic system. We conducted a case study on the development of a multi-robot system to illustrate the application of RefSORS in conjunction with the ArchSORS process (Chapter 4). Results indicate that the software architecture designed by the instantiation of RefSORS presents better modularity, coupling, and cohesion, if compared to the ones designed only using ArchSORS.

Finally, Chapter 6 concludes this thesis, revisiting the achieved contributions, summarizing limitations, and presenting perspectives of future research. The list of publications resulting from this work is presented in Appendix C.

Chapter

2

Background

2.1 Overview

This chapter provides an overview of the subjects that underlie the research developed in this thesis. The organization of the chapter is as follows. Section 2.2 introduces the terminology and main concepts of software architecture. Section 2.3 presents the SOA architectural style and its implementation technologies. Section 2.4 describes the general structure, development environments, and design approaches of robotic systems. Particularly, Section 2.4.5 characterizes the state-of-the-art of the development of Service-Oriented Robotic Systems (SORS) according to the results of an updated version of a systematic literature review initially published in (Oliveira et al., 2013b).

2.2 Software Architecture

Software architecture plays an essential role in determining system quality as it forms the backbone of any successful software-intensive system (Shaw and Clements, 2006). Decisions made at the architectural level directly impact on the achievement of business goals, functional and quality requirements (Bosch, 2000). According to Bass et al. (2012), software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. An architecture focuses on the representation of element interfaces and hides internal details on how such elements are implemented (Bass et al., 2012). Additionally, Jansen and Bosch (2005) define software architecture as a collection of explicit architectural design decisions made over time.

2.2.1 Terminology

The first reference to the “Software Architecture” phrase emerged in 1969 (Kruchten et al., 2006). From then until the late 1980s, the word architecture was mostly associated with system architectures, in the sense of hardware structure of a system (Kruchten et al., 2006). The concept of software architecture as it is understood nowadays only arose in the early 1990s. In the past two decades, several other terms associated with software architecture have been defined and used in the literature. The comprehension of these terms is fundamental for the establishment of a unified vocabulary and improvement in communication among practitioners and researchers of software architecture. Comprehensive lists of terms and concepts related to software architecture are available in (SEI, 2015c) and (Eeles, 2008). The most important terms in the context of the presented research are described as follows:

Stakeholder: refers to any individual, team or organization that is interested in a software system and plays a relevant role during its development process (ISO/IEC/IEEE, 2011). Examples of stakeholders include clients, users, software architects, software engineers, developers, and testers;

Concern: is an interest or need of one or more stakeholders on the software system under development (ISO/IEC/IEEE, 2011). Concerns can be, for instance, functionalities, qualities, purposes of use, and costs;

Concrete software architecture: is the software architecture of a given software system tailored for its particular set of stakeholders and concerns;

Architectural pattern: expresses fundamental structural organization schema for software systems. An architectural pattern describes a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for the organization of the relationships among them (Buschmann et al., 1996). According to Eeles (2008), an architectural pattern not only provides the structure of a solution for an usual problem in software design, but also describes the context in which such a problem may occur and the consequences associated with the pattern adoption. The Layers pattern, defined by Buschmann et al. (1996), is a well-known example of an architectural pattern that enables a software solution to be divided into groups of behaviour and separated into layers;

Architectural style: is a type of architectural pattern that can be used to start the process of moving from requirements to solution (Eeles, 2008). An architectural style defines a family of systems in terms of a pattern of structural organization, establishes vocabulary of components and connector types, and a set of constraints on how they can be combined (Shaw and Garlan, 1996). Examples of architectural styles include Client-server and Pipe-and-filters (Bass et al., 2012). The Client-server style defines the physical separation of client-side and server-side processing and enforces communication by protocol. Pipe-and-filter defines data transformation in a system through filters and communication among such filters via pipes;

Reference model: is an abstract representation of the elements in a given domain of interest, the behaviour of such elements, and the relationships among them (Bass et al., 2012). A reference model typically forms the conceptual basis for the development of more concrete elements, such as reference architectures (Eeles, 2008). OASIS (2006) is a well-known example of a reference model that defines the essence of SOA regardless of technology, standard or development approach;

Reference architecture: is a reference model mapped onto elements of software that cooperatively implement the functionalities defined in such a model (Angelov et al., 2009; Bass et al., 2012). It encompasses the knowledge on how to design concrete architectures of systems of a given application domain or technological domain (Nakagawa et al., 2011a). OASIS (2012) is an example of a reference architecture for SOA designed according to a reference model (OASIS, 2006). Section 2.2.2 provides additional details and examples of reference architectures as this topic is related to one of the contributions of this research;

Product line architecture: is an abstract software architecture for describing elements of a family of similar products developed by the Software Product Line (SPL) approach (Clements and Northrop, 2002). Product line architectures represent the kernel, optional, and variable components in the SPL, and their interconnections (Gomaa, 2004). The literature provides several examples of SPL and product line architectures that range from medical systems to avionics (SEI, 2015b).

The definition of these terms is important to avoid misconceptions, since some of them are frequently applied as synonyms. For instance, reference architectures are often used to refer to reference models, although the latter is more general than the former (Angelov et al., 2009; Bass et al., 2012; Nakagawa et al., 2014). Product line architectures are also mistakenly referred to as reference architectures, albeit the focus of product line architecture is on describing variabilities of a particular subset of software systems of a given domain (Nakagawa et al., 2011a). Despite reference models, architectural patterns, and reference architectures aim at facilitating the capture of elements of software architectures, they play clear and distinct roles in the architectural design (Bass et al., 2012). Figure 2.1 illustrates the relationship among reference model, architectural pattern, reference architecture, and concrete architectures.

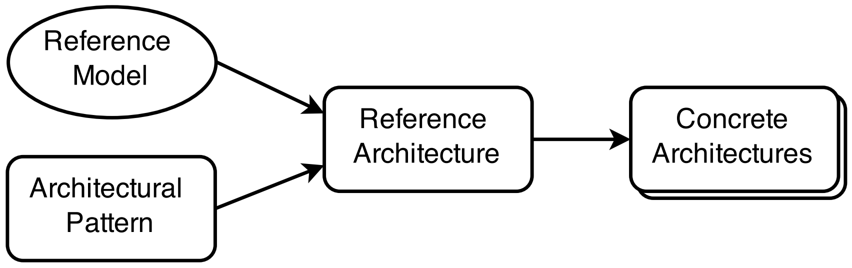


Figure 2.1: Relationship among reference model, architectural pattern, reference architecture, and concrete architectures (Bass et al., 2012)

2.2.2 Reference Architecture

Reference architecture has emerged as an important area of research in software architecture. It is considered a blueprint of software development, since it guides the design of concrete architectures of systems for a given application domain (Angelov et al., 2008; Muller, 2008; Nakagawa and Oquendo, 2012). Reference architectures can directly impact on the quality and design of a range of concrete architectures and software systems developed from them (Angelov et al., 2009). Therefore, they must consider business rules, architectural styles, best practices of software development, and software elements that support the design of systems of the application domain (Nakagawa et al., 2011a). Furthermore, reference architectures must be supported by a unified, unambiguous, and widely understood domain terminology. In this perspective, RAModel (Reference Architecture Model) (Nakagawa and Oquendo, 2012) provides a complete list of elements (and their relationships) that may be considered for engineering reference architectures.

Different institutions in both academia and industry have already proposed and used reference architectures in several application domains. There are examples of reference architectures designed for software engineering tools (Nakagawa et al., 2011b, 2007; Oliveira and Nakagawa, 2011), service-oriented systems (Arsanjani et al., 2007; OASIS, 2012; Oliveira et al., 2010; Oliveira and Nakagawa, 2011; The Open Group, 2015; Zimmermann et al., 2009), embedded systems (AUTOSAR, 2015; Batory et al., 1995; Eklund et al., 2005; Eklund and Bosch, 2014; UniversAAL Project, 2015), robotic systems (Albus, 2002; Alvarez et al., 2001; Clark, 2005; Hayes-Roth et al., 1995; Heisey et al., 2013; Nakagawa et al., 2014; Ortiz et al., 2005; Weyns and Holvoet, 2006), and so forth. However, no reference architecture has focused on the development of robotic systems based on SOA.

According to Muller (2008), a reference architecture can be used to facilitate the design of concrete architectures or as a standardization asset that supports interoperability among systems or components of systems. As shown in Figure 2.2, the same reference architecture can result in different concrete architectures, depending on the context and involved stakeholders. Due to the diversity of application domains and interests, reference architectures can be classified according to three dimensions, described as follows (Angelov et al., 2012, 2009):

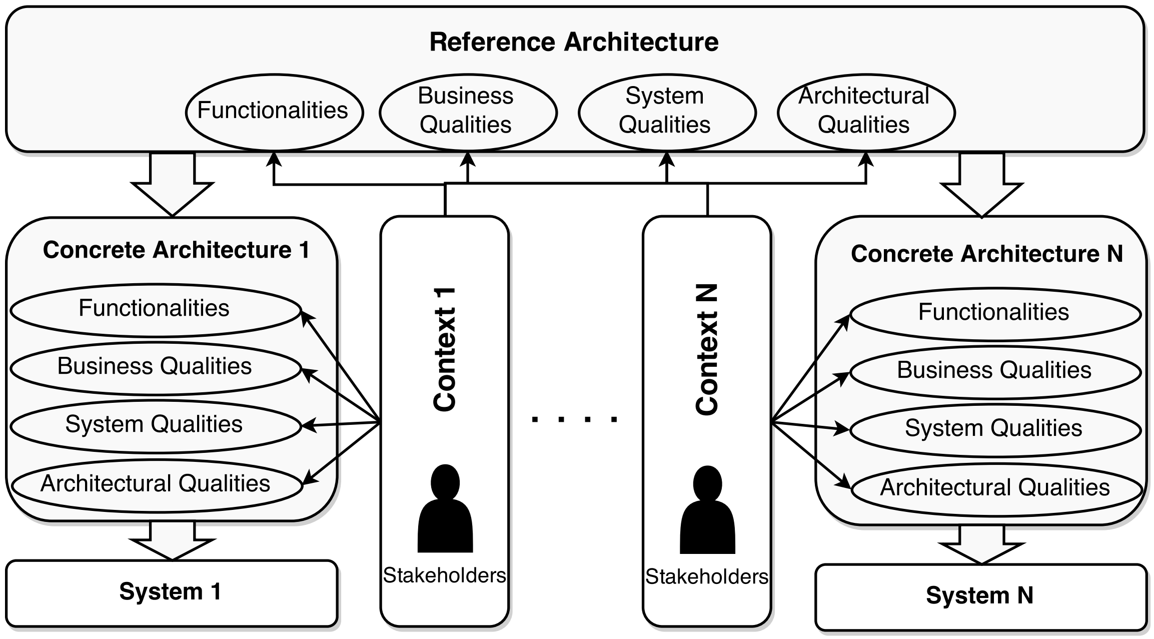


Figure 2.2: Role of stakeholders and contexts for reference and concrete architectures (Angelov et al., 2008)

Context dimension: reference architectures can be designed in the context of a *single organization* or *multiple organizations* that share a common characteristic, such as geographical location and market domain. Different types of organizations (e.g., *software organizations*, *user organizations*, *research centres*, and *standardization organizations*) are usually involved in the establishment of these architectures. Besides, such architectures can be designed before any existing systems (i.e., *preliminary*) or after accumulating the experience from the development of several systems (i.e., *classical*);

Goal dimension: as stated by Muller (2008), reference architectures can be designed with two main goals: *standardization* and *facilitation*. Reference architectures for *standardization* aim at improving interoperability among systems by promoting unified understanding of the domain at the architectural level. On the other hand, *facilitation* reference architectures aim at providing guidelines for the development of concrete architectures; and

Design dimension: reference architectures are represented by several types of elements, including components, interfaces, protocols, algorithms, policies, and guidelines. These elements can be described in different levels of detail (*detailed*, *semi-detailed*, and *aggregated*), formalism (*informal*, *semi-formal*, and *formal*), and abstraction (*concrete*, *semi-concrete*, and *abstract*).

The design and evaluation of reference architectures are often associated with their type. Whereas reference architectures designed in the classical manner usually involve software organizations, user organizations, and standardization organizations, preliminary reference architectures are frequently established by research centers (Angelov et al., 2012; Muller, 2008). Moreover, results and benefits are easier to estimate in classical reference architectures than in the preliminary ones (Angelov et al., 2008).

Since reference architectures can be the basis for several software systems, studies have focused on the design of this special type of architectures. Muller (2008) discussed a set of recommendations to create and maintain reference architectures. Angelov et al. (2012) described a framework for designing and analysing reference architectures. Galster and Avgeriou (2011) proposed a procedure to design and evaluate empirically-based reference architectures. In a more specific context, Dobrica and Niemela (2008) described an approach for designing reference architectures for embedded systems. Although these studies have provided important guidelines for the development of reference architectures, they either lack maturity or do not provide a detailed and general process for building such architectures. In this perspective, Nakagawa et al. (2014) proposed ProSA-RA, a process that systematizes the design, representation, and evaluation of reference architectures. ProSA-RA, illustrated in Figure 2.3, is divided into four steps:

- **Step RA-1: Information Source Investigation:** The main sources of information are investigated to help the understanding of the processes, activities, and tasks that could be supported by software systems in the domain. Examples of relevant sources of information include stakeholders (e.g., customers, users, and researchers), existing systems, publications (e.g., articles, books, and technical reports), reference models, other reference architectures, and ontologies and taxonomies of the domain;

- **Step RA-2: Architectural Analysis:** Information about the domain is obtained and three artifacts are established. Firstly, the requirements of software systems of

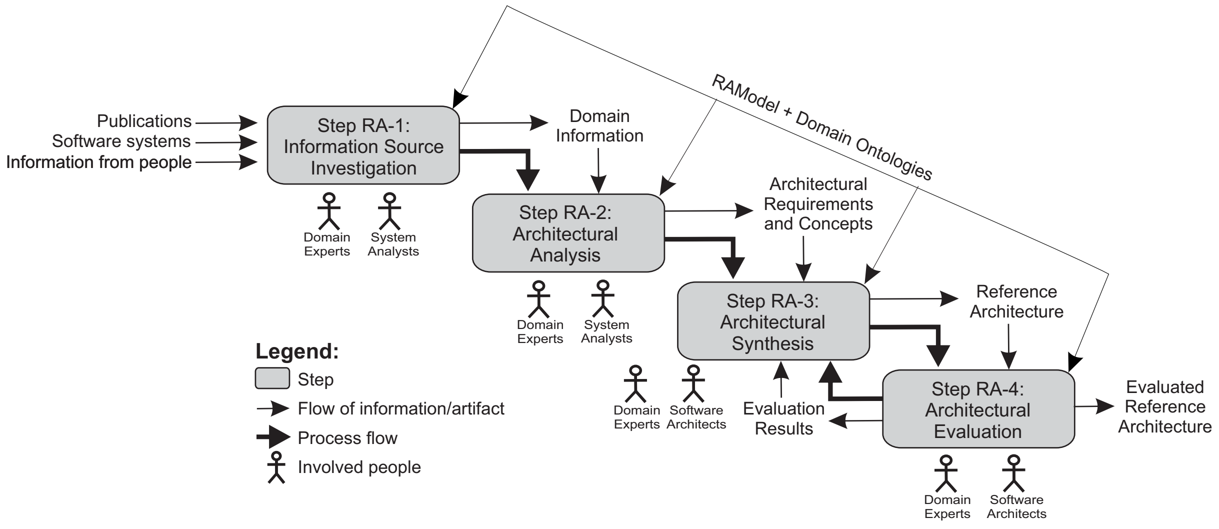

RAModel + Domain Ontologies
Publications
Software systems
Information from people
Step RA-1: Information Source Investigation
Domain Information
Domain Experts
System Analysts
Step RA-2: Architectural Analysis
Architectural Requirements and Concepts
Domain Experts
System Analysts
Step RA-3: Architectural Synthesis
Reference Architecture
Domain Experts
Software Architects
Evaluation Results
Step RA-4: Architectural Evaluation
Evaluated Reference Architecture
Domain Experts
Software Architects
Legend:
Step
Flow of information/artifact
Process flow
Involved people

Figure 2.3: Outline structure of ProSA-RA (Nakagawa et al., 2014)

the domain are elicited using the selected sources. Based on the requirements of the systems, a set of requirements for the reference architecture is then identified. After that, the set of concepts that must be considered in the reference architecture is established;

- **Step RA-3: Architectural Synthesis:** The architectural description of the reference architecture is created with the support of RAModel framework (Nakagawa and Oquendo, 2012). Different views of the reference architecture are described according to architectural styles and patterns probably identified in Step RA-1. These styles and patterns are the basis on which concepts of the reference architecture are organized. For instance, if SOA is used as an architectural style of the reference architecture, concepts are provided as independent services and communicate via standard interfaces and protocols; and

- **Step RA-4: Architectural Evaluation:** Finally, the reference architecture is evaluated for the identification of possible problems of omission, ambiguity, inconsistency, and missing information. For this, inspection checklists such as FERA (Framework for evaluation of reference architectures) can be applied (Santos et al., 2013). Results of the evaluation are used to improve the architectural description of reference architectures by performing new iterations of Step RA-3.

ProSA-RA is currently the most mature process available for establishing reference architectures. It has already been applied for the development of several reference architectures, including for robotics (Nakagawa et al., 2014). Therefore, we adopted ProSA-RA as the process for establishing the reference architecture for robotic systems based on SOA described in Chapter 6.

2.2.3 Architecture Description

Architecture description is considered the main artifact in expressing software architectures, as it plays a key role in communication, quality assessment, evaluation, and evolution (Clements et al., 2010; ISO/IEC/IEEE, 2011; Kruchten, 2009). Defining an adequate architecture description is fundamental for the effective use of software architectures during the life-cycle of software systems. From this perspective, the ISO/IEC/IEEE 42010:2011 standard (ISO/IEC/IEEE, 2011) specifies the manner in which architecture descriptions of systems should be organized and expressed. It presents the elements of the architecture description, addressing the creation, analysis, and sustenance of architectures of software intensive systems. The main elements for constructing architecture descriptions addressed by ISO/IEC/IEEE 42010:2011 standard are described as follows:

Model kind: defines conventions for a given type of modelling by describing rules, meta-models, templates, and operations for representing the relationships and information in an architecture description. Examples of model kinds include data flow diagrams (Gane and Sarson, 1977), class diagrams (OMG, 2015e), statecharts (Harel, 1987), Petri nets (Peterson, 1977), and state transition models (Gill, 1962);

Architecture viewpoint: establishes the conventions for the construction, interpretation, and use of architecture views for framing specific system concerns. In other words, a viewpoint formalizes the idea that there are different ways of looking at the same system;

Architecture view: addresses one or more concerns of stakeholders of the system in accordance with an architecture viewpoint. Architecture views are composed of one or more architecture models;

Architecture model: uses modelling conventions appropriate for the concerns to be addressed by one or more architecture views. These conventions are described by the model kind that governs the architecture description;

Architecture rationale: records explanation, justification or reasoning about architecture decisions that have been made. The rationale for a decision can include the basis for this decision, alternatives and trade-offs considered, as well as potential consequences and citations to sources of additional information; and

Correspondence: defines a relation between architecture elements by expressing architecture relations of interest within an architecture description. Correspondences can be governed by correspondence rules used to enforce relations within an architecture description (or between architecture descriptions).

More complex architectural assets, such as reference architectures and architectural styles, can be described based on these concepts. For instance, the SOA architectural style can be described by model kinds and a set of concerns. Besides, graphical and textual languages used to describe architectures can also be expressed in terms of model kinds, stakeholders, concerns, and correspondence rules (ISO/IEC/IEEE, 2011).

An Architecture Description Language (ADL) is any form of expression for use in architecture descriptions that provides one or more model kinds as a means to frame concerns for the audience of stakeholders (ISO/IEC/IEEE, 2011). This definition of ADL proposed by the ISO/IEC/IEEE 42010:2011 standard is broader than traditional definitions in the literature, which describe ADLs as formal languages for representing the architecture of software-intensive systems (Clements, 1996; Medvidovic and Taylor, 2000). Therefore, ADL is a language of any level of formality (formal, semi-formal or informal) that may support the creation, refinement, and validation of software architectures.

Formal ADLs are languages based on formal syntax and semantics, e.g., Wright (Allen, 1997), Rapide (Luckham, 1997), π-ADL (Oquendo, 2004a), π-AAL (Architecture Analysis Language) (Mateescu and Oquendo, 2006), and π-ARL (Architecture Refinement Language) (Oquendo, 2004b). Among the formal ADLs available in the literature, it is important to highlight the complementary languages π-ADL, π-AAL, and π-ARL, since they support the description of the dynamic nature of software architectures. These three languages form the basis of the π-ArchWare architecture-centric approach for developing software intensive systems, including service-based systems and embedded systems. π-ADL is a well-founded language based on the higher-order typed π-calculus (Milner, 1999) that focuses on the description of software architectures from both structural and behavioural viewpoints. Therefore, it can be used to express how these architectures evolve over the time. π-AAL is a well-founded theoretical language based on the modal μ-calculus (Kozen, 1983) that complements the former language by expressing correctness properties. It also provides automated support for the verification of such properties by using an analytical toolset based on theorem proving and model checking techniques. π-ARL is an architecture refinement language based on rewriting logic that supports property-preserving transformations and synthesis application through stepwise refinement. In other words, π-ARL enables incremental refinements in software architectures while keeping the consistency from higher abstract representations to more concrete ones.

Semi-formal ADLs are languages based on formal syntax and well-described semantics, such as AADL (Architecture Analysis and Design Language) (Allen et al., 2002) and UML (Unified Modeling Language) (OMG, 2015e). Semi-formal languages are often supported by graphical notations, which make them easier to be understood by non-technical stakeholders. AADL is an example of a domain-specific ADL suitable for the analysis and specification of complex real-time embedded systems. UML is the industry-standard language for specifying, visualizing, constructing, and documenting artifacts of software architectures. It provides a large set of diagrams that enables the description of structural and behavioural viewpoints of software architectures and has a large tooling support. As UML is widely adopted in both academia and industry, it has also been tailored to several specific areas. For instance, MARTE (Modeling and Analysis of Real-Time and Embedded Systems) (OMG, 2015d) is a UML profile for the model-driven development of embedded systems that adds notations for representing requirements related to schedulability, performance, and time constraints. Similarly, SysML (Systems Modeling Language) (SysML Partners, 2015) is a dialect of UML that supports specification, analysis, design, verification, and validation of systems. Elements of SysML enables the description of hardware, software, information, processes, personnel, and facilities involved in the development of a broad range of systems. In the context of SOA, the SoaML (Service oriented architecture Modeling Language) (OMG, 2015b) profile introduces elements for describing capabilities, service interfaces, contracts, protocols, participants (e.g., service consumer and service provider), as well as the relationship among these elements (for more information about SOA, see Section 2.3).

Informal ADLs are box-and-lines models highly dependent on textual explanations for adding meaning to their elements and relationships (Clements et al., 2010). Components (or boxes) represent modules and subsystems that provide the functionalities of a software system. Connectors (or lines) represent the communication among the components of this system. Informal descriptions are useful for providing an overview of the software system, but may lead to problems associated with ambiguity and misunderstandings. Therefore, it should not be used as the unique ADL of an architecture description (Land, 2002).

The adoption of an ADL during the design of a software architecture involves a trade-off between understandability and precision. Informal and semi-formal ADLs are often easier to understand, which facilitates the communication among technical and non-technical stakeholders. On the other hand, these languages may conceal inconsistence among models that can be propagated to subsequent stages of software development life-cycle (Mens et al., 2010). Formal ADLs increase precision and correctness at the cost of lowering understandability, limiting the use of the architecture description to technical stakeholders. Therefore, while informal and semi-formal languages are more suitable for describing information systems (e.g., desktop systems, Web-based systems, service-based systems), formal languages are frequently applied to the design of safe-critical systems (e.g, embedded real-time systems).

2.2.4 Software Architectures and Software Quality

Software systems have been increasingly used in several areas of society and the demand for these systems has been growing substantially. The achievement of goals and objectives regarding user satisfaction, business success, and human safety depends on high-quality software systems (ISO/IEC, 2011). The development of high-quality software systems is considered even more important in safe-critical domains, such as embedded systems and robotics, where failures may cause non-recoverable financial losses and serious damage to the human lives, environment and expensive equipment (Aguiar et al., 2010).

Throughout the history of software engineering, software quality improvement has been a highly important goal (ISO/IEC, 1999). According to the ISO/IEC 9126:2001 standard (ISO/IEC, 2001), software quality is the totality of features and characteristics of a software product affecting its ability to satisfy stated or implied needs. The achievement of quality attributes must be considered during design time, implementation, and deployment (Bass et al., 2012). Therefore, the adequate design of a software architecture is fundamental for the realization of several quality characteristics in a software system (Bass et al., 2012). In addition, the quality characteristics should also be evaluated at the architectural level.

In this perspective, software quality models have become well-accepted means of describing, managing, and predicting software quality. Over the years, a variety of quality models have been proposed to support the development of general software systems. McCall's Quality Model (McCall et al., 1977), considered the precursor of the actual models, established three major perspectives for defining and identifying the quality of a software product: product revision, product transition, and product operations. Each of these perspectives describes a set of quality attributes that refers to the ability of a software system to undergo changes, adapt to new environments, and adequately perform its functionalities. Similarly, Boehm's Quality Model (Boehm et al., 1976) qualitatively defined software quality by a given set of attributes and metrics. Another important quality model was proposed by the ISO/IEC 9126-1:2001 standard, which incorporated quality goals that encompass a large number of quality attributes.

Nowadays, one of the most important quality models used in software development is the ISO/IEC 25010:2011 standard (ISO/IEC, 2011), which cancelled and replaced its predecessor ISO/IEC 9126-1:2001. The ISO/IEC 25010:2011 standard defines quality characteristics relevant to all software systems. These characteristics are further subdivided into subcharacteristics, which can be measured by one or more quality properties. The model described by ISO/IEC 25010:2011 presents five characteristics associated with quality in use and eight about product quality. Quality in use characteristics focus on the interactions of users and the product in a particular context of use. These characteristics are described as follows (ISO/IEC, 2011):

Effectiveness: expresses accuracy and completeness with which users achieve specified goals;

Efficiency: expresses the amount of resources spent in relation to the accuracy and completeness with which users achieve goals. Examples of resources include time to complete a given task, materials, or financial cost of usage;

Satisfaction: is associated with the degree to which user needs are satisfied when a product or a system is executed in a specified context of use. Satisfaction expresses the response of the user to the interaction with a product or system. This quality characteristic encompasses usefulness, trust, pleasure, and comfort;

Freedom from risk: describes the degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment. Freedom from risk is defined by a function of the probability of occurrence of a threat and the adverse consequences associated with it. This quality characteristic can be subdivided into the following subcharacteristics: economic risk mitigation, health and safety risk mitigation, and environmental risk mitigation; and

Context coverage: expresses the degree to which a product or system can be used with effectiveness, efficiency, freedom from risk, and satisfaction in specified contexts of use and in contexts beyond the ones initially explicitly identified. Context coverage can be subdivided into context completeness and flexibility.

Product quality characteristics are related to static properties of software and dynamic properties of a computer system. The following product quality characteristics are described in the quality model (ISO/IEC, 2011):

Functional suitability: expresses the degree to which a software product or system provides functions that satisfy stated and implied needs when used under specified conditions. This characteristic encompasses functional completeness, functional correctness, and functional appropriateness;

Performance efficiency: is related to the amount of resources used under stated conditions. Resources include other software products, hardware configuration of the system, materials, and so forth. The performance efficiency characteristic can be subdivided into time behaviour, resource utilization, and capacity;

Compatibility: describes the degree to which a system can exchange information with other products, systems or components and perform its required functions, while sharing the same hardware or software environment. Compatibility can be subdivided into co-existence and interoperability;

Usability: expresses the degree to which a product can be used by specified users to achieve goals with effectiveness, efficiency, and satisfaction in a given context of use. This quality characteristic can be divided into appropriateness, recognizability, learnability, operability, user error protection, user interface aesthetics, and accessibility;

Reliability: describes the degree to which a system, product or component performs specific functions under certain conditions for a given period of time. It can be divided into the following subcharacteristics: maturity, availability, fault tolerance, and recoverability;

Security: expresses the degree to which a product or system protects data so that persons, other products or systems have an appropriate degree of data access to their levels of authorization. Security encompasses confidentiality, integrity, non-repudiation, accountability, and authenticity;

Maintainability: expresses the effectiveness and efficiency which a product or system can be modified by maintainers that aim at performing corrections, improvements or adaptations to meet changes in the environment or specifications. This quality characteristic involves modularity, reusability, analysability, modifiability, and testability; and

Portability: describes the degree of effectiveness and efficiency with which a system or product can be transferred from one hardware or software environment to another. It encompasses the subcharacteristics of adaptability, installability, and replaceability.

Besides ISO/IEC 25010:2011, it is also possible to identify studies in the literature focusing on proposing quality models and sets of quality characteristics for particular system domains, such as embedded systems (Ahrens et al., 2013; Carvalho and Meira, 2009; Sherman, 2008). These studies add quality characteristics relevant to a particular domain and adapt definitions of existing characteristics to better describe quality in this domain. For instance, Sherman (2008) adds durability as a relevant quality characteristic of embedded systems. The author also advocates that the portability of embedded systems is rather related to the definition of modifiability. Other studies, such as (Guessi et al., 2012) and (Oliveira et al., 2013a), identified the most relevant quality characteristics associated with embedded systems. These studies aimed at supporting prioritization of main quality characteristics during the development of embedded systems.

Apart from software system characteristics, there are quality characteristics associated with the software architecture itself that are important to achieve. Bass et al. (2012) describe three quality characteristics specific to software architectures:

Conceptual integrity: is related to the underlying theme or vision that unifies the software system at all levels. In other words, the software architecture of a system should describe similar functionalities in similar ways;

Correctness and Completeness: is associated with the ability of the software architecture to enable all specified requirements and runtime resources constraints to be met; and

Buildability: is related to the ability of a software architecture to enable a system to be completed by the available team in a timely manner and at reasonable cost. Buildability is strongly related to the decomposition of the software system into modules and can be achieved by maximizing the parallelism that may occur during the development of these modules.

The assessment of a software system in the early stages of development enables designers to discover and anticipate future problems in the project. Therefore, evaluating the quality of software architectures is fundamental for the design of high-quality software systems. According to Clements et al. (2002), architecture evaluation is a cheap way to avoid a software system project to fail. There are several methods in the literature that can be used to evaluate the quality characteristics of software architectures (Bengtsson et al., 2004; Galster et al., 2008; van Heesch et al., 2014; Kazman et al., 1994, 1998; SEI, 2015a; Tekinerdogan, 2004). Most of them are based on checklists, personal experience or usage scenarios and their results are qualitative rather than quantitative (Clements et al., 2002). Among the scenario-based evaluation methods, it is important to highlight SAAM (Software Architecture Analysis Method) (Kazman et al., 1994) and ATAM (Architecture Trade-off Analysis Method) (Kazman et al., 1998), which have been widely adopted in software projects and used as a basis for other methods focused on particular domains (Graaf et al., 2005; Kim et al., 2008; Tekinerdogan, 2004). Recently, methods based on decisions, such as DCAR (Decision-Centric Architecture Reviews) (van Heesch et al., 2014), have been proposed aiming at reducing time and cost in software architecture evaluations. Differently from qualitative evaluation methods, only few studies are available for the quantitative evaluation of software architectures (Galster et al., 2008).

2.3 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style that promotes reusability, flexibility, and scalability to software systems (Josuttis, 2007; Papazoglou et al., 2008). It has been increasingly used in both academia and industry, especially in the design of complex, distributed applications. SOA enables the integration of heterogeneous software systems by providing interoperability through language-independent, standardized protocols (Erl, 2005; Papazoglou and Heuvel, 2007). This architectural style also provides a uniform way to offer, discover, interact, and use competencies of software systems for the development of new applications. Therefore, complex software systems can be productively developed by assembling resources provided by other systems (Papazoglou et al., 2008).

2.3.1 Initial Concepts

In SOA, software resources are provided by well-defined, self-contained modules called services (Papazoglou and Heuvel, 2007). A service shares one or more functionalities that are independent of the state and context of other services (Josuttis, 2007). Services can be seen as black-boxes that hide implementation details and only communicate through interfaces. In service-oriented systems, all functionalities are offered as services, including business functionalities, system service functionalities, and business processes of any abstraction level (Papazoglou and Heuvel, 2007). Interactions in SOA involve two main concepts (Josuttis, 2007):

Service provider: a software system that implements a service (e.g., a business functionality) to be discovered and reused by other systems; and

Service consumer: a software system that reuses functionalities offered by other services, i.e., it consumes a provided service.

A service participant is a system that is a service provider, a service consumer, or both (Josuttis, 2007). A participant that combines the roles of service provider and service consumer is called service aggregator (Papazoglou and Heuvel, 2007). As illustrated in Figure 2.4, a service aggregator can act as a provider of a given system while it also requests services from other providers to produce more complete functionalities.

Services can be classified with respect to their purposes and roles played in an SOA (Josuttis, 2007; Papazoglou, 2003). According to Josuttis (2007), participants of an SOA can provide three main types of services:

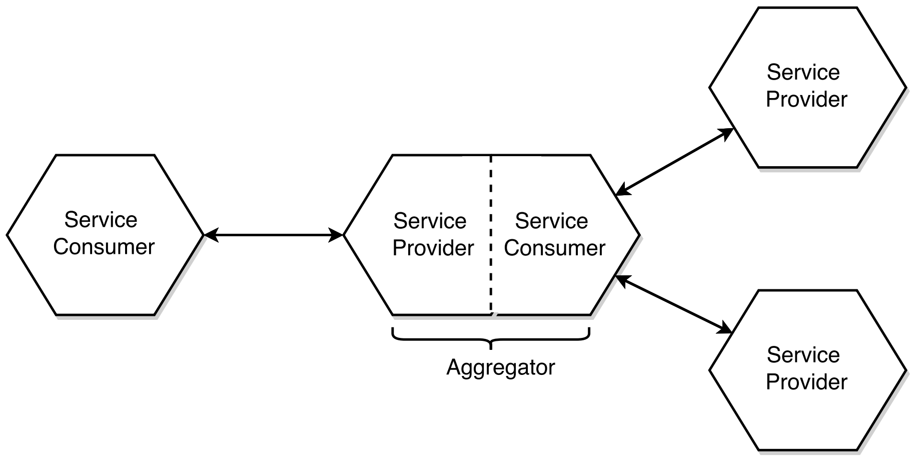


Figure 2.4: The role of a service aggregator (adapted from (Papazoglou and Heuvel, 2007))

Basic service: provides basic business functionalities that are meaningless if separated into multiple services. Basic services are usually short-term running and conceptually stateless. These services are designed to provide backend data or business logics to consumers of higher levels of abstraction. Services that provide data are responsible for reading and writing information of a backend system. Services that provide basic logic usually process input data and return corresponding results;

Composed service: describes services composed of basic services and/or other composed services. Composed services operate at a higher level of abstraction than basic services, but they are also short-term running and conceptually stateless. These services represent a short-running flow of activities inside a more complex business process; and

Process service: represents long-term workflows or business processes that are usually stateful, i.e., services that maintain their state over multiple interactions. Similarly to composite services, process services are also assembled by the composition of other services.

The interaction with a service is independent of its type, as the implementation details are hidden from consumers and the communication only depends on the service interface. For interacting with a service, it is first necessary to obtain its address (endpoint). In direct interactions, the service consumer is aware of the provider's endpoint at the design time and the communication is straightforward. In indirect interactions, providers publish standardized descriptions of their services in public and centralized registries so that they can be found and used by consumers (Papazoglou and Heuvel, 2007). Therefore, interactions may be performed by dynamic binding and service-oriented systems can communicate with service providers discovered at runtime (Josuttis, 2007). Figure 2.5 shows the interaction among service provider, service consumer, and service registry.

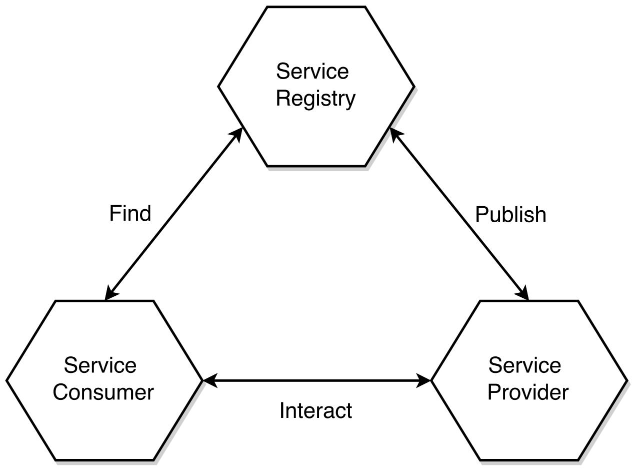


Figure 2.5: Interaction among service provider, service consumer and service registry (adapted from (Papazoglou and Heuvel, 2007))

Different participants can provide similar versions of a service in a service registry. Although these versions meet the same functional requirements, they usually display different quality characteristics. In this case, service consumers can choose the service to be invoked based on a set of software quality characteristics or specific Quality of Service (QoS) attributes (Canfora et al., 2008). For instance, a service consumer may decide to choose the fastest service, the most reliable, or one that presents a good compromise between both characteristics. The ISO 8402:2002 standard (ISO, 2002) describes a comprehensive set of QoS attributes, including price, availability, and reputation. Besides that, it is also possible to adopt additional quality attributes for specific domains. For example, a laser sensor service available in an application of the robotics domain could have measure precision and frequency as attributes. Using QoS attributes, service consumers may specify constraints on values of service quality characteristics which could impact on the choice (Canfora et al., 2005). Similarly, service providers can estimate ranges of QoS attributes values as additional information in contracts with potential users. This additional information is formally described by Service Level Agreement (SLA) (Canfora et al., 2008; Dan et al., 2004).

Apart from service registries, another important infrastructure of SOA is the Enterprise Service Bus (ESB) (Erl, 2005; Papazoglou and Heuvel, 2007). This infrastructure is the technical backbone that enables interoperability among services developed in different languages and platforms. According to Josuttis (2007), adoption of an ESB is essential for companies to fully achieve the advantages of SOA. An ESB acts as a middleware that supports several communication patterns over different protocols and provides important functionalities for service-oriented systems, including data transformation, routing, monitoring, security, reliability, and logging. Figure 2.6 illustrates how service consumers and service providers indirectly communicate through an ESB. In such a communication, all messages are received, mediated, and routed along this intermediary bus. Nowadays, several open source and proprietary ESBs are available: ServiceMix (Apache, 2015), OpenESB (LogCoy, 2015), MuleESB (Mule Soft, 2015), WSO2-ESB (WSO2, 2015), and JBoss ESB (JBoss Community, 2015). Among them, MuleESB is one of the most commonly used.

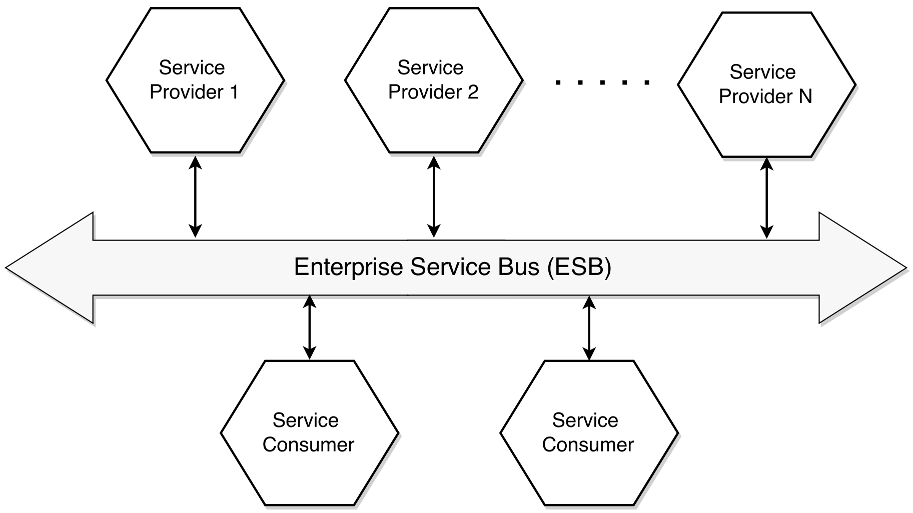


Figure 2.6: Service interaction through an ESB

2.3.2 Service Composition

Service composition is one of the most promising characteristics of SOA (Erl, 2005; Kazhamiakin et al., 2006). According to Kazhamiakin et al. (2006), service composition enables speeding up software systems development, improving reuse, and easing the interaction among complex services. Using composition, it is possible to design complex service-oriented systems by assembling functionalities provided by existing services. These functionalities are integrated and coordinated into workflows to model and execute new business processes. Developers can coordinate services that are partners in a composition according to two approaches (Josuttis, 2007; Peltz, 2003): orchestration and choreography. These coordination approaches are detailed as follows:

Orchestration: A central coordinator controls the execution of all functionalities provided by service partners according to the specified requirements (Josuttis, 2007). This coordinator centralizes all activities in the workflow, including the business logic and the order in which the service partners are invoked. Thus, the partners are not aware of the roles they play in the service composition or of being part of a higher level business process (Peltz, 2003). An orchestration is a service itself and can be composed by other business process. Figure 2.7 illustrates a generic representation of a service orchestration in which the central coordinator communicates with its service partners by exchanging messages; and

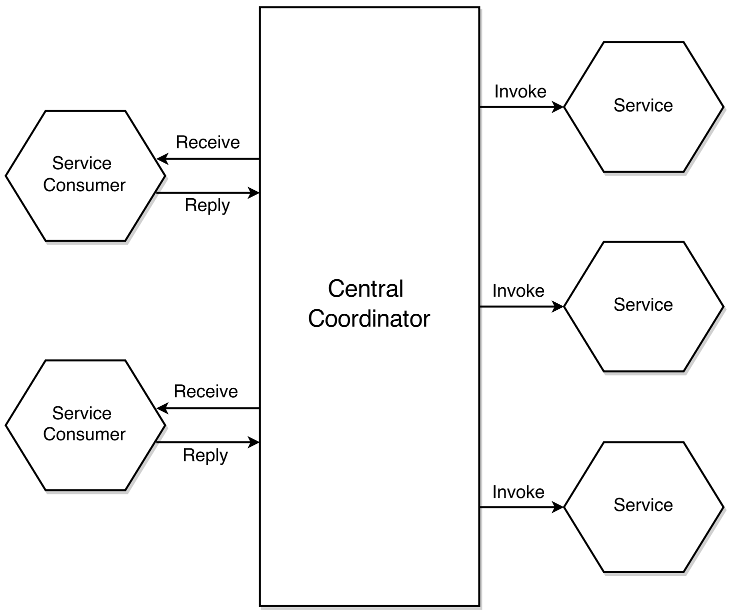


Figure 2.7: Service orchestration (adapted from (Peltz, 2003))

Choreography: No central coordinator controls the execution of the business process (Peltz, 2003). Differently from orchestration, all service partners are aware of the business process, operations to execute, and messages to be exchanged. Each service involved in the choreography knows when its operations must be executed and who to interact with. However, nobody is aware of or understands the process as a whole (Josuttis, 2007). Choreography is more collaborative and enables service partners to describe their participation using protocols (Josuttis, 2007). As a result, choreography has better scalability than orchestration, nonetheless are more difficult to coordinate (Josuttis, 2007). Tracking the current state of a choreography and discovering the cause for misbehaviours are often complex tasks (Josuttis, 2007). Figure 2.8 illustrates a generic representation of a service choreography.

2.3.3 Reference Models and Reference Architectures for SOA

As mentioned before, SOA increases flexibility, reusability, and productivity of the software systems development. However, there is a challenging downside to the adoption of this architectural style: service-oriented systems are usually difficult to be designed (Ar-

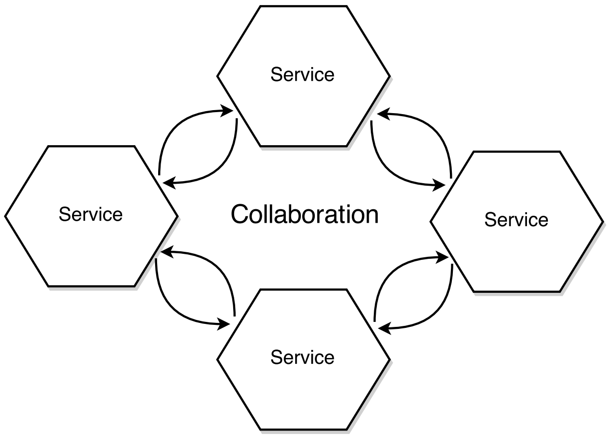


Figure 2.8: Service choreography (adapted from (Peltz, 2003))

sanjani et al., 2007). Development of service-oriented systems may involve integration of functionalities provided by several companies that follow different policies and governance. Therefore, architects must create software architectures using widely understood vocabulary and best design practices (Arsanjani et al., 2007). In this context, reference models and reference architectures are considered core assets to provide standardization, better communication, and reduce design complexity. Several companies, research institutes, and standardization organizations have been working on the establishment of reference architectures and reference models that support the development of systems based on SOA (Arsanjani et al., 2007; Dillon et al., 2007; OASIS, 2006, 2012; Oliveira and Nakagawa, 2011; The Open Group, 2015). Among such initiatives, it is important to highlight the reference model (OASIS, 2006) and the reference architecture (OASIS, 2012) proposed by the OASIS consortium[1], and the S3 (SOA Solution Stack) reference architecture (Arsanjani et al., 2007) proposed by IBM, which have been widely adopted in both academia and industry. Given its maturity, S3 has recently become part of an SOA technical standard (The Open Group, 2015).

OASIS Reference Model for SOA (SOA-RM) provides a common language for understanding SOA. It identifies the main characteristics of SOA and defines many of the important concepts necessary to comprehend what this architectural style is and what makes it important (OASIS, 2006). For instance, it describes the definition and the relationships among elements of a service, including service description, service interface, contract, and policy. OASIS Reference Architecture Foundation (SOA-RAF), originally defined in 2008, takes SOA-RM as its starting point to describe in an abstract manner how SOA-based systems can be realized (OASIS, 2012). SOA-RAF is a technology neutral reference architecture that follows the architecture description terminology defined by ISO/IEC/IEEE 42010:2011 and is divided into three views: (i) Participation in an SOA

[1]`https://www.oasis-open.org/`

Ecosystem focuses on the manner that participants are part of an SOA ecosystem; (ii) Realization of an SOA Ecosystem addresses the requirements for constructing an SOA-based system in an SOA ecosystem; and (iii) Ownership in an SOA Ecosystem describes what is meant to own an SOA-based system.

Differently from SOA-RAF, the S3 reference architecture provides an architectural definition of SOA based not only on concepts, but also on their implementation technologies. Figure 2.9 illustrates the nine layers of S3, which comprehend logical and physical aspects. The logical aspects include architectural building blocks, design decisions, options, and key performance indicators. The physical aspect encompasses the realization of the logical aspects using technology and products. Descriptions of the S3 layers are presented as follows (Arsanjani et al., 2007):

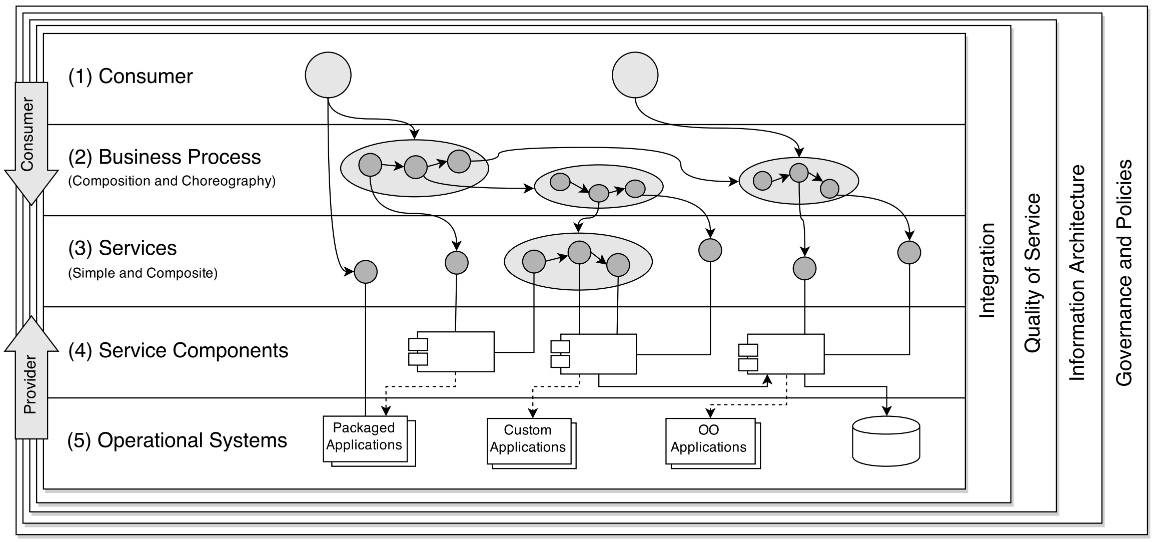


Figure 2.9: S3 reference architecture (adapted from (Arsanjani et al., 2007))

Operational systems: encompasses all application assets running in an operating environment that support SOA activities. Assets available in this layer can be custom, semi-custom or off-the-shelf;

Service components: contains software components that realize services and the operations these services provide. Service components reflect both functionality and quality characteristics of the service they represent;

Services: encompass all services defined within the SOA. As previously mentioned, services can be basic (i.e., simple) or composed;

Business process: represents business process services (i.e., process services) developed by the assembly of basic and composed services exposed in the service layer;

Consumer: represents the interaction of the service-oriented system with users or other services outside the application boundaries;

Integration: integrates the primary layers 2 through 4, mediating the requests of service consumers to the correct service provider. Capabilities provided in this layer include but are not limited to those found in an ESB;

Quality of service: is responsible for signing the noncompliance with service qualities in each SOA layer. It enables SOA to capture, monitor, log, and indicate noncompliance with quality requirements associated with the service qualities;

Information architecture: ensures that an organization includes the main considerations affecting data and information architectures; and

Governance and policies: encompasses all aspects of the management of the application life-cycle. It provides guidance and policies for managing SLAs regarding capacity, performance, security, and monitoring.

The instantiation of the S3 layers is aligned with the SOMA (Service-Oriented Modeling and Architecture) (Arsanjani et al., 2008) development method, which supports analysis, design, implementation, and deployment of service-oriented systems. SOMA is considered a mature method, since it has been successfully adopted in several projects in the industry.

As SOA-RAF and S3 are general and described in a high-level of abstraction, it is also possible to find reference architectures focused on specific domains (Costagliola et al., 2006; Murakami et al., 2007; Oliveira and Nakagawa, 2011; Peristeras et al., 2009). These reference architectures adds to SOA concepts the knowledge on how to design software systems in the application domain, such as on-line education (e-learning) (Costagliola et al., 2006) and software testing (Oliveira and Nakagawa, 2011). Although reference architectures can be considered the blueprint of software architecture design, no reference architecture has been defined to support the development of SORS.

2.3.4 Implementation Technologies

Web service is the most adopted way of realizing SOA (Erl, 2005). It enables applications developed in different programming languages and platforms to communicate over the Internet by means of standardized Web protocols. Web services are loosely-coupled software applications offered as a set of functionalities that can be composed to create complex business processes (Josuttis, 2007). WS-* standards and RESTful are currently the two main implementations of Web services (Pautasso et al., 2008).

WS-* standards encompass a set of languages and protocols based on the general purpose markup language XML (Extensible Markup Language) (W3C, 2015a). These standards are divided into two generations (Erl, 2005). The first generation forms the basis of the development of Web services and encompasses the following standards:

SOAP[2] (W3C, 2015b): is a lightweight protocol intended for the exchange of structured information in decentralized, distributed environments. SOAP enables Web services to execute RPC (Remote Procedure Call) communication over the Internet through several transport protocols, e.g., HTTP (Hypertext Transfer Protocol), HTTPS (HTTP Secure), and SMTP (Simple Mail Transfer Protocol) (Papazoglou and Heuvel, 2007);

Web Service Description Language (WSDL) (W3C, 2015c): is the language used to describe the interface of Web services. It encompasses all information necessary for the interaction with a service, including data types, port types, endpoint, and operations. Since operations and messages in WSDL are described abstractly, it must be used in conjunction with concrete protocols like SOAP; and

Universal, Discovery, Description and Integration (UDDI) (OASIS, 2015b): is the protocol that supports the description and discovery of Web services in service registries. It is designed to be interrogated by SOAP messages and provide access to WSDL documents.

The second generation standards were developed to complement the infrastructure provided by SOAP, WSDL, and UDDI to support the realization of SOA concepts, such as composition of business processes and specification of SLAs (Erl, 2007). Nowadays, different second generation standards are available for the development of Web services[3]. The two languages described bellow are examples of standards that implement the concepts of service orchestration and service choreography:

Web Services Business Process Execution Language (WS-BPEL) (OASIS, 2015a): is a language for describing and executing service orchestrations. Its elements enable invoking service partners, process responses, and deal with variables and control structures (Josuttis, 2007). An orchestration described in WS-BPEL is itself a service and, therefore, uses a WSDL document that describes its service interface; and

[2]Initially, SOAP was acronym for Simple Object Access Protocol. However, this definition is no longer used after Version 1.2.

[3]`http://en.wikipedia.org/wiki/List_of_web_service_specifications`, last accessed in January 13th, 2015.

Web Services Choreography Description Language (WS-CDL) (W3C, 2015d): is a language for the specification of business processes by choreography. It describes peer-to-peer collaborations between service partners by establishing a common view of their observable behaviour. The description of choreography in WS-CDL is a contract with multiple service partners that models the composition from a global point of view.

RESTful implementations are simpler alternatives to WS-* standards based on the Representational State Transfer (REST) architectural style (Fielding, 2000). Such implementations enable developers to create lightweight Web services that are scalable and easier to maintain (Guinard and Trifa, 2009; Pautasso et al., 2008). RESTful Web services are simple because they leverage well-known and widely adopted standards (HTTP, URI[4] and XML) and their necessary infrastructure has already become pervasive (Pautasso et al., 2008). Most programming languages and operating systems support clients and servers for HTTP. Effort to create a service consumer using RESTful Web services is reduced as developers can test it from an ordinary Web browser, without having to develop custom client-side software (Pautasso et al., 2008).

The REST architectural style is based on the concept of resource. Resources are representations of Web pages, images, files, and so forth. RESTful Web services expose resources based on four principles (Pautasso et al., 2008):

Resource identification through URI: establishes that resources must be identified by URIs to enable global addressing and service discovery. Resource URIs represent interaction targets of Web service consumers;

Uniform interface: establishes that resources of RESTful Web services should be manipulated by a fixed set of four operations: PUT, GET, POST, and DELETE. PUT creates a new resource that can be deleted using DELETE; GET retrieves the state of a resource in a given representation; and POST transfers new states onto a resource;

Self-descriptive messages: defines that resources must be decoupled from their representations. Therefore, the same resource can be accessed in different formats, e.g., HTML, JSON[5], and YAML[6]; and

Stateful interactions through hyperlinks: establishes that all interactions with resources must be self-contained and based on the concept of state transfer. In

[4]Uniform Resource Identifier (URI): `http://www.w3.org/Addressing/`. Last accessed in January 14, 2015.

[5]`http://www.json.org/`, last accessed in January 14th, 2015

[6]`http://www.yaml.org/`, last accessed in January 14th, 2015

stateful interactions, states can be embedded in response messages to inform valid future states of a resource.

WS-* standards and RESTful have similar purposes of use, but they display different conceptual and technological characteristics. According to Pautasso et al. (2008), WS-* Web services are more suitable for large-scale systems that involve complex business processes and constraints of security and reliability. On the other hand, RESTful Web services are particularly indicated to smaller *ad hoc* integration scenarios, where they significantly improve simplicity, performance, and loose integration of service-oriented systems. Despite both WS-* standards and RESTful were initially designed for Web-based systems, they have been adapted and used for the development of several embedded systems and robotic systems (Eisenhauer et al., 2009; Frenken et al., 2008; Guinard et al., 2010; Zeeb et al., 2007). Besides that, it is worth noting that Web services are not the only way to realize SOA. For instance, there are languages and protocols specially designed for developing service-oriented robotic systems (Ambroszkiewicz et al., 2010; Kononchuk et al., 2011).

2.4 Robotic Systems

Robotics has played an increasingly important role in several sectors of the society. Robots are no longer exclusively used to perform fast, repetitive tasks in controlled environments of factories. The actual generation of robots is being produced to operate along with humans and support daily activities inside hospitals (Pineau et al., 2003; Takahashi et al., 2010), houses (iRobots, 2015a,b), and on the streets (Fernandes et al., 2014; Thrun et al., 2006). According to Murphy (2000), robots can cooperate or even replace humans in several dangerous, tedious, and error-prone tasks. Their potential for improving quality of life and productivity has motivated both academia and industry to invest in robotics. For instance, in 2014, the European Commission announced a new partnership for a US$3.9 billion investment in robotics for the next six years[7].

2.4.1 Short History of Robots and Examples

The term robot derives from the Czech word "Robota", which means "servitude" or "forced labor". The word robotics, used to describe the study of robots, was coined in 1947 by science fiction writer Isaac Asimov in his book "I, Robot". The first modern robots emerged in the 1940s as manipulator arms and Automated Guided Vehicles (AGVs) for the industry (Dudek and Jenkin, 2010). Manipulator arms are fixed robots that move

[7]`http://www.eetimes.com/author.asp?doc_id=1322633`, last accessed in January 16th, 2015.

materials, tools, and other objects by arranging the position and angles of a set of joints. AGV is a mobile robot that follows markers or wires on the floor to transport materials from one place to another. These types of industrial robots usually have limited autonomy and only execute predefined routines. Most of them operate in controlled environments to avoid damage to humans. Even the AGVs that are able to detect objects blocking their path depend on humans for removing obstacles.

Over the past decades, several improvements in hardware devices and software algorithms have enabled researchers to develop robots of higher levels of autonomy. Hardware devices have become more precise, powerful, and cheaper. As a result, developers have created robotic systems of improved decision-making capacity and artificial intelligence. The interest in robots evolved from simply manufacturing to a broad range of application domains. Nowadays, robots support daily tasks, as well as complex, dangerous activities. For instance, iRobot Roomba (iRobots, 2015a), shown in Figure 2.10 (a), is an autonomous robot for vacuum cleaning. Husqvarna Automower (Husqvarna, 2015), illustrated in Figure 2.10 (b), is an example of robot for lawn mowing. Scooba (iRobots, 2015b) is a domestic robot designed by iRobot for floor scrubbing (Figure 2.10 (c)). Mars Pathfinder (Bajracharya et al., 2008) is a complex robot designed by the National Aeronautics and Space Administration (NASA) for exploring Mars (Figure 2.10 (d)). BigDog (Boston Dynamics, 2015), illustrated in Figure 2.10 (e), is as a four-legged robot designed to walk, run, climb, and transport heavy loads in rough terrains.

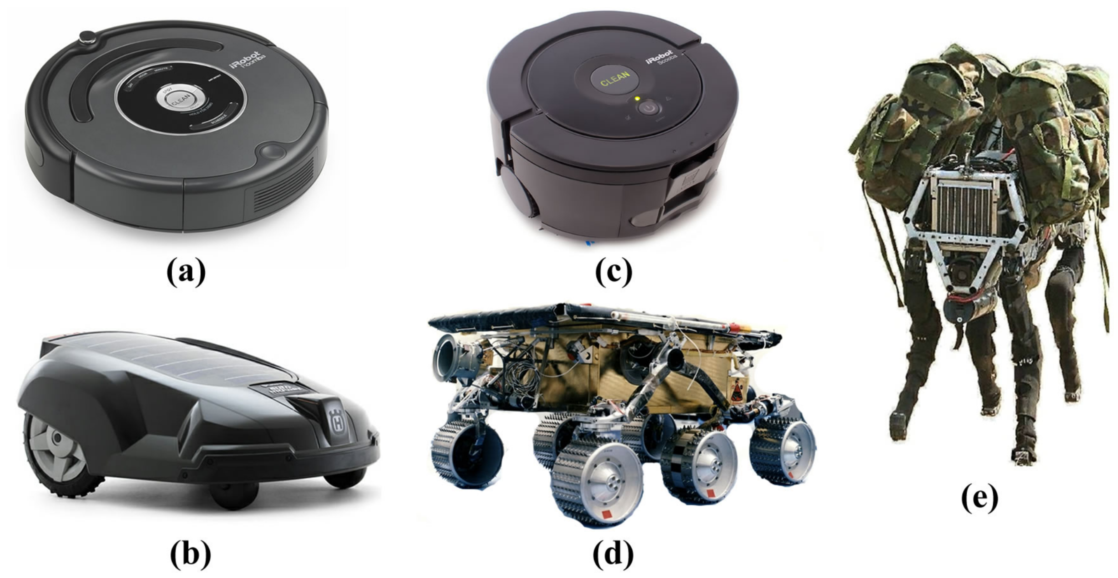


Figure 2.10: Examples of simple and complex robots: (a) Roomba, (b) Husqvarna Automower, (c) Scooba, (d) Mars Pathfinder, and (e) BigDog

Mobile robots have as main characteristic the ability of locomoting along the environment. This locomotion may be done in three levels of autonomy (teleoperated, semi-autonomous, and autonomous) and using different devices, such as wheels, legs, and helix. According to Romero et al. (2014), the most common types of mobile robots in the literature are:

Grounded: robots designed to operate in domestic environments, industries, and also rough terrains. They usually locomote using wheels, tracks or legs. Figure 2.11 (a) shows an example of autonomous grounded robot named Stanley (Thrun et al., 2006), which won the 2005 DARPA Grand Challenge after navigating 240km of desert in 6h54 (Thrun et al., 2005);

Aquatic: robots developed to operate on or under the water in rivers, oceans, and so forth. Underwater robots are commonly used in missions involving hard-to-reach places or risk to humans, e.g., in deepwater. Figure 2.11 (b) shows the autonomous underwater robot named Odyssey IV (Desset et al., 2005), which can operate in depths of up to six thousand meters; and

Aerial: robots able to autonomously fly according to pre-programmed flight plans or remotely controlled by teleoperation. Aerial robots, also known as Unmanned Aerial Vehicles (UAVs) or Drones, are frequently used in applications for agriculture, surveillance, and photometry. Figure 2.11 (c) shows an example of an aerial robot produced in Brazil called Tiriba (AGX, 2015).

The design of a robot is strongly related to its type and characteristics of the environment it will operate in. For instance, the design of a grounded mobile robot for a static indoor environment (e.g., industrial shop floor) may differ from another for navigating in a highly dynamic outdoor environment (e.g., an autonomous car on the streets). Despite differences may exist, all robots share the same general structure.

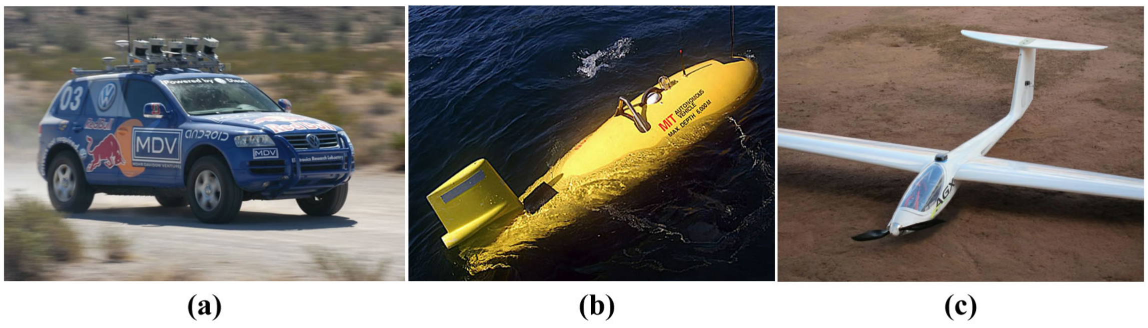


Figure 2.11: Examples of grounded, aquatic, and aerial robots: (a) Stanley, (b) Odyssey IV and (c) Tiriba.

2.4.2 General Structure of a Robot

The development of a robot is a multidisciplinary task and involves competencies from mechanical engineering, electrical engineering, automation engineering, computer engineering, and computer science (Romero et al., 2014). Mechanical engineering is the area responsible for developing the model and the physical structure of the robot. Electrical, automation, and computer engineerings usually focus on the development of electronic components (i.e., hardware devices). Computer science deals with the development of computer software systems for the control of the robot and its hardware devices to perform robotic activities (Romero et al., 2014). The software system used for controlling robots is known as robotic system.

The general structure of a robot can be defined by a relationship among perception, reasoning, decision making, and action (Iyengar and Elfes, 1991; Romero et al., 2014). Figure 2.12 illustrates the perception-decision-action loop, in which the robot senses its environment, decides on actions to perform, and executes them. Actions performed by a robot affect the environment surrounding it and, therefore, modify its perceptions for the next instant of time. Hardware devices used for data acquisition from the environment are called sensors. They enable the robot to obtain information about temperature of the environment, distance from other objects, possible collisions, and so forth. Actuators are hardware devices for interacting with the environment that enable robots to locomote, manipulate objects, and so forth.

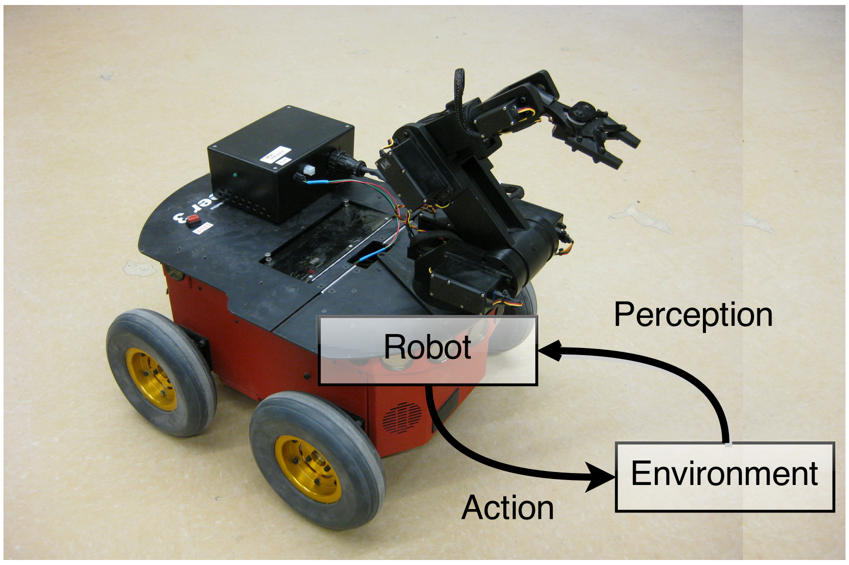


Figure 2.12: Perception-decision-action loop (adapted from (Wolf et al., 2009))

Several types of sensors and actuators are available for robotics. Although some may be used for the same purpose, they display different characteristics and limitations (Dudek and Jenkin, 2010; Romero et al., 2014). For instance, cameras and laser sensors can detect objects in the environment. While cameras may provide better range and less interference during daylight, laser sensors are more accurate for detecting objects in dark conditions. Different studies have described these characteristics and properties of hardware devices for robotics (Bekey, 2005; Romero et al., 2014; Siegwart and Nourbakhsh, 2004). According to Dudek and Jenkin (2010), the decision about the most suitable sensors and actuators is one of the most important activities during the development of a robot. Table 2.1 shows examples of sensors commonly used in robotics and Table 2.2 describes actuators for locomotion and object manipulation.

Table 2.1: Sensors available for robots (based on (Dudek and Jenkin, 2010))

Sensor	Purpose of use	Example
Position and orientation	Estimate the position and orientation of the robot	Global Positioning System (GPS), compass, inclinometer, and beacons
Contact	Detect the contact of the robot with objects	Bumper, whisker, and magnetic barrier
Obstacle detection	Estimate the distance between the robot and objects	infrared (IR), sonar, laser, and stereo cameras
Distance and velocity	Estimate the relative position and displacement of the robot	Odometer and angular potentiometer
	Estimate the displacement of the robot	Gyroscope and accelerometer
Communication	Send and receive data via signals	Optical sensors and radio-frequency (RF) sensors
Other	Capacitive sensors, inductive sensors, and pressure sensor	

Table 2.2: Actuators available for robots (based on (Dudek and Jenkin, 2010))

Atuador	Purpose of use	Example
Fixed basis	Manipulate objects	Industrial robotic arm
Mobile: tracks	Locomote on rough terrains	Military robots
Joint	Biped locomotion	Humanoid robots
	Quadruped locomotion	*BigDog*
	Hexapod locomotion	Spider robots
Mobile: helix and propellant	UAV with helix	Planes and Helicopters
	Water locomotion	Autonomous boats
	Underwater locomotion	Autonomous underwater vehicle
Other	Manipulation with sensory feedback	Tactile glove, force feedback gripper
	Trigger actuator	Soccer robots

Robotic control is the software module in a robotic system responsible for managing hardware devices and providing intelligent behaviour for the robot (Dudek and Jenkin, 2010). It processes data obtained from sensors and transforms them into information for supporting decision making. The robotic control also defines and executes action plans for the robot by sending commands to its actuators. Control architectures are the assets that represent the overall structure and functionalities of the robotic control. Definitions in the literature describe the control architecture of robotic systems as software architectures that represent components and the relationship among these components (Arkin, 1998; Mataric, 1992). However, the purpose of control architectures resembles more architectural styles (see Section 2.2.1), as they impose constraints on how software components of a robotic system interact. A control architecture is an abstract structure that describes a family of robotic systems of similar characteristics. According to Murphy (2000) and Romero et al. (2014), control architectures can be divided into:

Deliberative: robots perform activities based on predefined plans and using their internal model of the environment. All actions performed are defined prior their execution and possible changes in the environment are not taken into account. Deliberative architectures enable the robotic system to establish sets of actions that lead to more efficient solutions. However, these solutions are strongly dependent upon the correct representations of the environment. Therefore, this type of architecture is more suitable to static, controlled environments;

Reactive: actions are performed according to the state of the robot and the environment in each instant of time. The robotic system creates local models based on data obtained from sensors. Differently from deliberative architectures, no model describes the whole environment. Reactive architectures are usually simple and enable the robotic system to provide rapid reactions in highly dynamic environments. However, this type of architecture can produce only primitive behaviours and is more suitable for elementary actions, e.g., as avoidance of obstacles; and

Hybrid: they combine the main characteristics of deliberative and reactive architectures to produce more robust behaviours. The robotic system uses its global representation of the environment to create an initial plan before performing an activity. Along the execution, it reacts to changes in the environment updating its initial plan based on information from sensors. Therefore, it is possible, for instance, to plan and execute an optimal path avoiding unexpected obstacles along the navigation. Several complex robots have been designed by using hybrid control architectures, including the Stanley autonomous car presented in Section 2.4.1. Other examples of hybrid control architectures are described in (Romero et al., 2014; Siegwart and Nourbakhsh, 2004).

Regardless of the type of control architecture, the creation of robotic systems usually involves the development of the following functionalities:

Localization: consists in estimating the position of a robot in the environment through data from sensors. As any measuring instrument, sensors are imprecise and may return values slightly different from real ones (Thrun et al., 2005). Localization strategies use the available representation of the environment and algorithms that deal with uncertain values for reducing measurement errors. These algorithms are usually based on statistic methods, e.g., Kalman Filter (Leonard and Durrant-Whyte, 1991a), Monte-Carlo (Fox et al., 1999), and Markov chains (Fox et al., 1998);

Mapping: is the task of creating representations of the environment based on the estimated position of the robot and data from its sensors (Romero et al., 2014). Global maps are essential for the identification of efficient paths between an initial position and a goal position in the environment. Local maps help to avoid collisions with objects, humans or other robots along the path. Mapping may be a challenging task, since it is based on the current position of the robot, which is estimated by a representation of the environment that still under construction (Thrun et al., 2005). In this case, both tasks must be performed at the same time using techniques of Simultaneous Localization and Mapping (SLAM) (Leonard and Durrant-Whyte, 1991b); and

Navigation: consists in controlling the robot's speed and orientation for its moving from an initial position to a goal position (Thrun et al., 2005). It usually aggregates functionalities from control theory, path planning, and collision avoidance for assuring safety and reducing the risk of damage to the environment, humans or the robot itself (Romero et al., 2014). Depending on the control architecture, navigation can be based on global maps or purely reactive and guided by sensors.

Besides localization, mapping, and navigation, robotic systems may also encompass other tasks, depending on the intended use of the robot. For instance, robots that interact with humans require functionalities associated with gesture recognition or for interpreting voice commands. The taxonomy proposed in the context of this thesis, presented in Chapter 3, describes a broad set of tasks for robotic systems.

2.4.3 Development Environments for Robotic Systems

The development of robotic systems is a complex activity that involves creation and integration of complex algorithms, as well as the control of different hardware devices. In this perspective, development environments are tools created for supporting implementation, simulation, and testing of such systems. The adoption of development environments can provide several benefits: (i) cost reduction, as simulation can be used for the evaluation of a robotic system in different hardware configurations, prior buying any physical component; (ii) productivity, since simulation enables evaluation of hours of execution of a robotic system in only few minutes; (iii) safety, as simulation avoids damages to the robot or the environment caused by misbehaviours that may occur in early stages of development; and (iv) reliability, as the behaviour of the robot can be exhaustively tested in virtual environments for the reduction of possible faults.

Simulation environments are important modules of development tools that enable the creation of realistic models of the world. These models represent both the static and the dynamic of the environment, including forces as friction and gravity. In simulation environments, error models can be described for different sensors and actuators. The most used environments for the development and simulation of robotic systems are presented as follows:

Player, Stage, and Gazebo (Gerkey et al., 2003): Player is a development environment based on the client-server architectural style that supports over a dozen robotic platforms, as well as several types of sensors and actuators. It enables a software system to be written in different programming languages (e.g., C/C++, Java, and Python) and run on any computer with a network connection to the robot. Using Player, robotic systems are implemented as clients that manage robots provided through server interfaces. Therefore, switching from simulation to the real world is only a matter of changing the address of the virtual server to that of the actual robot. Player can be used in conjunction with two simulation environments: Stage and Gazebo. Stage is a two-dimensional bitmapped environment for the simulation of multiple robots. Gazebo is a multi-robot simulator for outdoor environments that provides both realistic sensor feedback and physically plausible interactions among objects;

Orca (Brooks et al., 2005): is an open source development environment for the creation of component-based robotic systems. It aims at improving modularity and reusability of robotic systems by providing means for the implementation of functionalities as self-contained components that interact via standard interfaces. Using ORCA, complex robotic systems can be developed by the assembly of reusable building-blocks available in public repositories;

Microsoft Robotic Developer Studio (MRDS) (Jackson, 2007): is a Windows-based environment for the development of service-oriented robotic systems. It uses a lightweight REST-style based on .NET and supports programming languages like Java and C#. It encompasses four main modules: (i) Concurrency and Coordination Runtime (CCR); (ii) Decentralized Software Services (DSS); (iii) Visual Programming Language (VPL); and (iv) Visual Simulation Environment (VSE). CCR enables the implementation of programs for handling asynchronous input from multiple sensors and output for actuators. DSS enables to access or set the state of a robotic system using a Web browser. VPL is a visual language for the creation of robotics systems by means of composing and connecting blocks of command representing services. VSE is a simulation environment that provides real-world physics and models for the representation of several robotic platforms; and

Robotic Operation System (ROS) (Straszheim et al., 2011): is an open source, meta-operating system that can be used for the development of SORS. Similarly to MRDS, it provides tools and libraries for the building, writing, and running of robotics services across multiple computers. Robotic systems implemented in ROS can communicate via different styles, including synchronous RPC and asynchronous data streaming. ROS also supports integration to multiple simulation environments, e.g., Stage and Gazebo. The ROS development environment is supported by dozens of companies and research institutes. The ROS repository, named ROS Wiki[8], currently lists over a hundred of services provided by the community for the development of robotic systems. However, it does not provide a mechanism that enables efficient description, classification, and discovery of these services.

Development environments have evolved over the years driven by the need of the robotics community for more reusable, scalable, and flexible robotic systems. This evolution is due mainly to the adoption of software architecture practices that potentially improve these quality characteristics, as the use of components and SOA.

2.4.4 Architectural Design of Robotic Systems

Robotic systems have become considerably large, complex, and integrated to other devices of the environment. Meanwhile, the increasing adoption of robots has demanded robotic systems of higher quality and better productivity. As a consequence, both industry and academia have focused their attention on the architectural design of these systems. Studies in the literature have described several architectural assets for guiding the development of robotic system, including reference models (Pires et al., 2011; Rodrigues et al., 2011), reference architectures (Albus, 2002; Clark, 2005; Hayes-Roth et al., 1995; Ortiz et al., 2005; Peters et al., 2000; Weyns and Holvoet, 2006), and design patterns (Fryer et al., 1997). These assets are commonly used with the support of different development approaches, as SPLs (Braga et al., 2011, 2012a,b; BRICS, 2015), Model-Driven Engineer-

[8]`http://www.ros.org/browse/list.php`, last accessed in January 21st, 2015.

ing (MDE) (Iborra et al., 2009; Schlegel et al., 2010), and Component-Based Software Engineering (CBSE) (BRICS, 2015; Iborra et al., 2009).

Regarding architectural assets, Graves and Czarnecki (2000) proposed a set of design patterns for the development of behaviour-based robotic systems. Pires et al. (2011) and Rodrigues et al. (2011) designed a reference model for describing missions for aerial robots (UAVs). Hayes-Roth et al. (1995) defined a reference architecture for supporting the design of intelligent, adaptive robotic systems. Focusing on scalability, Peters et al. (2000) described a reference architecture for indoor service robots[9]. Ortiz et al. (2005) proposed a component-based reference architecture for teleoperated service robots. Similarly, Albus (2002) also established a reference architecture for teleoperation, but focusing on the control of the robot by different types of devices. Clark (2005) designed a message-based reference architecture that aims at improving interoperability among subsystems and hardware devices. Finally, Weyns and Holvoet (2006) described a reference architecture for the development of multi-robotic systems.

With respect to development approaches, Fryer et al. (1997) investigated the use of object-orientation to improve modularity of robotic systems. Brugali and Scandurra (2009) discussed the use of CBSE for the design of robotic systems as a set of reusable architectural building blocks. In parallel, studies have associated MDE and SPL with CBSE for the development of robotic systems (Braga et al., 2012b; BRICS, 2015; Iborra et al., 2009). For instance, Iborra et al. (2009) used MDE and CBSE with reference architectures in a process to design robotic systems. Schlegel et al. (2010) described a development process based on MDE for the creation of component-based robotic systems. The BRICS project (BRICS, 2015) used CBSE and SPL to reduce the development efforts of engineering robotic systems. Braga et al. (2012a) and Braga et al. (2012b) used CBSE and SPL as a basis for the incorporation of certification activities into the design of aerial robots.

Similarly to systems for other domains, robotics has shown an increasing evolution from procedural paradigm and object-orientation to more modular and reusable forms of design. For instance, the architectural design of component-based robotic systems has been widely investigated in the recent literature. However, despite dedicated tooling already exists, the architectural design of robotic systems using SOA remains a new, promising topic of research.

[9]Notice that the term “service robot” is not associated with the use of SOA as architectural style, but with a particular type of robot that assists human beings in performing tasks that are usually dirty, dull, distant, dangerous or repetitive.

2.4.5 Service-Oriented Robotic Systems

SOA has raised considerable attention as an architectural style for the development of more flexible, integrable, and reusable robotic systems. Several researchers have created robotic systems based on SOA over the past years. This section describes the results of a systematic literature review conducted in the context of this thesis, and later updated, to characterize the state-of-the-art of the development of SORS. A systematic literature review is a rigorous, well-established approach to identify, evaluate, and interpret all available evidences regarding to a particular area or topic of interest (Kitchenham, 2004). Results of the first iteration of our systematic review are summarized in the paper (Oliveira et al., 2013b). Further details on the planning and conduction are available in a technical report (Oliveira et al., 2012). This systematic review was updated in January 2015 according to the same planning.

Table 2.3 shows the 57 included primary studies[10] and their year of publication. Additional information on the primary studies is also available in Appendix A. Column "Type" indicates if the primary study is a Technical Report (TR), Journal Article (JA) or Conference Paper (CP). Column "Criteria" points out the criteria applied to the inclusion of each study. Three Inclusion Criteria (IC) were used for the selection of primary studies in our systematic review: (IC1) the study proposes or reports on the design and development of an SORS; (IC2) the study proposes or reports on a technology for the development of SORS; and (IC3) the study presents a software engineering contribution (e.g., process, method, reference architecture, and ADL) for the design or implementation of SORS.

Table 2.3: Included primary studies

ID	Author	Year	Type	Criteria
S1	Lee et al.	2004	JA	IC1
S2	Ha et al.	2005	CP	IC1
S3	Kim et al.	2005	CP	IC1
S4	Narita et al.	2005	CP	IC2
S5	Ahn et al.	2006	CP	IC1, IC2
S6	Berná-Martínez et al.	2006	CP	IC1
S7	Amoretti et al.	2007	CP	IC1
S8	Coelho et al.	2007	CP	IC1
S9	Rahman et al.	2007	JA	IC1, IC2
S10	Tikanmaki and Roning	2007	CP	IC2
S11	Walter et al.	2007	CP	IC1
S12	Wu et al.	2007	JA	IC1

[10]The term "primary study" refers to any individual evidence that contributes to a systematic review.

Table 2.3: Included primary studies – *continued*

ID	Author	Year	Type	Criteria
S13	Yeom	2007	CP	IC1
S14	Awaad et al.	2008	CP	IC2
S15	Chen et al.	2008	CP	IC1
S16	Chen and Bai	2008	CP	IC1
S17	Hongxing et al.	2008	CP	IC1
S18	Lee et al.	2008	CP	IC1
S19	Majedi et al.	2008	CP	IC3
S20	Trifa et al.	2008	CP	IC1
S21	Tsai et al.	2008	JA	IC2
S22	Barbosa et al.	2009	CP	IC1
S23	Chen et al.	2009	CP	IC1
S24	Mokarizadeh et al.	2009	CP	IC1
S25	Pruter et al.	2009	CP	IC1
S26	Veiga et al.	2009	JA	IC1
S27	Ambroszkiewicz et al.	2010	JA	IC1, IC2
S28	Arumugam et al.	2010	CP	IC2
S29	Berná-Martínez and Maciá-Pérez	2010	CP	IC3
S30	Cepeda et al.	2010	CP	IC2
S31	Cesetti et al.	2010	CP	IC1
S32	Chen et al.	2010	CP	IC1
S33	Edwards et al.	2010	CP	IC1
S34	Pinto et al.	2010	CP	IC1
S35	Scotti et al.	2010	CP	IC1
S36	Blake et al.	2011	JA	IC1, IC2
S37	Kononchuk et al.	2011	CP	IC2
S38	Lindemuth et al.	2011	JA	IC1
S39	Waibel et al.	2011	JA	IC1
S40	Cepeda et al.	2011	CP	IC3
S41	Quintas et al.	2011	CP	IC1
S42	Brugali et al.	2012	CP	IC2
S43	Doriya et al.	2012	CP	IC1
S44	Doriya et al.	2012	CP	IC1
S45	Muhammad et al.	2012	CP	IC3
S46	Raffaeli et al.	2012	JA	IC1
S47	Zhou et al.	2012	CP	IC1
S48	Ebenhofer et al.	2013	CP	IC1
S49	Yang and Lee	2013	JA	IC2
S50	Brugali et al.	2014	CP	IC2

Table 2.3: Included primary studies – *continued*

ID	Author	Year	Type	Criteria
S51	Cai et al.	2014	JA	IC1
S52	Fluckiger and Utz	2014	JA	IC3
S53	Insaurralde and Petillot	2014	JA	IC1
S54	Koubaa	2014	CP	IC1, IC2
S55	Matta-Gomez et al.	2014	JA	IC1
S56	Oliveira et al.	2014	CP	IC3
S57	Oliveira et al.	2014	CP	IC2

During the conduction of this systematic review, we found overlapping primary studies reporting different stages or parts of the same research. Following the guidelines proposed by Kitchenham (2004), we did not consider these studies to avoid computing a same evidence twice. Only the most complete version of a research was taken into account during the data extraction. For instance, if a conference paper has an extended version in a journal, we only considered this most recent version. Table 2.4 shows the studies related to the development of SORS that were not included in the systematic review. Column "RS" presents the ID of the included study (listed in Table 2.3) that is directly related to the excluded study.

Table 2.4: Overlapping primary studies

ID	Author	Year	Type	RS
E1	Lee et al.	2003	CP	S1
E2	Ahn et al.	2006	CP	S5
E3	Kim et al.	2006	CP	S3
E4	Berná-Martínez et al.	2006	CP	S6
E5	Wu et al.	2006	CP	S12
E6	Ambroszkiewicz et al.	2007	TR	S27
E7	Veiga et al.	2007	CP	S26
E8	Tsai et al.	2008	CP	S21
E9	Lee et al.	2008	CP	S17
E10	Lee et al.	2008	CP	S17
E11	Tsai et al.	2008	CP	S21
E12	Du et al.	2011	CP	S32
E13	Remy and Blake	2011	JA	S36
E14	Insaurralde et al.	2012	CP	S53
E15	Insaurralde and Petillot	2013	CP	S53
E16	Lee and Yang	2013	CP	S49
E17	Yang and Lee	2013	CP	S49

Based on the included primary studies, we answered six research questions. Results are reported as follows.

Research Question 1 – Types of services for SORS: The development of robotic systems using SOA is still recent and there is no clear understanding on which modules of these systems should be provided as services. This research question investigated how SOA has been applied during the development of SORS. In the analysis of the 57 primary studies, we identified four different categories of services for SORS: (i) Sensors and Actuators, (ii) Tasks and Activities, (iii) Knowledge and Algorithms, and (iv) Whole Robot. The category "Sensors and Actuators" indicates that services are applied at the lowest level of the robotic system infrastructure, i.e., each device used to measure (using sensors) or interact (using actuators) with the environment is controlled and provides information as an independent service. In the "Tasks and Activities" category, services are used to offer functionalities related to the fundamental tasks of robotics (e.g., mapping and localization, described in Section 2.4.2), as well as more complex activities, such as surveillance and monitoring. Functionalities related to knowledge acquisition from different sources (e.g., the Internet) and supporting algorithms were considered services of the "Knowledge and Algorithms" category. The category "Whole Robot" indicates that the entire robot is considered a single service, i.e., Robot as a Service (RaaS) (Chen et al., 2010). Table 2.5 summarizes these categories and shows the total (column #) and percentage (column %) of primary studies that address each of them. Observe that studies in Column "Primary Studies" may address more than one category.

Table 2.5: Service development approaches in SOA-based robotic systems

Service Category	(#)	(%)	Primary Studies
Sensors and Actuators	33	57.89	S1, S6, S10, S11, S15, S16, S17, S21, S23, S26, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S40, S42, S43, S44, S45, S46, S50, S52, S53, S54, S55, S56, S57
Tasks and Activities	34	64.91	S1, S4, S5, S7, S8, S14, S16, S20, S21, S24, S27, S29, S31, S32, S33, S34, S35, S39, S40, S42, S43, S44, S45, S46, S47, S48, S49, S50, S51, S52, S53, S55, S56, S57
Knowledge and Algorithms	19	33.33	S3, S6, S11, S14, S18, S21, S23, S28, S29, S31, S35, S36, S37, S39, S40, S41, S46, S56, S57
Whole Robot	33	57.89	S1, S2, S3, S4, S5, S7, S8, S9, S10, S12, S13, S15, S19, S20, S22, S25, S28, S32, S38, S40, S42, S43, S44, S45, S47, S48, S50, S52, S53, S54, S55, S56, S57

Notice there is no predominance of a single category. Services have been designed in different manners and granularities to achieve multiple goals during SORS development. For instance, developers used SOA to overcome traditional problems regarding integration of heterogeneous, off-the-shelf sensors and actuators (as discussed in S16, S17, S26, and S29). Moreover, services were used to hide details of robotic task implementations for facilitating their reuse in several projects and robotic systems, as reported in S20, S21, S24, and S32. One third of the primary studies used service-orientation as a strategy to improve processing power and the knowledge of robotic systems. Services developed in the category "Knowledge and Algorithms" enabled fast, distributed implementations of demanding artificial intelligence algorithms, e.g., Q-Learning (Watkins and Dayan, 1992) for reinforcement leaning (as presented in S35 and S39). They also enabled the use of web services as sources of external knowledge for robots, so that the robots could learn and interact with objects not foreseen during design time (as proposed in S3 and S36). Finally, services used to wrap functionalities provided by robots reduce complexity and facilitate their coordination in complex missions (as discussed in S24 and S47).

Research Question 2 – Interactions between services of SORS: This research question investigated how services that are part of an SORS interact with each other. Based on the selected primary studies, we have identified five main types of interaction: (i) interaction of robotic control with its sensors and actuators (Robot to Device); (ii) interaction of the robot with other robots (Robot to Robot); (iii) interaction of the robot with a back-end computer[11] (Robot to Back-end); (iv) access of the robot to external services, i.e., services which are outside its working environment (Robot to ES); and (v) interaction of robot with other devices, such as electronic doors, light and temperature controllers (Robot to Envt.). Table 2.6 shows the types of interaction addressed by the primary studies. Observe some primary studies have addressed more than one type of interaction.

Most of the developed SORS (91.22%) are supported by or depend on a back-end computer to communicate, as they require a server that provides their functionalities as services. Moreover, robotic systems often use back-end computers as an alternative to increase their processing power. The second most common way of interaction occurs between the robotic control and the services that deal with hardware devices (45.61%). We have also identified studies (19.30%) that investigate the connection between the robotic system and external services. Those services are used to support robots to make more intelligent decisions (as in S18 and S36). Studies that use SOA to integrate robotic systems with elements of the environment, for instance a smart house, have also been found (S1, S2, S3, S23, and S49). Although this research question has evidenced that

[11]A back-end computer is a server that remotely supports processing, data storage or interaction among robots.

Table 2.6: Interactions among services of SORS

Ways of Interaction	(#)	(%)	Primary studies
Robot to Device	26	45.61	S5, S6, S15, S16, S21, S23, S26, S29, S30, S31, S32, S33, S35, S36, S40, S42, S43, S44, S45, S46, S48, S50, S52, S55, S56, S57
Robot to Robot	15	26.31	S1, S4, S7, S9, S30, S32, S37, S38, S42, S45, S50, S52, S55, S56, S57
Robot to Back-end	52	91.22	S1, S2, S3, S4, S5, S6, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44, S46, S47, S49, S50, S51, S52, S53, S54, S55, S56, S57
Robot to ES	11	19.30	S3, S18, S36, S39, S41, S42, S43, S44, S50, S56, S57
Robot to Envt.	9	15.79	S1, S2, S3, S23, S41, S47, S49, S56, S57

SORS are still dependent on external infrastructure, we believe this dependency tends to decrease as improvements are made in hardware and network technologies.

Research Question 3 – SORS development technologies: As mentioned in Section 2.4.2, robotic systems have several characteristics, such as resource limitations and real-time constraints, which differentiate them from information systems. This research question addresses technologies (in particular, protocols, programming languages, and frameworks) investigated and used to develop SORS. Although some of them were not initially designed for implementing robotic systems as services or collections of services, they have been used as innovative initiatives in the development of SORS. Table 2.7 lists the development technologies applied for the implementation of SORS and the primary studies that address each of them. Notice the technologies presented here might not be classified on the same abstraction level.

The SORS found in the systematic review were predominantly developed using Web service standards or variations of these standards. SOAP-based standards (WS-*) (36.84%) and REST (26.32%) are the technologies mostly addressed in primary studies. Together, they have been used in more than six out of ten SORS available in literature. Besides Web service standards, researchers have also used UPnP (5.26%) and CORBA (7.02%). In short, UPnP is a set of networking protocols that enables devices to seamlessly discover each other's presence in the network and establishes functional network services for data sharing and communications. CORBA enables separate pieces of software written in different languages and running on different computers to work with each other as a single application or a set of services. Although other technologies have been found, they seem to be isolated initiatives used only by their authors (e.g., S4, S9, S10, S14, S22, S27, and S37). Studies that do not report on the use of any technology are listed in the category DNR (Do Not Report).

Table 2.7: Languages, protocols, and frameworks for developing SORS

Implementation technology	(#)	(%)	Primary studies
SOAP (WS-*)	21	36.84	S2, S3, S6, S7, S8, S12, S13, S15, S18, S19, S23, S24, S29, S30, S31, S32, S36, S43, S46, S54, S55
REST	15	26.32	S15, S21, S23, S25, S30, S31, S32, S33, S34, S35, S39, S40, S44, S46, S55
CORBA (Vinoski, 1997)	4	7.02	S11, S17, S42, S52
UPnP (UPnP Forum, 2015)	3	5.26	S5, S13, S26
Simple XML	2	3.51	S22, S27
Java Messaging Service	1	1.75	S50
JINI (Waldo, 1999)	1	1.75	S1
MeRMaID	1	1.75	S22
Entish	1	1.75	S27
RoboCoP	1	1.75	S37
RoboLink	1	1.75	S4
Property	1	1.75	S10
SENORA	1	1.75	S9
XPERSIF	1	1.75	S14
DNR	13	22.81	S16, S28, S38, S35, S41, S45, S47, S48, S49, S51, S53, S56, S57

Results of this research question evidence most researchers have focused on languages and protocols widely accepted in commercial information systems (i.e., WS-* and REST) instead of those specially designed to fit the robots' needs. Even though generic-purpose technologies might not ensure important aspects of the robotics domain like message size and real-time constraints, they are easier to be integrated with other types of services and, therefore, have been more frequently adopted in robotic systems.

Research Question 4 – SORS development environments: This research question investigates environments that support the development of SORS. As mentioned in Section 2.4.3, development environments are important assets for creating robotic systems. In particular, when associated with simulators, they can reduce cost, increase productivity, and improve reliability of robotic systems. Table 2.8 lists the environments found by our systematic review.

Observe MRDS is the most used environment for the development of SORS. More than four out of ten primary studies reported the use of a development environment adopted MRDS. It is recognized as a landmark of SORS development and has been cited by almost all authors after 2007. However, there has been an increasing interest in the recent ROS development environment, since it is free, open source, and widely supported by robotics community. Although ROS has been less frequently addressed than MRDS – probably because this environment has just taken off – we believe it will be largely used in the

Table 2.8: Development environments used to create SORS

Development environment	(#)	(%)	Primary studies
MSRS	12	21.05	S15, S21, S23, S30, S31, S32, S35, S40, S43, S46, S55, S57
ROS	8	14.03	S28, S36, S39, S44, S50, S53, S54, S57
Player/Stage	2	3.51	S7, S33
LEGO Mindstorm	1	1.75	S6
YARP	1	1.75	S22
OROCOS	1	1.75	S42
4DIAC	1	1.75	S48
OpenRAVE	1	1.75	S49
XPERSim	1	1.75	S14
Own Environment	2	3.51	S3, S18
DNR	28	56.41	S1, S2, S4, S5, S8, S9, S10, S11, S12, S13, S16, S17, S19, S20, S24, S25, S26, S27, S29, S34, S37, S38, S41, S45, S47, S51, S52, S56

next years. Besides these service-oriented tools, researchers have also adapted other environments for developing and simulating SORS: Lego Mindstorm (LEGO, 2015), YARP (Metta et al., 2006), Player/Stage, OROCOS (OROCOS, 2015), 4DIAC (PROFACTOR, 2015), and OpenRAVE (Diankov, 2010). Furthermore, studies that propose their own environments were also found (S3, S14, and S18). Despite the importance of development environments, more than half of the primary studies did not report the use of such tools. These studies are found in the DNR category.

Research Question 5 – SORS development scenarios: This research question investigates scenarios in which SORS have been developed. We looked for characteristics that could influence the suitability of SOA for the development of robotic systems, as intended environment and type of robot. We also investigated the context in which SORS had been applied. Table 2.9 shows the characteristics addressed by each primary study. Column "Domain" indicates the areas of SORS reported in the primary studies. Column "App" classifies the studies according to their purpose of use, i.e., for academia (Acd), industry (Ind), or both. Column "Envt" indicates whether the SORS is destined for indoor (In) or outdoor (Out) environments. In column "Class", we classified a robot according to its mobility, as ground mobile (GM), aquatic (AQ), aerial (AE), and non-mobile (NM). Finally, in column "Type", we reported whether an SORS is destined for a single robot (SR), multiple robots (MR), or robot swarms (RS).

Notice most SORS have been developed for the academy (91.23%) and are generic (57.84%). Studies considered generic are those that address the SORS development itself, rather than possible application domains. Moreover, most SORS are grounded mobile (84.21%) and/or destined to indoor environment (89.47%). These SORS were developed

Table 2.9: Main characteristics of SORS

ID	Domain	App	Envt	Class	Type
S5, S6, S13, S14, S17, S29, S49	Generic	Acd	In	GM	SR
S18	Generic	Acd	In	NM	SR
S1, S4, S7, S9, S10, S13, S23, S15, S19, S27, S30, S33, S40, S42, S50, S51	Generic	Acd	In	GM	MR
S28, S32, S39, S41, S43, S44, S54	Cloud Computing	Acd	In	GM	MR
S22, S16	Monitoring	Acd	In	GM	MR
S36	Generic	Acd	In	NM	SR
S31, S46, S48	Manufacturing	Ind	In	GM	SR
S8, S37	Education	Acd	In	GM	MR
S2	Ubiquitous	Acd	In	GM	SR
S3	Ubiquitous	Acd	In	NM	SR
S38	Monitoring	Acd	Out	AE/AQ	MR
S20, S24, S47	Generic	Acd	In	GM	RS
S56	Generic	Acd	Out	GM	MR
S34, S53	Monitoring	Acd	Out	AQ	MR
S25	Automation	Both	In	GM	MR
S35	Generic	Both	In	GM	SR
S21	Gaming	Acd	In	GM	MR
S26	Manufacturing	Ind	In	NM	SR
S11	Pipe inspection	Acd	Out	AQ	SR
S12	Manufacturing	Ind	Ind	NM	MR
S52	Space exploration	Acd	Out	GM	MR
S45, S56, S57	Generic	Both	–	–	–

for robotic applications involving single robots, multiple robots, and in some cases, swarm of robots (S20, S24, and S47). Studies S45, S56, and S57 are generic software engineering guidelines that can be used in any type of robotic system. We believe the predominance of indoor and grounded (including non-mobile) robots can be related to physical limitations of the networks. In fact, this result seems coherent, since availability, reliability, and management of wireless networks are still a challenging topic in robotics. Another important fact is the growing interest of developers in providing a robot as a resource in cloud computing (S28, S32, S39, S41, S43, S44, and S54). The use of cloud computing in robotics is quite recent, and it seems to be a prominent research topic for the future (RobotShop, 2015; van de Molengraft, 2015).

Research Question 6 – Software engineering guidelines for SORS: In this research question, we investigated how software engineering has supported the design and implementation of SORS. As in any new type of software system, the development of SORS requires software engineering guidelines to become mature and more productive. Nevertheless, few studies have focused on guiding the creation of such systems. Only six studies are related to the design of robotic systems based on SOA. Study S29 describes an SOA-based integration model for robotic devices. In study S19, the authors propose a generic SOA-based model to be applied to various classes of pervasive computing applications, including robotics. Study S40 suggests a generic system architecture for robotic systems designed in the MDRS development environment. Study S45 proposes a UML profile for distributed embedded real-time systems that can be used for the design of SORS architectures. Focusing on space exploration, S54 details an architecture used for designing a family of NASA rovers. Finally, Study S56 reports the process proposed in the context of this thesis (See Chapter 4), which supports conception, detailing, and evaluation of SORS software architectures. Apart from these studies, no other was found in the systematic review. It is important to highlight that we have not considered specific architectures of SORS as software engineering guidelines, since they can not be applied for the development of other robotic systems.

Besides the primary studies found in the systematic review, we have also identified an additional work that investigates the architectural design of robotic systems based on SOA. The thesis presented by Hestand (2011) defines a set of metrics to be used when comparing the SOA approach with other architectures for integration on robotic systems. The work also proposes the adaptation of modules available for the Player environment to enable the development of SORS.

2.5 Final Remarks

This chapter presented the background for the contributions described in the remaining chapters. Firstly, the terminology and fundamental concepts of software architecture were discussed. Following, the theory associated with the SOA architectural style and its implementation technologies were addressed. Finally, an overview of robotics and robotic system development was provided. The chapter also characterized the state-of-the-art of SORS development through a systematic literature review. This review allowed us to identify limitations on the current research into the design of such systems. Despite the existence of some tooling and several SORS reported in the literature, these systems have been developed in an *ad hoc* manner. In particular, there is no support for the systematic identification of services for an SORS or how to structure these services as a software architecture. This lack of maturity during the design of SORS software architectures may impact the overall quality of the systems.

The contributions we present in the next chapters aim at overcoming some limitations of the SORS software architecture design. The next chapter reports the establishment of a comprehensive taxonomy of services for developing SORS, as well as the automation of a mechanism that enables classification and discovery services based on this a taxonomy. In Chapter 4, we propose a process aligned with the taxonomy to support the design of SORS software architectures. Chapter 5 describes a reference architecture that encompasses the knowledge of how to structure SORS software architectures designed by the proposed process.

Chapter

3

A Classification of Services for SORS

3.1 Overview

The potential of SOA in improving modularity, integrability, and flexibility has motivated researchers to develop their robotic systems as collections of services. As evidenced in the systematic review presented in Section 2.4.5, an increasing number of studies have reported the design of SORS. However, most of these systems have been developed in different manners, without a common understanding on how and which software modules should be provided as services. This lack of consensus hampers the reuse of services in other projects and reduces the capacity of SOA in raising productivity during the development of SORS.

In this sense, a classification of the types of services available for the development of SORS can provide two important contributions. First, it can be used to support the identification of services during the development of SORS software architectures. Second, it can facilitate the description, publication, and discovery of services for SORS, as well as enable the automation of such activities. Currently, none of the development environments available for SORS provide an efficient mechanism for publishing and discovering services. Developers have to manually search for services they need in repositories containing hundreds of different services, which is a time-consuming, error-prone task.

This chapter reports two complementary works on the classification of services for SORS. The first, presented in Section 3.2, describes a taxonomy of services for developing

SORS that supports the application of the process proposed in Chapter 4 and is also basis for the reference architecture detailed in Chapter 5. The second, presented in Section 3.3, describes a tool based on the proposed taxonomy that facilitates cataloging and discovery of services for SORS. Section 3.4 discusses results and limitations of both works.

This chapter is a summary of the papers *"Towards a Taxonomy of Services for Developing Service-Oriented Robotic Systems"*, published in the Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE'14) (Oliveira et al., 2014c), and *"Automating the Discovery of Services for Service-Oriented Robotic Systems"*, published in the Proceedings of the 11th IEEE Latin American Robotics Symposium (LARS'14) (Oliveira et al., 2014b). The latter was awarded as one of the best papers of the symposium and the authors were invited to submit an extended version. The extended paper, entitled *"RoboSeT: A Tool to Support Cataloging and Discovery of Services for Service-Oriented Robotic Systems"*, will be published in the Communications in Computer and Information Science series (Oliveira et al., 2015).

3.2 A Taxonomy of Services for SORS

We followed the systematic set of steps illustrated in Figure 3.1 to establish our taxonomy of services. In short, in Step 1, we first elicited services from different sources of information. In Step 2, we grouped the sets of services that display similar characteristics and purposes of use. In Step 3, these groups were described and organized into different abstraction levels, which resulted in the types of services and in the Robotics Services Dependency Stack (RSDS). Finally, in Step 4, the types of services and RSDS were evaluated in a survey applied to experts in the robotics area. Each step of the establishment of our service taxonomy is described in details as follows.

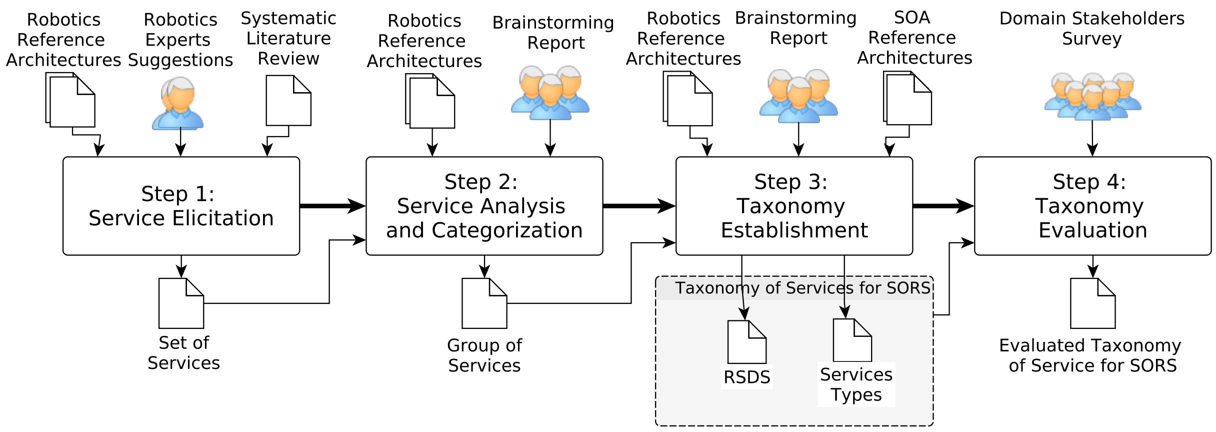

Robotics Reference Architectures
Robotics Experts Suggestions
Systematic Literature Review
Robotics Reference Architectures
Brainstorming Report
Robotics Reference Architectures
Brainstorming Report
SOA Reference Architectures
Domain Stakeholders Survey
Step 1: Service Elicitation
Step 2: Service Analysis and Categorization
Step 3: Taxonomy Establishment
Step 4: Taxonomy Evaluation
Set of Services
Group of Services
Taxonomy of Services for SORS
RSDS
Services Types
Evaluated Taxonomy of Service for SORS

Figure 3.1: Steps followed to establish the taxonomy of services

3.2.1 Step 1: Service Elicitation

Different information sources were selected for the elicitation of the types of services used in the development of SORS. These sources encompassed both theoretical and practical views of the robotic system development. The three sources were: (i) studies on the development of SORS identified in the first iteration of the systematic review discussed in Section 2.4.5; (ii) a set of reference architectures that encompass knowledge on how to structure robotic systems (Feitosa and Nakagawa, 2012); and (iii) expertise and knowledge of specialists on how to develop robotic systems. By investigating these sources, we obtained a broad set of services for analysis.

3.2.2 Step 2: Service Analysis and Categorization

Brainstorm meetings were organized for the analysis and classification of the services identified. During these meetings, we considered the expertise of specialists in software architecture, SOA, and robotics. Reference architectures for robotics used in Step 1 were again applied for the identification of the main modules and functionalities that should be considered in robotic systems. The following issues were mitigated during the categorization of services: (i) similar services with different names; (ii) services with the same name, but different functionalities; (iii) services lacking cohesion; and (iv) modules of the robotic systems that could be provided as services, but not identified in the information sources. As a result, we obtained five groups of services: Device Driver, Knowledge, Task, Robotic Agent, and Application. Afterwards, we identified the abstraction levels of each group of services.

3.2.3 Step 3: Taxonomy Establishment

We proposed the taxonomy of services for SORS based on the groups of services identified. It is composed of two main parts: (i) RSDS, which establishes dependencies between groups of services, and (ii) description of all types of services within the groups. We adopted the overall structure of the layered S3 reference architecture (see Section 2.3.3) to define the dependency stack. In particular, we considered in RSDS the dependencies established by the primary layers of S3 for operational infrastructure, simple services, composed services, and business processes. Figure 3.2 shows RSDS and its layers, which represent the groups of services. In RSDS, less abstract services provide functionalities for the higher abstract ones (e.g., Device Driver services support Task services). Moreover, services in lower layers, such as software abstraction of hardware devices, are more fine grained and provide functionalities that can be used in different application domains.

Services in higher-level layers coordinate services in lower-level layers to perform complete activities and are, therefore, more dependent of the application domain.

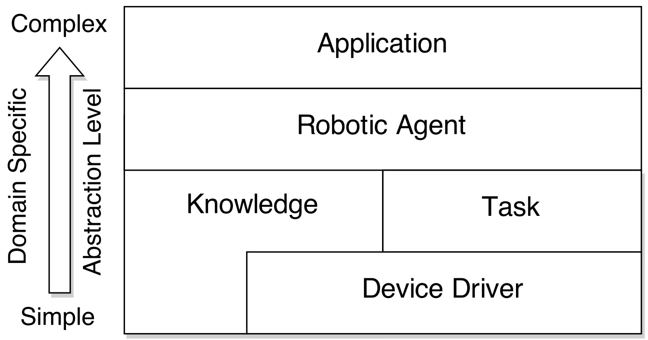


Figure 3.2: Robotics Services Dependency Stack

The Device Driver layer encompasses basic services that control hardware devices, providing their functionalities to the higher layers. The Knowledge layer represents basic and/or composed services that manage information used by robotic systems for making their decisions. This information can be based on data from sensors, databases, or both. Supported by Device Driver and Knowledge services, the Task layer groups composed services that provide tasks of robotics (e.g., mapping or localization) according to different behaviours. The Robotic Agent layer represents business processes that encapsulate the system controlling a robot as a service. Robotic agent services perform different tasks – using Task services – based on information gathered from Knowledge services. Finally, the Application layer encompasses high-level process services that coordinate one or more Robotic Agents in performing more complex activities, like surveillance and entertainment.

None of these five layers are mandatory during the design of SORS. For instance, a SORS can be designed without the use of lower level layers by creating coarse grained robotics services (e.g., a monolithic service that includes both task coordination and hardware abstraction). However, this practice may reduce flexibility and potential of reuse. A detailed description of each group of services is provided as follows. Observe that the taxonomy describes modules of robotic systems and, therefore, some content may have already been discussed along the background (see Section 2.4).

Device Driver Services: This group includes services that encapsulate hardware drivers, acting as a Hardware Abstraction Layer (HAL). Device driver services are used as design solutions to provide better integration among heterogeneous off-the-shelf resources. There are two sets of service types in this group: *Actuators* and *Sensors.* Services of this group are illustrated in Figure 3.3 and described as follows. Notice some devices currently available can combine more than one type of service, e.g., for sensing position and orientation.

In these cases, the device interface is a composition of the interfaces of its provided services.

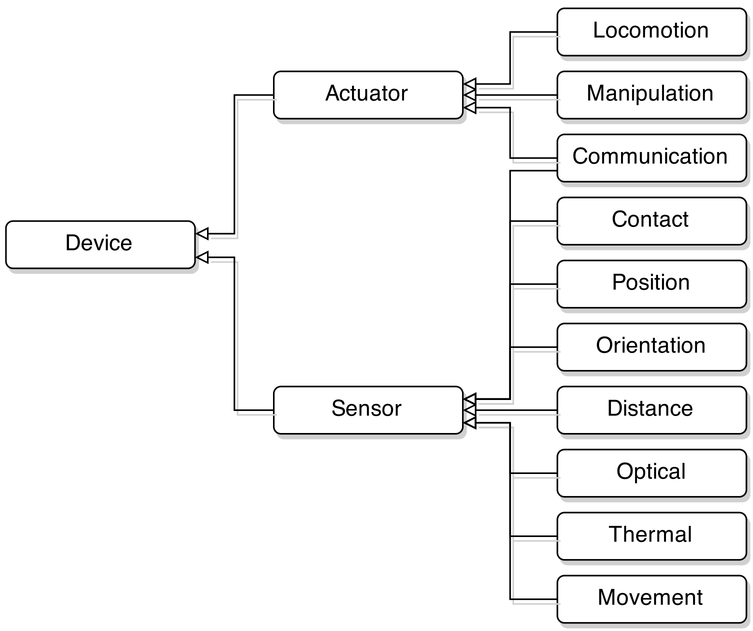


Figure 3.3: Device Driver service group

Locomotion: provide mobility to the robot. Examples of locomotion driver services include software for controlling wheels, joints, and helix;

Manipulation: enables the robot to manipulate or hold objects of the environment. Drivers for controlling arms and grippers are considered manipulation device services;

Communication: plays the role of both actuator and sensor. Communication is often bidirectional and the robotic system can use this type of service to feel the environment or interact with it. Drivers for multimedia, network, and radio frequency are examples communication services;

Contact: controls sensors that detect whether the robot is in contact with objects in the environment. Drivers for barrier and bumper are examples of contact services;

Position: obtains information of the location of the robot in the environment. A well-known example of position service is the GPS device driver service;

Orientation: provides information on the orientation of the robot in the environment. Inclinometer driver and compass driver are examples of orientation services;

Distance: measures the distance between the robot and the objects in the environment. Drivers for lasers and sonars are examples of distance measurement services;

Optical: converts images of the environment into digital information that can be processed. Drivers for stereo cameras are examples of optical sensor services;

Thermal: is used to measure temperature of objects inside the environment. Drivers for thermal cameras are well-known examples of thermal sensor services; and

Movement: measures the distance travelled by the robot. Encoder driver can be considered a movement service.

Knowledge Services: This group comprises services that gather, interpret, store, and share information necessary for the accomplishment of tasks and control of the robot as a whole. This information enables robotic systems to learn about characteristics of their environment and objects in it. Knowledge services use not only data from sensors, but also semantic information from a wide range of sources like ontologies and machine learning datasets. Services that belong to this group are shown in Figure 3.4 and detailed as follows.

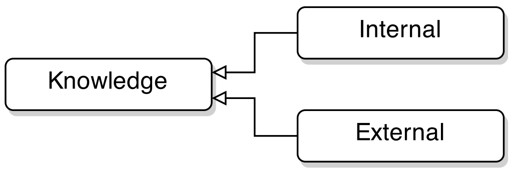


Figure 3.4: Knowledge service group

Internal: gathers, stores, and shares information obtained inside the boundaries of the environment. This information can be obtained from both sensors and databases hosted in a back-end server. Information obtained from sensors is generally produced by Task services and then stored in an internal knowledge service to be shared among robots. Two examples of internal knowledge are: (i) a service hosted in a back-end server that provides information on how to deal with a given object in the environment; and (ii) a service used to share the current representation of the environment among multiple robots; and

External: obtains and interprets general purpose information obtained from sources outside the environment. Through this type of service, a Web-enabled robot can search, access, and obtain information from machine-readable content on the Internet or even ordinary Web sites, portals, and wikis. External sources of knowledge enables the robotic system to learn procedures, semantic concepts, and relationships that were not considered during the design-time. An example of external knowledge is the service used by an autonomous robot for learning how to make pancakes based on information from the Web, described in (Tenorth et al., 2011).

Task Services: This group encompasses services that perform the tasks usually provided by robotic systems. Task services enable robots to perform simple activities, such as move from a position to another, and can be implemented according to different behaviours, i.e., a robot can move to another position either by following a wall or keeping the distance between walls. The group can be divided into seven main types, as shown in Figure 3.5. Details of these services are discussed as follows.

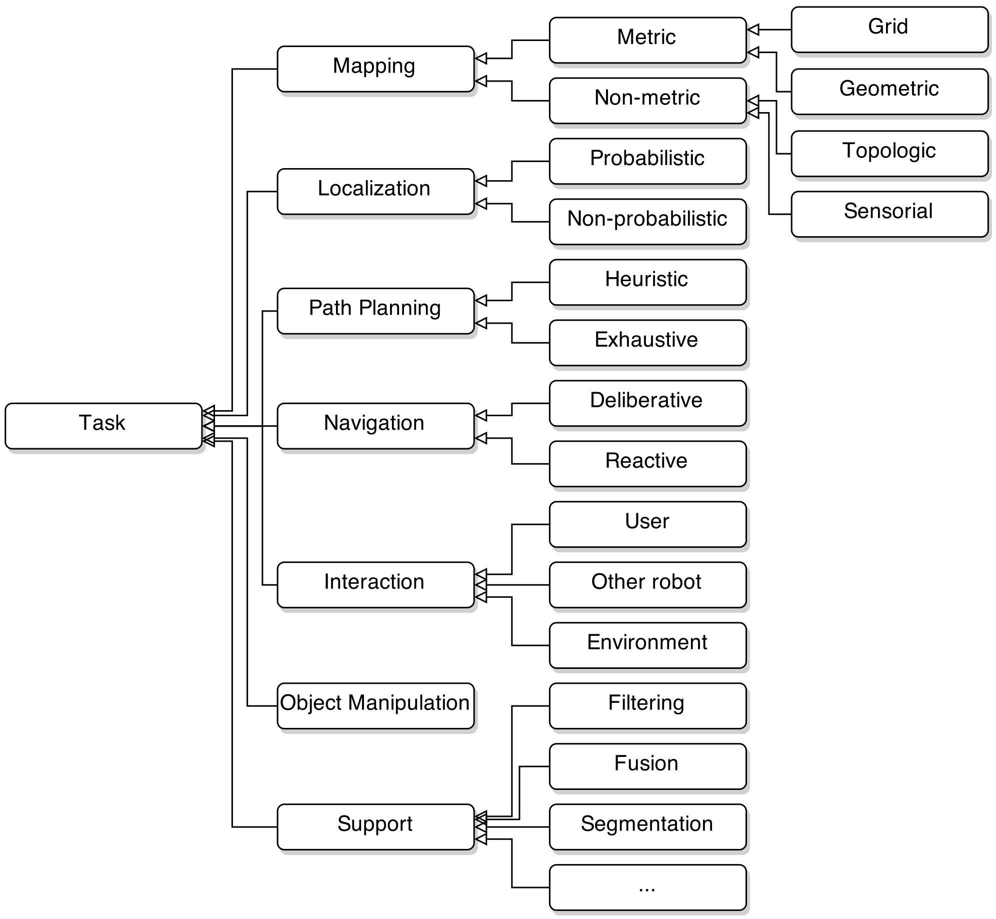


Figure 3.5: Task service group

Mapping: encapsulates algorithms that estimate and build representations of the environment where the robot is. There are two types of mapping services, those that use Metric Maps and those that use Non-metric Maps. Metric maps are 2D/3D representations that use coordinates – *Geometric* or in a *Grid* – to describe the real location of objects inside an environment. Non-metric maps are logical representations that can be either *Topological* (e.g., an adjacency graph) or *Sensorial* (e.g., sequence of images);

Localization: estimates the position of the robot. Two types of localization services exists: *Probabilistic* and *Deterministic*. Kalman filter is a well-known example algorithm that can be provided as Probabilistic localization service. Moreover, Deterministic localization service can be any localization algorithm based solely on data from GPS or other type of position service;

Path Planning: defines good (or optimal) paths between two or more positions in the environment. These services are often supported by mapping services and are developed based on two main strategies: (i) *Heuristic Search* (e.g., A-Star (A*) (Hart et al., 1968)) and (ii) *Exhaustive Search* (any type of service that creates a path by examining all possible paths);

Navigation: is used by the robot to navigate in the environment. The classification of a navigation service is associated with the control architecture it implements, i.e., *Deliberative* or *Reactive*. A service based on Deliberative navigation performs a path according to a predefined plan. Reactive navigation service controls the robot based on its current sensory data, e.g., using Potential Fields (Borenstein and Koren, 1989). A robotic system can combine these two types of services to provide a more robust hybrid control;

Interaction: enables the robot to work together with the *Environment* and *Other Robots*, or interact with a *User*. Interaction services support data exchange and procedures invocation between the robotic system and other systems. For instance, these services can be used to request to a system controlling a smart house to open the door of a room;

Object Manipulation: provides algorithms to support physical interactions with objects inside the environment. Object manipulation services encapsulate control algorithms that coordinate actuator devices, e.g., arms and grippers; and

Support: provides general purpose functionalities that support the development of robotic systems. These functionalities involve data filtering, data fusion, math calculations, point cloud processing, segmentation of images from cameras, and so forth. As support services are not only available for robotics, a finite number of subtypes of services that belong to this group can not be determined.

Robotic Agent Services: This group encompasses services that coordinate other services located in less abstract layers (i.e., Task, Knowledge, and Device Driver). Providing a robotic agent as a service enables robots to be remotely controlled and eases coordination of multi-robotic systems. As illustrated in Figure 3.6, two types of Robotic Agent services exist: *Non-mobile* and *Mobile*.

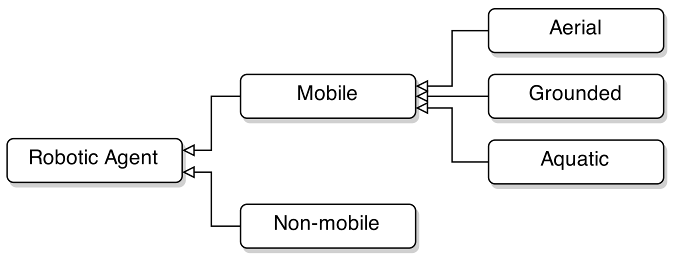


Figure 3.6: Robotic Agent service group

Non-mobile: provides functionalities of a robot without mobility capability. An example of a non-mobile service is a robotic system that controls an industrial robot designed to manipulate objects; and

Mobile: provides functionalities related to both locomotion and object manipulation. Due to the different types of mobile robots, as well as their distinct representations of position and orientation, mobile robotic services can be divided into three categories: *Aerial*, *Grounded*, and *Aquatic*.

Application Services: This group comprises services that manage robots to perform more complex activities. They are orchestrators that acquire knowledge through robotic agent services, process it, and then request a set of tasks that satisfy a given activity. This type of service enables designers to focus on the application itself rather than on the details of implementation of a robotic agent. Figure 3.7 shows the three different types of Application services, which are described as follows.

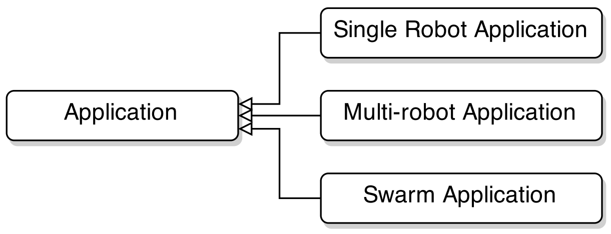


Figure 3.7: Application service group

Single Robot Application: describes, coordinates, and monitors high-level robotic activities. Robotic vacuum cleaning and intrusion detection are examples of this type of service;

Multi-robot Application: describes, coordinates, and monitors multiple robotic agents (i.e., multiple robots) for performing a given application. This type of service allocates tasks for robots according to their specific features and availability. The coordination of multiple robotic arms in a factory line is an example of this type of service; and

Swarm Application: coordinates and monitors a large amount of simple robots to perform cooperative applications. In a swarm application, all robots have the same service interface and provide the same functionalities. The measurement of the temperature of an environment by a robotic swarm is an example of this type of service.

3.2.4 Step 4: Taxonomy Evaluation

Our taxonomy was evaluated by a group of ten experts in robotics, invited through important discussion lists[1], who read the taxonomy documentation and then answered questions in an on-line survey[2]. The group comprised software architects, software engineers, software developers, and research team leaders from six different institutions in five countries, from both academia and industry. During the survey, we evaluated both acceptance and comprehension of the service taxonomy and the results are provided as follows. A discussion on these results is presented in Section 3.4.

Three main aspects were considered in the acceptance evaluation: (i) RSDS, (ii) groups of services, and (iii) taxonomy as a whole. For each aspect, we proposed statements that should be answered by experts according to a Likert scale (Wohlin and Andrews, 2003) with the following options: Strongly Agree, Agree, Tend to Agree, Neutral, Tend to Disagree, Disagree, and Strongly Disagree. A Neutral answer was given whenever the interviewee felt unsure about agreeing or disagreeing with a given statement. Text fields were also provided after each statement so that interviewees could justify possible disagreements.

Three statements were proposed in RSDS: ST1 - "*The layers of RSDS are sufficient to describe the main parts and organization of a robotic system*"; ST2 - "*The dependencies between layers of RSDS are coherent*"; ST3 - "*Layers of RSDS are disjoint, i.e., there is a clear separation among layers*". Table 3.1 shows the summary of answers for each statement in percentage terms. RSDS was considered (or tended to be considered) complete and coherent by 70% of the experts (i.e., more than 70% of answers about ST1 and ST2 were strongly agreed, agreed, or tended to agree). Besides, 60% of the experts were in favor of (strongly agreed, agreed, or tended to agree) and 30% were neutral on a clear separation between layers of RSDS. The negative answers in the three statements were given by the same interviewee, who was not in favor of the classification of robotics services into layers.

Three statements were proposed for the evaluation of the five groups of services: ST1 - "*The group of services is complete.*"; ST2 - "*The group of services is correct.*";

[1]The robotics-worldwide list is considered the most important forum of robotics. The robotica-l and robotics-australia-nz-list lists are the main forums of robotics community in Brazil and Australia/Asia, respectively.

[2]`http://goo.gl/mJkQTd`

Table 3.1: Acceptance of RSDS

Statement	Strongly Agree	Agree	Tend to Agree	Neutral	Tend to Disagree	Disagree	Strongly Disagree
ST1	30	40	10	10	10	0	0
ST2	30	30	10	20	10	0	0
ST3	20	20	20	30	0	10	0

and ST3 - "*The group of services has an adequate level of abstraction.*". Table 3.2 shows the results in percentage terms. A high degree of acceptance is observed in the Device Driver, Task, Robotic Agent, and Application groups, where an average of 87.5% of answers were positive (i.e., strongly agreed, agreed, and tended to agree). Except for few disagreements, positive answers were also given by experts regarding the Knowledge group. These disagreements are mainly related to the fact that this group comprises only two types of service (i.e., Internal and External). However, a deeper classification would probably lead to an incomplete set of subcategories, as we could not identify all possible types of internal and external sources of knowledge. Notice that a similar classification was also adopted elsewhere (Blake et al., 2011).

Table 3.2: Acceptance of the groups of services

Type	ST	Strongly Agree	Agree	Tend to Agree	Neutral	Tend to Disagree	Disagree	Strongly Disagree
Device Driver	ST1	30	30	20	0	20	0	0
	ST2	30	40	10	20	0	0	0
	ST3	30	30	40	0	0	0	0
Knowledge	ST1	10	30	20	20	20	0	0
	ST2	10	30	20	40	0	0	0
	ST3	0	40	10	20	10	0	20
Task	ST1	10	60	20	0	10	0	0
	ST2	10	50	30	10	0	0	0
	ST3	20	40	30	0	0	0	10
Robotic Agent	ST1	40	30	20	10	0	0	0
	ST2	30	30	20	20	0	0	0
	ST3	30	30	20	10	0	10	0
Application	ST1	20	40	30	10	0	0	0
	ST2	20	40	20	20	0	0	0
	ST3	30	30	40	0	0	0	0

Four statements were proposed for the evaluation of the overall acceptance of the proposed taxonomy: ST1 - "*I believe the taxonomy is clear and well-described*"; ST2 - "*I believe the taxonomy was defined adequately*"; ST3 - "*I believe the taxonomy is useful for describing services for diverse types of robotic applications*"; and ST4 - "*I believe the taxonomy is useful for describing services of different types of robotic systems*". Table 3.3 shows the results in percentage terms. It can be observed that an average of 80% have strongly agreed, agreed or tended to agree with all statements. Moreover, none of them have disagreed with the statements aforementioned. These results indicate the proposed taxonomy can describe and classify services currently available for the development of SORS.

Table 3.3: Overall acceptance of the taxonomy

Statement	Strongly Agree	Agree	Tend to Agree	Neutral	Tend to Disagree	Disagree	Strongly Disagree
ST1	30	30	20	20	0	0	0
ST2	40	30	0	30	0	0	0
ST3	50	10	30	10	0	0	0
ST4	20	40	20	20	0	0	0

The experts' understanding of the taxonomy was also assessed during the survey. They were asked to classify services using the taxonomy to answer different questions. For example, we requested them to indicate the group in which a service that provides a collision avoidance algorithm should be classified; or if a service that controls a robot to monitor an office should be classified as a Task service or an Application service. Table 3.4 summarizes the topics evaluated, the number of multi-choice questions for each topic (Questions (#)), number of choices available for each question (Choices (#)), and average of correct answers (Avg. score). Questions about the Robotic Agent service group were not asked, since the only difference among services in this group is the type of robot adopted and service interface associated with them.

Table 3.4: Questions and scores obtained in the assessment

Topic	Questions (#)	Choices (#)	Avg score (%)
RSDS	8	5	78.8
Device Driver	6	5	90.0
Knowledge	4	2	80.3
Task	6	4	88.3
Application	4	2	80.0

Notice that 78.8% of the questions about the RSDS were correctly answered. Besides that, between 80% and 90% of the 20 questions about the types of service in the five groups were correctly answered. These results evidence a good comprehension of our taxonomy and its use by the experts for the classification of services for SORS. We believe that the understanding of the taxonomy can be even higher as other robotics services available in the literature are classified into the proposed service groups. The larger the number of examples of services classified in a service group, the easier the decision on the fitting of a new service in such group.

The classification described in the taxonomy provided a conceptual base for automating cataloging and discovery of services for SORS. Next section presents a mechanism that we have developed to enable developers to discover and reuse services for robotic systems.

3.3 Automating Cataloging and Discovery of Services for SORS

The development environments available for SORS enable researchers to create services that can be used in different projects. These environments also provide functionalities for the construction of robotic systems by means of composition of services into business processes. However, finding the appropriate service for reuse during the development of SORS is currently a difficult task. No development environment provides a mechanism that adequately supports publication and discovery of services for SORS. Developers should either create services for their robotic system from scratch or try to find equivalent implementations in repositories containing hundreds of different services. For instance, to find a service in the ROS Wiki, one needs to read the description of all services that seem to provide the intended functionalities (i.e., services are not classified into categories and searches are based on service names). Moreover, users of such a repository must follow daily updates in the site to be aware of any new content available.

To facilitate the discovery of services at design time, we developed RoboSeT (Robotics Services Semantic Search Tool), a mechanism that enables classification and search of services. RoboSeT is composed of two main parts: on-line service repository and plug-ins that can be locally integrated into development environments. The service repository enables developers to publish robotics services hosted in different version control systems, such as Git[3] and SVN[4], and describe these services using the taxonomy proposed in the previous section. Each service registered by a service provider is classified according to the type it belongs in the taxonomy and, therefore, can be discovered semantically. Service consumers can search for services by using either the Web interface provided by RoboSeT or a plug-in installed in their machines. Plug-ins are applications integrated into development environments that access the service repository and allows developers to search and obtain services. Using RoboSeT, services for SORS can be discovered and integrated into local projects transparently, i.e., the service consumer does not need to know where the service is located or who the service provider is. Figure 3.8 illustrates the overall organization of RoboSeT. Further details about its development and functionalities

[3]`http://git-scm.com/`
[4]`http://subversion.tigris.org/`

are provided as follows. Graphical examples and tutorials on how to use its functionalities are available in the Website[5].

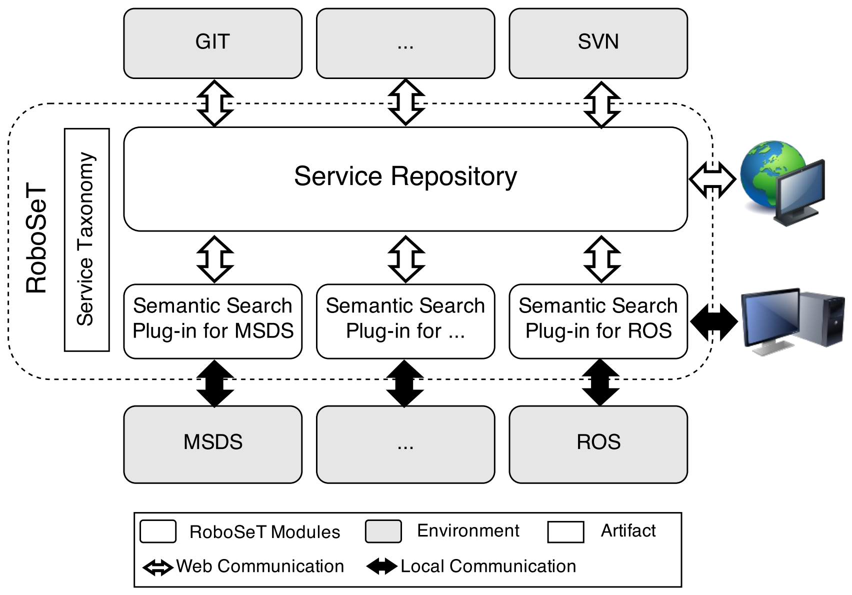


Figure 3.8: Overview of RoboSeT

3.3.1 Services Repository

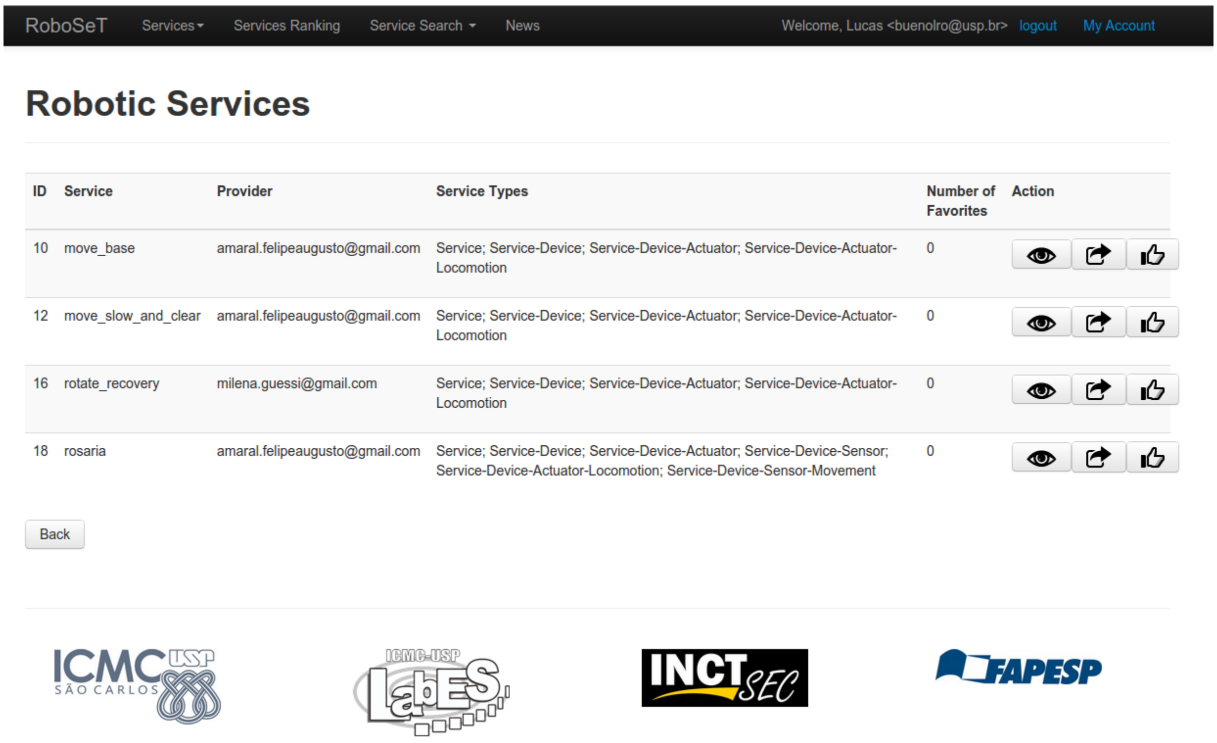
RoboSeT enables users to store and classify information about robotics services in a repository, as well as search for services to be reused in their robotic systems. Registered service providers can publish and describe their services by using the taxonomy. These services are linked to a control version repository where they are hosted, and the provider describes how they work, license types, versions, and so forth. Service consumers use the taxonomy to search for services that provide functionalities they need.

For instance, a consumer that needs a service for robotic localization based on a high precision GPS (Global Localization System) can search for a service whose type is "Service/Task/Localization/Non-probabilistic" (for more information, see Figure 3.5). This query retrieves all services in the repository that provide a functionality of the Task type, related to Localization, and based solely on information of a sensor. A less precise search, like "Service/Task/Localization", also retrieves services of the first query and all other services described as a subtype of Localization, e.g., probabilist localization services.

Figure 3.9 illustrates results of a search performed in the RoboSeT service repository. In this screen, service consumers obtain a summary of the services retrieved in the search, including names, providers, and related service types. Buttons in the right side of the screen can also be used for obtaining further information on a service, accessing its hosting

[5]http://www.labes.icmc.usp.br:8595/RegistroServicoWeb/

address, or endorsing it. Endorsements are used to recommend a given service and indicate to other potential consumers that the service is a worthwhile choice. As discussed in Section 2.3.1, reputation is one of the attributes that may indicate the quality of a service. In addition to searches using the taxonomy, RoboSeT also provides the following functionalities:



Robotic Services

ID	Service	Provider	Service Types	Number of Favorites	Action
10	move_base	amaral.felipeaugusto@gmail.com	Service; Service-Device; Service-Device-Actuator; Service-Device-Actuator-Locomotion	0	
12	move_slow_and_clear	amaral.felipeaugusto@gmail.com	Service; Service-Device; Service-Device-Actuator; Service-Device-Actuator-Locomotion	0	
16	rotate_recovery	milena.guessi@gmail.com	Service; Service-Device; Service-Device-Actuator; Service-Device-Actuator-Locomotion	0	
18	rosaria	amaral.felipeaugusto@gmail.com	Service; Service-Device; Service-Device-Actuator; Service-Device-Sensor; Service-Device-Actuator-Locomotion; Service-Device-Sensor-Movement	0	

Figure 3.9: Result of a search in the service repository

Account management: an account is required for the access to the service repository. Three possible types of user are available: consumer, provider, and administrator. Consumers can search for and obtain services in the repository. Providers can obtain services for reuse and also publish their own services in the system. Administrators manage accounts, the elements of the service taxonomy, and other important aspects of the systems. The account management enables users to edit information and promote their account types from a consumer to a provider. Administrators can also invite consumers and providers to administrate the service repository;

Service management: enables service consumers to get information on services they have already obtained, such as configuration instructions, comments from other users, and bug reports. Service providers can also manage their published services and register new services in the repository;

Service ranking: a ranking is provided for the most relevant services available in the repository, therefore, users can find services of better reputation and data about their providers. Similarly to the service search functionality, additional information on a given service can be obtained and the repository where the service is hosted can be accessed;

Service search: additionally to the search by type, it is possible to look for services using search strings. This type of search complements the semantic search and enables users to find services by their names and providers. Services are also obtained through searches in parts of the text contained in the service description;

Service detailing: enables service consumers to obtain all information available about a service, including its full description, service dependencies, number of users, number of endorsements, versions, and license of use. Comments made by previous consumers and reported bugs are also available as references. A complete list of quality attributes is provided to support consumers in identifying services of higher quality. Services in the repository can be graded by their consumers in each quality attribute of ISO/IEC 25010:2011 (see Section 2.3.1); and

News about services: news is automatically generated based on logs of the system and provided to users in a customized way. Users are notified on updates in the services they are using, reported or fixed bugs, and so forth. They are also informed on any changes in the service repository, including as updates in both taxonomy or quality attributes used for the evaluation of services for SORS.

We adopted the MVC (Model-View-Controller) and the DAO (Data Access Object) architectural patterns to design a more modular system. RoboSeT modules were divided into four layers, namely: Model, View, Controller, and Database. These layers were organized into packages labelled by stereotypes (e.g., classes into Servlets package are associated with Controller layer), as shown in Figure 3.10. The Entities package contains classes that represent entities of RoboSeT, such as those related to published services and users of repository. The DAO package contains classes responsible for the persistence of entities in the database. Since we adopted Hibernate[6] framework, XML files for each entity were placed in the Mappings package. The JSP package contains Web pages developed by using the Twitter Bootstrap[7] framework. The Controllers package contains classes responsible for the communication among graphical interface, the entities, and the DAO package. Different servlets were created and placed in the package Servlets for the integration of the graphical interface into the controllers.

[6]`hibernate.org/`
[7]`getbootstrap.com/`

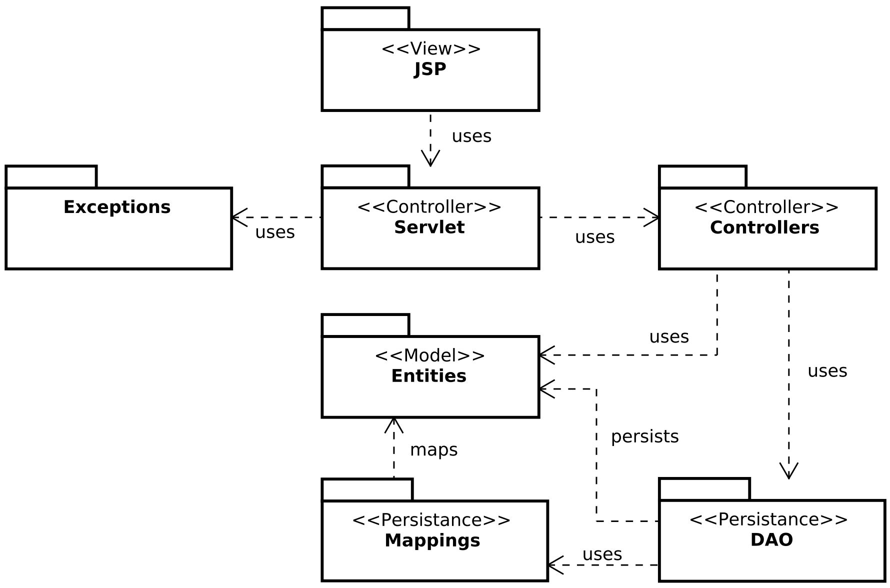


Figure 3.10: Diagram of packages of the service repository

3.3.2 Semantic Search Plug-in

A RoboSeT plug-in is a local application that remotely searches services in the repository and integrates results into the project of a robotic system. Therefore, developers designing a robotic system can create a project in their development environment using the plug-in, search for services necessary for their system, and reuse them. Only services unavailable in the repository and services used to integrate functionalities of the robotic system have to be implemented. Services used to integrate and coordinate other services are more domain-specific and should be implemented according to the requirements of each project. We have created a plug-in for the ROS development environment to exemplify functionalities that should be provided by these applications. Although this implementation is specific for ROS, the functionalities can be adapted to any other SORS development environment. The following activities were automated by the plug-in for supporting creation of robotic systems:

Project creation: as this plug-in acts as a layer on top of ROS, we have developed a functionality to abstract the creation of projects in this environment. Therefore, the whole ROS file-system can be built using the plug-in, including the appropriate build and manifest files. The functionality implemented in this plug-in works as a proxy for the respective functionality in ROS (i.e., the roscreate-pkg command);

Identification of types of services: during the design time, developers should identify services that will be used for the creation of a robotic system. They analyse, group, and map the robotic system requirements into the service types of the taxonomy.

To support this activity, the last version of the taxonomy available in the repository is obtained every time the plug-in starts, so that developers can get any additional information on a given type of service in their local machines (e.g., parent type, description, and examples);

Service search: searches for each type of service identified for the robotic system are performed in the repository by using the plug-in. As a result, all services that have matched the searched types are presented along with their ID, description, provider information, license type, and number of endorsements;

Service selection: searches for a given type of service can retrieve more than one service implementation. Therefore, the plug-in enables developers to obtain additional information about services in the repository. The number of recommendations received and score of each quality attribute can help developers to choose the service to be used;

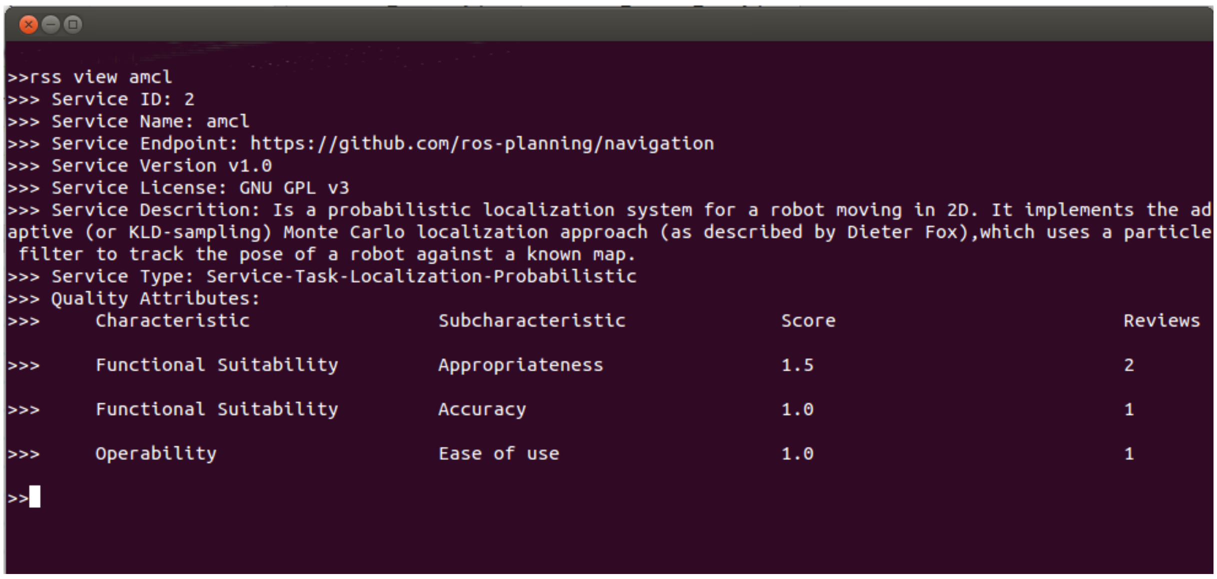
Service obtaining: the services that will be used in a robotic system can be obtained by the plug-in. A request using the service ID is sent to the service repository, which answers with the url of the location of the service. The service is automatically accessed in the version control repository and downloaded into the local environment. Notice robotic systems are real-time systems and most of their services must be deployed and executed locally;

Service deployment: services obtained by the plug-in are deployed inside the current ROS project to be integrated with other services being developed;

Service evaluation: users can evaluate the quality of services they are using at any time. Five levels of quality can be assigned to each quality attribute of a service: (1) unsatisfying, (2) needs improvement, (3) regular, (4) good, and (5) excellent. As mentioned in the previous section, a service can be evaluated according to any quality attributes of ISO/IEC 25010:2011. We adopted this quality model as a quality reference to avoid different interpretations of quality attributes. As a standard, it provides detailed and widely accepted descriptions of software quality attributes that can be used as a common vocabulary; and

Comments and bug report: when necessary, users can provide comments and report errors in the services directly from the plug-in. Comments can be either improvement requests or tips that might be useful for other developers of SORS.

The plug-in we designed for ROS has been implemented in Java and has a command-line user interface. Figure 3.11 illustrates such an interface being used to obtain information about a probabilistic localization service named *amcl*. We opted for a command-line plug-in to make it more familiar to ROS developers, who already use this type of interface to interact with the development environment. A help command is available to support developers in learning how to use functionalities available. Currently, this plug-in does not support publication of services directly into the service repository. Nevertheless, new services developed in projects that use the plug-in can be published through the Web interface and reused in other projects.



```
>>rss view amcl
>>> Service ID: 2
>>> Service Name: amcl
>>> Service Endpoint: https://github.com/ros-planning/navigation
>>> Service Version v1.0
>>> Service License: GNU GPL v3
>>> Service Descrition: Is a probabilistic localization system for a robot moving in 2D. It implements the ad
aptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox),which uses a particle
 filter to track the pose of a robot against a known map.
>>> Service Type: Service-Task-Localization-Probabilistic
>>> Quality Attributes:
>>>     Characteristic              Subcharacteristic           Score                       Reviews

>>>     Functional Suitability      Appropriateness             1.5                         2

>>>     Functional Suitability      Accuracy                    1.0                         1

>>>     Operability                 Ease of use                 1.0                         1

>>>
```

Figure 3.11: Interface used to integrate the plug-in with ROS

3.3.3 RoboSeT Usage Example

In order to illustrate the use of RoboSeT, this section describes the design of a robotic system of robust navigation capability. Robust navigation enables a robotic system to guide robots through the environment without risk, avoiding collision with humans, objects, and other robots. The design of a robotic system for robust navigation involves coordination of different tasks, including path planning, motion control, and sensor data processing. Functionalities associated with the development of the system are described as follows:

Motion Planning: creates a global path between a given starting position and the goal position represented by a sequence of intermediate points in a static environment;

Trajectory generation: defines the velocity of each part of the path. The trajectory is represented as a sequence of planned intermediate positions and associated velocities;

Obstacle detection and representation: represents objects in the environment by using data from different sensors. These objects and their respective positions are used to update the map of the environment;

Obstacle avoidance: adapts the pre-defined global trajectory to avoid unexpected objects in the path, e.g., humans or other robots. It enables the robot to navigate in dynamic environments safely;

Position and velocity control: generates instant linear and angular velocity so that the robot can navigate along the computed trajectory. Local navigation is strongly related to the kinematic model used by the robot; and

Localization: estimates the position of the robot with respect to a global map by using different types of sensors, e.g., odometer, camera, and laser rangefinder.

Pioneer P3-DX[8] robot was adopted as a base platform in this study. It moves using two-wheel differential drive with rear balancing caster. The two front wheels are independent and can have different speeds if necessary. Each wheel has an encoder sensor that supports the localization task. Moreover, we also used a laser sensor to provide information about objects surrounding the robot. Given these hardware specifications, drivers for robot and laser were considered in the design of the robotic system.

The first step to design the robotic system was investigate services available for reuse. After creating an ROS project, each functionality necessary to develop the robotic system was classified according to the service types available in the taxonomy. Table 3.5 shows functionalities of the robotic system and types of services they are associated with.

Table 3.5: Functionalities of the robotic system and related service types

Functionality	Service type
Motion Planning	Service/Task/Path planning
Trajectory generation	Service/Task/Path planning
Obstacle detection and representation	Service/Task/Mapping
Obstacle avoidance	Service/Task/Path planning
Position and velocity control	Service/Task/Navigation
Localization	Service/Task/Localization
Encoder controller	Service/Device/Sensor/Movement
Differential drive controller	Service/Device/Actuator/Locomotion
Laser controller	Service/Device/Sensor/Distance

The search for reusable services was then performed based on the identified types. Each service type was applied to the plug-in developed for ROS so that services available

[8]`http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx`, last accessed in February 5th, 2015.

in the service repository could be found. Results of the search were analysed and the most adequate services for our robotic system were selected. Task service and Device Driver service were the only two groups of the taxonomy necessary for designing robotic systems with robust navigation capabilities. However, design of more complex SORS may require other groups of service. Table 3.6 shows services identified for functionalities related to the Task group and the service types associated with them.

Table 3.6: Functionalities of Task group, service types, and services identified for reuse

Functionality	Task service types	ROS service
Motion Planning	Path planning/Heuristic Search	NavfnROS, CarrotPlanner
Trajectory generation	Path planning/Heuristic Search	TrajectoryPlannerROS, DWA-PlannerROS
Obstacle detection and representation	Mapping/Metric/Grid	CostMap2D
Obstacle avoidance	Path planning/Heuristic Search	TrajectoryPlannerROS, DWA-PlannerROS
Position and velocity control	Navigation	MoveBase
Localization	Localization/Probabilistic	Amcl

The *CostMap2D* service implements a 2D costmap that takes in sensor data from the world and builds a 2D or 3D occupancy grid of the data. *NavfnROS* and *CarrotPlanner* are two complementary implementations of *BaseGlobalPlanner* interface for ROS. *Navfn-ROS* is an A* path-planner for maps described by occupancy grids. *CarrotPlanner* is a simpler planner that calculates a straight line between the robot position and the goal position and checks collisions along the path. Both path planners are complementary services that can be used to calculate a global path. *TrajectoryPlannerROS* and *DWA-PlannerROS* can be used to generate a trajectory for a defined global path. These services produce velocity commands based on the map, the global path, and unexpected objects close to the robot, as well as provide functionalities associated with obstacle avoidance. The localization of the robot can be estimated by the *Amcl* service, which uses Monte Carlo probabilistic method to reduce the error of encoder measurements. Observe that some task services in the repository were able to provide more than one type of functionality. For instance, services *TrajectoryPlannerROS* and *DWAPlannerROS* were associated with both trajectory generation and obstacle avoidance.

We also obtained services for controlling hardware devices used by services of task group. Table 3.7 shows the services identified for functionalities related to the Device Driver group and the service types associated with them. *RosAria* service provides means for controlling the differential drive and the encoder of Pioneer P3-DX. *SICK Toolbox* service was identified for the control of the SICK[9] laser rangefinder. Similarly to services in task group, *RosAria* provided functionalities in more than one type of service.

Table 3.7: Functionalities of Device Driver group, service types, and services identified for reuse

Functionality	Device service types	ROS service
Encoder controller	Sensor/Movement	RosAria
Differential drive controller	Actuator/Locomotion	RosAria
Laser controller	Sensor/Distance	SICK Toolbox

Figure 3.12 illustrates the software architecture we designed for the robotic system using the services found in RoboSeT repository. In this architecture, *CostMap2D* builds its map based on information provided by *SICK Toolbox* service. *MoveBase* service orchestrates mapping (*CostMap2D*) service and other services for localization (*Amcl*) and path planning (*NavfnROS*, *CarrotPlanner*, *TrajectoryPlannerROS*, and *DWAPlannerROS*) to generate robust navigation commands. These commands are provided to *RosAria* service, which acts as an interface to control the Pioneer P3-DX robot. As the robot moves in the environment, *RosAria* service collects data from the odometer sensor. The odometry information is consumed by *Amcl* service and used to estimate the current localization of the robot.

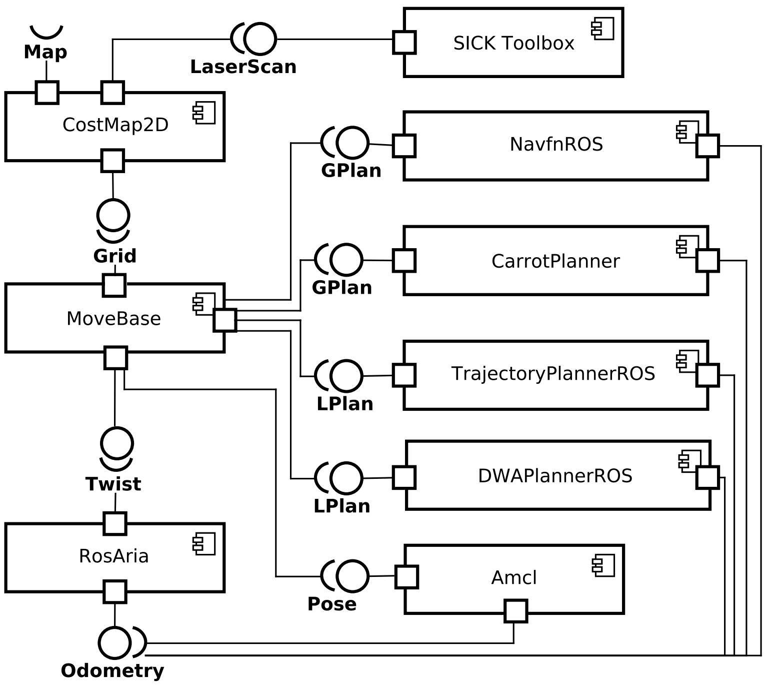


Figure 3.12: Software architecture of the robotic system

[9]`http://www.sick.com/group/EN/home/products/product_portfolio/laser_measurement_systems/Pages/laser_measurement_technology.aspx`, last accessed in February 5th, 2015.

The navigation system described in the software architecture is similar to the architecture of the 2D Navigation Stack[10] available for ROS. In fact, services identified by the plug-in are part of this ROS stack, but registered in the service repository as independent services. Currently, the services identified for robust navigation are strongly dependent on *MoveBase* to start and, therefore, to work properly. However, a study has already demonstrated these services can work independently after a refactoring process (Brugali et al., 2012a). Benefits and limitations of RoboSeT are discussed in the following section.

3.4 Discussion of Results and Limitations

The definition of a widely accepted classification of elements of a given domain is a difficult task. Aiming at establishing a representative taxonomy of services for SORS, we considered a large amount of information sources, including results of a systematic review and the knowledge of specialists in robotics. We also submitted this taxonomy to be evaluated by the robotics community and observed its acceptance and comprehension. Results indicate our taxonomy can be used for the classification of services for SORS. Results also evidenced a good comprehension of the taxonomy and its service groups. Minor issues were pointed out, mainly related to the Knowledge group and the completeness of the Device Driver group. Few interviewees referred services of Knowledge group as part of the Task group, albeit these groups play different roles. Knowledge services are specially dedicated to manage and share information in robotic systems, which is sometimes produced by services from Task group. Regarding the completeness of the Device Driver group, we highlight that our taxonomy does not encompass all types of drivers for sensors and actuators available for embedded systems, but the ones frequently used in the development of SORS. It is worth mentioning this taxonomy can be updated as the research area evolves.

The proposed taxonomy took the first step towards a better communication among developers of SORS and enabled the creation of a mechanism to support classification and discovery of robotics services. The example in the previous section illustrated how services can be identified, obtained, and reused to develop an SORS. Although services used in this example are also available in ROS Wiki, their localization depends on the developers' previous knowledge. Without the support provided by RoboSeT, researchers not aware of the existence of these services need to manually search them among hundreds of other services. RoboSeT also promotes indirect communication among service providers and service consumers. These characteristics can yield benefits in three different perspectives: service consumer, service provider, and robotic system user.

[10]`http://wiki.ros.org/navigation`, last accessed in February 5th, 2015.

From the service consumer perspective, RoboSeT can facilitate discovery of services and, therefore, improve reuse during development of robotic systems. It is intuitive that services easier to be found are more likely to be reused. However, the reuse of a service does not depend only on its discovery, but on its documentation and suitability to the robotic system under development. RoboSeT provides functionalities that enable consumers to obtain structured information on services being searched, including documentation, comments from other users, and quality attributes. These functionalities aim at facilitating the identification of the most suitable service for each robotic system. As a direct consequence of reuse, RoboSeT intends to improve productivity in the development of robotic systems. Although quantitative evidences are still necessary, studies in the literature have already shown that reuse improvements positively influence productivity in the development of software systems (Mohagheghi and Conradi, 2007).

As a counterpart, service providers of RoboSeT receive in-use feedback from the robotics community about services they have published. Services provided to the community are generally executed in several environment configurations and on different robotic platforms, which represents an important corpus of evaluation. Comments, suggestions, bug reports, and quality evaluation of these services are organized in the Web interface of RoboSeT for each service provider. Providers can use the My Services section in the Web interface as a guide to improve the quality of their services. By improving the quality of the service they provide to the community, providers can also improve the overall quality of their own robotic systems.

Collaboration among service providers and service consumers through RoboSeT can also provide benefits from the perspective of robotic systems users. As service providers receive feedback for their services and use it to improve the overall quality of their systems, higher quality robotic systems are made available for end users. Besides, reuse improvements in the development of software systems can reduce costs and result in more affordable systems (Mohagheghi and Conradi, 2007).

Despite the RoboSeT benefits, it is also important highlight limitations of such mechanism. Similarly to any repository of services, the success of RoboSeT strongly depends on the cooperation of the robotics community. We are aware that without adoption of RoboSeT by the community, few services will be available for searching, fewer consumers will be interested in searching for them, and weak feedbacks will be offered as counterpart to providers. Therefore, we have designed RoboSeT to be as flexible as possible to stimulate its adoption. The taxonomy can be evolved and modified according to the community needs and new quality attributes can be proposed for the evaluation of services. Besides that, we aim at releasing RoboSeT and its plug-in for ROS as open source software to encourage and support development of plug-ins for other environments.

3.5 Final Remarks

This chapter has addressed the establishment of a taxonomy of services for SORS and its use to develop a tool that supports publication and discovery of robotics services. This taxonomy can contribute to the robotics area by providing a common vocabulary and knowledge on how to describe services for SORS. RoboSeT tool advances the current means of publication and discovery of services for SORS, enabling developers to identify, select, obtain, and deploy services in a transparent, automated manner. Results of a survey applied to specialists in robotics indicate the taxonomy is representative for classifying services for SORS. The example described in Section 3.3.3 shows that RoboSeT can be used to support discovery and reuse of services during the development of an SORS.

The taxonomy proposed in this chapter can be used as guide for identifying services that will constitute SORS software architectures. Next chapter presents a process for the development of these architectures.

Chapter

4

A Process for the Development SORS Software Architectures

4.1 Overview

Software architecture design directly impacts on the success of software systems projects. Performing this activity in an adequate manner is fundamental for the development of high-quality systems (Bass et al., 2012). As discussed in Section 2.4.4, software architecture design has been considered an important concern of robotics in recent years. However, SORS software architectures are still created without support of a systematic process and depends on the designers' personal experience in both robotics and SOA. Traditional design approaches, including those for object-oriented software, do not adequately support the creation of software systems based on SOA (Arsanjani et al., 2008). Current approaches for the development of SOA-based systems do not consider particularities of robotic systems design, such as limited processing capability and real-time constraints. Therefore, the proposal of a process for supporting the SORS architectural design can facilitate the development of these systems and positively impact on their overall quality.

This chapter introduces ArchSORS (Architectural Design of Service-Oriented Robotic System), a process that aims at systematizing the design of SORS software architectures. Section 4.2 details the ArchSORS process. Section 4.3 describes an experiment for the evaluation of the process. Results and limitations of this experiment are discussed in Section 4.4.

Contents of this chapter were previously presented in the paper *"Towards a Process to Design Architectures of Service-Oriented Robotic Systems"*, published in the Proceedings of the 8th European Conference on Software Architecture (ECSA'14) (Oliveira et al., 2014a).

4.2 Establishment of ArchSORS

ArchSORS is a process that intends to foster the systematic development of SORS software architectures. In order to design this process, we considered different sources of information on the development of software architectures, SOA-based systems, and robotic systems. ArchSORS encompasses the steps proposed by Hofmeister et al. (2007) to design software architectures (architectural analysis, architectural synthesis, and architectural evaluation) and the phases proposed by the consolidated SOMA method (Arsanjani et al., 2008) for the development of service-oriented systems. It is also based on the SORS software architectures identified through the systematic review reported in Section 2.4.5 and reference architectures that encompass knowledge of how to structure robotic systems (see Section 2.4.4). Figure 4.1 shows the overall structure of ArchSORS.

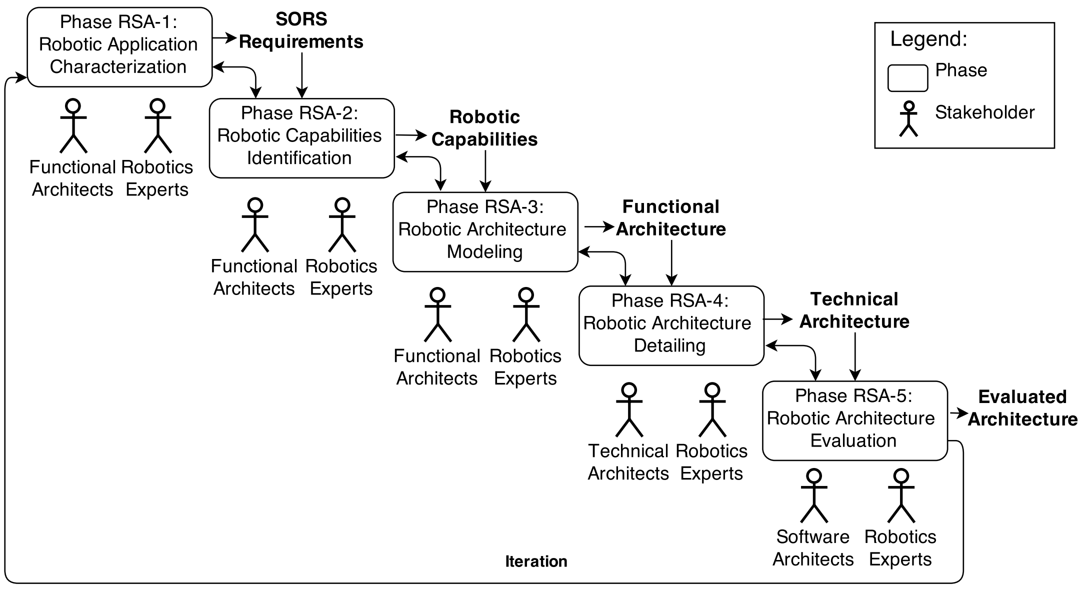


Figure 4.1: ArchSORS: a process to develop SORS software architectures

ArchSORS is divided into five phases applied in an iterative, incremental manner. These phases can also be guided by reference architectures, which enables the development of SORS software architectures through stepwise refinements of abstract templates.

In short, in order to establish a software architecture using ArchSORS, it is first necessary to characterize the robotic application and to produce a document of requirements (Step RSA-1). Following, in Step RSA-2, these requirements are used to model the application flow and identify capabilities that the robotic system should provide. Then, in Step RSA-3, a functional architecture is described and represented in terms of services used to provide the identified capabilities. In Step RSA-4, services of the functional architecture are further described and decisions about hardware infrastructures are made, which result in the technical architecture of an SORS. Finally, in Step RSA-5, the SORS software architecture is evaluated by using architectural analysis methods. If necessary, the evaluated architecture is refined through new iterations. Software architects (functional and technical) and robotics experts are involved and conduct phases of the process.

Each phase of ArchSORS is supported by a method that describes activities to be performed and their order of precedence. These activities are further detailed into sets of tasks. Milestones indicate the end of a method and the completion of a key deliverable. We described ArchSORS in SPEM (Software & Systems Process Engineering Metamodel Specification) (OMG, 2015c), which is a standard for process representation. We also created a *method plugin* in IBM Rational Method Composer[1] to enable the ArchSORS instantiation for particular projects and its integration into project management tools, e.g., Microsoft Project[2]. The ArchSORS phases are detailed as follows[3].

4.2.1 Phase RSA-1: Robotic Application Characterization

The design of an SORS software architecture requires the characterization of the application (mission) in which the robotic system will be used. Therefore, robotics experts describe the robotic application in terms of goals, main activities, as well as characteristics of the system being developed and its operating environment. They also identify applicable policies, rules, and constraints related to the operation of the robotic system in this environment. Characteristics of this application are used to guide elicitation of the SORS requirements. These requirements are then reported in a document, which is the deliverable that indicates the end of this phase. Figure 4.2 illustrates a method containing a set of activities to be performed in Phase RSA-1. Descriptions of these activities are presented as follows.

RSA-A 1.1 – Initiate project activities: The main goals and characteristics of the robotic application are defined, described, and documented. Robotics specialists organize brainstorm meetings to identify: (i) goals related to the robotic application; (ii) activities

[1]`http://www-03.ibm.com/software/products/en/rmc`, last accessed in February 13th, 2015.
[2]`http://www.microsoft.com/project`, last accessed in February 13th, 2015.
[3]Additional documentation on ArchSORS is available at the process website: `http://goo.gl/ykQ2d9`.

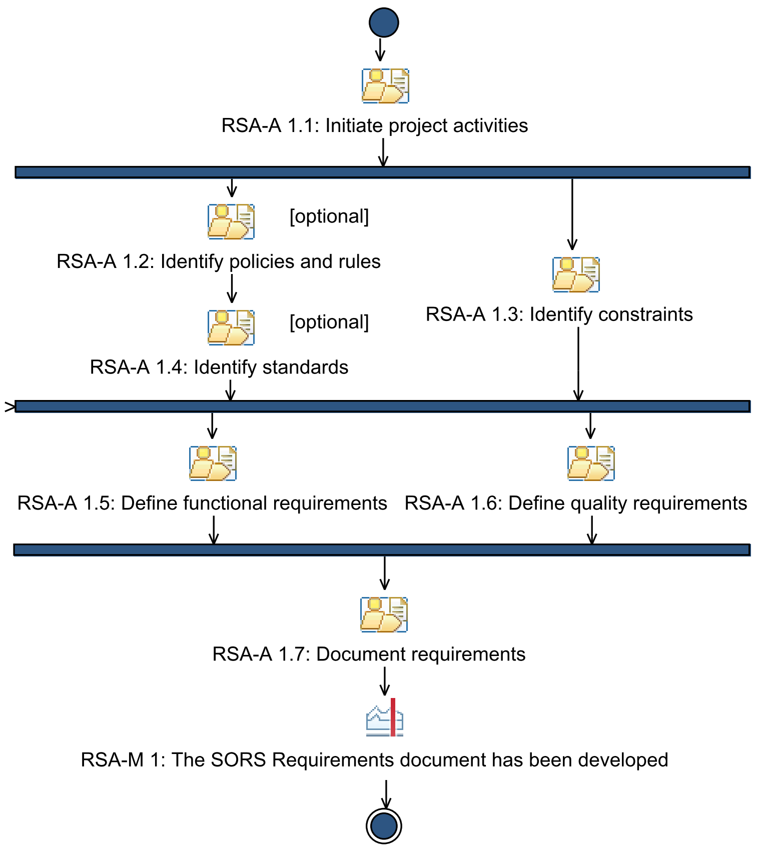

RSA-A 1.1: Initiate project activities
[optional]
RSA-A 1.2: Identify policies and rules
RSA-A 1.3: Identify constraints
[optional]
RSA-A 1.4: Identify standards
RSA-A 1.5: Define functional requirements
RSA-A 1.6: Define quality requirements
RSA-A 1.7: Document requirements
RSA-M 1: The SORS Requirements document has been developed

Figure 4.2: Method for the characterization of robotic applications

the robotic system should perform to achieve these goals; (v) the environment where the robotic system will operate (indoor, outdoor or both); (iv) type of robotic system to be developed, i.e., if the application involves a single robot, a team of robots, or a swarm; and (v) type of robot (or types of robots) to be used, the characteristics related to its mobility (if it will be mobile or non-mobile), how it will move through the environment, its size, and so forth. At this point, no assumption is made on which hardware devices will be used with the robotic system. Figure 4.3 shows the tasks of Activity RSA-1.1 and their order precedence.

RSA-A 1.2 – Identify policies and rules: Robotic applications must conform with applicable policies and rules to be commercialized and used. Therefore, robotics experts and functional architects identify policies and rules related to the robotic system. A policy, for instance, is defined by a law that regulates the operation of a given type of robotic system. Rules are restrictions on the robotic system design and operation that must be respected to comply with a policy. For example, to comply with a given law, the robotic system should enforce safety by using redundant, independent sensors to measure the distance from the objects.

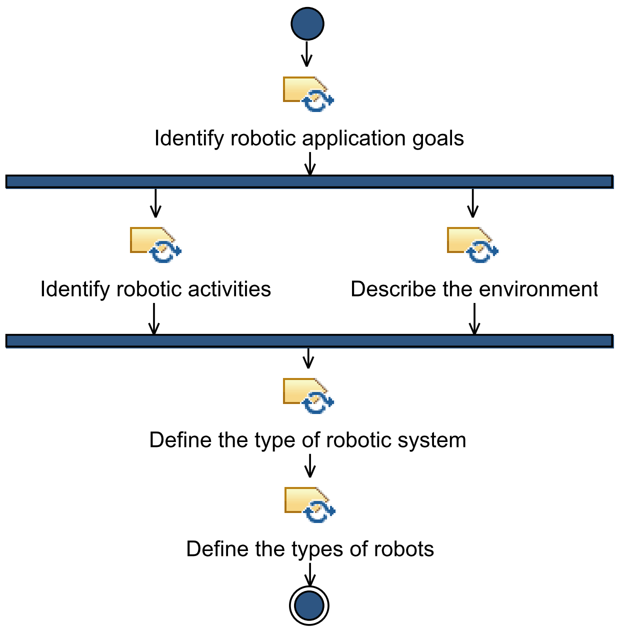

Identify robotic application goals
Identify robotic activities
Describe the environment
Define the type of robotic system
Define the types of robots

Figure 4.3: Tasks of Activity RSA-1.1

RSA-A 1.3 – Identify constraints: Based on the decisions made in Activity RSA-A 1.1, constraints related to the robotic application are identified. These constraints are associated with both hardware requirements and real-time operation. Infrastructure requirements, including battery consumption, processing power, network availability, and robot autonomy are considered in the identification of hardware constraints. Usage scenarios are identified and described in the definition of real-time constraints. Afterwards, constraints associated with these scenarios are detected and prioritized. As robotic systems are often used in safe-critical domains, real-time constraints are very important and they must guide the rationale behind service identification and composition.

RSA-A 1.4 – Identify standards: Robotic systems may need to be certificated to ensure compliance with policies imposed on its operation. Standards are applied to both robotic system and its development process for the obtainment of certification. Therefore, at this point, all standards related to the SORS are identified. Different standards can be applied to a robotic system depending on its own characteristics and the environment where it will be used. For instance, standards as DO-178C (RTCA, 2011) should be considered in the development of UAVs. For road vehicles, standards such as ISO 26262:2011 (ISO, 2011) are also considered very important. The adoption of standards directly impacts on the software architecture to be designed.

RSA-A 1.5 – Define functional requirements: Based on the outcomes of the previous activities, a set of functional requirements is elicited and a draft of a document is created. These requirements represent functionalities the SORS should provide to perform the robotic application.

RSA-A 1.6 – Define quality requirements: In this activity, quality requirements of SORS are identified based on: (i) application goals, (ii) policies and rules, (iii) constraints, and (iv) standards. Afterwards, robotic experts and functional architects organize brainstorm meetings to prioritize quality requirements of the robotic system. A quality requirement can be considered very important, important, less important, and unimportant. Requirements considered very important are critical to the success of the system and must be addressed. However, considering a quality requirement unimportant does not mean it should be avoided, but designers do not need to make extra effort on it. In order support this activity, we performed a secondary study (Oliveira et al., 2013a) to identifying the most addressed quality requirements in the embedded system domain. Results of this study indicate that requirements such as reliability, security, safety, and efficiency should be considered very important. Figure 4.4 illustrates the three tasks of Activity RSA-1.6.

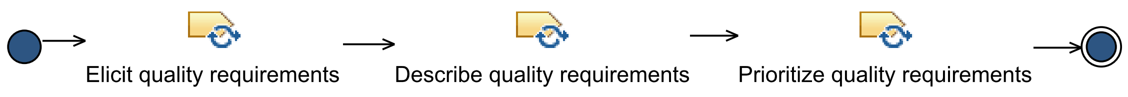

Elicit quality requirements
Describe quality requirements
Prioritize quality requirements

Figure 4.4: Tasks of Activity RSA-1.6

RSA-A 1.7 – Document SORS requirements: A document is created based on the functional and quality requirements elicited in the previous two activities. The deliverable containing the SORS requirements will guide the description of the robotic application flow and support the identification of robotic capabilities. Therefore, it should be reviewed by all stakeholders to ensure correctness, completeness, and accordance with the robotic application goals.

4.2.2 Phase RSA-2: Robotic Capabilities Identification

Robotics specialists and functional architects use SORS requirements to describe the application in terms of functionalities, flow between these functionalities, and capabilities responsible for providing them. For this, the application flow is modelled and then decomposed into different robotic capabilities. A capability is a service candidate that may either be available or need to be designed and developed. The taxonomy presented in Section 3.2 can be used to identify capabilities during this phase. Other capability identification methods, such as those reported by Huergo et al. (2014), may also complement the guidelines proposed in this phase. Figure 4.5 shows a method for guiding conduction of Phase RSA-2. Activities in this method are described as follows. The milestone is represented by the completion of a document reporting the identified SORS capabilities.

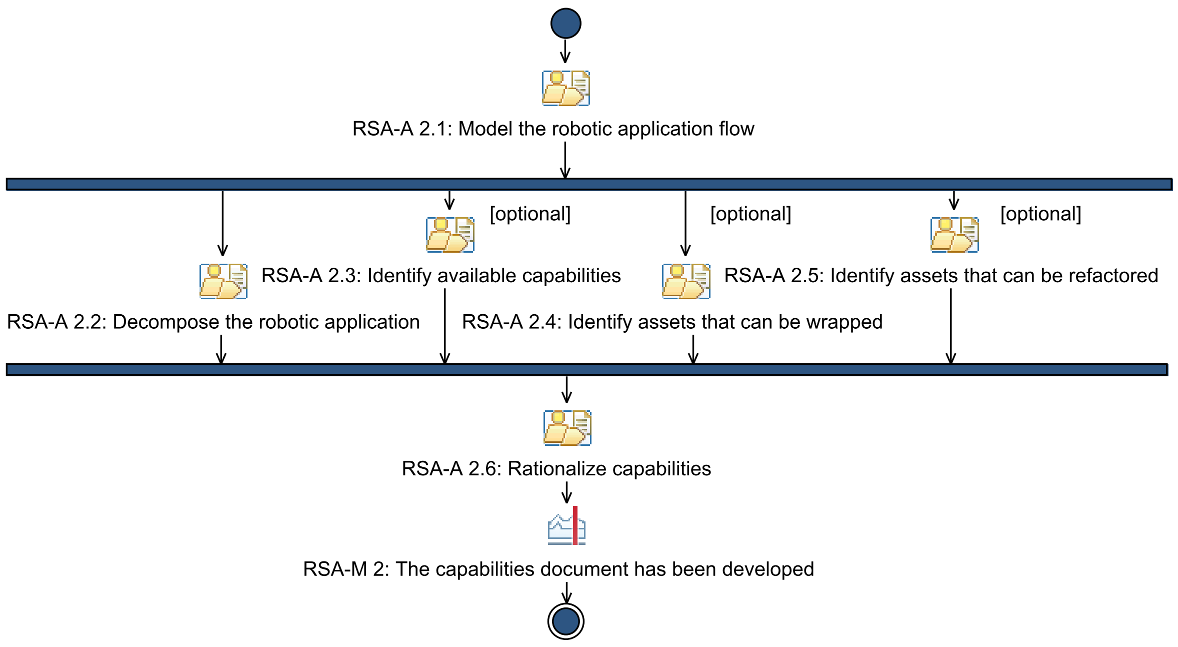

RSA-A 2.1: Model the robotic application flow
[optional]
[optional]
[optional]
RSA-A 2.3: Identify available capabilities
RSA-A 2.5: Identify assets that can be refactored
RSA-A 2.2: Decompose the robotic application
RSA-A 2.4: Identify assets that can be wrapped
RSA-A 2.6: Rationalize capabilities
RSA-M 2: The capabilities document has been developed

Figure 4.5: Method for the robotic capability identification

RSA-A 2.1 – Model the robotic application flow: The activities of the robotic application are identified and represented by description languages, such as UML activity diagrams, Business Process Model and Notation (BPMN) (OMG, 2015a), and π-ADL. The robotic application flow is described in terms of: (i) functionalities performed in parallel; (ii) functionalities performed in sequence, i.e., that depend on the result of the execution of previous functionalities; and (iii) functionalities based on the combination of results from other functionalities. Thereafter, the model is reviewed for checking whether it has fulfilled all functional and quality requirements. Figure 4.6 illustrates the sequence of tasks performed during RSA-A 2.1.

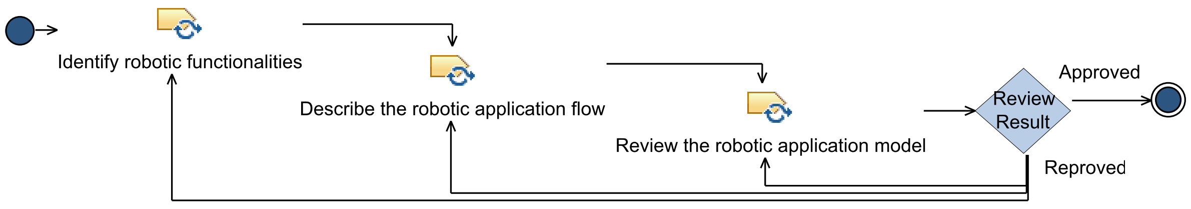

Identify robotic functionalities
Describe the robotic application flow
Review the robotic application model
Review Result
Approved
Reproved

Figure 4.6: Tasks of Activity RSA-2.1

RSA-A 2.2 – Decompose the robotic application: Based on the defined model, the robotic application is decomposed into capabilities, which encompass a set of functionalities the robotic system will provide. According to the taxonomy of services for SORS (see Section 3.2), capabilities identified in this activity are mainly related to: (i) device drivers, (ii) robotics tasks, (iii) knowledge, and (iv) robotic agents. The latter type is often identified in multi-robotic applications and swarms, where different robots must interact with each other.

RSA-A 2.3 – Identify available capabilities: Robotics experts investigate available capabilities that can be reused. These capabilities are identified from different sources: (i) robotic systems developed in previous projects; (ii) development environments, as ROS and MRSD, which provide a set of native services for SORS; (iii) repositories of services for SORS, as RoboSeT (see Section 3.3) and ROS Wiki; and (iv) general purpose repositories (e.g., service brokers) from which functionalities considered non-critical can be remotely accessed. Notice capabilities identified with the support of the service taxonomy can be directly searched by using RoboSeT. Figure 4.7 shows the tasks performed during activity RSA-A 2.3.

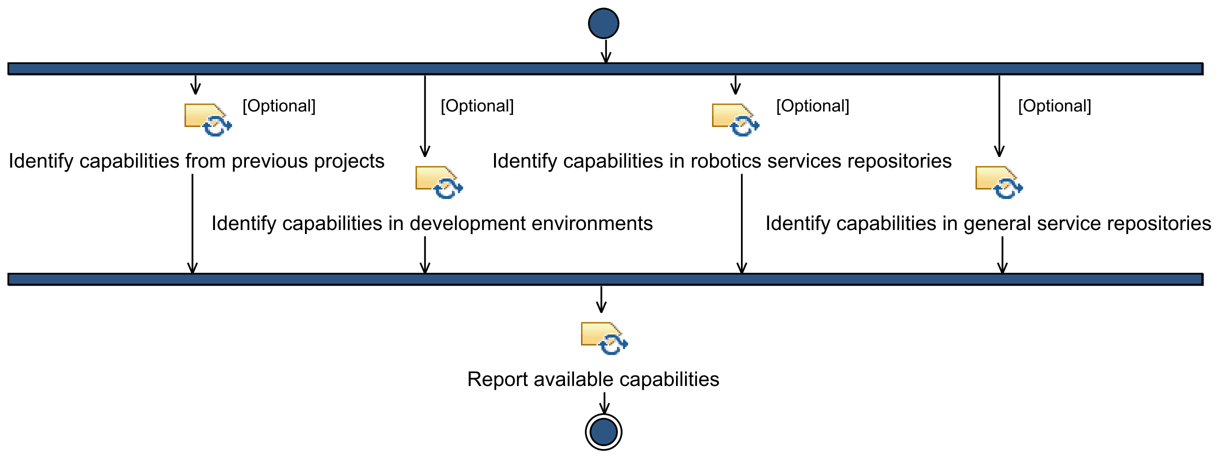

[Optional]
Identify capabilities from previous projects
[Optional]
Identify capabilities in development environments
[Optional]
Identify capabilities in robotics services repositories
[Optional]
Identify capabilities in general service repositories
Report available capabilities

Figure 4.7: Tasks of Activity RSA-2.3

RSA-A 2.4 – Identify assets that can be wrapped: Additionally to the available capabilities, previous projects of non-service-oriented robotic systems are investigated for identifying assets to be provided as capabilities. These assets are packages, software modules, legacy applications, and algorithms (e.g., for localization and mapping) that can be wrapped and then provided as services for the robotic system.

RSA-A 2.5 – Identify assets that can be refactored: Assets useful for the robotic system but that can not be provided directly as robotic capabilities should also be identified. These assets must be refactored in order to be reused as capabilities of the robotic system. For instance, a mapping algorithm tangled into a robotic control system developed in a previous project can be refactored to be used as an independent, self-contained service.

RSA-A 2.6 – Rationalize capabilities: Capabilities necessary for developing the robotic system are compared to the ones available for reuse. Discussions are made to decide which capabilities will be exposed as services (service components) and which capabilities will be provided as components (technical components) that support these services. The decision is made taking into account the quality requirements defined in the previous phase, as it can directly affect the achievement of quality attributes, such as performance, safety, and dependability. As a result, a document is created to report: (i) capabilities related to the robotic application; (ii) functionalities provided by each capability; (iii) architectural elements used to provide each capability; and (iv) design rationale behind the decisions. Figure 4.8 shows the tasks performed during activity RSA-A 2.6.

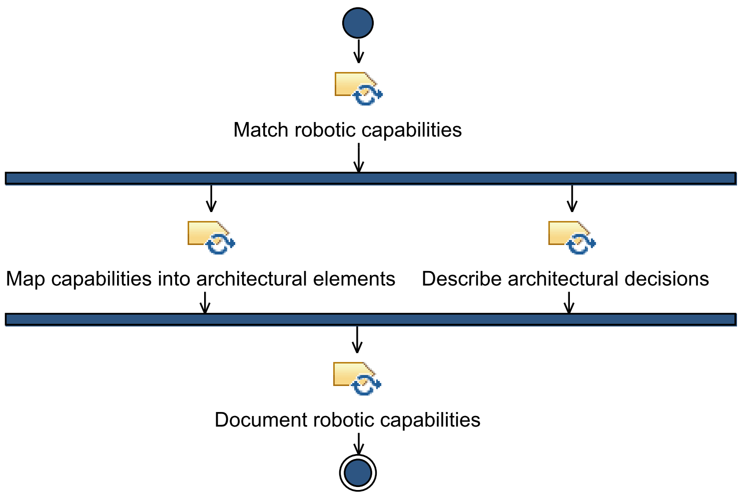

Match robotic capabilities
Map capabilities into architectural elements
Describe architectural decisions
Document robotic capabilities

Figure 4.8: Tasks of Activity RSA-2.6

4.2.3 Phase RSA-3: Robotic Architecture Modeling

Services and technical components that will expose the robotic capabilities are described, modelled, and composed. Functional architects create and document different models to represent interfaces, contracts, participants, as well as relationships among services and between services and their technical components. These models are designed based on constraints and quality attributes that each service should comply with. Figure 4.9 illustrates a method to conduct Phase RSA-3. Activities of this method are described as follows. The SORS functional architecture is the resulting deliverable document and its development represents a project milestone.

RSA-A 3.1 – Specify robotics services: The document that contains information on the robotic capabilities is updated and roles played by each service are described in de-

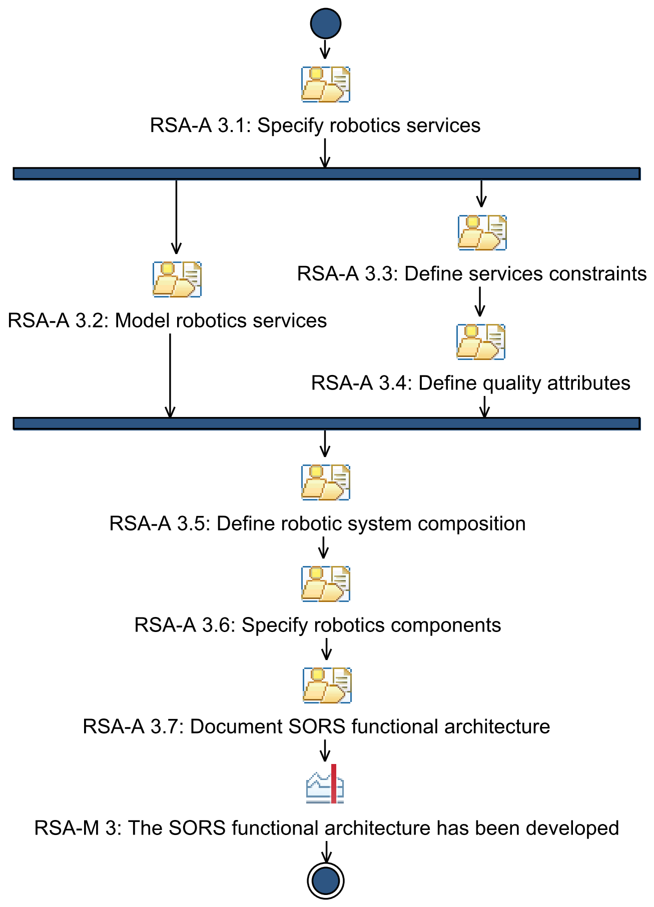

RSA-A 3.1: Specify robotics services
RSA-A 3.3: Define services constraints
RSA-A 3.2: Model robotics services
RSA-A 3.4: Define quality attributes
RSA-A 3.5: Define robotic system composition
RSA-A 3.6: Specify robotics components
RSA-A 3.7: Document SORS functional architecture
RSA-M 3: The SORS functional architecture has been developed

Figure 4.9: Method for the robotic architecture modelling

tails. The resulting document links requirements of the robotic system to the requirements provided by each service.

RSA-A 3.2 – Model robotics services: Functional architects design the services of the robotic system. As mentioned before, different types of ADLs can be used to describe interfaces, contracts, and functionalities in the software architecture. SoaML and π-ADL are, respectively, examples of semi-formal and formal languages to this purpose. Figure 4.10 shows the tasks performed during activity RSA-A 3.2. In SORS, service interfaces are associated with contracts that usually enforce three types of interaction: (i) synchronous Remote Procedure Call (RPC), used to request simpler functionalities, as obtaining of the instant position of the robot; (ii) asynchronous RPC, which enables the request of more complex functionalities, as navigation between positions in the environment; and (iii) service subscription, which is a long-term interaction commonly established to obtain data from sensors. In a service subscription, the service client implements a handler method to receive notifications from its service provider. These notifications can be either periodic (e.g., measures of a laser sensor) or event-driven (e.g., a bumper being pressed).

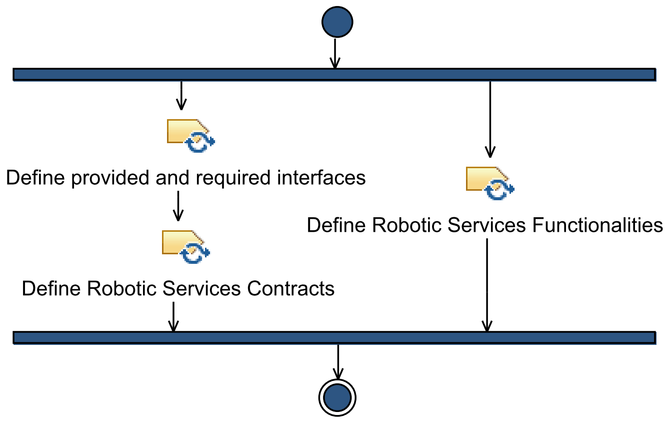

Define provided and required interfaces
Define Robotic Services Functionalities
Define Robotic Services Contracts

Figure 4.10: Tasks of Activity RSA-3.2

RSA-A 3.3 – Define service constraints: To ensure compliance with the overall robotic system constraints, each service must guarantee its individual set of constraints. The clear description of constraints at architectural level is crucial for the determination of the participant (i.e., concrete service) that will provide a given service. Therefore, the document of capabilities is updated with information about the constraints of each robotics service in the software architecture.

RSA-A 3.4 – Describe quality attributes: The quality attributes of all services in the software architecture are defined according to the quality requirements of the robotic system and services constraints identified in the previous activity. This information is included in the document of capabilities to describe how functionalities of robotics services should be provided.

RSA-A 3.5 – Define services composition: The composition of robotics services is defined according to obligations of consumers and providers established in the service contracts. Moreover, important details concerning interactions among service partners are described, such as services to be deployed in the same infrastructure (e.g., in the same server). This information will be used to support decisions made during the design of the technical architecture described in the next phase. Models such as SoaML Service Architecture diagram can be used to represent relationships among services in the software architecture.

RSA-A 3.6 – Specify robotics components: Robotics services are often abstractions of functionalities provided by the coordination of one or more technical components, i.e., robotic capabilities not directly exposed as services. Therefore, relationships between services and their technical components are described and modelled by using different types of representations, e.g., UML component diagrams.

RSA-A 3.7 – Document SORS functional architecture: The document of capabilities is again updated with all the developed models, the design rationale applied in the modelling, and all useful information regarding the functional and quality aspects of the robotic system. The resulting document reports the SORS functional architecture to be used as input for the following phase.

4.2.4 Phase RSA-4: Robotic Architecture Detailing

Software architects and robotics experts design the SORS technical architecture by detailing its functional architecture in terms of software modules, technologies, and hardware devices to be used to develop services of the system. Figure 4.11 shows the method that guides conduction of this phase. Descriptions of activities in the method are presented as follows. The completion of the SORS technical architecture document is the milestone of Phase RSA-4.

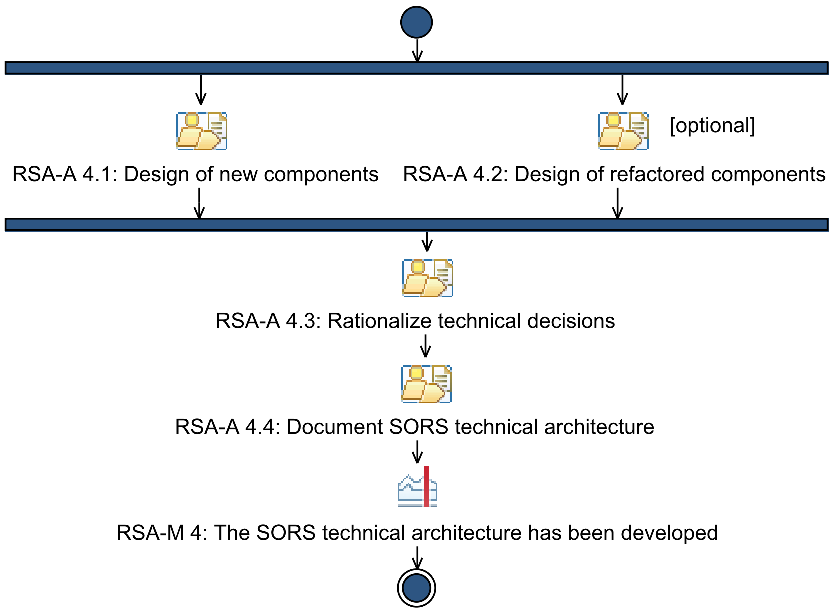

[optional]
RSA-A 4.1: Design of new components
RSA-A 4.2: Design of refactored components
RSA-A 4.3: Rationalize technical decisions
RSA-A 4.4: Document SORS technical architecture
RSA-M 4: The SORS technical architecture has been developed

Figure 4.11: Method for the robotic architecture detailing

RSA-A 4.1 – Design of new components: Services unavailable for reuse that must be developed are further described and represented. Different models, including state and sequence diagrams, can be created by architects to illustrate both design and runtime aspects of the services. The internal structure of the services can be represented by ordinary object-oriented (OO) modelling and different design patterns.

RSA-A 4.2 – Design of refactored components: Services that provide capabilities from existing robotics assets are designed with the support of refactoring techniques available in the literature (Mens and Tourwe, 2004). Therefore, software architects analyse the documentation and the source code of these assets to create new diagrams representing the components to be refactored for the robotic system.

RSA-A 4.3 – Rationalize technical decisions: Technical architects and robotics experts decide on hardware infrastructures and implementation strategies (e.g., types of algorithms) to be used during the robotics services realization. Furthermore, decisions are made on how the services of the robotic system will be deployed, i.e., which services must be locally provided (inside the robot) and which services can be remotely used (outside the robot). These decisions are supported by information on concerns and quality requirements (e.g., reliability, performance, and security) described in the functional architecture document. As a result, a document reporting the rationale behind service concretization is created. Figure 4.12 illustrates the tasks performed during Activity RSA-A 4.3.

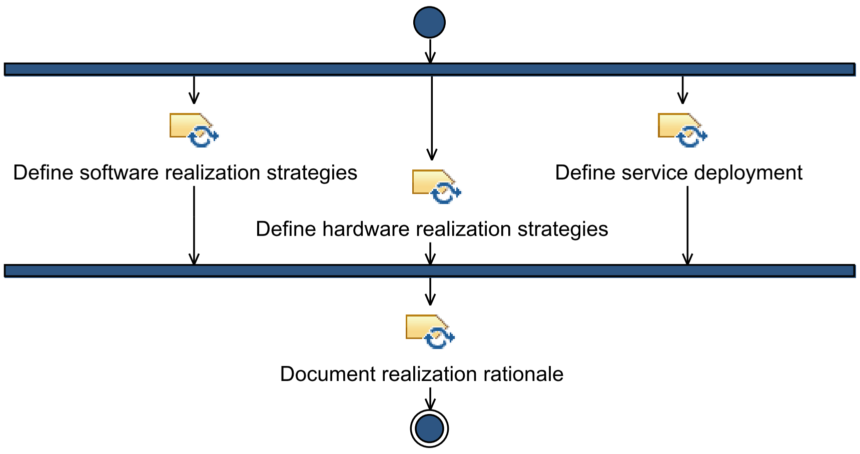

Define software realization strategies
Define service deployment
Define hardware realization strategies
Document realization rationale

Figure 4.12: Tasks of Activity RSA-4.3

RSA-A 4.4 – Detail SORS concrete architecture: Finally, the overall structure of the functional architecture is described in a document containing all information related to its design. Textual descriptions of the diagrams and design decisions are documented. Additional views of the architecture, such as deployment view, can be also created.

4.2.5 Phase RSA-5: Robotic Architecture Evaluation

The SORS software architecture is evaluated for ensuring its compliance with requirements and constraints of the system. Moreover, the architectural description itself is evaluated to identify and eliminate defects related to omission, ambiguity, inconsistency, as well as strange and incorrect information. Different evaluation methods for assessing software architectures already exist in the literature, e.g., inspection check lists and scenario-based methods. Section 2.2.4 discussed several studies that can be used to this purpose. Since evaluation methods are not mutually exclusive, they can be applied in a complementary manner. As a result, a more reliable software architecture can be achieved. Figure 4.13 shows activities to be performed in the evaluation of SORS software architectures.

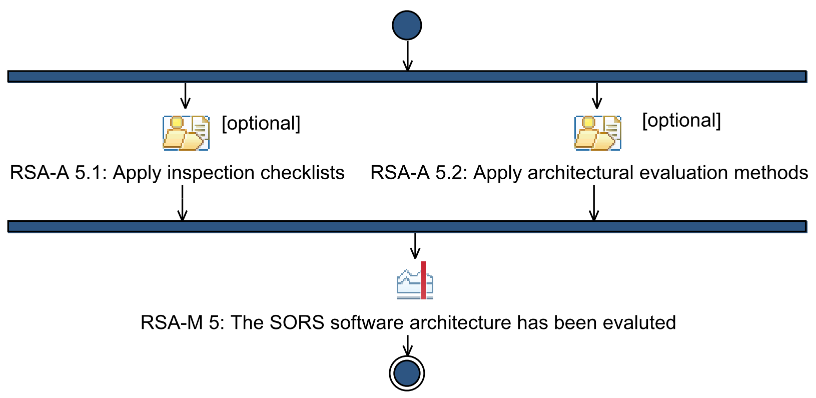

[optional]
RSA-A 5.1: Apply inspection checklists
[optional]
RSA-A 5.2: Apply architectural evaluation methods
RSA-M 5: The SORS software architecture has been evaluted

Figure 4.13: Method for robotic architecture evaluation

4.3 Experimental Evaluation of ArchSORS

In order to evaluate ArchSORS, we performed a controlled experiment with students enrolled in a preparatory course for the French national robotics competition[4]. This experimental study aimed at checking whether ArchSORS could design software architectures with higher quality than those currently developed in an *ad hoc* manner, i.e., without considering quality attributes such as reusability, modifiability, and maintainability. We carried out our study using the systematic process proposed by Wohlin et al. (2012), which divides an experiment into five phases: (i) scoping, (ii) planning, (iii) operation, (iv) analysis and interpretation, (v) and presentation.

4.3.1 Scope of the Experiment

The objective of our experimental study was outlined by the Goal, Question, Metric template (GQM) (Basili et al., 1999). According to this template, the experiment had the *objective* of analysing ArchSORS *for the purpose of* evaluation *with respect to* modularity, coupling, and cohesion of the SORS software architectures *from the point of view of* the professor and researchers *in the context of* a preparatory course for a national robotics competition. Based on this goal, we established four research questions (RQ):

- **RQ 1:** Can ArchSORS produce SORS software architectures of higher modularity than that of architectures developed in an *ad hoc* manner?

- **RQ 2:** Can ArchSORS produce less coupled SORS software architectures than those currently developed in an *ad hoc* manner?

[4]`www.robafis.fr`, last accessed in February 12th, 2015.

- **RQ 3:** Can ArchSORS produce SORS software architectures less dependent on the control service than those currently developed in an *ad hoc* manner?

- **RQ 4:** Can ArchSORS produce more cohesive SORS software architectures than those currently developed in an *ad hoc* manner?

In this experiment, we have considered modularity, coupling, and cohesion as indicators of quality since they can affect not only the aforementioned quality attributes, but also other attributes such as buildability and complexity (Galster et al., 2008). We also considered that SORS software architectures that are strongly dependent (i.e., coupled) of coarse-grained control services tend to be less maintainable and reusable.

4.3.2 Planning the Experiment

The experiment was performed in the context of a Software Engineering for Robotics course at a French national Engineering School. During the course, 30 students of the second and third years (equivalent to fourth and fifth years at a University) of Computer Engineering and Automation Engineering were trained to participate in a robotics competition. Students (hereafter referred to as subjects) were chosen by convenience, as they had prior knowledge on both robotics and SOA and represented a sample from possible robotics developers. We randomly split the subjects into two groups: (i) one to design the software architecture of an SORS using ArchSORS and (ii) another to design it in an *ad hoc* manner (i.e., the Control group). Four hypotheses were defined in our experiment, one for each research question. These hypotheses are formally described as follows:

1. **Modularity improvement:**

   Null hypothesis, H_0: There is no difference in the modularity (Mod) of SORS software architectures designed using ArchSORS and architectures designed in an *ad hoc* manner, i.e., H_0: $Mod(ArchSORS) = Mod(Control)$.

   Alternative hypothesis, H_1: $Mod(ArchSORS) > Mod(Control)$.

2. **Coupling reduction:**

   Null hypothesis, H_0: The coupling ($Coup$) of SORS software architectures designed using ArchSORS and architectures designed in an *ad hoc* manner are the same, i.e., H_0: $Coup(ArchSORS) = Coup(Control)$.

   Alternative hypothesis, H_1: $Coup(ArchSORS) < Coup(Control)$.

3. **Control service dependency reduction:**

   Null hypothesis, H_0: Services of SORS are usually strongly connected to the main service in the architecture, called control. Software architectures designed using ArchSORS show the same dependency from the control service (Dep_{max}) of architectures designed in an *ad hoc* manner. Formally,

   H_0: $Dep_{max}(ArchSORS) = Dep_{max}(Control)$.

   Alternative hypothesis, H_1: $Dep_{max}(ArchSORS) < Dep_{max}(Control)$.

4. **Cohesion improvement:**

   Null hypothesis, H_0: There is no difference in the cohesion (Coh) of SORS software architectures designed using ArchSORS and architectures designed in an *ad hoc* manner. Formally,

   H_0: $Coh(ArchSORS) = Coh(Control)$.

   Alternative hypothesis, H_1: $Coh(ArchSORS) > Coh(Control)$.

We adopted four metrics (i.e., dependent variables) to test our hypotheses: (i) Modularity Factor (MF); (ii) Coupling Factor (CpF); Max Connections per Component (Max CpC); and (iv) Cohesion Factor (CF). Metrics MF, CpF, and CF were proposed by Galster et al. (2008), which provide the most validated set of metrics for the prediction of quality in software architectures we have found in the literature. Metric Max CpC was proposed by ourselves, aiming at investigating the tendency of designers to concentrate most functionalities of SORS inside the control service, which can cause reuse and maintainability problems. These four metrics range from zero to one. For metrics MF and CF, the higher the result, the better they are (i.e., more modular and more cohesive). For metrics CpF and Max CpC, the lower the result, the better they are (i.e., less coupled and less dependent on a single component). A brief and informal description of these metrics is provided as follows. Further details and usage examples can be found in (Galster et al., 2008).

- **Modularity Factor (MF):** describes the degree at which the system is composed of independent services, such that a change in one service exerts a minimal impact on the others. This metric is computed (in its non-normalized form) by dividing the number of requirements by the number of services in the architecture;

- **Coupling Factor (CpF):** measures the strength of association established by a connection between two architecture artifacts, i.e., the degree of interdependence between services. Coupling is calculated by summing the number of connections of all pair-wise sets of services in the architecture and dividing it by the difference between the square of the number of requirements and the number of requirements;

- **Max Connections per Component (Max CpC):** expresses the degree of dependency between the most coupled service in the architecture and the other services, i.e., it indicates how changes in the main component can impact the whole system; and

- **Cohesion Factor (CF):** estimated how requirements and tasks are related within a service. This is expressed by the ratio between the maximal possible (theoretical) number of dependencies among requirements within a service and the actual number of dependencies.

We have used the structural view of the software architecture, described by a UML component diagram, to calculate these metrics. In this diagram, services were represented by components and service usages by dependency connectors. We also implemented a script to automate the metric computation and reduce possible human mistakes.

4.3.3 Experiment Operation

The experiment was divided into two parts and conducted in a single day. In the first part, subjects received training and performed a pilot study similar to the experiment. The group using ArchSORS was trained on how to design SORS according to the process and SoaML. The control group received a review on how to create models using UML. In the pilot study, both groups were asked to design an architecture of a robotic system proposed in a previous edition of the RobAFIS competition. Doubts and misunderstandings on the experiment were then clarified. Since subjects were free to decide on their participation in the experiment, four of them did not attend the second part. As a consequence, the amount of subjects in the group that used ArchSORS was reduced from 15 to 11. However, we decided to leave the groups unbalanced as subjects in the control group did not receive training on ArchSORS.

In the second part, we performed the controlled experiment. The two groups of subjects received the specification of an extended, more complex version of a RobAFIS project (see Appendix B). As in the pilot study, they also received the document of requirements associated with the project. As expected outcome, subjects had to provide: (i) three diagrams and (ii) a table containing the description and the list of requirements related to each service. Subjects in ArchSORS group were asked to design a BPMN model, an SoaML Capability diagram, and a UML Component diagram representing the Service Architecture. In the control group, subjects were free to choose two diagrams they considered most pertinent, plus a UML component diagram. No limit of time was imposed during the experiment conduction. After the experiment execution, all data were validated to eliminate possible mistakes.

4.3.4 Analysis and Interpretation of Results

A summary of the results obtained by the subjects in the four metrics (MF, CpF, Max CpC, and CF) is shown in Table 4.1. Subjects of ArchSORS group are indicated by the letter ‘P’ and the ones in the control group by the letter ‘C’. For analysing these data, we applied descriptive statistics. Table 4.2 shows the mean, median, minimum value (Min), maximum value (Max), and standard deviation (SD) of the results obtained by the subjects in both groups. Figure 4.14 shows the box plots for the four evaluated metrics.

Table 4.1: Summary of the experiment results

ArchSORS process				
Subject	MF	CpF	Max CpC	CF
P01	0.500	0.145	0.328	0.596
P02	0.333	0.198	0.389	0.648
P03	0.333	0.148	0.296	0.526
P04	0.310	0.154	0.333	0.516
P05	0.238	0.200	0.556	0.541
P06	0.405	0.143	0.256	0.770
P07	0.333	0.143	0.538	0.765
P08	0.429	0.137	0.333	0.485
P09	0.238	0.200	0.556	0.400
P10	0.310	0.192	0.467	0.547
P11	0.286	0.220	0.345	0.600
Control (*ad hoc*)				
Subject	MF	CpF	Max CpC	CF
C01	0.143	0.400	0.667	0.155
C02	0.286	0.167	0.818	0.282
C03	0.214	0.222	0.875	0.335
C04	0.214	0.222	0.750	0.256
C05	0.190	0.250	1.000	0.232
C06	0.214	0.222	1.000	0.259
C07	0.238	0.222	0.900	0.352
C08	0.190	0.464	0.615	0.234
C09	0.238	0.200	0.889	0.294
C10	0.214	0.222	0.500	0.231
C11	0.262	0.182	1.000	0.342
C12	0.214	0.222	0.875	0.330
C13	0.214	0.222	0.875	0.289
C14	0.214	0.278	0.900	0.318
C15	0.238	0.222	0.700	0.413

Table 4.2: Experimental results of the metrics in the two treatments

Metric	Treatment	Mean ($\overline{x}$)	Median ($\tilde{x}$)	Max	Min	SD (σ)
MF	ArchSORS	0.337	0.333	0.500	0.238	0.079
	Control	0.218	0.214	0.286	0.143	0.032
CpF	ArchSORS	0.170	0.154	0.220	0.137	0.031
	Control	0.248	0.222	0.464	0.167	0.080
Max CpC	ArchSORS	0.400	0.345	0.566	0.256	0.110
	Control	0.824	0.880	1.000	0.500	0.149
CF	ArchSORS	0.581	0.547	0.770	0.400	0.112
	Control	0.288	0.290	0.413	0.155	0.062

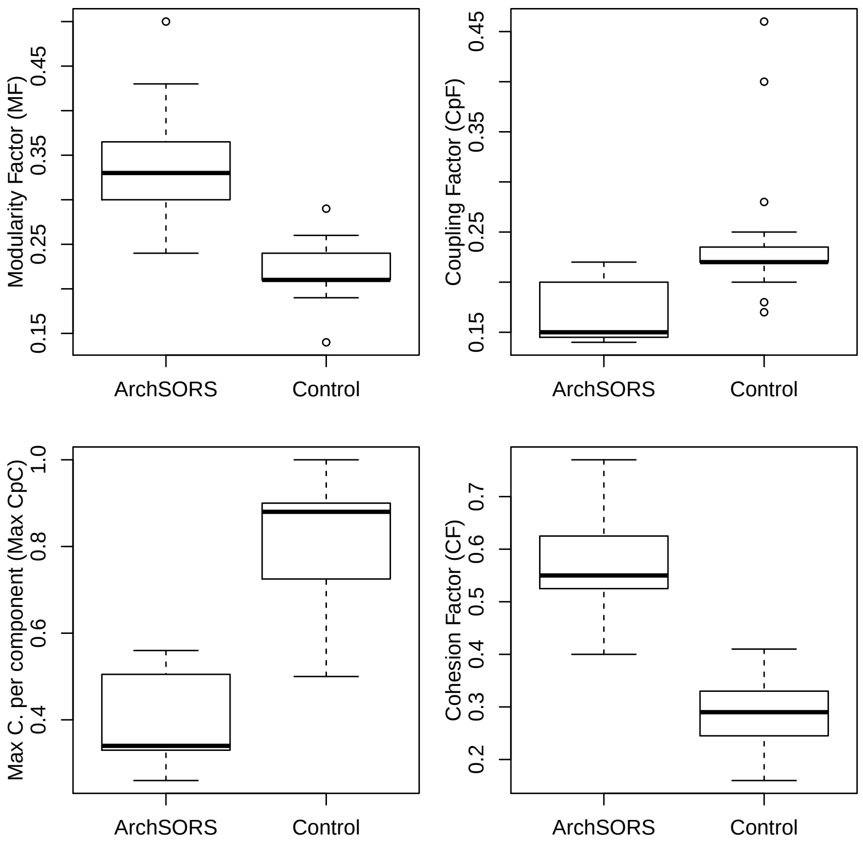


Figure 4.14: Box plots of the results for MF, CpF, Max CpC, and CF

Subjects of ArchSORS group developed their SORS software architecture with higher modularity and cohesion. The mean values of CpF and Max CpC also indicate these architectures were less coupled and dependent from the control service, which suggests a positive influence of ArchSORS on the development of SORS software architectures. We applied a statistical test to ensure that the differences between the means of the two treatments were significant and not caused by chance. For identifying the proper test, we first analysed the data distribution. Histograms shown in Figure 4.15 indicate that data were not normally distributed, which is confirmed by the normal probability plots illustrated in Figure 4.16. In a normal probability plot (chi-square test), deviations from the straight line indicate lack of data normality (Wohlin et al., 2012).

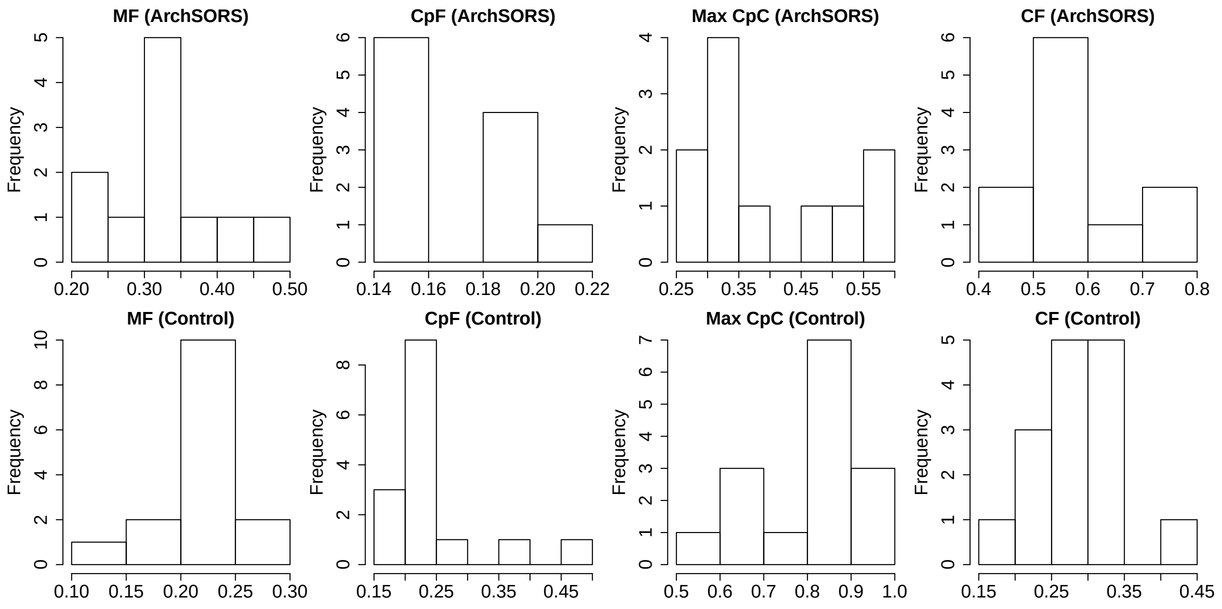


Figure 4.15: Histogram of the results

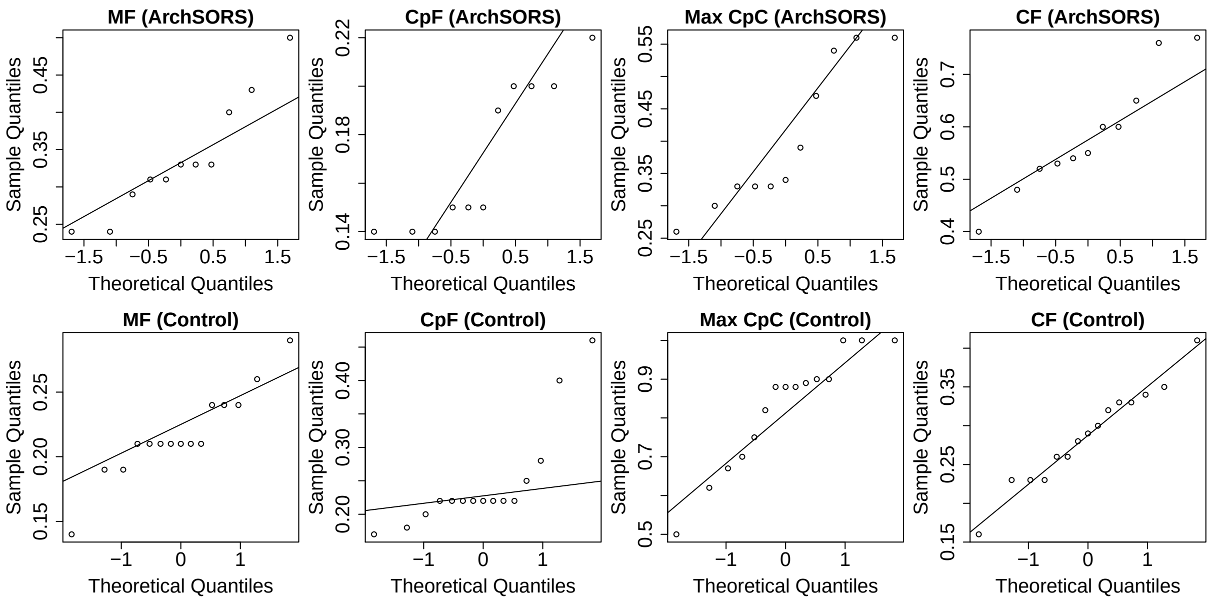


Figure 4.16: Normal probability plot of the results

As the data were skewed and the two sample sizes were unequal, the most appropriate statistical test is Mann-Whitney (Wohlin et al., 2012). Results obtained from the application of this test show a statistically significant difference, with *P-Value* < 0.001

(one-tailed error), between the medians of metrics MF, CpF, Max CpC, and CF for the development using ArchSORS and the development in *ad hoc* (Control). Therefore, the null hypotheses that the differences between the medians arose due to sampling effects can be rejected in favour of the alternative hypotheses that the adoption of ArchSORS has positive influence on the quality of SORS software architectures.

4.4 Discussion and Threats to Validity

Results of the experiment indicate that ArchSORS can support the design of SORS software architectures of higher quality. Galster et al. (2008) advocate the metrics we adopted in our study impact on different quality attributes, such as modifiability, reusability, complexity, and buildability. Modifiability expresses cost of change and probability of change propagation. Software architectures that are more modular and cohesive can be more easily modified (Galster et al., 2008; Pressman, 2001). Similarly, an architecture that presents good modularization and whose services are designed with a well-defined objective (highly cohesive) is more likely to be reusable. A low degree of coupling goes along with independent artifacts and also leads to a higher chance of later reuse. Complexity describes the degree at which an architecture can be handled intellectually. Higher coupling usually involves more complex interfaces, which leads to higher complexity (Galster et al., 2008). Buildability enables systems to be completed in a timely manner and its improvement is related to the ability of maximizing parallelism that can occur in development. Additionally, high cohesion facilitates construction of individual artifacts and a system that is more modularized can be assigned to more development teams. These design-time attributes, along with the four metrics, can also impact on runtime quality attributes like availability, reliability, and performance. However, determining the impact of ArchSORS in runtime quality attributes demands long-term studies, such as evaluations through case studies.

We were aware of possible threats to the validity of our experiment and tried to mitigate them. For avoiding problems of internal validity, we decided to perform the experiment in a single day, keeping the two groups separated during the conduction. This measure aimed at avoiding communication among subjects. Moreover, to motivate participation of subjects and, at the same time, avoid competition, the professor proposed grades for the commitment of the students, rather than for their results. As we decided to perform the experiment in a single day, some activities of ArchSORS had to be simplified or omitted, which could result in a threat to *construction validity*. Nevertheless, even a simplified version of ArchSORS was able to positively impact on the quality of the SORS architectures. For reducing possible issues associated with the *conclusion validity*, we carefully chose the statistical test and number of subjects. The main threat of our study is related to the *external validity*, because of the choice of students as subjects instead of professionals. However, obtaining the commitment of a comprehensive amount of professionals for our experiment was a very difficult and cost expensive choice. Subjects that participated in our study had competencies on both robotics and SOA and represented the population usually associated with the development of SORS, i.e., the researchers in academia. Therefore, we believe our results are still valid.

4.5 Final Remarks

This chapter presented ArchSORS, a systematic process for designing SORS software architectures. The establishment of this process was based on consolidated guidelines to design software architectures, SOA-based systems, and robotic systems. Each phase of ArchSORS was detailed by a method represented in SPEM notation. Finally, the impact of ArchSORS on the quality of SORS software architectures was evaluated in a controlled experiment.

The contributions of this chapter are threefold: (i) a process that positively impacts on coupling, cohesion, and modularity of SORS software architectures and, consequently, improves the overall quality of their resulting systems; (ii) a SPEM *method plugin* that enables the instantiation of this process for particular projects, as well as the monitoring of its activities directly into project management tools; and (iii) an experimental evaluation of ArchSORS, including the use of metrics to an early assessment of software architectures. The next chapter describes a reference architecture established to facilitate the application of ArchSORS in the design of indoor grounded mobile SORS.

Chapter

5

A Reference Architecture for Indoor Grounded Mobile SORS

5.1 Overview

The development of SORS involves integration and coordination of functionalities designed by multiple teams in different contexts. A common understanding on how to structure SORS software architectures is, therefore, fundamental to the success of these systems. Reference architectures are recognized as important assets for improving communication, promoting standardization, and reducing design complexity of software architectures. Although several reference architectures for robotics exists, none of them support the design of robotic systems based on SOA. Therefore, the proposal of a reference architecture for SORS would facilitate the development of such systems and contribute to the consolidation of SOA in robotics.

This chapter reports on the establishment of RefSORS (Reference Architecture for Service-Oriented Robotic Systems), a reference architecture for indoor grounded mobile SORS. This reference architecture is aligned with the ArchSORS process and aims at facilitating the application of its phases. RefSORS is focused on indoor grounded applications because most of robotic systems based on SOA are intended for this particular type of environment (see Section 2.4.5, Research Question 5). We adopted the ProSA-RA process described in Section 2.2.2 to guide the establishment of RefSORS. The conduction of this process was supported by the RAModel reference model, as it provides a broad list of elements that a reference architecture should encompass.

The organization of this chapter follows the steps of ProSA-RA. Section 5.2 details the sources of information used to establish RefSORS. Section 5.3 reports on the requirements of the reference architecture elicited from these information sources. Section 5.4 describes architectural views of RefSORS. Finally, Section 5.5 presents a case study on the development of a SORS using RefSORS in conjunction with ArchSORS.

5.2 Step RA-1: Information Source Investigation

The first step to establish our reference architecture was to identify information sources to be used in the elicitation of requirements. We considered four sets of sources regarding robotics and SOA: Set 1 – Service-oriented robotic systems identified in the literature; Set 2 – Guidelines to the development of service-oriented systems; Set 3 – Reference architectures for robotic systems; and Set 4 – Control architectures of the robotics domain. Following the guidelines proposed by ProSA-RA, we also used results of systematic reviews to identify information sources in Sets 1, 2, and 3 (Feitosa and Nakagawa, 2012; Oliveira et al., 2010, 2013b). The sets of information sources considered in this step are detailed as follows:

Set 1 – Service-oriented robotic systems: The main concern of our reference architecture is to understand and represent the knowledge on how to structure SORS software architectures. We investigated 39 studies on the development of SORS that were identified in the first iteration of the systematic review detailed in Section 2.4.5. As result, we elicited the types of services related to the creation of such systems and described them into a taxonomy (see Section 3.2). The types of services were then organized into five groups, considered the as concepts of robotics in our reference architecture: (i) Device driver, (ii) Task, (iii) Knowledge management, (iv) Robotic agent, and (v) Robotic application. Additionally, we analyzed the SORS software architectures available in the studies to comprehend how the types of services within the groups are related to each other;

Set 2 – Guidelines to the development of service-oriented systems: Reference models and reference architectures for SOA encompass the knowledge on how to design service-oriented systems. Therefore, these assets can be basis for the establishment of other domain-specific reference architectures. We used a systematic review on reference models and reference architectures for SOA, conducted in a previous work (Oliveira et al., 2010), as a starting point for the identification of concepts associated with service-orientation. We also revisited the literature to obtain guidelines published more recently, such as the SOA-RA technical standard (The Open Group, 2015). Particular attention has been paid to the S3 and OASIS reference architectures, as they are considered the most consolidated ones (see Section 2.3.3). As a result, we identified the main concepts related to SOA for our reference architecture: (i) Service description, (ii) Service publication, (iii) Service interaction, (iv) Service composition, and (v) Quality of service;

Set 3 – Reference architectures for the robotics domain: Reference architectures support the reuse of design expertise in the development of a set of software systems of a similar domain. Therefore, we investigated seven reference architectures for mobile robotic systems identified by the systematic review reported in (Feitosa and Nakagawa, 2012). The following functionalities of mobile robotic systems were identified: sensor data processing, control, collision detection, mapping, localization, planning, user interaction, communication, decision judgement, and robots interaction. Notice all of these functionalities were encompassed by the concepts already identified in Set 1, which evidences that such concepts are comprehensive enough for representing the main characteristics of robotics domain. Although any additional concept of robotics was identified in Set 3, functionalities reported by reference architectures in the systematic review were important for the elicitation of requirements described in the next section; and

Set 4 – Control architectures of the robotics domain: Another important source of information considered in the establishment of our reference architecture is the set of control architectures available in the robotics literature. Although RefSORS is focused on how robotic systems are composed using services rather than on how robots are controlled, these architectures describe the different manners that modules of robotic systems can be structured and communicate. Particularly, we investigated the overall structure of deliberative, reactive, and hybrid control architectures discussed in Section 2.4.2, as well as examples of their implementations reported in (Dudek and Jenkin, 2010; Romero et al., 2014).

5.3 Step RA-2: Architectural Analysis

The architectural requirements of RefSORS were elicited based on the sources of information identified in the previous step. An architectural requirement is a requirement of a reference architecture that describes the common functionalities of a class of systems of a given domain (Nakagawa et al., 2014). During the design of concrete architectures from reference architectures, each architectural requirement will be instantiated as one or more system requirements, depending on the context of development and the stakeholders' concerns. We identified 43 architectural requirements for our reference architecture and classified them into two groups: requirements of the robotics domain and requirements related to SOA.

5.3.1 Architectural Requirements of Robotics Domain

The Architectural Requirements of the Robotics domain (AR-R) were obtained by analysing the information sources described by Sets 1, 3, and 4. They represent functionalities that a reference architecture must encompass to enable the design of indoor grounded mobile robotic systems. The requirements related to the robotics domain are presented as follows:

AR-R [1]: The reference architecture must enable the development of robotic systems able to perform different types of activities in indoor environments;

AR-R [2]: The reference architecture must enable the development of robotic systems able to perform activities in both structured and dynamic environments;

AR-R [3]: The reference architecture must enable the development of robotic systems able to coordinate a single robot, multiple robots, and swarms;

AR-R [4]: The reference architecture must enable the development of robotic systems able to coordinate both homogeneous and heterogeneous teams of robots;

AR-R [5]: The reference architecture must enable the development of robotic systems able to perform activities in different ways, by adapting their behaviour at runtime;

AR-R [6]: The reference architecture must enable the development of robotic systems that can be deployed in different robotic platforms;

AR-R [7]: The reference architecture must enable the development of robotic systems based on different control architectures, such as reactive, deliberative, and hybrid;

AR-R [8]: The reference architecture must enable the development of robotic systems with different levels of autonomy, such as teleoperated, semi-autonomous, and autonomous;

AR-R [9]: The reference architecture must enable the development of robotic systems with different levels of criticality;

AR-R [10]: The reference architecture must enable the development of robotic systems able to communicate with a back-end server;

AR-R [11]: The reference architecture must enable the development of robotic systems that support the communication among robots;

AR-R [12]: The reference architecture must enable the development of robotic systems able to communicate with humans in the environment;

AR-R [13]: The reference architecture must enable the development of robotic systems able to manipulate objects inside the environment using different strategies;

AR-R [14]: The reference architecture must enable the development of robotic systems able to perform localization using different strategies;

AR-R [15]: The reference architecture must enable the development of robotic systems able to navigate in the environment using different strategies;

AR-R [16]: The reference architecture must enable the development of robotic systems able to plan local and global paths using different strategies;

AR-R [17]: The reference architecture must enable the development of robotic systems able to map the environment using different strategies and maps representations, such as topological and metric;

AR-R [18]: The reference architecture must enable the development of robotic systems able to map the environment cooperatively, i.e., by using data acquired from different robots;

AR-R [19]: The reference architecture must enable the development of robotic systems whose tasks, such as image segmentation, can be processed inside robots or remotely by a back-end server;

AR-R [20]: The reference architecture must enable the development of robotic systems able to produce and store information using data acquired from the environment;

AR-R [21]: The reference architecture must enable the development of robotic systems whose robotic agents can share information among them;

AR-R [22]: The reference architecture must enable the development of robotic systems able to store and acquire information in/from a back-end computer;

AR-R [23]: The reference architecture must enable the development of robotic systems able to acquire information from sources outside the environment, such as Web services and machine-readable Wikis;

AR-R [24]: The reference architecture must enable the development of robotic systems that can coordinate, monitor, and acquire data from different types of sensors;

AR-R [25]: The reference architecture must enable the development of robotic systems able to acquire data from sensors spread over the environment;

AR-R [26]: The reference architecture must enable the development of robotic systems able to share sensor data among different robots;

AR-R [27]: The reference architecture must enable the development of robotic systems that can coordinate and monitor different types of resources, such as battery and processor;

AR-R [28]: The reference architecture must enable the development of robotic systems able to share information about resources state (e.g., battery state of charge) among its robots;

AR-R [29]: The reference architecture must enable the development of robotic systems that can coordinate and monitor different types of actuators;

AR-R [30]: The reference architecture must enable the development of robotic systems that can interact with actuators spread over the environment;

AR-R [31]: The reference architecture must allow the development of robotic systems by its partial instantiation, i.e., functional robotic systems can be designed using only part of the services of the reference architecture; and

AR-R [32]: The reference architecture must allow the development of robotic systems that are compliant with applicable laws and standards.

These architectural requirements are directly related to the concepts of robotics described in the previous step (see Section 5.2). The mapping shown in Table 5.1 is important to bridge the architectural requirements and the reference architecture description detailed in Section 5.4, which is based on the concepts of the domain. Requirements related to the reference architecture itself are referred in the table as "General".

5.3.2 Architectural Requirements of SOA

Additionally to requirements from robotics domain, we also identified 11 Architectural Requirements related to SOA (AR-S). These requirements were elicited from the information sources reported in Set 2 and describe characteristics that a reference architecture must encompass to enable the design of software modules of robotic systems as services. The list of this requirements is presented as follows.

Table 5.1: Mapping of architectural requirements into concepts of robotics

Concept of robotics	Architectural requirements
Robotic Application	AR-R [1], AR-R [2], AR-R [3], AR-R [4], AR-R [5]
Robotic Agent	AR-R [6], AR-R [7], AR-R [8], AR-R [9]
Task	AR-R [10], AR-R [11], AR-R [12], AR-R [13], AR-R [14], AR-R [15], AR-R [16], AR-R [17], AR-R [18], AR-R [19]
Knowledge	AR-R [20], AR-R [21], AR-R [22], AR-R [23]
Device Driver	AR-R [24], AR-R [25], AR-R [26], AR-R [27], AR-R [28], AR-R [29], AR-R [30]
General	AR-R [31], AR-R [32]

AR-S [1]: The reference architecture must enable the development of services for SORS that provide normative descriptions related to their correct use;

AR-S [2]: The reference architecture must enable the development of services for SORS that provide semantic descriptions for their classification in service repositories;

AR-S [3]: The reference architecture must enable the development of services for SORS that can be published and discovered by service consumers;

AR-S [4]: The reference architecture must enable the development of services for SORS that can be invoked directly or through mediators;

AR-S [5]: The reference architecture must enable the development of services for SORS able to evolve without affecting their interaction with service consumers;

AR-S [6]: The reference architecture must enable the development of SORS composed by the orchestration of services into business processes;

AR-S [7]: The reference architecture must enable the development of SORS that result from the collaboration of services in a choreography;

AR-S [8]: The reference architecture must enable the development of SORS that are scalable and can incrementally evolve through the addition of new services;

AR-S [9]: The reference architecture must enable the development of SORS that can be composed by other systems or used by client applications;

AR-S [10]: The reference architecture must enable the development of SORS that encompass or are able to use mechanisms to capture, monitor, log, and signal non-compliance with quality requirements; and

AR-S [11]: The reference architecture must enable the development of services for SORS that contain information related to their quality characteristics.

Table 5.2 shows the mapping between the architectural requirements related to SOA and their associated concepts identified in the previous step (see Section 5.2).

Table 5.2: Mapping of architectural requirements into concepts of SOA

Concept of SOA	Architectural requirements
Service Description	AR-S [1], AR-S [2], AR-S [11]
Service Publication	AR-S [2], AR-S [3]
Service Interaction	AR-S [4], AR-S [5], AR-S [6], AR-S [7],
Service Composition	AR-S [6], AR-S [7], AR-S [8], AR-S [9]
Quality of Service	AR-S [10], AR-S [11]

All architectural requirements and concepts identified in this step are inputs to build the architecture description of RefSORS.

5.4 Step RA-3: Architectural Synthesis

Architectural descriptions are important resources of reference architectures used to transmit knowledge of the domain to different stakeholders. An adequate representation of this knowledge should be documented in different levels of abstraction and according to multiple architectural views. RefSORS is described by combining informal notation and the semi-formal languages SoaML and UML. It encompasses five architectural views: (i) Conceptual view, which provides a general overview of interactions among the concepts associated with SORS; (ii) Capability view, which represents functionalities related to these concepts in terms of robotic capabilities; (iii) Service Interface and Contracts view, which represents capabilities in terms of software elements; (iv) Service Architecture view, which illustrates how services interact in a SORS; and (v) Service Deployment view, which guides the deployment of services of a SORS. These views are detailed in the following sections.

5.4.1 Conceptual View

High-level informal representations of reference architectures are useful artifacts to describe interactions among concepts of the domain. They are detailed using informal, non-technical language that can be easily understood by all stakeholders. We designed RefSORS based on RSDS proposed in Chapter 3 and the overall structure of S3, which is aligned with the SOMA method used in the establishment of ArchSORS. Figure 5.1 shows the Conceptual view of RefSORS, which is composed by the four layers:

Robotic Agent Service Layer: contains the elements frequently encompassed by robotic systems. It is composed of four groups of services: (i) `Device Driver`

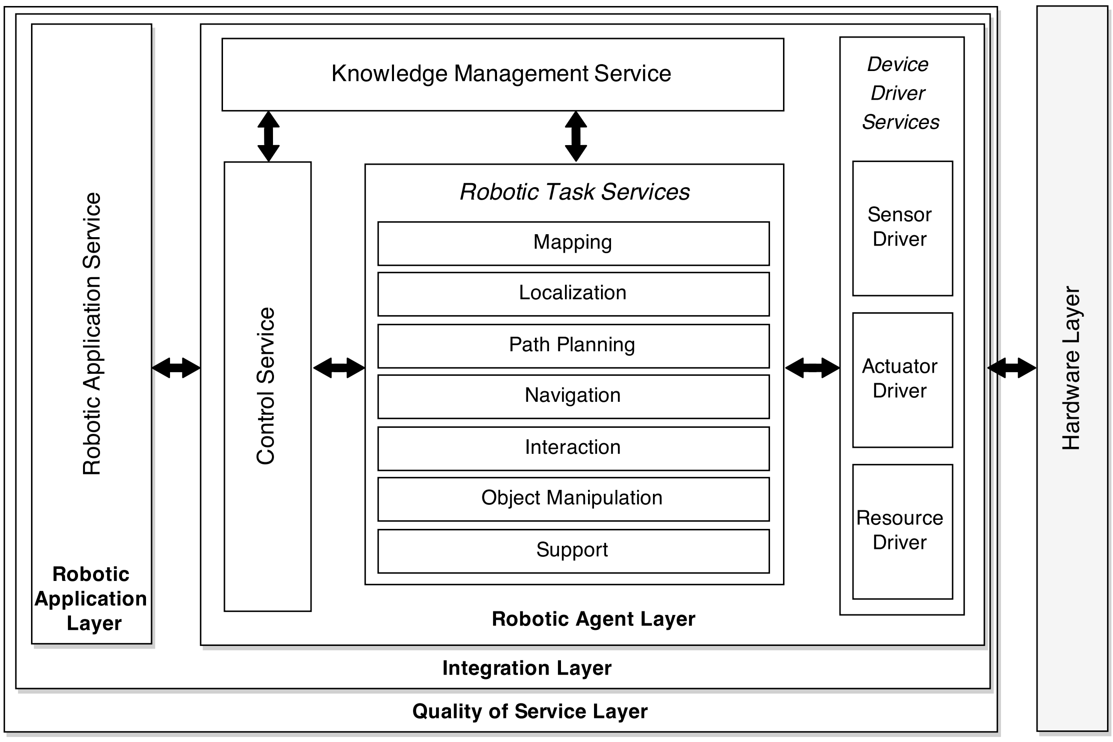


Figure 5.1: Conceptual view of RefSORS

`Services` coordinate and monitor sensors, actuators, and resources of the robot; (ii) `Knowledge Management Services` are responsible for the management and the processing of useful information for the robotic system, such as the characteristics of the environment and the objects inside it; (iii) `Robotic Task Services` encompass the core tasks of robotics, including mapping, localization, and navigation; and (iv) `Control Service` coordinates tasks performed by a robotic agent to execute more complex activities, e.g., the transportation of objects;

Robotic Application Layer: coordinates one of more robotic agents to perform a robotic application (i.e., the robotic mission). In this layer, both orchestration and choreography strategies can be used for the management of the robotic activities necessary to achieve the application goals. The robotic application service is responsible for the identification of the most suitable robotic agent services (i.e., robots) to perform each activity, taking into account their current status, characteristics, and provided capabilities;

Integration Layer: mediates, routes, and transports requests between service consumers and service providers. This layer enables integration and composition of services of all levels within the robotic system. It also connects the SORS and its potential clients. The integration layer provides means for service discovery and supports the indirect communication among participants of an SORS; and

Quality of Service Layer: supervises services contained in the other three layers (`Robotic Agent Layer`, `Robotic Application Layer`, and `Integration Layer`) to check their compliance with SORS quality requirements. In other words, it observes and signals when a quality requirement is not fulfilled by a service of the SORS. This layer ensures that robotic systems encompass adequate levels of reliability, security, availability, and safety. The inclusion of a quality of service layer in RefSORS is considered very important, as services for SORS can be developed by several institutions and present different characteristics. Robotic systems are often created for safety-critical contexts and their overall quality depends on the quality of the services of which they are composed.

The Conceptual view of RefSORS addresses the following architectural requirements: AR-R [1] – AR-R [9], AR-R [11] – AR-R [17], AR-R [20], AR-R [23], AR-R [24], AR-R [26] – AR-R [32], AR-S [3], AR-S [4], AR-S [6], AR-S [7], and AR-S [9] – AR-S [11].

5.4.2 Capability View

The Capability view enables transforming elements of the robotics domain represented in the Conceptual view into software artifacts adequate for the description of services. A capability is a representation of a set of functionalities of the robotic system that can be realized and provided as service. Figure 5.2 shows RefSORS in terms of its capabilities, provided functionalities, and dependency relationships. This view is described by a SoaML Capability diagram, which adopts UML stereotyped classes to represent capabilities and usage links to indicate dependencies between these capabilities.

The `Robotic Application` capability coordinates activities executed by one or more `Robotic Agents` to accomplish the robotic system goals. Each activity is an abstract workflow of short-term robotic tasks performed by a `Robotic Agent`. Tasks composition is coordinated by the `Control` capability, which uses information of the `Knowledge Management` capability to request functionalities from capabilities of `Navigation`, `Localization`, `Mapping`, `Interaction`, `Object Manipulation`, and `Support`. The `Control` also monitors resources of the robot, such as battery consumption, through `Resource Driver` capabilities. Capabilities related to robotic tasks are supported by `Sensor Driver` and `Actuator Driver` capabilities, as well as by other task capabilities. For instance, the `Navigation` capability may use the `Path Planning` capability to obtain information on the navigable path before for driving the robot from a position to another. It may also use the `Mapping` capability to obtain the map of the environment, the `Localization` capability to estimate the position of the robot, the `Sensor Driver` capabilities to receive sensory

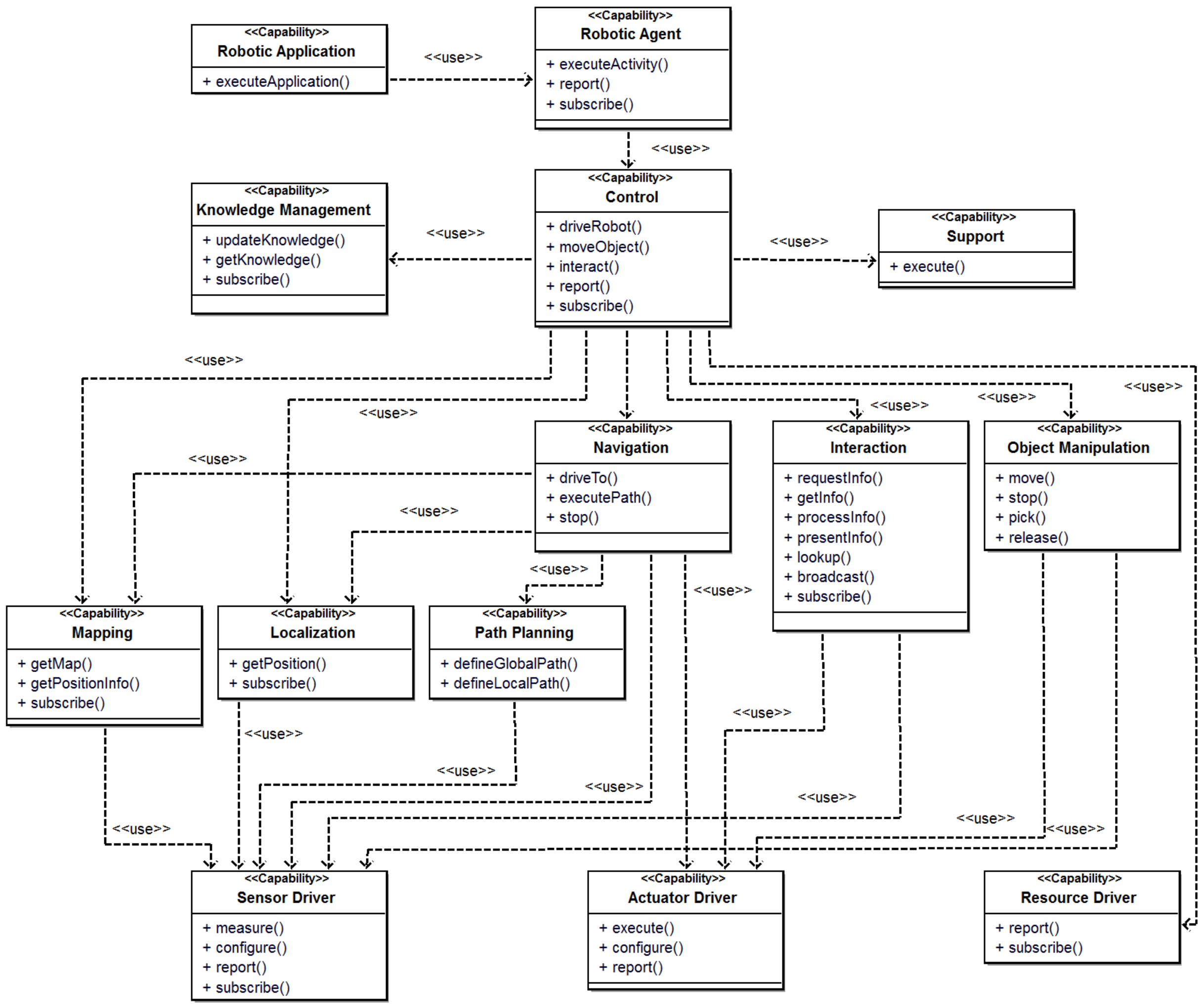


Figure 5.2: Capability view of RefSORS

information, and the `Actuator Driver` capabilities to control the actuators responsible for the locomotion of the robot.

Each capability of the reference architecture can be instantiated as one or more concrete capabilities during the design of SORS software architectures. Besides, capabilities can be omitted depending on the goals of the robotic system or development concerns. A capability in this view represents a service candidate in the development of SORS and, therefore, it can be exposed by a service interface. Service interfaces and contracts express how capabilities are provided as services and how usage dependencies are translated into service interactions. This information is described in the Service Interface and Contracts view.

The following architectural requirements are addressed by the Capability view: AR-R [1] – AR-R [9], AR-R [11] – AR-R [18], AR-R [20], AR-R [21], AR-R [24], AR-R [26] – AR-R [29], and AR-R [31].

5.4.3 Service Interface and Contracts View

This view represents in details how capabilities of SORS are exposed by services and the agreements of these services with potential service consumers. Functionalities of services are provided through standard service interfaces, which are described in SoaML by UML classes with the <<`Service Interface`>> stereotype. Service contracts represent communication agreements between partners of a service collaboration and are detailed by one or more description documents called protocols. The use of a service contract enforces design compatibility between services by demanding both provided and required interfaces. Service protocols describe the order that functionalities should be requested and how responses take place. The service interfaces, contracts, and protocols of each service of RefSORS are detailed as follows. Additional documentation on the services and possible types of instantiation are available in the service taxonomy discussed in Section 3.2.3. Notice that reference architectures are represented at high-level of abstraction. Attributes of all functionalities were omitted as they may differ from a particular instantiation to another.

Sensor driver service: exposes the capability of controlling sensors used to obtain information on the environment. Figure 5.3 shows the `SensorDriverService` interface, which implements a provider interface composed by four functionalities: (i) `measure` informs the instant value obtained by the sensor; (ii) `configure` modifies settings of the sensor such as precision and periodicity; (iii) `report` provides the current status of a sensor and its settings; and (iv) `subscribe` indicates that a service consumer wants to be notified about updates on the sensor state with a given periodicity or in case of an event (e.g., the pressing of a contact sensor). The use of subscription demands to service consumers the implementation of the `SensorDriverHandler` required interface. It also enforces a contract whose protocol defines that a `subscribe` request must be performed in order to enable the consumer to receive updates on a sensor. This request is followed by successive response messages sent to the `handleEvent` functionality implemented by the service consumer.

Actuator driver service: exposes the capability of controlling actuators of the robot. Figure 5.4 shows the `ActuatorDriverService` interface and its provided functionalities: (i) `execute` enacts a behaviour on the actuator device, such as stopping an engine or moving an arm joint; (ii) `configure` changes the settings of the actuator, e.g., its linear and angular velocities; and (iii) `report` informs on the current state of the actuator. The three functionalities provided by the actuator driver service are simple and stateless. Therefore, this service does not demand required interfaces nor protocols between participants. Notice that `execute` is a generic representation of functionality that can be instantiated

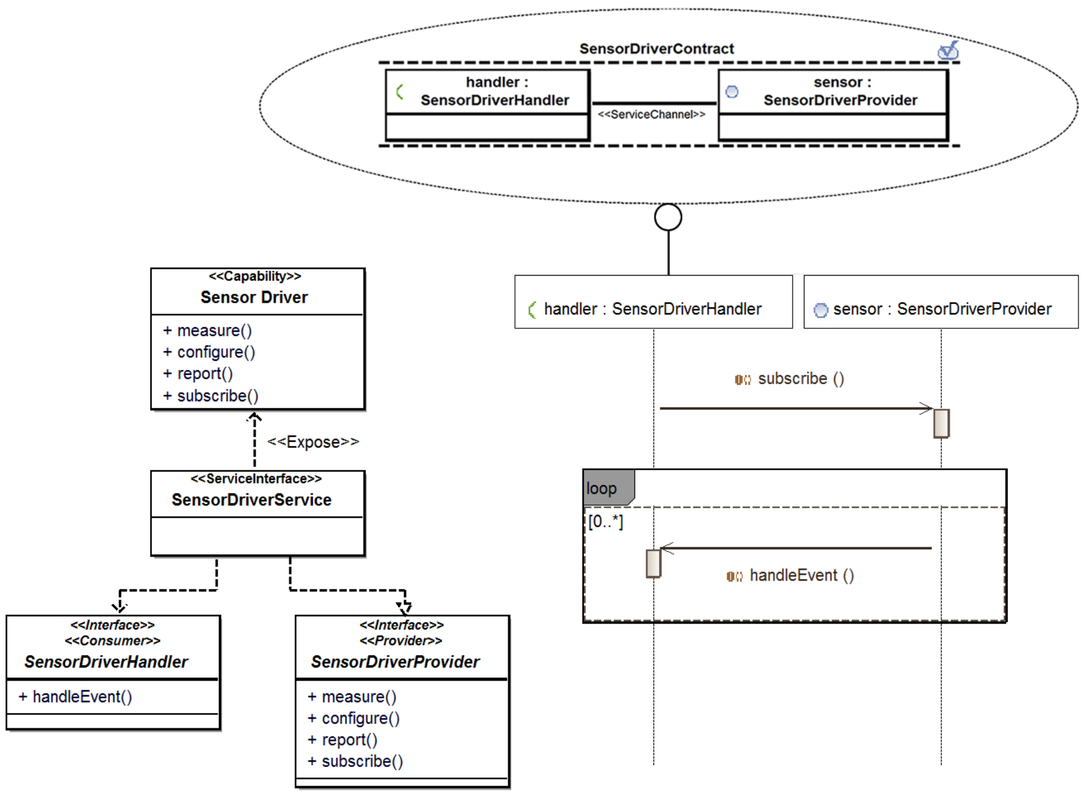


Figure 5.3: Service interface, contract, and protocol of the sensor driver service

by concrete actuator services as one or more functionalities. For instance, while designing a locomotion driver service one may instantiate the `execute` functionality as `start` and `stop` functionalities.

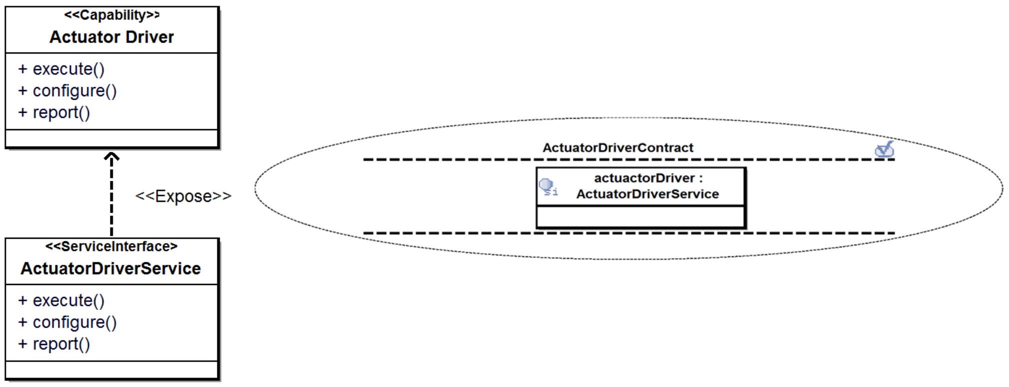


Figure 5.4: Service interface and protocol of the actuator driver service

Resource driver service: exposes the capability of monitoring resources of the robot like battery, memory, and processor. Figure 5.5 depicts the interfaces and protocol associated with this service, as well as the following functionalities: (i) `report` provides a description of the resource's state, e.g., as the percentage of charge of the battery; and (ii) `subscribe` enables service consumers to be periodically updated on the resource's state. The subscription functionality provided by the `ResourceDriverService` demands required interface and protocol similar to those described for the `SensorDriverService`. In this sense, service consumers must send a `subscribe` request to receive updates on the `handleResourceSubscription` functionality.

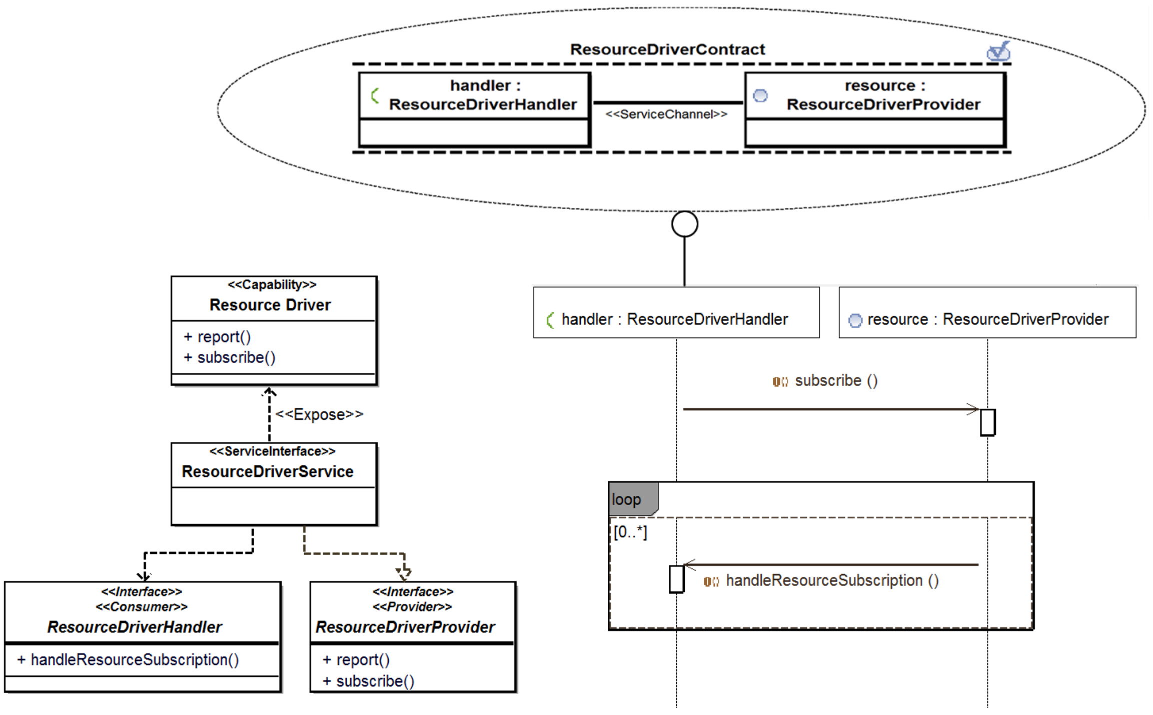


Figure 5.5: Service interface, contract, and protocol of the resource driver service

Localization service: exposes the capability of estimating the position of the robot in the environment. Figure 5.6 shows the service interfaces and protocol related to `LocalizationService`. The functionality `getPosition` provides the current estimated localization of the robot by using a simple request-response protocol. The `subscribe` functionality sends periodic updates on the robot's position and, therefore, demands the use of both provided and required interfaces. The protocol requires that service consumers send `subscribe` requests to be notified on the `handleLocalizationSubscription` functionality.

Mapping Service: exposes the capability of creating representations of the environment and the objects inside it. Figure 5.7 illustrates the interfaces and protocol of `MappingService`. This service offers three functionalities through its service interface: (i) `getMap` provides the current representation of the environment; (ii) `getPositionInfo` informs on the status of a given position in the map, e.g., whether it is navigable of not; and (iii) `subscribe` is used to indicate that a consumer wants to receive periodic updates on the map of the environment. The protocol of `MappingService` contract is similar to other protocols of subscription described in the aforementioned robotics services.

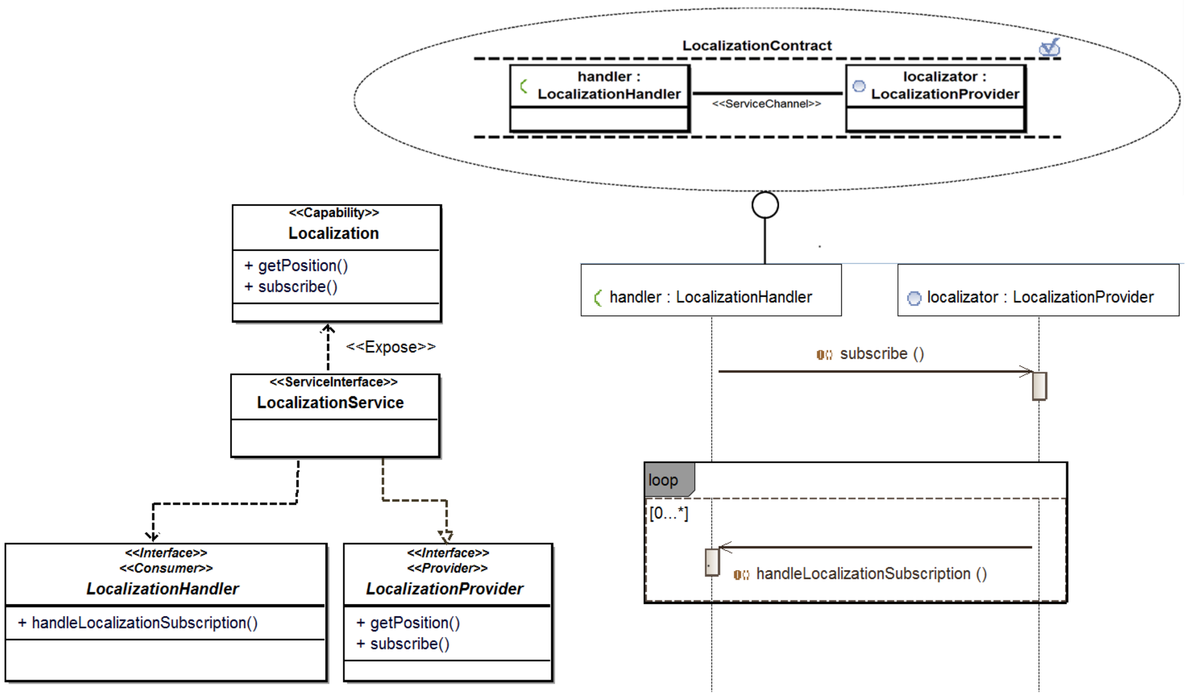


Figure 5.6: Service interface, contract, and protocol of the localization service

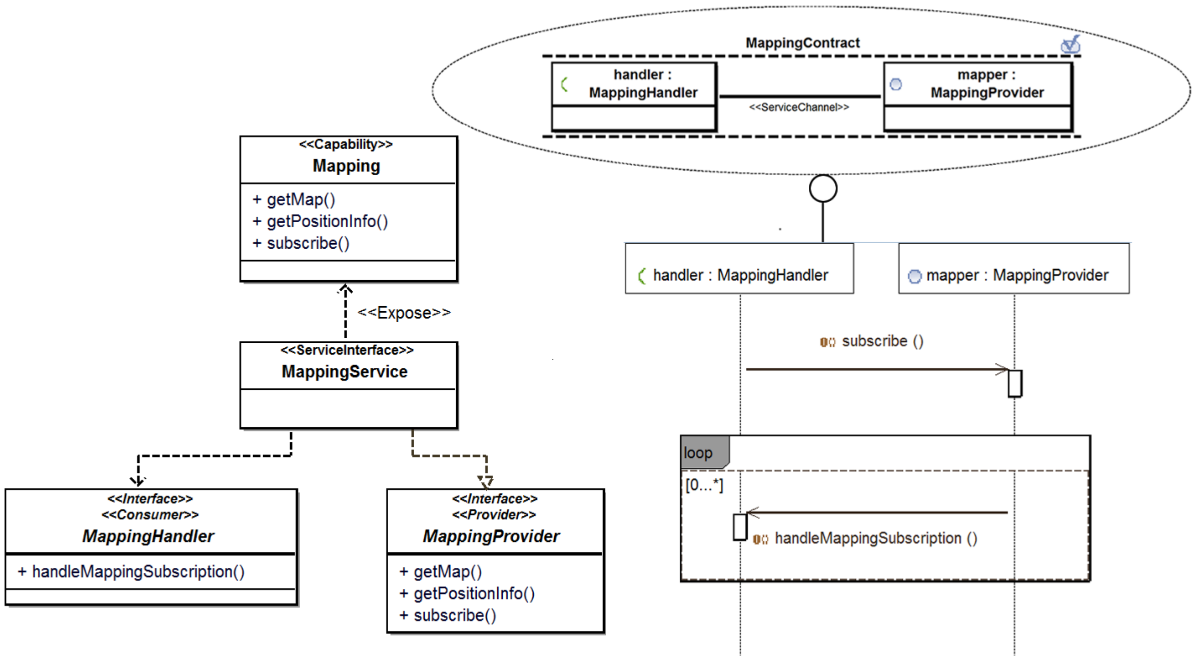


Figure 5.7: Service interface, contract, and protocol of the mapping service

Path planning service: exposes the capability of generating navigable paths for the robotic agent. Figure 5.8 depicts the interfaces involved in the use of this service, as well as its encompassed functionalities: (i) `defineGlobalPath` establishes an initial path from a starting position to a goal position and (ii) `defineLocalPath` adapts the global path to avoid collisions with unpredicted obstacles in dynamic environments. The use of

`PathPlanningService` requires the fulfilment of a contract that includes the two complementary protocols shown in Figure 5.9. The first defines that consumers must provide a functionality named `executePath` to handle the result of a `defineLocalPath` request. This behaviour enables asynchronous execution of such a functionality so that its consumer will not be blocked during the processing. The second protocol enforces that a global path must be defined before a local one, since the latter is defined based on the former.

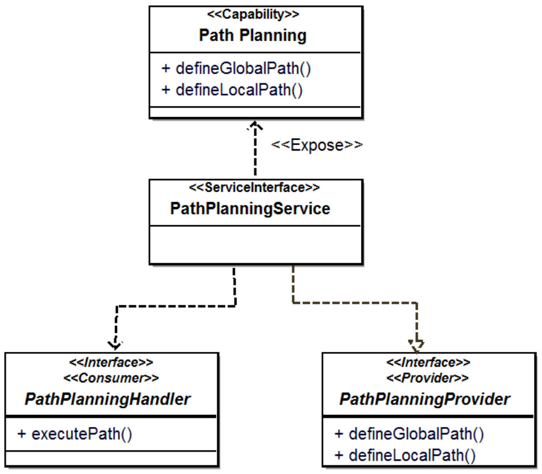


Figure 5.8: Service interface of the Path Planning service

Navigation service: exposes the capability of driving the robot through the environment. Figure 5.10 illustrates the interfaces, contract, and protocol of `NavigationService`. Three main functionalities are provided by this service: (i) `driveTo` requests the navigation from a position to another according to a given velocity; (ii) `executePath` handles path planning responses; and (iii) `stop` sets robot's velocity to zero, which can be used in emergency situations. `NavigationService` demands a `NavigationHandler` required interface from its consumers to ensure the implementation of a `handleNavigationResult` functionality. It enables the service contract to enforce asynchronous communication between service consumer and service provider. The protocol associated with `NavigationContract` prescribes that results of `driveTo` requests must be later notified to `handleNavigationResult` by callback messages.

Object manipulation service: exposes the capability of manipulating objects in the environment. Figure 5.11 shows the manipulation service interface, which provides the following functionalities: (i) `move` translates and/or rotates an object; (ii) `stop` interrupts the procedure of moving an object; (iii) `pick` is used to pick an object; and (iv) `release` frees the object. Notice the instantiation of these functionalities may vary depending on the type of actuator used in the robot. For instance, an object manipulation service that

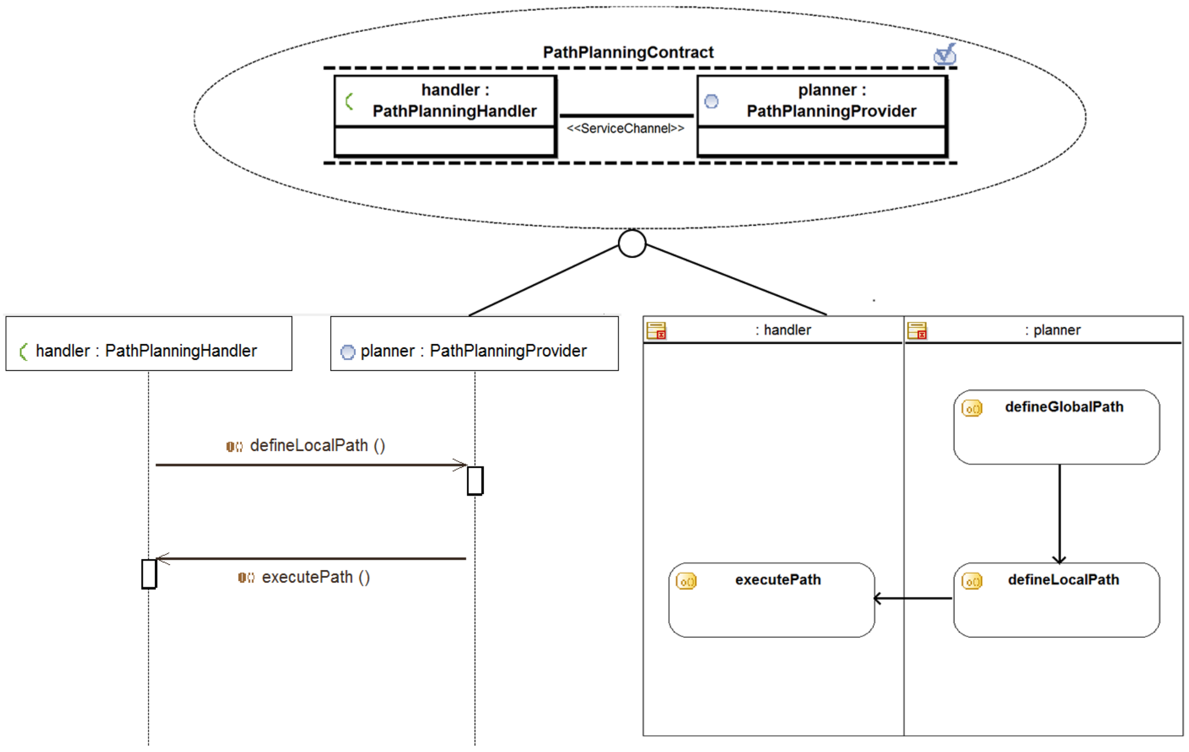


Figure 5.9: Service contract and protocols of Path Planning service

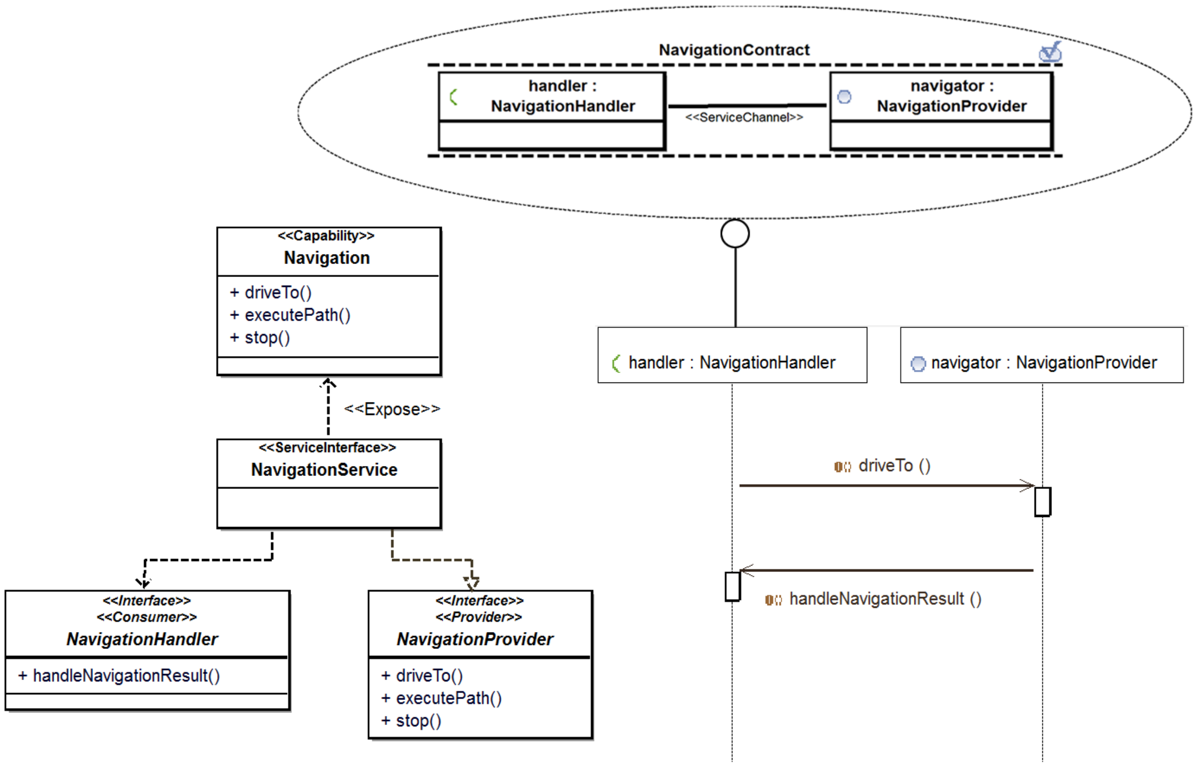


Figure 5.10: Service interface, contract, and protocol of the navigation service

uses a linear forklift actuator would only need to provide `move` and `stop` functionalities.

The `move` functionality is driven by a contract whose protocol enforces asynchronous interaction by callback. Similarly to the `NavigationService` protocol, results of `move` requests are later answered to the handler functionality provided by the service consumer.

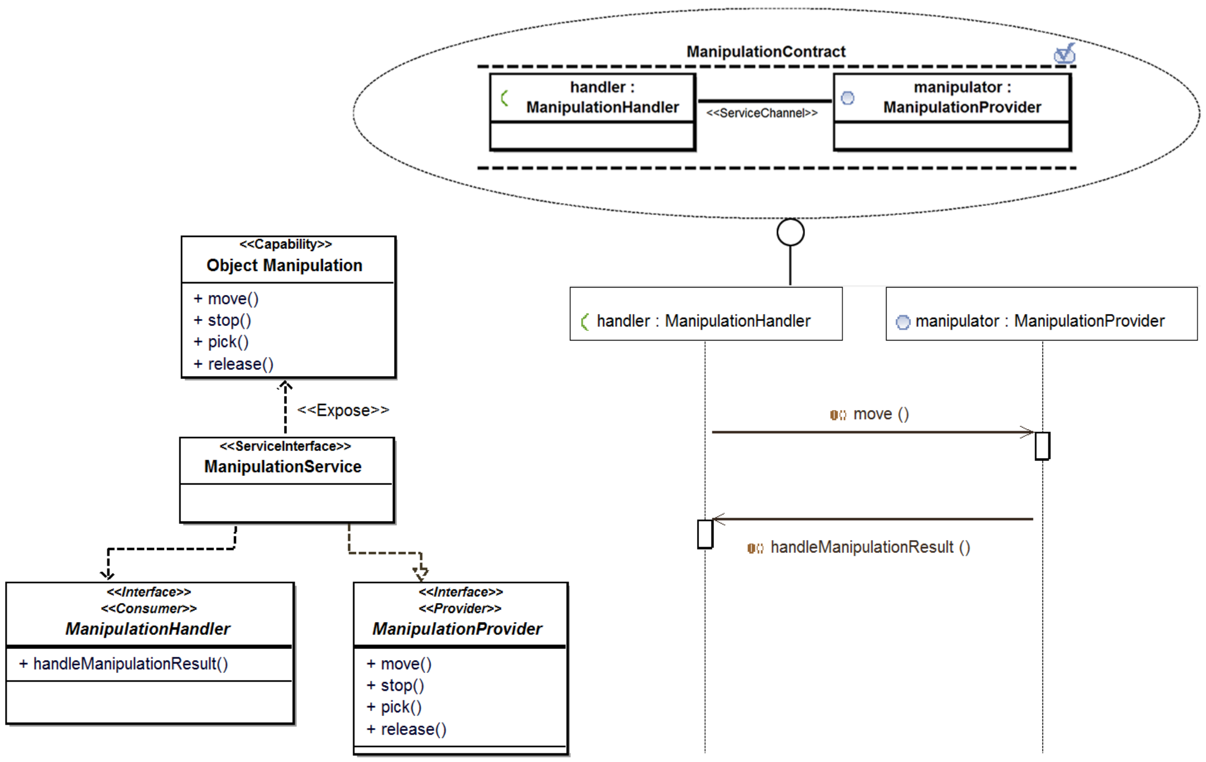


Figure 5.11: Service interface, contract, and protocol of the object manipulation service

Interaction service: exposes the capability of interacting with humans, other robotic agents, and systems in the environment. Figure 5.12 illustrates the interfaces, contract, and protocol of the `InteractionService`. The following functionalities are provided through its interface: (i) `requestInfo` demands information from other participant of the SORS, such as a robotic agent; (ii) `getInfo` obtains information from a source that the robotic agent has a long-term interaction, e.g., a back-end server; (iii) `processInfo` transforms raw data (or signal) into useful information; (iv) `presentInfo` presents information about the robotic agent; (v) `lookup` finds possible collaborations at runtime, e.g., another robotic agent registered in a service broker; (vi) `broadcast` sends information on the robotic agent to all participants of a SORS; and (vii) `subscribe` enables service consumers to be notified on a given type of information. The `subscribe` functionality demands a compatible required interface and the fulfilment of a service subscription protocol.

Knowledge service: exposes the capability of managing knowledge from neural networks, information datasets, ontologies, or any similar source of information that can be useful for the robotic system. The following functionalities are provided through the

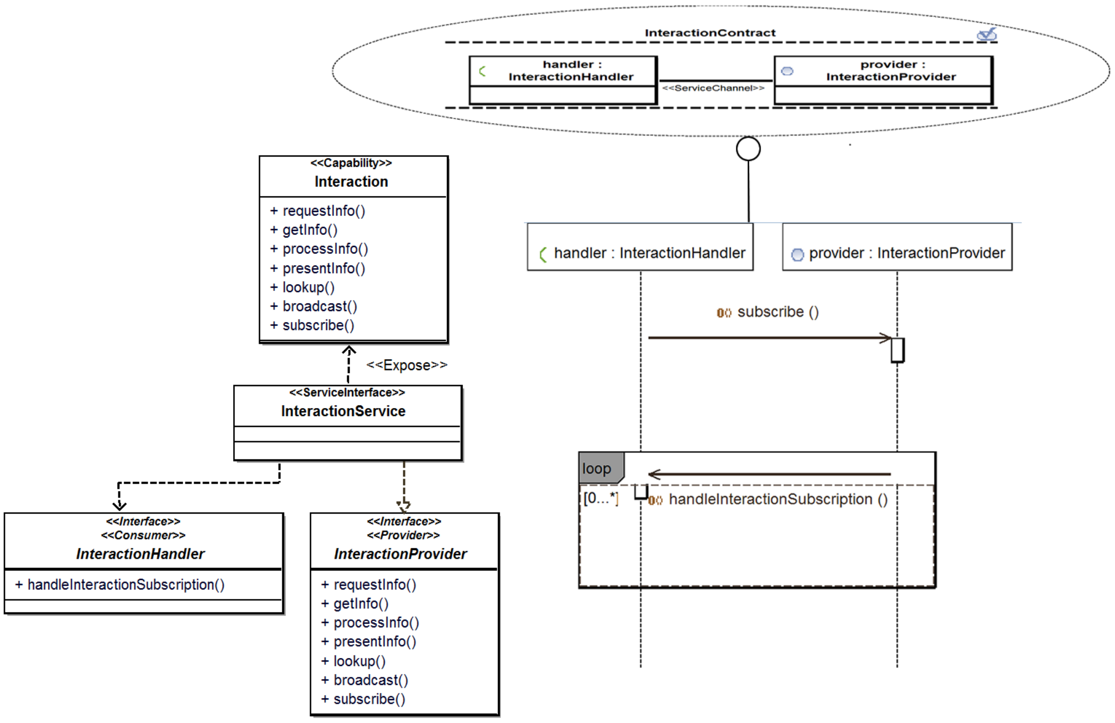


Figure 5.12: Service interface, contract, and protocol of interaction service

`KnowledgeService` interface: (i) `updateKnowledge` updates the current knowledge of the robotic system on a given topic; (ii) `getKnowledge` provides the current knowledge on a topic to service consumers; and (iii) `subscribe` enables consumers to receive updates on a particular topic. Figure 5.13 illustrates the service interface, contract, and subscription protocol of the knowledge service.

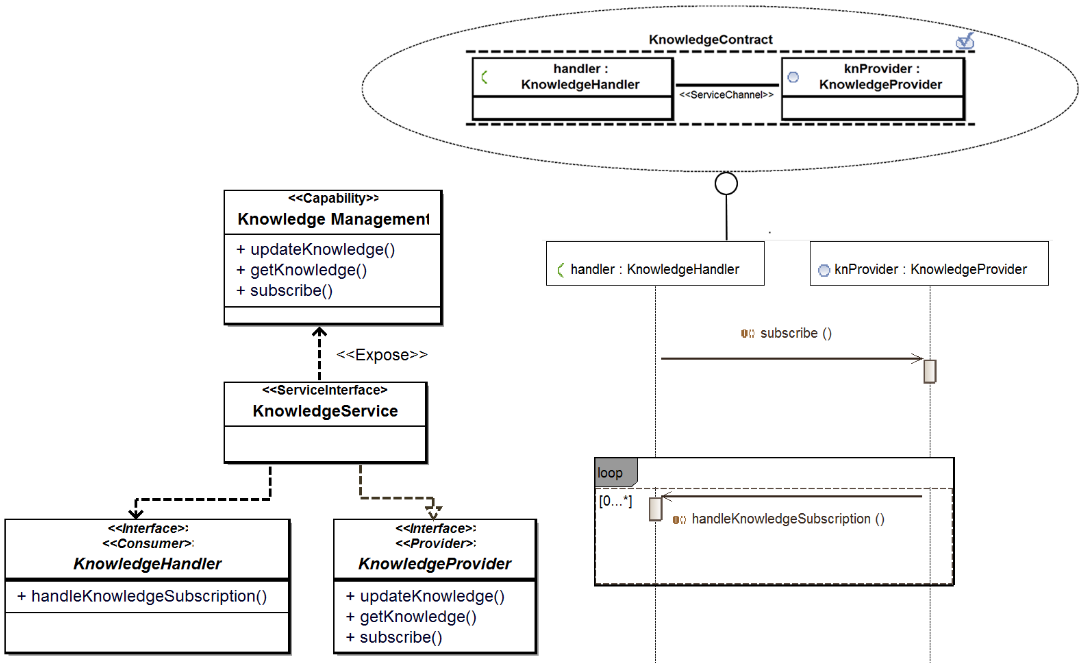


Figure 5.13: Service interface, contract, and protocol of the knowledge service

Support service: exposes the general capability of supporting other services of the robotic system. It represents services for image segmentation, math, and so forth. Figure 5.14 shows the service interface of `SupportService`, which provides the `execute` abstract functionality. This functionality can be instantiated in different manners depending on the type of support service being designed. For instance, while designing an image segmentation support service one may instantiate the `execute` functionality as `segmentImage`. The interaction with support service demands the fulfilment of a contract that enforces asynchronous communication, which ensures that service consumers will not be blocked during their requests.

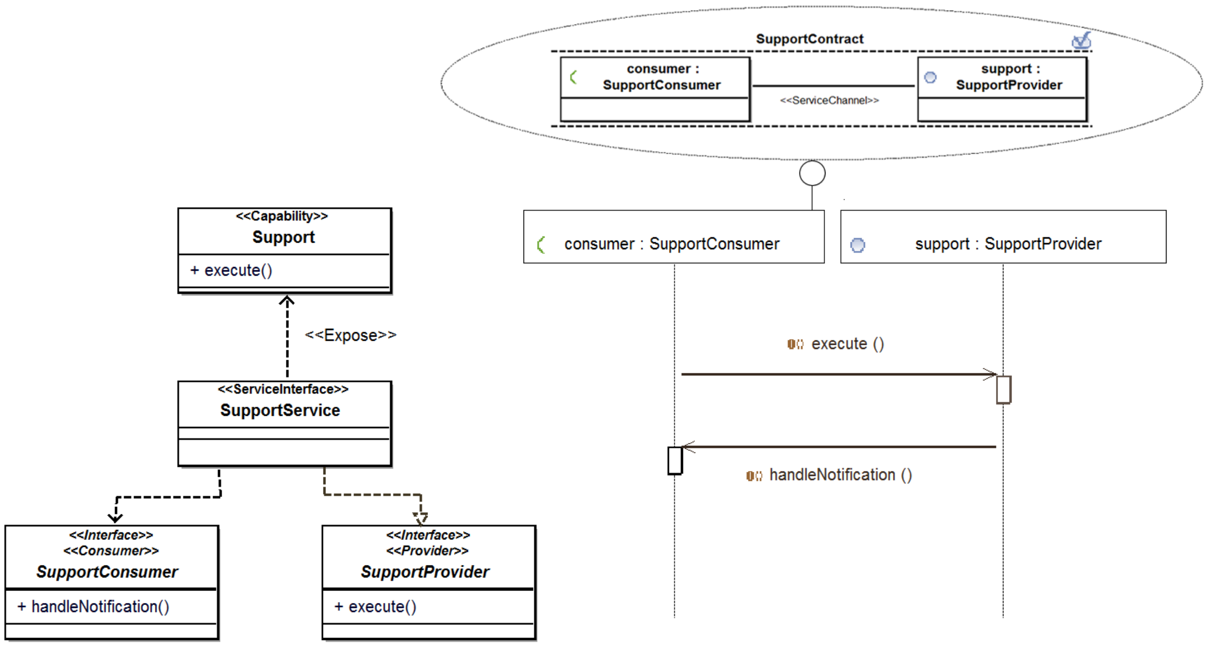


Figure 5.14: Service interface, contract, and protocol of the support service

Control service: exposes the capability of coordinating the services of a robotic agent. Figure 5.15 shows the interfaces of the control service, which provides facade functionalities for navigation, object manipulation, and interaction. Five main functionalities are provided through the `ControlService` interface: (i) `driveRobot` coordinates different services to drive the robot from a position to another; (ii) `moveObject` controls several services for the manipulation of objects; (iii) `interact` manages services to enable the communication of the robotic agent with humans, other agents, or systems used to control devices in the environment; (iv) `report` provides information on the current status of the control, such as the position of the robot or the battery level of charge; and (v) `subscribe` enables consumers to be periodically notified about updates on the state of the robotic control.

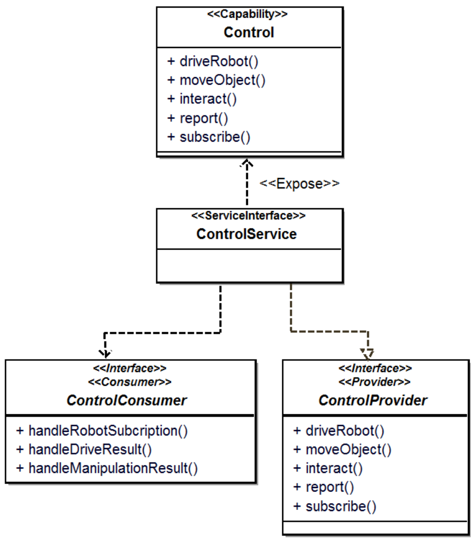


Figure 5.15: Service interface of control service

The use of `ControlService` demands the fulfilment of a contract that enforces interactions in accordance with the three services it acts as facade. It also requires from consumers an interface that is compatible with such a contract. Figure 5.16 illustrates the three protocols encompassed by the `ControllerContract`. The protocols on the left and the centre enforce asynchronous interaction regarding `driveRobot` and `moveObject` functionalities, respectively. The one on the right side describes a service subscription behaviour. Consumers of control service must provide handler functionalities for each of these protocols.

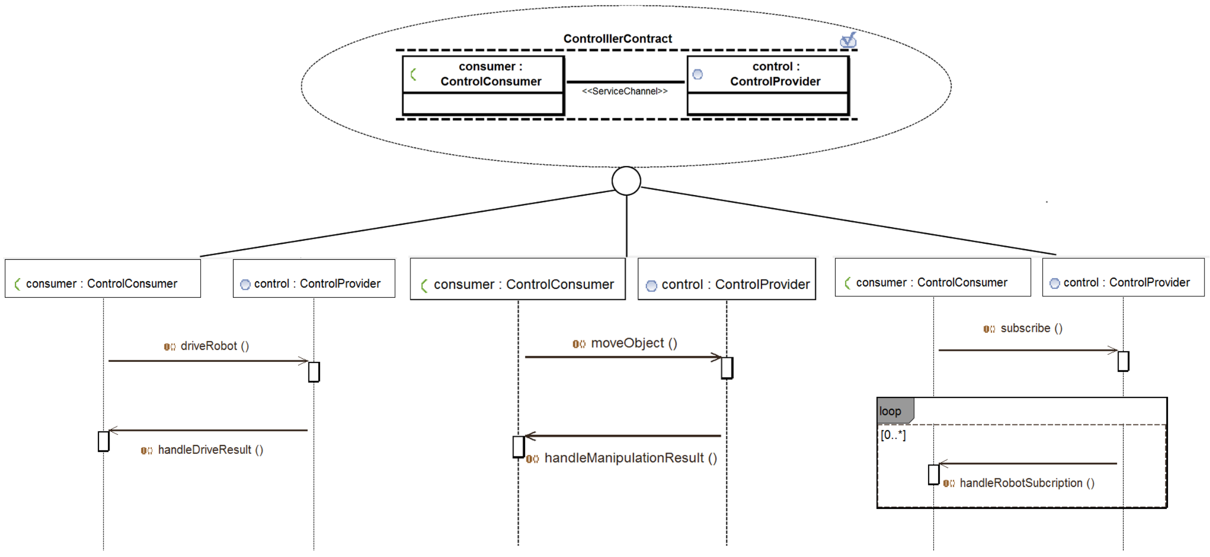

ControlllerContract
consumer : ControlConsumer
<<ServiceChannel>>
control : ControlProvider
consumer : ControlConsumer
control : ControlProvider
driveRobot ()
handleDriveResult ()
consumer : ControlConsumer
control : ControlProvider
moveObject ()
handleManipulationResult ()
consumer : ControlConsumer
control : ControlProvider
subscribe ()
loop
[0..*]
handleRobotSubcription ()

Figure 5.16: Service contract and protocols of the Control service

Robotic agent service: exposes the capability of a robotic agent in performing different activities. This service hides technical details of complex algorithms and hardware devices to facilitate the composition of heterogeneous robots into application workflows. Figure 5.17 shows the `RoboticAgentService` interface, which encompasses the following functionalities: (i) `executeActivity` enacts the execution of a workflow of robotic tasks necessary for performing an activity, such as the transportation of an object; (ii) `report` provides information on the current status of the robotic agent; and (iii) `subscribe` enables service consumers to receive periodic updates. Notice that the `executeActivity` represented in the reference architecture can result in the design of one or more functionalities in the software architectures, as robotic agents usually provide multiple activities.

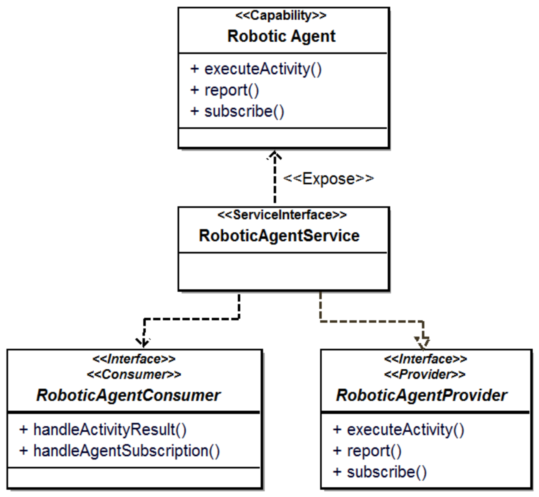


Figure 5.17: Service interface of the robotic agent service

`RoboticAgentService` demands both provided and required interfaces, as well as the fulfillment of a contract encompassing two protocols. The `executeActivity` uses asynchronous communication and requires that consumers provide a callback functionality named `handleActivityResult` to receive response messages. Service subscription also requires from service consumers a handler functionality named `handleAgentSubscrition`. Figure 5.18 shows the robotic agent service contract and its protocols.

Robotic application service: exposes the capability of coordinating one or more robotic agents to perform a robotic application. This service offers the general `executeApplication` functionality, which is used to enact long-term business processes provided by robotic systems. The instantiation of this functionality depends on the robotic systems goals established for each project. Figure 5.19 depicts the interfaces, contract, and protocol of robotic application service. As the completion of a robotic application demands considerable amount of time, interactions with the `executeApplication` functionality must be asynchronous. Therefore, the service protocol enforces that consumers must provide a `handleApplicationResult` to be later notified on robotic application results.

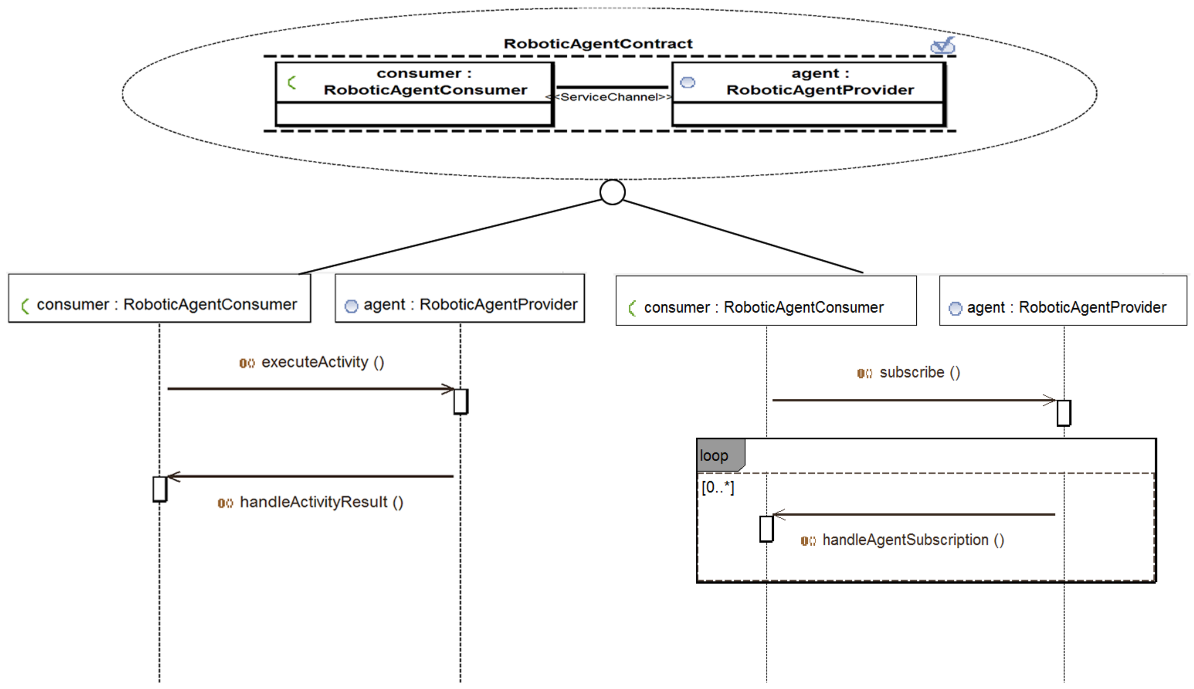

RoboticAgentContract
consumer : RoboticAgentConsumer
<<ServiceChannel>>
agent : RoboticAgentProvider
consumer : RoboticAgentConsumer
agent : RoboticAgentProvider
executeActivity ()
handleActivityResult ()
consumer : RoboticAgentConsumer
agent : RoboticAgentProvider
subscribe ()
loop
[0..*]
handleAgentSubscription ()

Figure 5.18: Service contract and protocols of the robotic agent service

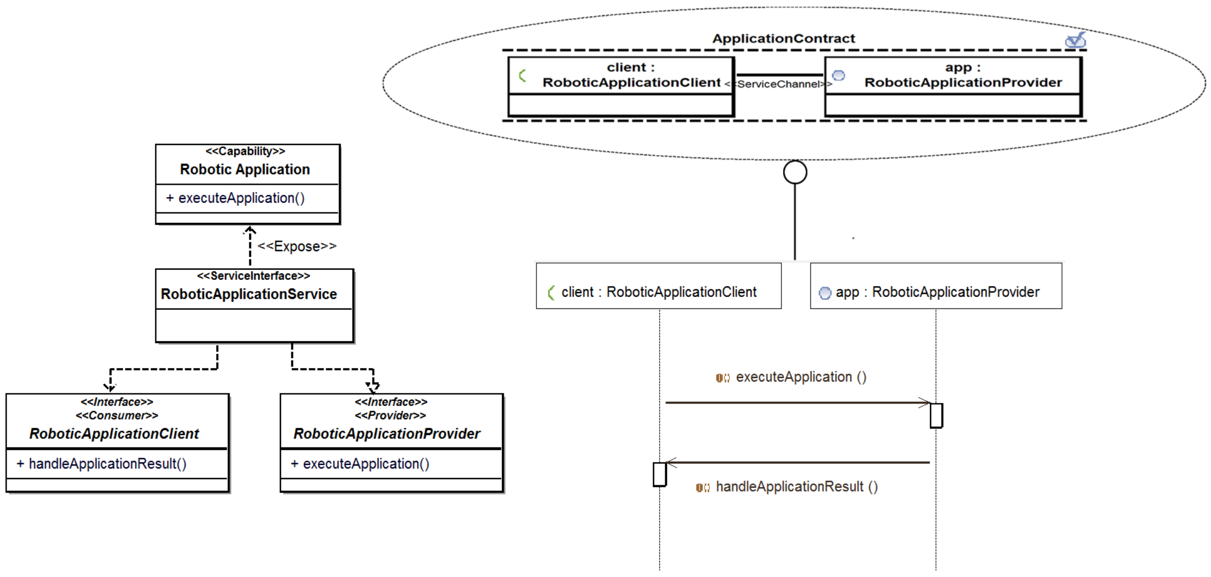


Figure 5.19: Service interface, contract, and protocol of the robotic application service

Table 5.3 shows the architectural requirements addressed by each service in the Service Interface and Contracts view of RefSORS.

Table 5.3: Architectural requirements addressed by Service Interface and Contracts view

Service	Architectural requirement
Sensor driver	AR-R [20], AR-R [24] – AR-R [26], AR-S [1]
Actuator driver	AR-R [29], AR-R [30], AR-S [1]
Resource driver	AR-R [27], AR-R [28], AR-S [1]
Localization	AR-R [14], AR-R [19] – AR-R [21], AR-S [1], AR-S [5]
Mapping	AR-R [17], AR-R [18] – AR-R [21], AR-S [1], AR-S [5]
Path planning	AR-R [16], AR-R [19] – AR-R [21], AR-S [1], AR-S [5]
Navigation	AR-R [15], AR-S [1], AR-S [5], AR-S [6]
Object manipulation	AR-R [13], AR-S [1], AR-S [5]
Interaction	AR-R [10], AR-R [11], AR-R [12], AR-S [1], AR-S [5]
Knowledge	AR-R [20] – AR-R [23], AR-S [1], AR-S [5]
Support	AR-R [19], AR-S [1], AR-S [5]
Control	AR-R [7] – AR-R [9], AR-S [1], AR-S [5], AR-S [6]
Robotic agent	AR-R [1], AR-R [2], AR-R [6] – AR-R [9], AR-R [21], AR-S [1], AR-S [5], AR-S [6], AR-S [8], AR-S [9]
Robotic application	AR-R [1] – AR-R [5], AR-R [21], AR-S [1], AR-S [5] – AR-S [9]

5.4.4 Service Architecture View

Services are abstract specifications of capabilities that can be implemented and provided by different participants of an SOA. The Service Architecture view of RefSORS represents participants of an SORS, their realization of service interfaces, communication through service channels, and fulfilment of service contracts. This view is described by the SoaML Service Architecture diagram shown in Figure 5.20, in which participants are represented by stereotyped components, contracts by dashed ellipses, and contract fulfilments by dashed lines. The structure of interaction between robotics services in the Service Architecture view is similar to the one illustrated in the Capability view. However, the first one adds semantics of interaction by contract, which denotes that services are not simply connected, but they communicate according to specific protocols and compatible interfaces.

Participants that realize sensor driver services, actuator driver services, and resource driver services are responsible for interfacing with the hardware of robots and, therefore, form the basis of the SORS service architecture. They provide functionalities that support other participants, such as `Mapper` and `PathPlanner`, to perform their services. Participants that realize services associated with robotics tasks offer, in turn, functionalities to other tasks and to the `Robot Controller` participant. `Knowledge Management` and `Support` participants are also orchestrated by the `Robot Controller`. The `Robotic Agent` encompasses functionalities executed by the aforementioned participants, providing their services for the `Application`. Interactions between participants of the service architecture are guided by contracts described in the Service Interface and Contracts view.

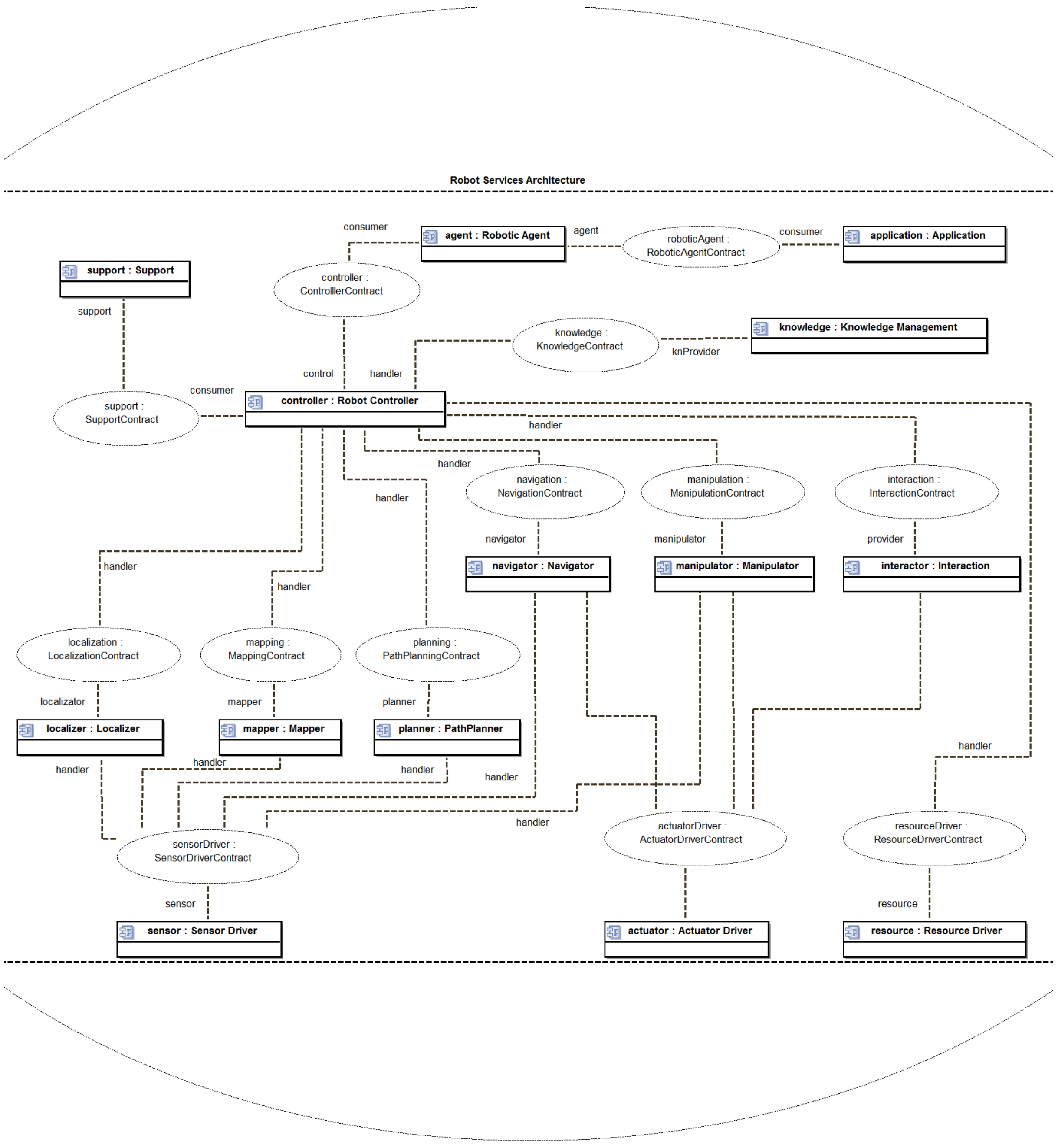

Robot Services Architecture
agent : Robotic Agent
roboticAgent : RoboticAgentContract
application : Application
support : Support
controller : ControlllerContract
knowledge : KnowledgeContract
knowledge : Knowledge Management
support : SupportContract
controller : Robot Controller
navigation : NavigationContract
manipulation : ManipulationContract
interaction : InteractionContract
navigator : Navigator
manipulator : Manipulator
interactor : Interaction
localization : LocalizationContract
mapping : MappingContract
planning : PathPlanningContract
localizer : Localizer
mapper : Mapper
planner : PathPlanner
sensorDriver : SensorDriverContract
actuatorDriver : ActuatorDriverContract
resourceDriver : ResourceDriverContract
sensor : Sensor Driver
actuator : Actuator Driver
resource : Resource Driver
consumer
agent
support
control
handler
knProvider
navigator
manipulator
provider
localizator
mapper
planner
sensor
resource

Figure 5.20: Service Architecture view of RefSORS

The following architectural requirements of RefSORS are addressed by the Service Architecture view: AR-R [1] – AR-R [8], AR-R [11] – AR-R [18], AR-R [20], AR-R [21], AR-R [24], AR-R [26] – AR-R [29], AR-S [1], and AR-S [6] – AR-S [10].

5.4.5 Service Deployment View

Robotic systems are frequently used in safe-critical applications in which quality characteristics such as dependability and performance are not only desirable but mandatory.

Most of runtime quality attributes can be affected by the network and hardware infrastructures adopted in the service deployment. The Service Deployment view of RefSORS describes the infrastructures involved in the execution of an SORS and details how services should be deployed to mitigate possible problems of QoS. Figure 5.21 illustrates a UML Deployment diagram that represents this view.

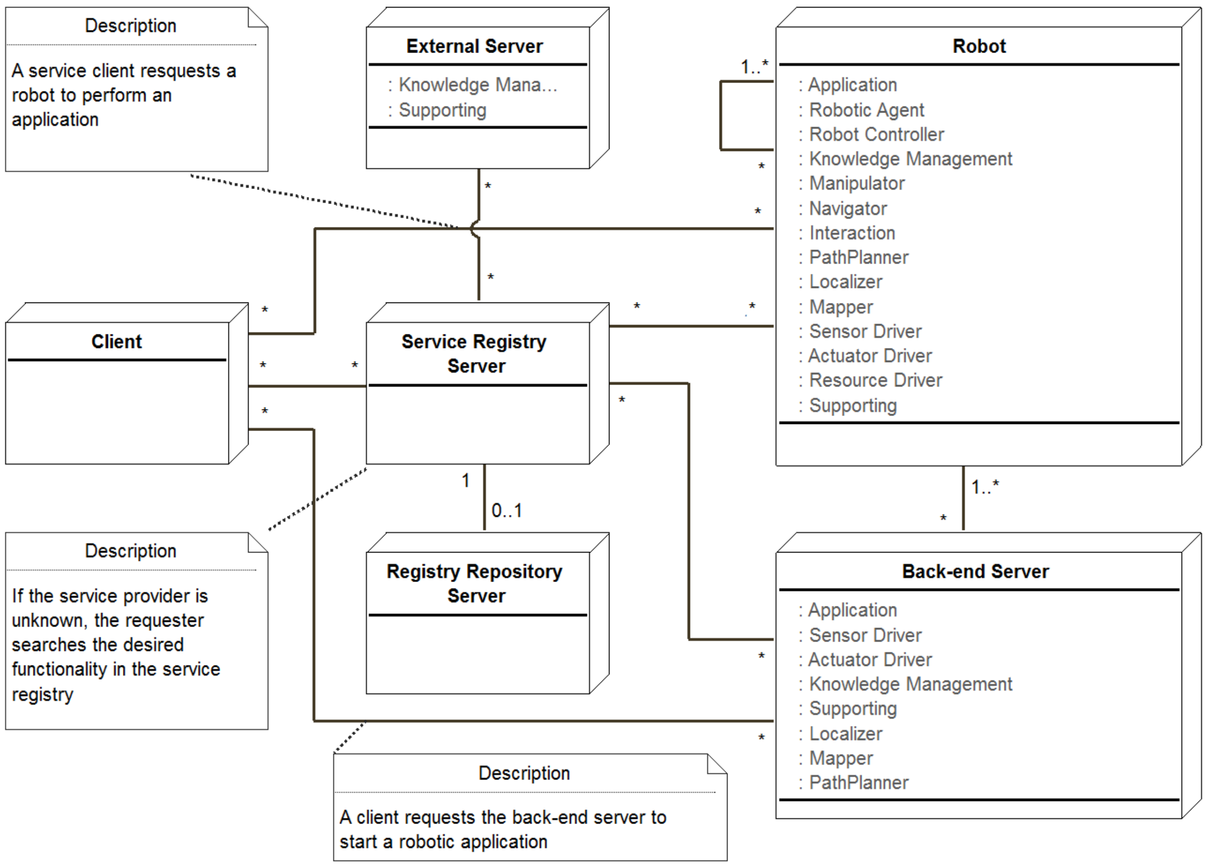


Figure 5.21: Service Deployment view of RefSORS

According to this view, services of SORS can be deployed into `Robots`, local `Back-end Servers`, and `External Servers`. Services that are fundamental for the correct operation of the robotic system in real-time, e.g., navigation and control, must be deployed into the robot or in a dedicated computer attached to it. Services providing functionalities that are not critical for the robotic system operation can be deployed into a `Back-end Server`. These services can be used to provide additional processing power for robotic systems, store large datasets or other sources of knowledge, and control sensors and actuators spread over the environment. Support and knowledge management services that are complementary and do not interfere in the real-time operation of the robotic system can be accessed from `External Servers`. For instance, machine-readable data available on the Internet can be used to obtain additional semantic information on objects inside the environment, as reported by Blake et al. (2011).

Consumers hosted in `Client` servers can access services for SORS both directly or thought a `Service Registry Server`. Direct communication is used when a consumer knows the end-point of the service provider at design time. This interaction is often adopted between services associated with real-time operation, to reduce problems regarding availability and other QoS attributes. Communication thought `Service Registry Server` enables robotic systems to transparently discover additional functionalities at runtime. `Service Registry Servers` are brokers in which service descriptions are registered and stored (using `Registry Repository Server`) to be later discovered. Indirect communication between services increases flexibility of service-oriented systems, as new functionalities can be discovered and added on demand. However, dynamic discovery and dynamic binding of services are still open issues in the domain of robotics. Currently, services for SORS can be indirectly discovered (as discussed in Section 3.3.1), but their composition is done at design time.

The Deployment view addresses the following architectural requirements: AR-R [1], AR-R [8], AR-R [10], AR-R [11], AR-R [19], AR-R [21] – AR-R [23], AR-R [25], AR-R [28], AR-R [30], AR-S [2] – AR-S [5], and AR-S [8] – AR-S [11].

5.5 Step RA-4: Architectural Evaluation

RefSORS was independently evaluated by three external reviewers to identify possible problems of omission, inconsistency, and missing information. This activity was performed incrementally and we updated RefSORS after each review based on the pointed mistakes. Firstly, we submitted RefSORS to an expert in architectural representation to verify three main issues: (i) if RefSORS provides an adequate set of architectural views; (ii) if these views are described using a level of formalism appropriate to stakeholders; and (iii) if models in the views are represented using adequate description language. After that, an expert in reference architectures evaluated RefSORS to check the completeness of information related to its construction and general adequacy of its documentation. Finally, RefSORS was reviewed by an expert in reference architectures for embedded systems to evaluate its pertinence with respect to the robotics domain.

These inspections pointed out lack of information on: (i) relation between architectural requirements and architectural views; (ii) points of variability and how they affect the rest of the architecture; and (iii) how to deploy participants described in the Service Architecture view. The first problem was mitigated by adding a list of addressed architectural requirements to the documentation of each architectural view. The problem regarding points of variability was not tackled in the current version of the reference architecture and is considered an object of future research. RefSORS was described in SoaML and this language does not support description of variability. The third problem was solved by adding the Service Deployment view, which complements other views described in SoaML with a UML deployment diagram. Minor problems of inconsistency between views were also identified and later corrected in the reference architecture documentation. As a result, a more consistent and complete architectural description of RefSORS was achieved.

Additionally to the architectural evaluation, we also conducted a case study to provide evidences on the viability of RefSORS and observe possible benefits of its adoption. This study is discussed as follows.

5.6 Case Study

We conducted a two-step case study on the instantiation of RefSORS for the development of an indoor grounded mobile SORS. In the first step, a student was selected to design an SORS software architecture for the same project proposed in the second part of the experiment reported in Section 4.3 (see Appendix B). We reproduced the same schedule, activities, and conditions imposed to the group that designed the SORS using the ArchSORS process, but introducing RefSORS as additional support. After that, we computed the four metrics (MF, CpF, Max CpC, and CF) for the software architecture and compared them to the results obtained by the group in the experiment that only used ArchSORS. In the second step, we asked the same student to instantiate the other views of RefSORS following ArchSORS and to implement the robotic system.

The project considered in the case study involves the development of a robotic system that controls two handling robots in an industrial shop floor. Robots in this project must transport products manufactured in two different production units and place them into their respective storage units based on the product type. Initially, both robots are in standby mode in their rest areas. At the signal given by the operator, one of the two robots must: (i) pick up a product in the production unit; (ii) bring the product to the storage unit; and (iii) return to a rest area to remain in standby mode until the next transportation request. Robots used in this project are both Pioneer P3-DX platforms equipped with a laser rangerfinder, a pan-tilt-zoom (PTZ) camera, a GPS sensor, and a two-axis gripper.

Phase RSA-1 of ArchSORS was not necessary in this case study – neither in the experiment – because the description of the project already provided requirements for the robotic system. In Phase RSA-2, system requirements were mapped into architectural requirements of RefSORS to identify concepts of the domain addressed by the project. After that, the Conceptual view of RefSORS (Section 5.4.1) was used to identify lanes of BPMN diagrams created to represent the application flow. The first BPMN diagram, shown in Figure 5.22, was created to describe the application flow in terms of Robotic Application and its Robotic Agents. Activities of this diagram were then further decomposed into more concrete BPMN diagrams that illustrate the flow of robotic tasks, such as the mapping of the environment, manipulation of products, and transport of products from production units to storage units.

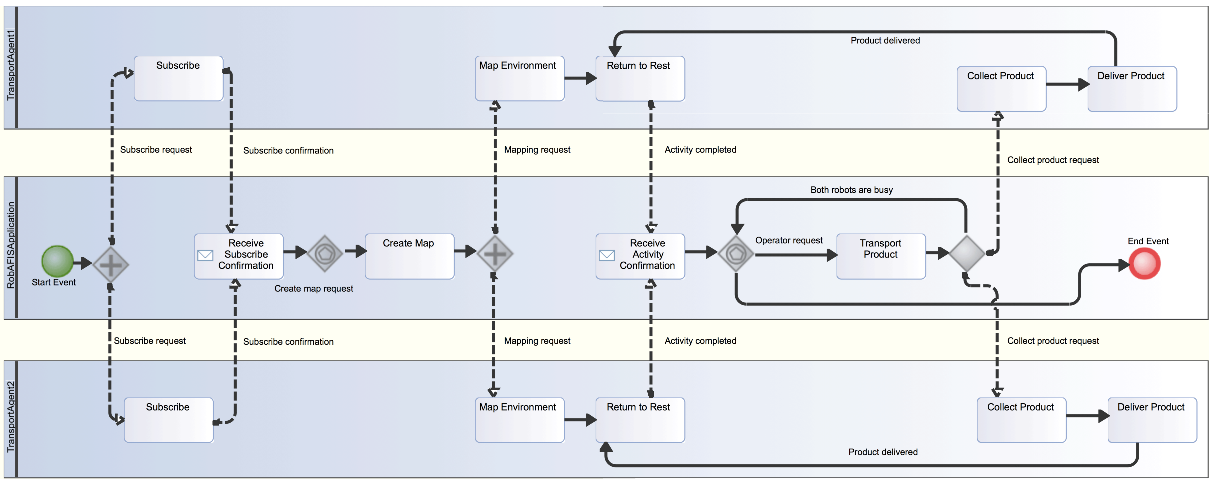


Figure 5.22: BPMN diagram designed to represent the application flow

The capability view of the software architecture was created based on functionalities described in the BPMN diagrams and the template proposed in the Capability view of RefSORS (Section 5.4.2). As shown in Figure 5.23, the `Sensor Driver` capability of RefSORS was instantiated into four capabilities: `GPS Driver`, `Camera Driver`, `Laser Driver`, and `Sonar Driver`. The `Actuator Driver` capability of RefSORS was used to identify capabilities for controlling of the Pioneer P3-DX base platform (`Base Driver`) and the two-axis gripper (`Gripper Driver`). Besides that, the `Knowledge Management` and `Support` capabilities of RefSORS were instantiated as capabilities for storing the map of the environment (`Map Information`) and processing images (`Image Processing`), respectively. The use of a `Map Information` capability was necessary to enable both robots to cooperatively build the map of the environment.

In Phase RSA-3, all capabilities identified for the robotic system were modelled as services according to guidelines provided in the Service Interface and Contracts view of RefSORS (Section 5.4.3). Requirements of the robotic system addressed by each service were also documented as prescribed in the Activity RSA-A 3.1 of ArchSORS. After that, participants designed to provide services of the robotic system were connected to represent the Service Architecture shown in Figure 5.24. The structure and the contract fulfilments of this functional architecture were defined based on the Service Architecture view of RefSORS (Section 5.4.4). Protocols associated with the contracts in the software

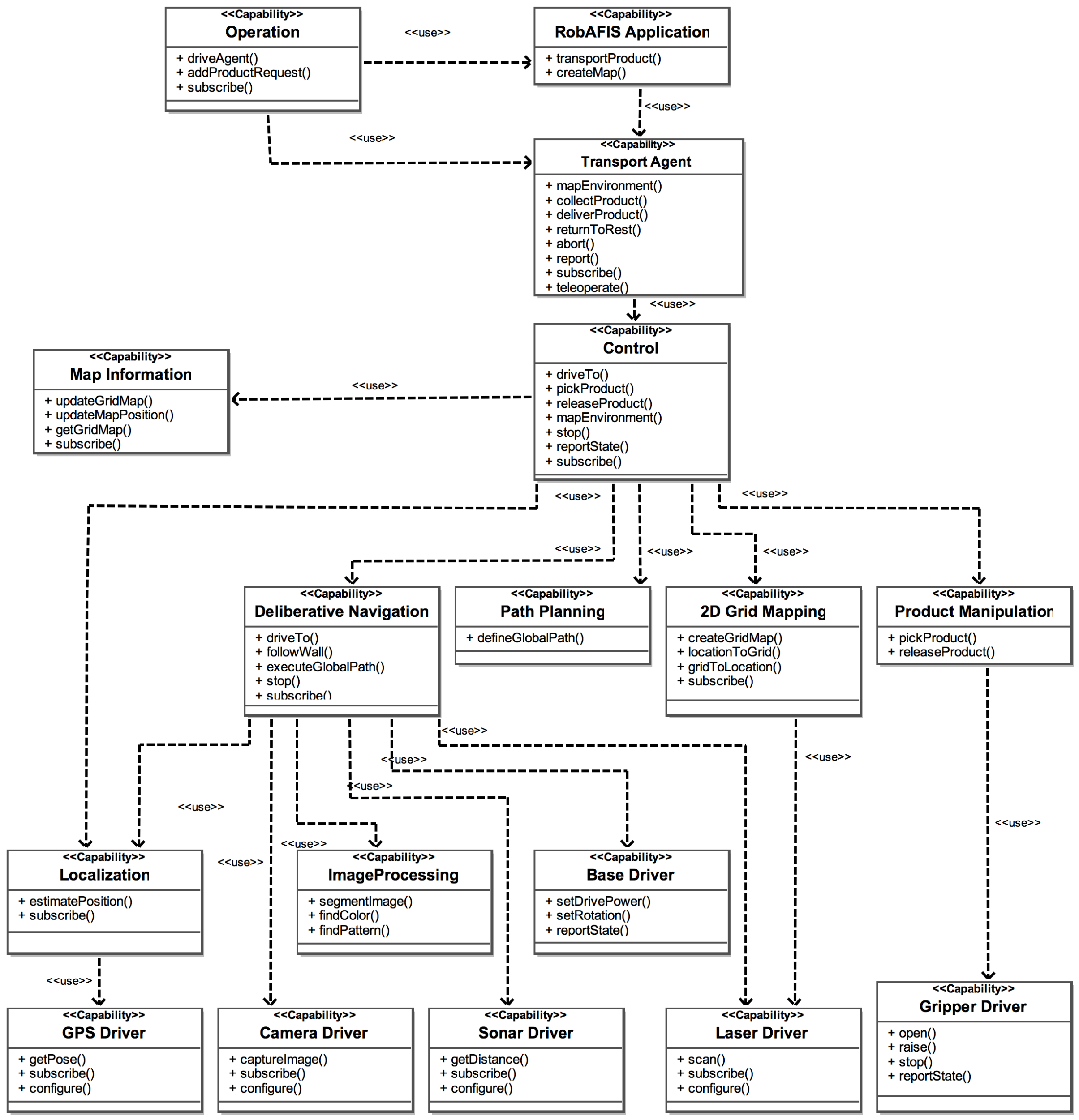


Figure 5.23: Capability view of the software architecture

architecture followed the behaviours of synchronous and asynchronous communication and service subscription established in RefSORS.

Similarly to the experiment previously conducted (detailed in Section 4.3), we evaluated the SORS functional architecture and its set of services using the same four metrics related to coupling, cohesion, and modularity. Results shown in Figure 5.25 indicate that the software architecture designed from the instantiation of RefSORS is more modular and cohesive than the mean (and median) of the architectures designed only using ArchSORS (i.e., it displays higher values for MF and CF). The software architecture of the case study is also less coupled than the mean (and median) of the architectures produced in the experiment (i.e., it displays a lower value for CpF), albeit slightly more centred in

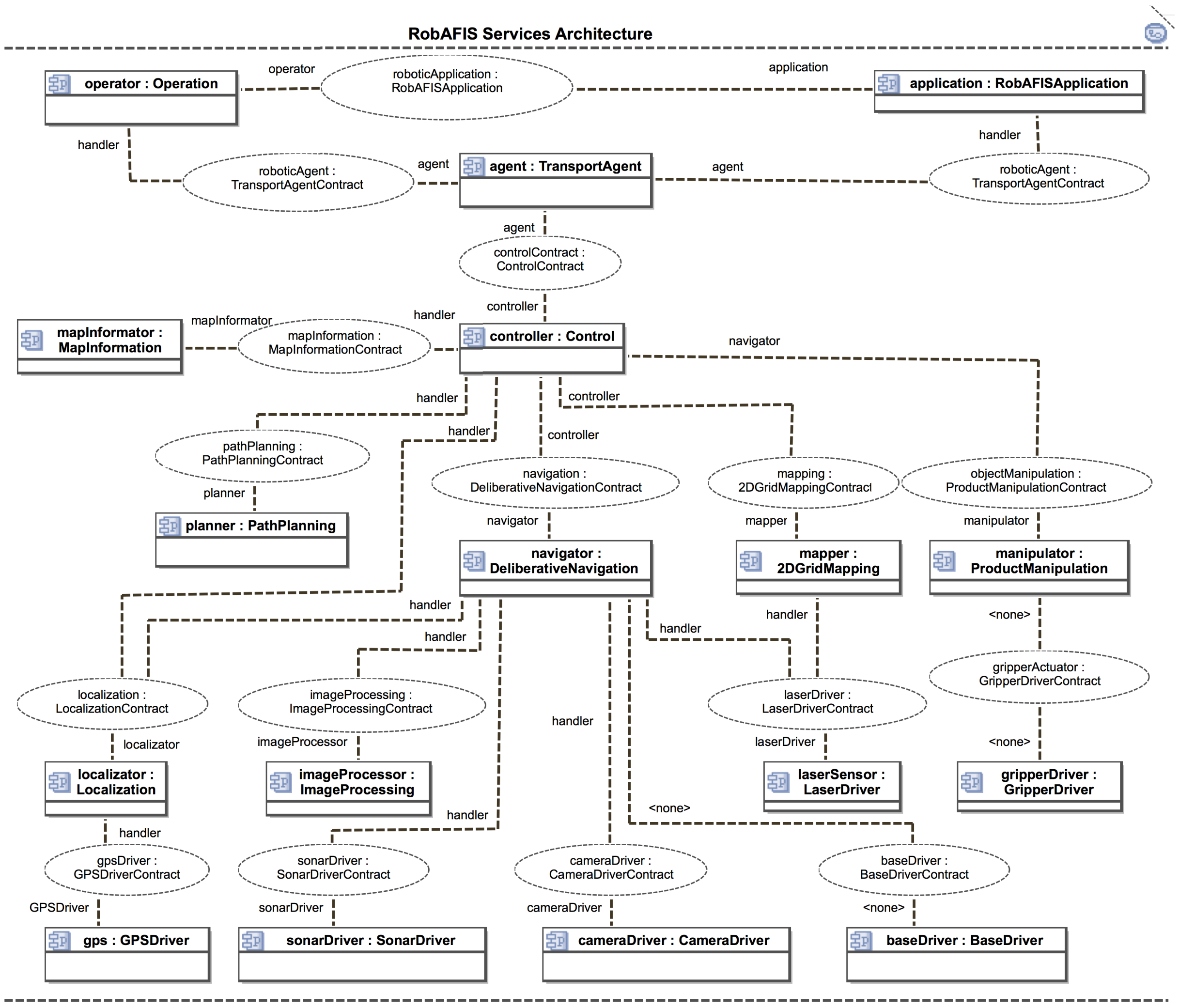

RobAFIS Services Architecture
operator : Operation
operator
roboticApplication : RobAFISApplication
application
application : RobAFISApplication
handler
roboticAgent : TransportAgentContract
agent
agent : TransportAgent
agent
roboticAgent : TransportAgentContract
handler
agent
controlContract : ControlContract
controller
mapInformator : MapInformation
mapInformator
mapInformation : MapInformationContract
handler
controller : Control
navigator
handler
controller
handler
controller
pathPlanning : PathPlanningContract
planner
planner : PathPlanning
navigation : DeliberativeNavigationContract
navigator
navigator : DeliberativeNavigation
mapping : 2DGridMappingContract
mapper
mapper : 2DGridMapping
objectManipulation : ProductManipulationContract
manipulator
manipulator : ProductManipulation
<none>
gripperActuator : GripperDriverContract
<none>
gripperDriver : GripperDriver
handler
handler
handler
handler
handler
localization : LocalizationContract
localizator
localizator : Localization
imageProcessing : ImageProcessingContract
imageProcessor
imageProcessor : ImageProcessing
laserDriver : LaserDriverContract
laserDriver
laserSensor : LaserDriver
<none>
handler
handler
gpsDriver : GPSDriverContract
GPSDriver
gps : GPSDriver
sonarDriver : SonarDriverContract
sonarDriver
sonarDriver : SonarDriver
cameraDriver : CameraDriverContract
cameraDriver
cameraDriver : CameraDriver
baseDriver : BaseDriverContract
<none>
baseDriver : BaseDriver

Figure 5.24: Service Architecture view of the software architecture

the control and navigation services (i.e., the most coupled services). Although the value of Max CpC in the case study is higher than most of software architectures designed using only ArchSORS, the result is much lower than those obtained by architectures created in *ad hoc* manner (see Table 4.1).

In the second part of the case study, the functional architecture was further detailed to enable the implementation of the robotic system. In Phase RSA-4, different services of the system were described by object-oriented diagrams representing their strategies of implementation. For instance, the mapping service was designed to enable creation of an occupancy grid. The path planning service was modelled to used this occupancy grid to execute an A* algorithm. The navigation service was designed to be deliberative, as the robotic application is executed in a controlled environment. Additionally, the deployment diagram of the software architecture was created by instantiating the Deployment view of RefSORS. The infrastructure described in Figure 5.26, as well as the hardware realization strategies, followed specifications established by the proposed project. Therefore, only two

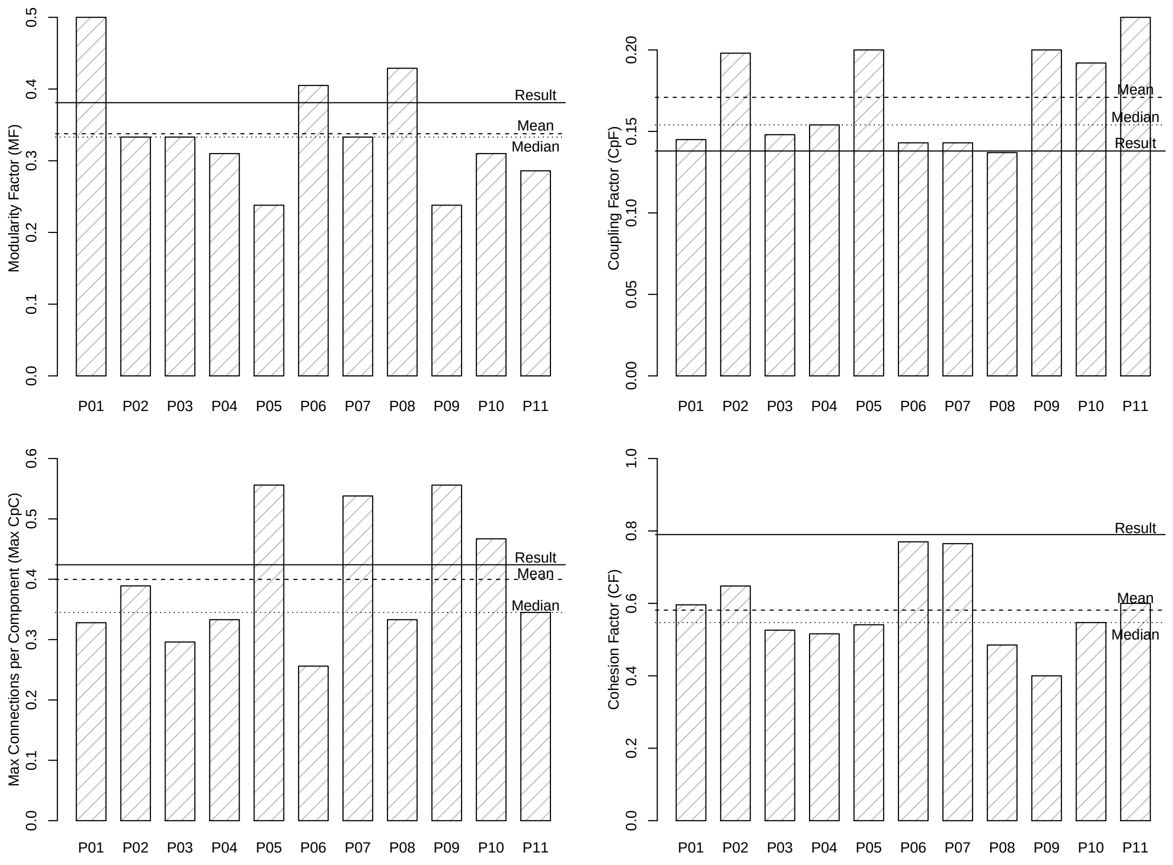


Figure 5.25: Comparison between results of the case study and results of subjects of the experiment

robots and a supporting laptop were considered. Due to the dimensions of the project, activities of Phase RSA-5 were not performed during the case study.

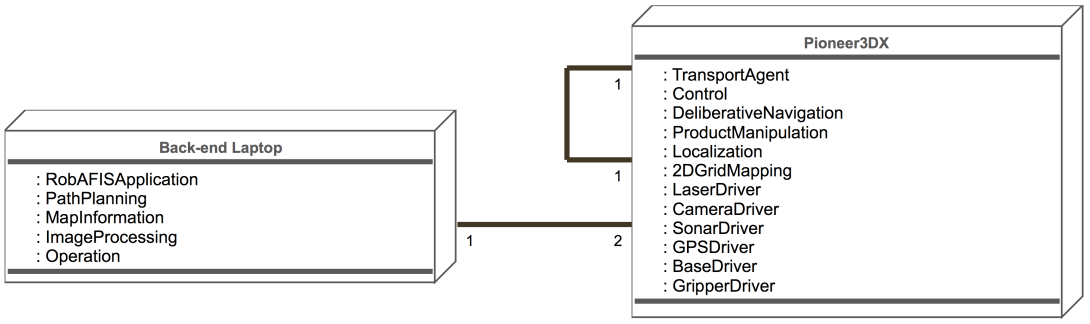


Figure 5.26: Deployment view of the software architecture

The robotic system designed in this case study was implemented in the ROS development environment. Both robots and the industrial shop floor were simulated using the Gazebo virtual environment. Figure 5.27 shows the robots cooperatively mapping the shop floor before starting the transport of products. In order to create the occupancy grid map, the robots execute a wall following navigation until they reach back their rest areas. This task is partially illustrated in images (1) to (4). This grid map is later used to calculate optimal paths from the rest areas to the production units by using the A* algorithm.

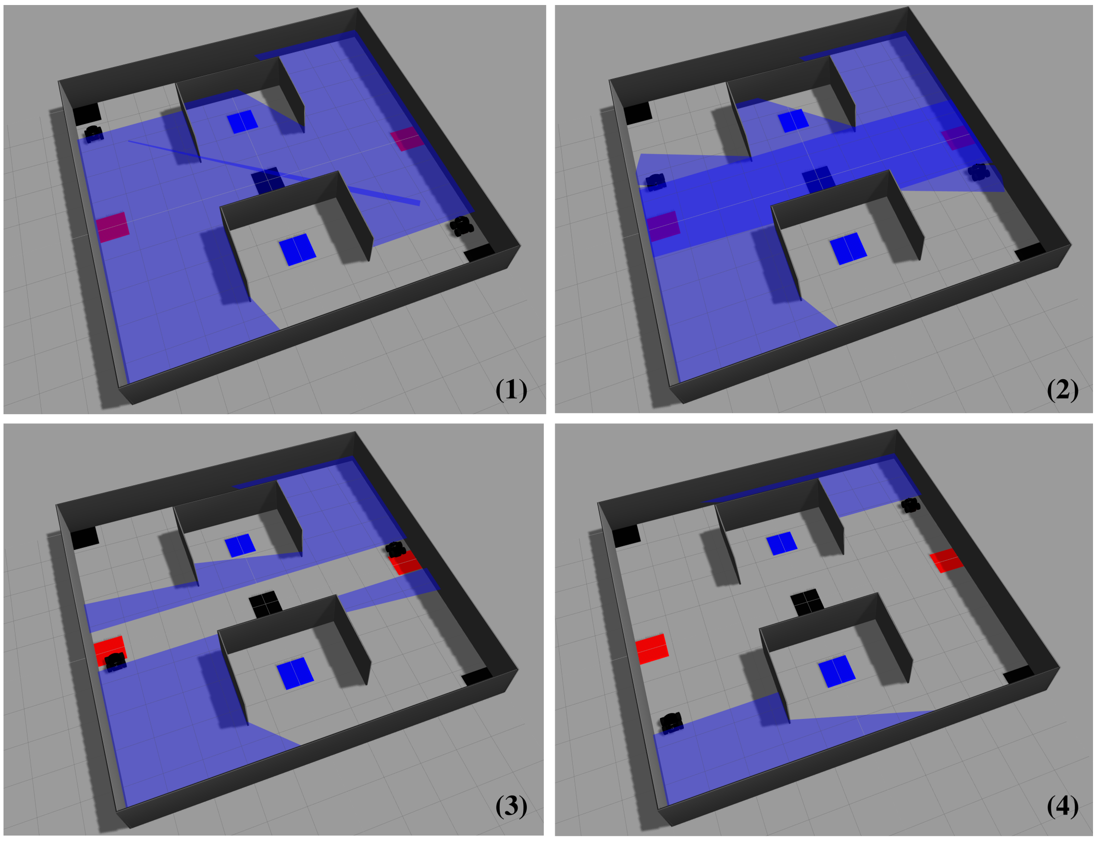


Figure 5.27: Simulated robots mapping the shop floor

5.7 Discussion of Results and Limitations

The case study was conducted to illustrate an instantiation of RefSORS to design a grounded mobile robotic system. We opted for using the same project proposed in the experiment to observe the impact of associating RefSORS with the ArchSORS process. Results indicate that RefSORS had positive influence on the modularity, cohesion, and coupling of the designed architecture. The result for the metric indicating the dependency from the control service was also good, but not better than results obtained in the experiment. Therefore, in general, the adoption of RefSORS led to a software architecture that presented good compromise between the three metrics proposed by Galster et al. (2008) and the metric introduced in the scope of this work. As discussed in Section 4.4, an architecture that presents good modularity, cohesion, and coupling is more likely to result in a system of higher quality. However, the achievement of quality characteristics such reusability, maintainability, and buildability still depends on how the system is implemented (Bass et al., 2012).

Results obtained by our case study can not be generalized to other contexts, as it involved the development of a single system. However, considering that RefSORS provides an architectural template for the design of SORS, we believe that software architectures designed in the future may present similar results. In this sense, we plan to conduct other case studies to strengthen the evidences obtained thus far. Besides this quantitative analysis, it is worth mentioning that RefSORS is aligned with several reference architectures of robotics and SOA. Services addressed by RefSORS were based on the taxonomy described in Section 3.2, which was considered by a board of ten experts as correct and complete. This reference architecture presents, therefore, good perspectives to be adopted and contribute to the design of SORS software architectures.

5.8 Final Remarks

This chapter presented RefSORS, a reference architecture for indoor grounded mobile SORS. The establishment of RefSORS was guided by the taxonomy of services reported in Chapter 3, several reference architectures for robotic systems and SOA-based systems, and the main control architectures of robotics. RefSORS was described in five views, represented in informal and semi-formal notations, to address different concerns and stakeholders of SORS projects. The documentation of RefSORS was incrementally reviewed by three independent researchers to mitigate issues related to its completeness and correctness. Additionally, we performed a case study to illustrate the applicability of RefSORS. Results indicate that RefSORS can contribute to the development of SORS software architectures, including those ones designed using the ArchSORS process. However, other case studies are necessary and will be conducted to confirm the obtained results, identify possible improvements for the reference architecture, and increase the confidence on its adoption.

The next chapter concludes this thesis, summarizing the main contributions, discussing general limitations, and mentioning future work.

Chapter

6

Conclusion

Robotics is experiencing rapid growth and robots are already supporting several application domains. The demand for robots of higher autonomy and decision making capacity is challenging engineers, architects, and developers to create considerably large robotic systems in timely manner. Several architectural styles have already been investigated to improve both productivity and quality in the development of such systems. Particularly, SOA has been increasingly adopted as the underlying architectural style of many robotic systems in recent years. Nevertheless, most of SORS are still designed in *ad hoc*, which reduces the potential of SOA in promoting reusability, modularity, and other important quality attributes. In this perspective, research focusing on the architectural design of SORS is important to consolidate the adoption of SOA in robotics, as well as foster productivity and quality in the development of these systems.

This thesis contributed in this sense, supporting a better understanding and systematization of the architectural design of SORS. Achievements of this work include the definition and evaluation of a taxonomy of services for the robotics domain, the creation of a mechanism for the publication and discovery of services, the proposal of a process for designing SORS software architectures, and the establishment of a reference architecture for indoor grounded mobile SORS.

The set of contributions described along the thesis are revisited in Section 6.1. Section 6.2 summarizes the limitations of the work, how these limitations can be overcome, and directions for further research.

6.1 Revisiting the Thesis Contributions

This section summarizes the main contributions of this thesis.

Definition of a taxonomy of services for SORS: we defined a taxonomy of services for the robotics domain. This taxonomy, presented in Section 3.2.3, was established according to 39 studies on the development of SORS (see Section 2.4.5), several reference architectures for robotic systems, and the knowledge of experts. We applied a survey to ten specialists in robotics and the results evidenced the taxonomy is comprehensive enough for the classification of services available for SORS. This taxonomy can, therefore, be used for improving communication among developers of robotics, supporting the development of SORS;

Development of a mechanism for cataloging and discovering services for SORS: we designed and implemented a mechanism, named RoboSeT, to support classification and discovery of services for SORS (see Section 3.3). RoboSeT automates the taxonomy proposed in this thesis to overcome limitations of existing service repositories in robotics, enabling services to be transparently discovered, obtained, and evaluated. Preliminary results of a case study indicate that RoboSeT can foster service reuse and, therefore, improve productivity in the development of SORS;

Establishment of a process for the design of SORS software architectures: we proposed a process, named ArchSORS, to systematize the design of SORS software architectures (see Chapter 4). For each phase of ArchSORS, we described a method containing a set of activities, tasks, work products, and key deliverables. We also represented ArchSORS using the SPEM 2.0 notation and created an IBM RMC *method plugin*, which enables instantiations of the process and its integration into project management tools;

Experimental evaluation of the proposed process: we evaluated ArchSORS in an experimental study, reported in Section 4.3. The controlled experiment compared the current *ad hoc* approach used to create SORS software architectures and the development using ArchSORS. Results provide evidences that ArchSORS can produce architectures of higher modularity and cohesion that are also less coupled and centered in the control service. The improvements achieved in these four metrics suggest the proposed process can positive impact on important quality attributes of SORS, such as modifiability, reusability, complexity, and buildability; and

Establishment of a reference architecture for indoor grounded mobile SORS: We proposed a reference architecture, named RefSORS, to facilitate the design of software architectures for indoor grounded mobile SORS (see Chapter 5). RefSORS is aligned with the ArchSORS process (see Chapter 4) and describes how services reported in the taxonomy (see Section 3.2.3) can be designed and composed in a SORS. Preliminary results of a case study indicate that software architectures designed from the instantiation of RefSORS have better modularity, coupling, and cohesion when compared to the ones designed only using ArchSORS.

The achievements of this thesis contribute to the areas of Software Architecture and Robotics, as they advance the current state-of-the-art on the architectural design of robotic systems based on SOA.

6.2 Limitations and Future Work

This section describes limitations of this thesis and how they can be tackle in the future. Notice that specific limitations have already been discussed in previous chapters. Therefore, we herein focus on broader limitations to be overcome and possible improvements that can be made during short and medium term research. We conclude pointing out future directions of research in the areas associated with the design of software architectures for SORS. It worth highlighting that we intent to deal with these issues during the postdoctoral project planned to shortly after the completion of this doctorate.

Evaluation of the taxonomy of services for SORS: the evaluation of the taxonomy proposed in Chapter 3 involved ten experts in robotics. This limited number of interviewees threats the generalization of the achieved results. Therefore, we plan to conduct broader evaluations of the taxonomy as soon as we obtain the commitment of more experts. This will support us to revisit and update the taxonomy with the aim of gaining more confidence on its correctness and completeness, fostering its adoption as an widely accepted classification of services for robotics;

Limitations of RoboSeT: the mechanism described in Chapter 3 advances the current practice in classifying and discovering services for SORS. However, improvements can still be implemented to promote its adoption by the robotics community. For instance, the inclusion of new services in RoboSeT is currently done manually, but it could be automated to facilitate the importing of services already registered in the ROS Wiki. Another possible extension is the publication of services directly from the plug-ins installed in the local machines, which would ease the sharing of services through RoboSeT;

Limitations of the ArchSORS process: the ArchSORS process described in Chapter 4 provides guidelines that facilitate the development of SORS software architectures. However, additional information and support could be provided to facilitate the application of its phases. For instance, we plan to enrich the proposed methods by providing document templates and further directions on how to conduct their activities;

Limitations in the experimental study: the experiment used to evaluate ArchSORS involved 30 subjects and was performed in a single day. The obtained results provide encouraging evidences, but the scope was limited and opportunities for future evaluations are manifold. First, an experiment involving professionals from industry could be performed to enable a stronger generalization of the results. It is also worth conducting case studies in larger and longer projects to observe qualitative aspects involving the development of SORS software architectures according to ArchSORS;

Evaluation of the RefSORS: we conducted a case study to evaluate the RefSORS reference architecture and the preliminary results were promising. However, as discussed in Section 2.2.2, the benefits of adopting reference architectures are inherently difficult to estimate and generalize. Therefore, we plan to conduct other case studies in order to provide additional evidences that increase the confidence on the adoption of this reference architecture.

6.3 Possible Extensions

Many opportunities of research emerged during the development of this thesis. They represent perspectives of future research that can contribute to the areas of Robotics and Software Architecture. Some of them are described as follows.

Integration of robotic systems in the context of systems-of-systems: most of robotic systems are still developed to operate isolatedly and can not interact with other types of systems, which hampers their use into larger and more complete applications running in the context of Systems-of-Systems (SoS). SoS are complex, large-scale software systems in which operationally and managerially independent systems cooperate to provide new, unique features that can not be provided by any constituent separately (Dagli and Kilicay-Ergin, 2008). In this perspective, research on the integration of robotic systems in the context of SoS can contribute to increase the use of robots in many areas of society benefited by SoS, including national security and elderly care;

Development of an ADL for service-oriented embedded systems: several description languages, e.g., as SysML and MARTE, are currently available for the design of embedded systems. However, these languages do not support the representation of elements of SOA, such as service interfaces, contracts, and protocols. Although a proposal of a UML profile that combine MARTE and SoaML already exists (Muhammad et al., 2012), further investigation is still necessary and can contribute to design of a whole class of systems, including SORS;

Support to the instantiation of reference architectures: different processes and methods in the literature support the establishment of reference architectures. For instance, ProSA-RA has already attained considerable maturity and can be applied in different application domains. Nevertheless, instantiation of reference architectures is still performed in *ad hoc* and depends exclusively on the expertise of software architects. Future research focused on guiding and facilitating this activity can contribute to increase the adoption of reference architectures;

Support to the dynamic discovery and binding: several techniques and technologies currently support discovery and binding of Web services at runtime, which increases flexibility of service-based systems. However, most of safety-critical embedded systems based on SOA are still developed by assembling services identified at design time. In this perspective, researches proposing techniques and technologies for enabling dynamic discovery and binding of services for embedded systems involving real-time and safety constraints are still necessary; and

Evolution of the taxonomy into an ontology: the taxonomy proposed in this thesis aims at improving communication among developers by identifying the main types of services that can be used to create robotic systems. The evolution of such a taxonomy into an ontology can be an important contribution towards the dynamic discovery, binding, and reconfiguration of services for robotic systems at runtime. Therefore, investigating the relationship among the taxonomy proposed herein and available ontologies for non-service-oriented robotic systems, such as the IEEE 1872-2015 standard (IEEE, 2015), can be considered a promising extension of this work.

References

Affonso, F. J.; Felizardo, K. R.; Oliveira, L. B. R.; Nakagawa, E. Y. Reference architectures for self-managed software systems: a systematic literature review. In: *Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS'14)*, Maceió, Brazil, 2014, p. 21–30.

Aguiar, A.; Filho, S.; Magalhães, F.; Casagrande, T.; Hessel, F. Hellfire: A design framework for critical embedded systems' applications. In: *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED'10)*, San Jose, USA, 2010, p. 730–737.

AGX VANT Tiriba. Online, `http://www.agx.com.br/` - Accessed in January 20th 2015, 2015.

Ahn, S. C.; Lee, J.-W.; Lim, K.-W.; Ko, H.; Kwon, Y.-M.; Kim, H.-G. Requirements to UPnP for robot middleware. In: *Proceedings of the 19th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*, Beijing, China, 2006a, p. 4716–4721.

Ahn, S. C.; Lee, J.-W.; Lim, K.-W.; Ko, H.; Kwon, Y.-M.; Kim, H.-G. UPnP SDK for robot development. In: *Proceedings of the 2nd SICE-ICASE International Joint Conference (SICE-ICASE'06)*, Bexco, Korea, 2006b, p. 363–368.

Ahrens, D.; Frey, A.; Pfeiffer, A.; Bertram, T. Objective evaluation of software architectures in driver assistance systems. *Computer Science - Research and Development*, v. 28, p. 23–43, 2013.

Albus, J. S. 4D/RCS - A reference model architecture for intelligent unmanned ground vehicles. *Unmanned Ground Vehicle Technology*, v. 4715, p. 303–310, 2002.

Allen, R. *A formal approach to software architecture.* PhD Thesis, Carnegie Mellon University, School of Computer Science, 1997.

Allen, R.; Vestal, S.; Cornhill, D.; Lewis, B. Using an architecture description language for quantitative analysis of real-time systems. In: *Proceedings of the 3rd International Workshop on Software and Performance (WOSP'02)*, Rome, Italy, 2002, p. 203–210.

Alvarez, B.; Iborra, A.; Alonzo, A.; De la Puente, J. A. Reference architecture for robot teleoperation: Development details and practical use. *Control Engineering Practice*, v. 9, n. 4, p. 395–402, 2001.

Ambroszkiewicz, S.; Bartyna, W.; Faderewski, M.; Terlikowski, G. Multirobot system architecture: environment representation and protocols. *Bulletin of the Polish Academy of Sciences-Technical Sciences*, v. 58, n. 1, p. 3–13, 2010.

Ambroszkiewicz, S.; Bartyna, W.; Faderewski, M.; Terlikowski, G.; Cetnarowicz, K. *Interoperability in open heterogeneous multirobot systems.* Technical report, Association for the Advancement of Artificial Intelligence (AAAI), Arlington, USA, 2007.

Amoretti, M.; Zanichelli, F.; Conte, G. A service-oriented approach for building autonomic peer-to-peer robot systems. In: *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'07)*, Paris, France, 2007, p. 137–142.

Angelov, S.; Grefen, P.; Greefhorst, D. A framework for analysis and design of software reference architectures. *Information and Software Technology*, v. 54, n. 4, p. 417 – 431, 2012.

Angelov, S.; Grefen, P. W. P. J.; Greefhorst, D. A classification of software reference architectures: Analyzing their success and effectiveness. In: *Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA'09)*, Cambridge, UK, 2009, p. 141–150.

Angelov, S.; Trienekens, J. J.; Grefen, P. Towards a method for the evaluation of reference architectures: Experiences from a case. In: *Proceedings of the 2nd European Conference on Software Architecture (ECSA'08)*, Paphos, Cyprus: Springer-Verlag, 2008, p. 225–240 (LNCS v.5292).

Apache ServiceMix. Online, `http://servicemix.apache.org/home.html` - Accessed in January 8th 2015, 2015.

Arkin, R. C. *Behavior-based robotics.* The MIT Press, 1998.

Arsanjani, A.; Ghosh, S.; Allam, A.; Abdollah, T.; Gariapathy, S.; Holley, K. SOMA: A method for developing service-oriented solutions. *IBM Systems Journal*, v. 47, n. 3, p. 377–396, 2008.

Arsanjani, A.; Zhang, L.-J.; Ellis, M.; Allam, A.; Channabasavaiah, K. S3: A service-oriented reference architecture. *IT Professional*, v. 9, n. 3, p. 10–17, 2007.

Arumugam, R.; Enti, V. R.; Bingbing, L.; Xiaojun, W.; Baskaran, K.; Kong, F. F.; Kumar, A.; Meng, K. D.; Kit, G. W. DAvinCi: A cloud computing framework for service robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'10)*, Anchorage, Alaska, 2010, p. 3084–3089.

AUTOSAR AUTOSAR (AUTomotive Open System ARchitecture). Online, `http://www.autosar.org/` - Accessed in January 4th 2015, 2015.

Awaad, I.; Hartanto, R.; Leon, B.; Ploger, P. A software system for robotic learning by experimentation. In: *Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'08)*, Venice, Italy: Springer-Verlag, 2008, p. 99–110 (LNCS v. 5325).

Bajracharya, M.; Maimone, M. W.; Helmick, D. Autonomy for mars rovers: Past, present, and future. *IEEE Computer*, v. 41, p. 44–50, 2008.

Barbosa, M.; Bernardino, A.; Figueira, D.; Gaspar, J.; Gonçalves, N.; Lima, P. U.; Moreno, P.; Pahliani, A.; Santos-Victor, J.; Spaan, M. T. J.; Sequeira, J. ISROBOT-NET: A testbed for sensor and robot network systems. In: *Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*, St. Louis, USA, 2009, p. 2827–2833.

Basili, V. R.; Shull, F.; Lanubile, F. Building knowledge through families of experiments. *IEEE Transaction on Software Engineering*, v. 25, n. 4, p. 456–473, 1999.

Bass, L.; Clements, P.; Kazman, R. *Software architecture in practice.* SEI Series in Software Engineering, 3 ed. Addison-Wesley, 2012.

Batory, D.; Coglianese, L.; Goodwin, M.; Shafer, S. Creating reference architectures: an example from avionics. In: *Proceedings of the 1st ACM SIGSOFT Symposium on Software Reusability (SSR'95)*, Seattle, USA, 1995, p. 27–37.

Bekey, G. A. *Autonomous robots: From biological inspiration to implementation and control.* Cambridge, England: The MIT Press, 2005.

Bengtsson, P.; Lassing, N.; Bosch, J.; van Vliet, H. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, v. 69, n. 1-2, p. 129–147, 2004.

Berná-Martínez, J. V.; Maciá-Pérez, F. Model of integration and management for robotic functional components inspired by the human neuroregulatory system. In: *Proceedings of the 15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'10)*, Bilbao, Spain, 2010, p. 1–4.

Berná-Martínez, J. V.; Maciá-Pérez, F.; Gilart-Iglesias, V.; Marcos-Jorquera, D. Robotic architecture based on electronic business models - from physics components to smart services. In: *Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics (ICINCO'06)*, Setúbal, Portugal, 2006a, p. 544–547.

Berná-Martínez, J. V.; Maciá-Pérez, F.; Ramos-Morillo, H.; Gilart-Iglesias, V. Distributed robotic architecture based on smart services. In: *Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN'06)*, Singapore, 2006b, p. 480–485.

Blake, M. B.; Remy, S. L.; Wei, Y.; Howard, A. M. Robots on the web. *IEEE Robotics & Automation Magazine*, v. 18, n. 2, p. 33–43, 2011.

Boehm, B. W.; Brown, J. R.; Lipow, M. Quantitative evaluation of software quality. In: *Proceedings of the 2nd International Conference on Software Engineering (ICSE'76)*, San Francisco, USA, 1976, p. 592–605.

Borenstein, J.; Koren, Y. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, v. 19, n. 5, p. 1179–1187, 1989.

Bosch, J. *Design and use of software architectures: Adopting and evolving a product-line approach*. 1 ed. New York, NY, USA: Addison-Wesley, 2000.

Boston Dynamics BigDog - The Most Advanced Rough-Terrain Robot on Earth. Online, `http://www.bostondynamics.com/` - Accessed in January 4th 2015, 2015.

Braga, R. T. V.; Branco, K. R. L. J. C.; Trindade Junior, O.; Masiero, P. C. ProLiCES: An Approach to Develop Product Lines for Safety-Critical Embedded Systems. In: *Proceedings of the 37th Latin-American Informatics Conference (CLEI'11)*, Quito, Equador, 2011, p. 991–1006.

Braga, R. T. V.; Trindade, Jr., O.; Branco, K. R. L. J. C.; Lee, J. Incorporating certification in feature modelling of an unmanned aerial vehicle product line. In: *Proceedings of the 16th International Software Product Line Conference (SPLC'12)*, Salvador, Brazil, 2012a, p. 249–258.

Braga, R. T. V.; Trindade Junior, O.; Branco, K. R. L. J. C.; Neris, L. O.; Lee, J. Adapting a software product line engineering process for certifying safety critical embedded systems. In: *Proceedings of the 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP'12)*, Magdeburg, Germany, 2012b, p. 352–363.

BRICS Best practice in robotics. Online, `http://www.best-of-robotics.org/` - Accessed in January 10th 2015, 2015.

Brooks, A.; Kaupp, T.; Makarenko, A.; Williams, S.; Orebäck, A. Towards component-based robotics. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, Alberta, Canada, 2005, p. 163–168.

Brugali, D.; Da Fonseca, A.; Luzzana, A.; Maccarana, Y. Developing service oriented robot control system. In: *Proceedings of the 8th International Symposium on Service Oriented System Engineering (SOSE'14)*, Oxford, UK, 2014, p. 237–242.

Brugali, D.; Gherardi, L.; Biziak, A.; Luzzana, A.; Zakharov, A. A reuse-oriented development process for component-based robotic systems. In: *Proceedings of the 4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'12)*, Tsukuba, Japan, 2012a, p. 361–374.

Brugali, D.; Gherardi, L.; Klotzbücher, M.; Bruyninckx, H. Service Component Architectures in Robotics: The SCA-Orocos Integration. In: Hähnle, R.; Knoop, J.; Margaria, T.; Schreiner, D.; Steffen, B., eds. *Leveraging Applications of Formal Methods, Verification, and Validation*, Communications in Computer and Information Science, Springer-Heidelberg, p. 46–60, 2012b.

Brugali, D.; Scandurra, P. Component-based robotic engineering (Part I). *IEEE Robotics Automation Magazine*, v. 16, n. 4, p. 84–96, 2009.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture: A system of patterns.* John Wiley & Sons, Inc., 1996.

Cai, Y.; Tang, Z.; Zhao, C. New layered SOA-Based architecture for multi-robots cooperative online SLAM. *Chinese Journal of Electronics*, v. 23, n. 1, p. 25–30, 2014.

Canfora, G.; Di Penta, M.; Esposito, R.; Villani, M. L. An approach for QoS-aware service composition based on genetic algorithms. In: *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO'05)*, Washington DC, USA, 2005, p. 1069–1075.

Canfora, G.; Di Penta, M.; Esposito, R.; Villani, M. L. A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software*, v. 81, n. 10, p. 1754–1769, 2008.

Carvalho, F.; Meira, S. Towards an embedded software component quality verification framework. In: *Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'09)*, Potsdam, Germany, 2009, p. 248–257.

Cepeda, J.; Soto, R.; Gordillo, J.; Chaimowicz, L. Towards a service-oriented architecture for teams of heterogeneous autonomous robots. In: *Proceedings of the 10th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence and Applications (MICAI'11)*, Puebla, Mexico, 2011, p. 102–108.

Cepeda, J. S.; Chaimowicz, L.; Soto, R. Exploring Microsoft Robotics Studio as a mechanism for service-oriented robotics. In: *Proceedings of the 7th Latin American Robotics Symposium and Intelligent Robotic Meeting (LARS'10)*, São Bernardo do Campo, Brazil, 2010, p. 7 –12.

Cesetti, A.; Scotti, C. P.; Buo, G. D.; Longhi, S. A service oriented architecture supporting an autonomous mobile robot for industrial applications. In: *Proceedings of the 18th Mediterranean Conference on Control Automation (MED'10)*, Marrakech, Morocco, 2010, p. 604–609.

Chen, Y.; Abhyankar, S.; Xu, L.; Tsai, W. T.; García-Acosta, M. Developing a security robot in service-oriented architecture. In: *Proceedings of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'08)*, Kunming, China, 2008, p. 106–111.

Chen, Y.; Bai, X. On robotics applications in service-oriented architecture. In: *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops (ICDCS'08)*, Beijing, China, 2008, p. 551–556.

Chen, Y.; Du, Z.; García-Acosta, M. Robot as a service in cloud computing. In: *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE'10)*, Nanjing, China, 2010, p. 151–158.

Chen, Y.; Sabnis, A.; García-Acosta, M. Design and performance evaluation of a service-oriented robotics application. In: *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS'09)*, Montreal, Canada, 2009, p. 292–299.

Clark, M. N. JAUS compliant systems offers interoperability across multiple and diverse robot platforms. In: *Proceedings of the AUVSI's Symposium Unmanned Systems North America (AUVSI'05)*, Baltimore, USA, 2005, p. 249–255.

Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. *Documenting software architectures: Views and beyond.* The SEI Series in Software Engineering. Addison-Wesley, 2010.

Clements, P.; Kazman, R.; Klein, M. *Evaluating software architectures: Methods and case studies.* The SEI Series in Software Engineering. Boston, MA: Addison-Wesley, 2002.

Clements, P.; Northrop, L. *Software product lines: Practices and patterns.* Boston, MA: Addison-Wesley, 2002.

Clements, P. C. A survey of architecture description languages. In: *Proceedings of the 8th International Workshop on Software Specification and Design (IWSSD '96)*, Schloss Velen, Germany, 1996, p. 16–25.

Cloutier, R.; Muller, G.; Verma, D.; Nilchiani, R.; Hole, E.; Bone, M. The concept of reference architectures. *Systems Engineering*, v. 13, n. 1, p. 14–27, 2010.

Coelho, P. R.; Sassi, R. F.; Cardozo, E.; Guimarães, E. G.; Faina, L. F.; Lima, A. Z.; Pinto, R. P. A web lab for mobile robotics education. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'07)*, Roma, Italy, 2007, p. 1381–1386.

Costagliola, G.; Ferrucci, F.; Fuccella, V. SCORM run-time environment as a service. In: *Proceedings of the 6th International Conference on Web engineering (ICWE'06)*, Palo Alto, USA, 2006, p. 103–110.

Dagli, C. H.; Kilicay-Ergin, N. *System of systems architecting* John Wiley & Sons, p. 77–100, 2008.

Dan, A.; Davis, D.; Kearney, R.; Keller, A.; King, R.; Kuebler, D.; Ludwig, H.; Polan, M.; Spreitzer, M.; Youssef, A. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, v. 43, n. 1, p. 136–158, 2004.

Desset, S.; Damus, R.; Hover, F.; Morash, J.; Polidoro, V. Closer to deep underwater science with ODYSSEY IV class Hovering Autonomous Underwater Vehicle (HAUV). In: *Proceedings of the 2005 IEEE Oceans – Europe (OCEANS'05)*, Brest, France, 2005, p. 758–762.

Diankov, R. *Automated construction of robotic manipulation programs.* PhD Thesis, Carnegie Mellon University, Robotics Institute, 2010.

Dillon, T. S.; Wu, C.; Chang, E. Reference architectural styles for service-oriented computing. In: *Proceedings of the 4th International Conference of Network and Parallel Computing (NPC'07)*, Dalian, China: Springer-Verlag, 2007, p. 543–555 (LNCS v. 4672).

Dobrica, L.; Niemela, E. An approach to reference architecture design for different domains of embedded systems. In: *Proceedings of the 8th International Conference on Software Engineering Research and Practice (SERP'08)*, Las Vegas, USA, 2008, p. 287–293.

Doriya, R.; Chakraborty, P.; Nandi, G. Robot-cloud: A framework to assist heterogeneous low cost robots. In: *Proceedings of the 1st International Conference on Communication, Information Computing Technology (ICCICT'12)*, Mumbai, India, 2012a, p. 1–5.

Doriya, R.; Chakraborty, P.; Nandi, G. Robotic services in cloud computing paradigm. In: *Proceedings of the 1st International Symposium on Cloud and Services Computing (ISCOS'12)*, Mangalore, India, 2012b, p. 80–83.

Du, Z.; Yang, W.; Chen, Y.; Sun, X.; Wang, X.; Xu, C. Design of a robot cloud center. In: *Proceedings of the 10th International Symposium on Autonomous Decentralized Systems (ISADS'11)*, Kobe, Japan, 2011, p. 269 –275.

Dudek, G.; Jenkin, M. *Computational principles of mobile robotics.* 2 ed. New York, USA: Cambridge University Press, 2010.

Ebenhofer, G.; Bauer, H.; Plasch, M.; Zambal, S.; Akkaladevi, S.; Pichler, A. A system integration approach for service-oriented robotics. In: *Proceedings of the 1st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'13)*, Cagliari, Italy, 2013, p. 1–8.

Edwards, R.; Parker, L. E.; Resseguie, D. R. Robopedia: Leveraging sensorpedia for web-enabled robot control. In: *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM'10)*, Mannheim, Germany, 2010, p. 183–188.

Eeles, P. Understanding architectural assets. In: *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA'08)*, Vancouver, Canada, 2008, p. 267–270.

Eisenhauer, M.; Rosengren, P.; Antolin, P. A development platform for integrating wireless devices and sensors into ambient intelligence systems. In: *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON'09)*, Rome, Italy, 2009, p. 1–3.

Eklund, U.; Örjan Askerdal; Granholm, J.; Alminger, A.; Axelsson, J. Experience of introducing reference architectures in the development of automotive electronic systems. *ACM SIGSOFT Software Engineering Notes*, v. 30, n. 4, p. 1–6, 2005.

Eklund, U.; Bosch, J. Architecture for embedded open software ecosystems. *Journal of Systems and Software*, v. 92, n. 1, p. 128–142, 2014.

Erl, T. *Service-oriented architecture: Concepts, technology, and design.* Upper Saddle River, New Jersey, USA: Prentice Hall., 2005.

Erl, T. *SOA principles of service design.* Upper Saddle River, New Jersey, USA: Prentice Hall., 2007.

Feitosa, D.; Nakagawa, E. Y. An investigation into reference architectures for mobile robotic systems. In: *Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA'12)*, Lisbon, Portugal, 2012, p. 465–471.

Fernandes, L. C.; Souza, J. R.; Pessin, G.; Shinzato, P. Y.; Sales, D.; Mendes, C.; Prado, M.; Klaser, R.; Magalhães, A. C.; Hata, A.; Pigatto, D.; Branco, K. C.; Grassi Jr., V.; Osório, F. S.; Wolf, D. F. CaRINA Intelligent Robotic Car: Architectural design and applications. *Journal of Systems Architecture*, v. 60, n. 4, p. 372–392, 2014.

Fielding, R. *Architectural styles and the design of network-based software architectures.* PhD Thesis, University of California, Irvine, Califonia, USA, 2000.

Fluckiger, L.; Utz, H. Service oriented robotic architecture for space robotics: Design, testing, and lessons learned. *Journal of Field Robotics*, v. 31, n. 1, p. 176–191, 2014.

Fox, D.; Burgard, W.; Dellaert, F.; Thrun, S. Monte carlo localization: Efficient position estimation for mobile robots. In: *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, Orlando, USA, 1999, p. 1–7.

Fox, D.; Burgard, W.; Thrun, S. Markov localization for reliable robot navigation and people detection. In: *Selected Papers from the International Workshop on Sensor Based Intelligent Robots*, London, UK: Springer-Verlag, 1998, p. 1–20.

Frenken, T.; Spiess, P.; Anke, J. A flexible and extensible architecture for device-level service deployment. In: *Proceedings of the 1st European Conference on Towards a*

Service-Based Internet (ServiceWave'08), Madrid, Spain: Springer-Verlag, 2008, p. 230–241 (LNCS v. 5377).

Fryer, J. A.; McKee, G. T.; Schenker, P. S. Configuring robots from modules: An object oriented approach. In: *Proceedings of the 8th International Conference on Advanced Robotics (ICAR'97)*, Monterey, USA, 1997, p. 907–912.

Galster, M.; Avgeriou, P. Empirically-grounded reference architectures: A proposal. In: *Proceedings of the 7th ACM SIGSOFT International Conference on Quality of Software Architectures (QoSA'11) and the 2nd ACM SIGSOFT International Symposium on Architecting Critical Systems (ISARCS'11)*, Boulder, Colorado, USA, 2011, p. 153–158.

Galster, M.; Eberlein, A.; Moussavi, M. Early assessment of software architecture qualities. In: *Proceedings of the 2nd International Conference on Research Challenges in Information Science (RCIS'08)*, Marrakech, Marroco, 2008, p. 81–86.

Gane, C.; Sarson, T. *Structured systems analysis: Tools and techniques.* Englewood Cliffs, USA: McDonnell Douglas Systems Integration Co., 1977.

Gerkey, B. P.; Vaughan, R. T.; Howard, A. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In: *Proceedings of the 11th International Conference on Advanced Robotics (ICAR'03)*, Coimbra, Portugal, 2003, p. 317–323.

Gill, A. *Introduction to the theory of finite-state machines.* New York: McGraw-Hill, 1962.

Gomaa, H. *Designing software product lines with UML: From use cases to pattern-based software architectures.* Object Technology Series. Addison-Wesley, 2004.

Graaf, B.; van Dijk, H.; van Deursen, A. Evaluating an embedded software reference architecture – industrial experience report. In: *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, Manchester, UK, 2005, p. 354–363.

Graciano Neto, V. V.; Garcés-Rodríguez, L. M.; Guessi, M.; Oliveira, L. B. R.; Oquendo, F. On the equivalence between reference architectures and metamodels. In: *Proceedings of the 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA'15) – held in conjunction with the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA'15)*, Montreal, Canada, 2015, p. 1–4.

Graciano Neto, V. V.; Guessi, M.; Oliveira, L. B. R.; Oquendo, F.; Nakagawa, E. Y. Investigating the model-driven development for systems-of-systems. In: *Proceedings of the 2nd International Workshop on Software Engineering for Systems-of-Systems (SESoS'14) – held in conjunction with the 8th European Conference on Software Architecture (ECSA'14)*, Vienna, Austria, 2014, p. 1–8.

Graves, A.; Czarnecki, C. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, v. 30, n. 1, p. 36–41, 2000.

Guessi, M.; Cavalcante, E.; Oliveira, L. B. R. Characterizing architecture description languages for software-intensive systems-of-systems. In: *Proceedings of the 3rd International Workshop on Software Engineering for Systems-of-Systems (SESoS'15) – held in conjunction with the 37th International Conference on Software Engineering (ICSE'15)*, Florence, Italy, 2015a, p. 1–8.

Guessi, M.; Nakagawa, E. Y.; Oquendo, F.; Maldonado, J. C. Architectural description of embedded systems: a systematic review. In: *Proceedings of the 3rd International Symposium on Architecting Critical Systems (ISARCS'12)*, Bertinoro, Italy, 2012, p. 31–40.

Guessi, M.; Oliveira, L. B. R.; Garcés-Rodríguez, L. M.; Oquendo, F. Towards a formal description of reference architectures for embedded systems. In: *Proceedings of the 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA'15) – held in conjunction with the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA'15)*, Montréal, Canada, 2015b, p. 1–5.

Guessi, M.; de Oliveira, L. B. R.; Nakagawa, E. Y. Representation of reference architectures: A systematic review. In: *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE'11)*, Miami Beach, USA, 2011, p. 782–785.

Guinard, D.; Trifa, V. Towards the web of things: Web mashups for embedded devices. In: *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, Madrid, Spain, 2009, p. 1–8.

Guinard, D.; Trifa, V.; Karnouskos, S.; Spiess, P.; Savio, D. Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, v. 3, n. 3, p. 223 –235, 2010.

Ha, Y.-G.; Sohn, J.-C.; Cho, Y.-J. Service-oriented integration of networked robots with ubiquitous sensors and devices using the semantic web services technology. In: *Proceedings of the 18th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, Alberta, Canada, 2005, p. 3947–3952.

Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. 3, p. 231–274, 1987.

Hart, P. E.; Nilsson, N. J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics Computing*, v. 2, n. 4, p. 100–107, 1968.

Hayes-Roth, B.; Pfleger, K.; Lalanda, P.; Morignot, P.; Balabanovic, M. A domain-specific software architecture for adaptive intelligent systems. *IEEE Transactions on Software Engineering*, v. 21, n. 4, p. 288–301, 1995.

van Heesch, U.; Eloranta, V. P.; Avgeriou, P.; Koskimies, K.; Harrison, N. Decision-centric architecture reviews. *IEEE Software*, v. 31, n. 1, p. 69–76, 2014.

Heisey, C.; Hendrickson, A.; Chludzinski, B.; Cole, R.; Ford, M.; Herbek, L.; Ljungberg, M.; Magdum, Z.; Marquis, D.; Mezhirov, A.; Pennell, J.; Roe, T.; Weinert, A. A reference software architecture to support unmanned aircraft integration in the national airspace system. *Journal of Intelligent and Robotic Systems: Theory and Applications*, v. 69, n. 1-4, p. 41–55, 2013.

Hestand, P. D. *A service oriented architecture for robotic platforms*. PhD Thesis, 2011.

Hofmeister, C.; Kruchten, P.; Nord, R. L.; Obbink, H.; Ran, A.; America, P. A general model of software architecture design derived from five industrial approaches. *Journal of System and Software*, v. 80, n. 1, p. 106–126, 2007.

Hongxing, W.; Shiyi, L.; Ying, Z.; Liang, Y.; Tianmiao, W. A middleware based control architecture for modular robot systems. In: *Proceedings of the 4th IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications (MESA'08)*, Beijing, China, 2008, p. 327–332.

Huergo, R. S.; Pires, P. F.; Delicato, F. C.; Costa, B.; Cavalcante, E.; Batista, T. A systematic survey of service identification methods. *Service Oriented Computing and Applications*, v. 8, n. 3, p. 199–219, 2014.

Husqvarna Husqvarna Automower: Robotic lawn mower. Online, `http://www.husqvarna.com/us/products/robotic-mowers/` - Accessed in January 4th 2015, 2015.

Iborra, A.; Caceres, D.; Ortiz, F.; Franco, J.; Palma, P.; Alvarez, B. Design of service robots. *IEEE Robotics Automation Magazine*, v. 16, n. 1, p. 24–33, 2009.

IEEE *Standard Ontologies for Robotics and Automation (IEEE 1872-2015).* Standard 1872-2015, IEEE/ORA, 2015.

Insaurralde, C.; Cartwright, J.; Petillot, Y. Cognitive control architecture for autonomous marine vehicles. In: *Proceedings of the 6th IEEE International Systems Conference (SysCon'12)*, Vancouver, Canada, 2012, p. 117–124.

Insaurralde, C.; Petillot, Y. Intelligent autonomy for collaborative intervention missions of unmanned maritime vehicles. In: *Proceedings of the 2013 MTS/IEEE OCEANS*, San Diego, USA, 2013, p. 1–6.

Insaurralde, C. C.; Petillot, Y. R. Capability-oriented robot architecture for maritime autonomy. *Robotics and Autonomous Systems*, v. 67, p. 87–104, 2015.

iRobots iRobot Roomba Vacuum Cleaning Robot. Online, `http://www.irobot.com/us/learn/home/roomba.aspx` - Accessed in January 20th 2015, 2015a.

iRobots iRobot Scooba: Floor Washing Robot. Online, `http://www.irobot.com/For-the-Home/Floor-Scrubbing/Scooba.aspx` - Accessed in January 20th 2015, 2015b.

ISO *Quality Vocabulary (ISO 8402, Part of the ISO 9000/2002).* Standard 8402/2002, International Organization for Standardization (ISO), 2002.

ISO *Software Compliance: Achieving Functional Safety in the Automotive Industry (ISO 26262).* Standard 26262/2011, ISO, 2011.

ISO/IEC *Information technology – Software product evaluation – Part 1: General overview (ISO/IEC 14598-1).* Standard 14598/1999, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC), 1999.

ISO/IEC *Software engineering – Product quality – Part 1: Quality model (ISO/IEC 9126-1).* Standard 9126/2001, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC), 2001.

ISO/IEC *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* Tech Report 25010/2011, ISO/IEC, 2011.

ISO/IEC/IEEE *Recommended Practice for Architectural Description of Software-Intensive Systems (ISO/IEC/IEEE 42010).* Standard 42010/2011, International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE), 2011.

Iyengar, S. S.; Elfes, A. *Autonomous mobile robots: Control, planning, and architecture*, v. 2. IEEE Computer Society Press, 1991.

Jackson, J. Microsoft Robotics Studio: A technical introduction. *IEEE Robotics & Automation Magazine*, v. 14, n. 4, p. 82–87, 2007.

Jansen, A.; Bosch, J. Software architecture as a set of architectural design decisions. In: *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, Pittsburgh, Pennsylvania, USA, 2005, p. 109–120.

JBoss Community JBoss ESB. Online, `http://www.jboss.org/jbossesb/` - Accessed in January 8th 2015, 2015.

Josuttis, N. M. *SOA in practice: The art of distributed systems design.* O'Reilly, 2007.

Kazhamiakin, R.; Pistore, M.; Santuari, L. Analysis of communication models in web service compositions. In: *Proceedings of the 15th International Conference on World Wide Web (WWW'06)*, Edinburgh, Scotland, 2006, p. 267–276.

Kazman, R.; Bass, L.; Abowd, G.; Webb, M. SAAM: A method for analyzing the properties of software architectures. In: *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, Sorrento, Italy, 1994, p. 81–90.

Kazman, R.; Klein, M.; Barbacci, M.; Longstaff, T.; Lipson, H.; Carriere, J. The architecture tradeoff analysis method. In: *Proceedings of the 4th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98)*, Monterey, CA, USA, 1998, p. 68–78.

Kim, B. K.; Miyazaki, M.; Ohba, K.; Hirai, S.; Tanie, K. Web services based robot control platform for ubiquitous functions. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'05)*, Barcelona, Spain, 2005, p. 691–696.

Kim, B. K.; Tomokuni, N.; Ohara, K.; Ohba, K.; Tanikawa, T.; Hirai, S. Ubiquitous function services based control for robots with ambient intelligence. In: *Proceedings of the 32nd IEEE Annual Conference of Industrial Electronics (IECON'06)*, Paris, France, 2006, p. 4546 –4551.

Kim, T.; Ko, I. Y.; Kang, S. W.; Lee, D. H. Extending ATAM to assess product line architecture. In: *Proceedings of the 8th IEEE International Conference on Computer and Information Technology (CIT'08)*, Sydney, Australia, 2008, p. 790–797.

Kitchenham, B. *Procedures for performing systematic reviews.* Technical Report TR/SE-0401, Software Engineering Group, Department of Computer Science Keele University, United King and Empirical Software Engineering, National ICT Australia Ltd, Australia, 2004.

Kononchuk, D.; Kandoba, V.; Zhigalov, S.; Abduramanov, P.; Okulovsky, Y. RoboCoP: A protocol for service-oriented robot control systems. In: *Proceedings of the 4th International Conference on Research and Education in Robotics (EUROBOT'11)*, Prague, Czech Republic: Springer-Heidelberg, 2011, p. 158–171 (CCIS v.161).

Koubaa, A. A service-oriented architecture for virtualizing robots in robot-as-a-service clouds. In: *Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS'14)*, Luebeck, Germany, 2014, p. 196–208 (LNCS v. 8350.

Kozen, D. Results on the propositional μ-calculus. *Theoretical Computer Science*, v. 27, n. 3, p. 333–354, 1983.

Kruchten, P. Documentation of software architecture from a knowledge management perspective – design representation. In: Ali Babar, M.; Dingsøyr, T.; Lago, P.; van Vliet, H., eds. *Software Architecture Knowledge Management*, Springer-Heidelberg, p. 39–57, 2009.

Kruchten, P.; Obbink, H.; Stafford, J. The past, present, and future for software architecture. *IEEE Software*, v. 23, n. 2, p. 22–30, 2006.

Land, R. *A brief survey of software architecture.* Technical report, Department of Computer Engineering, Mälardalen University, Sweden, 2002.

Lee, J.; Kim, J.; Lee, B. Semantic and dynamic web service of SOA based smart robots using web 2.0 OpenAPI. In: *Proceedings of the 6th International Conference on Software Engineering Research, Management and Applications (SERA'08)*, Prague, Czech Republic, 2008a, p. 255–260.

Lee, J.; Kim, J.; Lee, B.; Wu, C. Utilizing semantic web 2.0 for self-reconfiguration of SOA based agent applications in intelligent service robots. In: *Proceedings of the 8th IEEE International Conference on Computer and Information Technology (CIT'08)*, Sydney, Australia, 2008b, p. 784–789.

Lee, J.; Kim, J.; Lee, C.; Lee, B. Agent based dynamic adaptation of intelligent robots using enterprise service bus. In: *Proceedings of the 4th International Conference on Information Science and Security (ICISS'08)*, Hyderabad, India, 2008c, p. 94 –97.

Lee, K. K.; Zhang, P.; Xu, Y. A service-based network architecture for wearable robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03)*, Taipei, Taiwan, 2003, p. 1671–1676.

Lee, K. K.; Zhang, P.; Xu, Y.; Liang, B. An intelligent service-based network architecture for wearable robots. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 34, n. 4, p. 1874–1885, 2004.

Lee, W.-P.; Yang, T.-H. Using knowledge ontologies and neural networks to control service-oriented robots. In: *Proceedings of the World Congress on Engineering (WCE'13)*, London, UK, 2013, p. 1–6.

LEGO Lego mindstorm. Online, `http://mindstorms.lego.com/` - Accessed in January 27th 2015, 2015.

Leonard, J.; Durrant-Whyte, H. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, v. 7, n. 3, p. 376–382, 1991a.

Leonard, J.; Durrant-Whyte, H. Simultaneous map building and localization for an autonomous mobile robot. In: *Proceedings of the 4th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'91)*, Osaka, Japan, 1991b, p. 1442–1447.

Lindemuth, M.; Murphy, R.; Steimle, E.; Armitage, W.; Dreger, K.; Elliot, T.; Hall, M.; Kalyadin, D.; Kramer, J.; Palankar, M.; Pratt, K.; Griffin, C. Sea robot-assisted inspection. *IEEE Robotics & Automation Magazine*, v. 18, n. 2, p. 96 –107, 2011.

LogCoy The Open Service Bus - OpenESB. Online, `http://www.open-esb.net/` - Accessed in January 8th 2015, 2015.

Luckham, D. C. Rapide: a language and toolset for simulation of distributed systems by partial orderings of events. In: *Proceedings of the DIMACS workshop on Partial order methods in verification (POMIV'96)*, Princeton, USA, 1997, p. 329–357.

Majedi, M.; Osman, K.; Boyd, M. A generic service oriented architectural model for pervasive applications: A case study in internet-based multiple robot control. In: *Proceedings of the 3rd International Conference on Pervasive Computing and Applications (ICPCA'08)*, Alexandria, Egypt, 2008, p. 54 –59.

Mataric, M. Behavior-based control: Main properties and implications. In: *Proceedings of the Workshop on Architectures for Intelligent Control Systems (WAIC'92), IEEE International Conference on Robotics and Automation (ICRA'92)*, Nice, France, 1992, p. 46–54.

Mateescu, R.; Oquendo, F. π-AAL: an architecture analysis language for formally specifying and verifying structural and behavioural properties of software architectures. *ACM SIGSOFT Software Engineering Notes*, v. 31, n. 2, p. 1–19, 2006.

Matta-Gomez, A.; Cerro, J. D.; Barrientos, A. Multi-robot data mapping simulation by using microsoft robotics developer studio. *Simulation Modelling Practice and Theory*, v. 49, p. 305–319, 2014.

McCall, J.; Richards, P.; Walters, G. *Factors in software quality*, v. 1-3. The National Technical Information Service (NTIS), 1977.

Medvidovic, N.; Taylor, R. N. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, v. 26, n. 1, p. 70–93, 2000.

Mens, T.; Magee, J.; Rumpe, B. Evolving software architecture descriptions of critical systems. *Computer*, v. 43, n. 5, p. 42–48, 2010.

Mens, T.; Tourwe, T. A survey of software refactoring. *IEEE Transactions on Software Engineering*, v. 30, n. 2, p. 126–139, 2004.

Metta, G.; Fitzpatrick, P.; Natale, L. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, v. 3, n. 1, p. 043–048, 2006.

Milner, R. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.

Mohagheghi, P.; Conradi, R. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, v. 12, n. 5, p. 471–516, 2007.

Mokarizadeh, S.; Grosso, A.; Matskin, M.; Kungas, P.; Haseeb, A. Applying semantic web service composition for action planning in multi-robot systems. In: *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW'09)*, Venice, Italy, 2009, p. 370–376.

Muhammad, W.; Radziah, M.; Dayang, N. SOA4DERTS: A Service-Oriented UML profile for distributed embedded real-time systems. In: *Proceedings of the IEEE Symposium on Computers and Informatics (ISCI'12)*, Penang, Malaysia, 2012, p. 64–69.

Mule Soft MuleESB. Online, `http://www.mulesoft.org/what-mule-esb` - Accessed in January 8th 2015, 2015.

Muller, G. *A reference architecture primer.* Whitepaper, Eindhoven University of Technology, Eindhoven, 2008.

Murakami, E.; Saraiva, A. M.; Ribeiro, Junior, L. C. M.; Cugnasca, C. E.; Hirakawa, A. R.; Correa, P. L. P. An infrastructure for the development of distributed service-oriented information systems for precision agriculture. *Computers and Electronics in Agriculture*, v. 58, n. 1, p. 37–48, 2007.

Murphy, R. R. *Introduction to AI Robotics.* 1 ed. Cambridge, MA, USA: MIT Press, 2000.

Nakagawa, E. Y.; Antonino, P. O.; Becker, M. Reference architecture and product line architecture: A subtle but critical difference. In: *Proceedings of the 5th European Conference on Software Architecture (ECSA'11)*, Essen, Alemanha: Springer-Heidelberg, 2011a, p. 207–211 (LNCS v.6903).

Nakagawa, E. Y.; Ferrari, F. C.; Sasaki, M. M.; Maldonado, J. C. An aspect-oriented reference architecture for software engineering environments. *Journal of Systems and Software*, v. 84, n. 10, p. 1670 – 1684, 2011b.

Nakagawa, E. Y.; Gonçalves, M. B.; Guessi, M.; Oliveira, L. B. R.; Oquendo, F. The state of the art and future perspectives in systems of systems software architectures. In: *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-Systems (SESoS'13) – held in conjunction with the 7th European Conference on Software Architecture (ECSA'13)*, Montpellier, France, 2013, p. 13–18.

Nakagawa, E. Y.; Guessi, M.; Feitosa, D.; Maldonado, J. C. Consolidating a process for the design, representation, and evaluation of reference architectures. In: *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA'14)*, Sydney, Australia, 2014, p. 143–152.

Nakagawa, E. Y.; Oquendo, F. RAModel: A Reference Model for Reference Architectures. In: *Proceedings of the 10th Joint Working Conference on Software Architecture (WICSA'12) and 6th European Conference on Software Architecture (ECSA'12)*, Helsinki, Finland, 2012, p. 297–301.

Nakagawa, E. Y.; Simão, A. S.; Ferrari, F.; Maldonado, J. C. Towards a reference architecture for software testing tools. In: *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE'07)*, Boston, USA, 2007, p. 157–162.

Narita, M.; Shimamura, M.; Oya, M. Reliable protocol for robot communication on web services. In: *Proceedings of the 4th International Conference on Cyberworlds (CW'05)*, Singapore, 2005, p. 210–217.

OASIS *Reference model for service oriented architecture 1.0.* Technical report, Advanced Open Standards for the Information Society (OASIS), 2006.

OASIS *Reference architecture foundation for service oriented architecture version 1.0.* Technical report, Advanced Open Standards for the Information Society (OASIS), 2012.

OASIS – Advanced Open Standards for the Information Society (OASIS), Web Services Business Process Execution Language Version 2.0 (WS-BPEL). Online, `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html` - Accessed in January 13th 2015, 2015a.

OASIS – Advanced Open Standards for the Information Society (OASIS), Universal, Discovery, Description and Integration (UDDI). Online, `http://www.oasis-open.org/specs/#uddiv2` - Accessed in January 13th 2015, 2015b.

Oliveira, L. B. R.; Baldochi Jr., L. A.; Nakagawa, E. Y. Development of a complex system based on architectural and design patterns. In: *Proceedings of the 7th IADIS International Conference on Applied Computing (IADIS-AC'11)*, Rio de Janeiro, Brazil, 2011, p. 123–130.

Oliveira, L. B. R.; Felizardo, K. R.; Feitosa, D.; Nakagawa, E. Y. Reference models and reference architectures based on service-oriented architecture: A systematic review. In: *Proceedings of the 4th European Conference on Software Architecture (ECSA'10)*, Copenhague, Denmark: Springer-Heidelberg, 2010, p. 360–367 (LNCS v.6285).

Oliveira, L. B. R.; Guessi, M.; Feitosa, D.; Manteuffel, C.; Galster, M.; Oquendo, F.; Nakagawa, E. Y. An investigation on quality models and quality attributes for embedded systems. In: *Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA'13)*, Venice, Italy, 2013a, p. 523–528.

Oliveira, L. B. R.; Leroux, E.; Felizardo, K. R.; Oquendo, F.; Nakagawa, E. Y. Towards a process to design architectures of service-oriented robotic systems. In: *Proceedings* *of the 8th European Conference on Software Architecture (ECSA'14)*, Vienna, Austria, 2014a, p. 218–225 (LNCS v. 8627).

Oliveira, L. B. R.; Martins, D. B.; Amaral, F. A.; Oquendo, F.; Nakagawa, E. Y. Automating the discovery of services for service-oriented robotic systems. In: *Proceedings of the 11th IEEE Latin American Robotics Symposium (LARS'14)*, São Carlos, Brazil, 2014b, p. 151–156.

Oliveira, L. B. R.; Martins, D. B.; Amaral, F. A.; Oquendo, F.; Nakagawa, E. Y. RoboSeT: A Tool to Support Cataloging and Discovery of Services for Service-Oriented Robotic Systems. In: *Communications in Computer and Information Science (CCIS)*, Springer, (to appear), 2015, p. 1–25.

Oliveira, L. B. R.; Nakagawa, E. Y. A service-oriented reference architecture for the software testing domain. In: *Proceedings of the 5th European Conference on Software Architecture (ECSA'11)*, Essen, Alemanha: Springer-Heidelberg, 2011, p. 405–421 (LNCS v.6903).

Oliveira, L. B. R.; Osório, F. S.; Nakagawa, E. Y. *A systematic review on service-oriented robotic systems development.* Technical Report RT N. 373, Prof. Achilles Bassi Library, Institute of Mathematical and Computer Sciences (ICMC), University of São Paulo (USP), 2012.

Oliveira, L. B. R.; Osório, F. S.; Nakagawa, E. Y. An investigation into the development of service-oriented robotic systems. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing (ACM/SAC'13)*, Coimbra, Portugal, 2013b, p. 223–226.

Oliveira, L. B. R.; Osório, F. S.; Oquendo, F.; Nakagawa, E. Y. Towards a taxonomy of services for developing service-oriented robotic systems. In: *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE'14)*, Vancouver, Canada, 2014c, p. 344–349.

Oliveira, R. A. P.; Oliveira, L. B. R.; Cafeo, B. B. P.; Durelli, V. H. S. On using mutation testing for teaching programming to novice programmers. In: *Proceedings of the 22nd International Conference on Computers in Education (ICCE'14)*, Nara, Japan, 2014d, p. 394–396.

OMG Business Process Model and Notation, Version 2.0. Online, `http://www.bpmn.org/` - Accessed in February 11th 2015, 2015a.

OMG Service Oriented Architecture Modeling Language (SoaML) Version 1.3. Online, `http://www.omg.org/spec/SoaML/1.0.1` - Accessed in January 11th 2015, 2015b.

OMG Software and Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Online, `http://www.omg.org/spec/SPEM/2.0/` - Accessed in February 8th 2015, 2015c.

OMG The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. Online, `http://www.omgwiki.org/marte/` - Accessed in January 11th 2015, 2015d.

OMG Unified Modeling Language (UML). Online, `http://www.omg.org/spec/UML/2.4.1/` - Accessed in January 11th 2015, 2015e.

Oquendo, F. π-ADL: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Software Engineering Notes*, v. 29, n. 3, p. 15–28, 2004a.

Oquendo, F. π-ARL: an architecture refinement language for formally modelling the stepwise refinement of software architectures. *ACM SIGSOFT Software Engineering Notes*, v. 29, n. 5, p. 40–59, 2004b.

OROCOS Open robot control software - orocos. Online, `http://www.orocos.org/` - Accessed in January 27th 2015, 2015.

Ortiz, F.; Alonso, D.; Alvarez, B.; Pastor, J. A reference control architecture for service robots implemented on a climbing vehicle. In: *Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies (ADA'10)*, York, UK: Springer-Verlag, 2005, p. 13–24 (LNCS v. 3555).

Papazoglou, M. Service-oriented computing: concepts, characteristics and directions. In: *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE'03)*, Rome, Italy, 2003, p. 3–12.

Papazoglou, M. P.; Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, v. 16, n. 3, p. 389–415, 2007.

Papazoglou, M. P.; Traverso, P.; Dustdar, S.; Leymann, F. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, v. 17, n. 2, p. 223–255, 2008.

Pautasso, C.; Zimmermann, O.; Leymann, F. RESTful web services vs. "big" web services: making the right architectural decision. In: *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*, Beijing, China, 2008, p. 805–814.

Peltz, C. Web Services Orchestration and Choreography. *IEEE Computer*, v. 36, n. 10, p. 46–52, 2003.

Peristeras, V.; Fradinho, M.; Lee, D.; Prinz, W.; Ruland, R.; Iqbal, K.; Decker, S. CERA: A collaborative environment reference architecture for interoperable CWE systems. *Service Oriented Computing and Applications*, v. 3, n. 1, p. 3–23, 2009.

Peters, L.; Pauly, M.; Arghir, A. Servicebots - a scalable architecture for autonomous service robots. In: *Proceedings of the 9th IEEE International Conference on Fuzzy Systems (FUZZ'00)*, San Antonio, USA, 2000, p. 1013–1016.

Peterson, J. L. Petri nets. *ACM Computer Surveys*, v. 9, n. 3, p. 223–252, 1977.

Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; Thrun, S. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, v. 42, n. 3-4, p. 271–281, 2003.

Pinto, J.; Martins, R.; Sousa, J. B. Towards a REST-style architecture for networked vehicles and sensors. In: *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM'10)*, Mannheim, Germany, 2010, p. 745 –750.

Pires, R. M.; Rodrigues, D.; Branco, K. R. L. J. C. MOSA - Mission Oriented Sensor Arrays. In: *Anais do 37º Latin-American Informatics Conference (CLEI'11)*, Quito, Equador, 2011, p. 1–10.

Pressman, R. S. *Software engineering: A practitioner's approach.* 5th ed. McGraw-Hill Higher Education, 2001.

PROFACTOR Open Source for Distributed Industrial Automation - 4DIAC. Online, `http://www.fordiac.org/` - Accessed in January 27th 2015, 2015.

Pruter, S.; Golatowski, F.; Timmermann, D. Adaptation of resource-oriented service technologies for industrial informatics. In: *Proceedings of the 35th IEEE Annual Conference of Industrial Electronics (IECON'09)*, Porto, Portugal, 2009, p. 2399–2404.

Quintas, J.; Menezes, P.; Dias, J. Cloud robotics: Towards context aware robotic networks. In: *Proceedings of the 2nd IASTED International Conference on Robotics (Robo'11)*, Pittsburgh, USA, 2011, p. 420–427.

Raffaeli, R.; Cesetti, A.; Angione, G.; Lattanzi, L.; Longhi, S. Virtual planning for autonomous inspection of electromechanical products. *International Journal on Interactive Design and Manufacturing*, v. 6, n. 4, p. 215–231, 2012.

Rahman, M. A.; Miah, M. S.; Gueaieb, W.; Saddik, A. E. SENORA: A P2P service-oriented framework for collaborative multirobot sensor networks. *IEEE Sensors Journal*, v. 7, n. 5, p. 658–666, 2007.

Remy, S. L.; Blake, M. B. Distributed service-oriented robotics. *IEEE Internet Computing*, v. 15, n. 2, p. 70 –74, 2011.

RobotShop My robots. Online, `http://www.myrobots.com/` - Accessed in January 27th 2015, 2015.

Rodrigues, D.; de Melo Pires, R.; Estrella, J. C.; Marconato, E. A.; Trindade Jr., O.; Branco, K. R. L. J. C. Using SOA in Critical-Embedded Systems. In: *Proceedings of the 4th IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing (CPSCom'11)*, Dalian, China, 2011, p. 733–738.

Romero, R. A. F.; Prestes, E.; Osório, F.; Wolf, D. *Mobile robotics.* 1 ed. GEN LTC, (in Portuguese), 2014.

RTCA, I. *Final Report for Clarification of DO-178C – Software Considerations in Airborne Systems and Equipment Certification.* Tech Report DO-178C, RTCA, 2011.

Santos, J. F. M.; Guessi, M.; Galster, M.; Feitosa, D.; Nakagawa, E. A checklist for evaluation of reference architectures for embedded systems. In: *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering (SEKE'13)*, Boston, USA, 2013, p. 451–454.

Schlegel, C.; Steck, A.; Brugali, D.; Knoll, A. Design abstraction and processes in robotics: From code-driven to model-driven engineering. In: *Proceedings of the 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'10)*, Darmstadt, Germany, 2010, p. 324–335.

Scotti, C. P.; Cesetti, A.; Buo, G. D.; Longhi, S. Service oriented soft real-time implementation of SLAM capability for mobile robots. In: *Proceedings of the 7th IFAC Symposium on Intelligent Autonomous Vehicles (SIAV'10)*, Lecce, Italy, 2010, p. 545–550.

SEI CBAM: Cost Benefit Analysis Method. `http://www.sei.cmu.edu/architecture/tools/cbam/index.cfm` – Accessed in January 4th 2015, 2015a.

SEI Software Engineering Institute (SEI) – Catalog of Software Product Lines. Online, `http://www.sei.cmu.edu/productlines/casestudies/catalog/index.cfm` - Accessed in January 4th 2015, 2015b.

SEI Software Engineering Institute (SEI) – Software Architecture Glossary. Online, `http://www.sei.cmu.edu/architecture/start/glossary/` - Accessed in January 4th 2015, 2015c.

Shaw, M.; Clements, P. The golden age of software architecture. *IEEE Software*, v. 23, n. 2, p. 31–39, 2006.

Shaw, M.; Garlan, D. *Software architecture: Perspectives on an emerging discipline.* Prentice-Hall, 1996.

Sherman, T. Quality attributes for embedded systems. In: Sobh, T., ed. *Advances in Computer and Information Sciences and Engineering*, Springer-Netherlands, p. 536–539, 2008.

Siegwart, R.; Nourbakhsh, I. R. *Autonomous robots: From biological inspiration to implementation and control.* The MIT Press, 2004.

Straszheim, T.; Gerkey, B.; Cousins, S. The ROS build system. *IEEE Robotics & Automation Magazine*, v. 18, n. 2, p. 18–19, 2011.

Swisslog RoboCourier Autonomous Mobile Robot. Online, `http://www.swisslog.com/en/Products/HCS/Automated-Material-Transport/RoboCourier-Autonomous-Mobile-Robot` - Accessed in March 4th 2015, 2015.

SysML Partners Systems Modeling Language (SysML). Online, `http://www.sysml.org/` - Accessed in January 11th 2015, 2015.

Takahashi, M.; Suzuki, T.; Shitamoto, H.; Moriguchi, T.; Yoshida, K. Developing a mobile robot for transport applications in the hospital domain. *Robotics and Autonomous Systems*, v. 58, n. 7, p. 889–899, 2010.

Tekinerdogan, B. ASAAM: Aspectual software architecture analysis method. In: *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, Oslo, Norway, 2004, p. 5–14.

Tenorth, M.; Klank, U.; Pangercic, D.; Beetz, M. Web-enabled robots. *IEEE Robotics & Automation Magazine*, v. 18, n. 2, p. 58–68, 2011.

The Open Group Service Oriented Reference Architecture (SOA-RA). Online, `http://www.opengroup.org/soa/source-book/soa_refarch/` - Accessed in January 13th 2015, 2015.

Thrun, S.; Burgard, W.; Fox, D. *Probabilistic robotics.* Cambridge, UK: The MIT Press, 2005.

Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J.; Halpenny, M.; Hoffmann, G.; Lau, K.; Oakley, C.; Palatucci, M.; Pratt, V.; Stang, P.; Strohband, S.; Dupont, C.; Jendrossek, L.-E.; Koelen, C.; Markey, C.; Rummel, C.; van Niekerk, J.; Jensen, E.; Alessandrini, P.; Bradski, G.; Davies, B.; Ettinger, S.; Kaehler, A.; Nefian, A.; Mahoney, P. Stanley: The robot that won the DARPA Grand Challenge: Research Articles. *Journal of Robotic Systems*, v. 23, n. 9, p. 661–692, 2006.

Tikanmaki, A.; Roning, J. Property service architecture for distributed robotic and sensor systems. In: *Proceedings of the 4th International Conference on Informatics in Control, Automation and Robotics (ICINCO'07)*, Angers, France, 2007, p. 226–233.

Trifa, V. M.; Cianci, C. M.; Guinard, D. Dynamic control of a robotic swarm using a service-oriented architecture. In: *Proceedings of the 13th International Symposium on Artificial Life and Robotics (AROB'08)*, Oita, Japan, 2008, p. 119–122.

Tsai, W. T.; Huang, Q.; Sun, X. A collaborative service-oriented simulation framework with Microsoft Robotic Studio. In: *Proceedings of the 41st Annual Simulation Symposium (ANSS'08)*, Ottawa, Canada, 2008a, p. 263–270.

Tsai, W. T.; Sun, X.; Chen, Y.; Huang, Q.; Bitter, G.; White, M. Teaching service-oriented computing and STEM topics via robotic games. In: *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08)*, Orlando, USA, 2008b, p. 131 –137.

Tsai, W. T.; Sun, X.; Huang, Q.; Karatza, H. An ontology-based collaborative service-oriented simulation framework with Microsoft Robotics Studio. *Simulation Modelling Practice and Theory*, v. 16, n. 9, p. 1392–1414, 2008c.

UniversAAL Project The UniversAAL Reference Architecture. Online, `http://www.universaal.org/images/stories/deliverables/D1.3-B.pdf` - Accessed in January 4th 2015, 2015.

UPnP Forum Universal Plug and Play (UPnP). Online, `http://www.upnp.org/` - Accessed in January 27th 2015, 2015.

van de Molengraft, M. J. Robo Earth. Online, `http://www.roboearth.org/what-is-roboearth` - Accessed in January 27th 2015, 2015.

Veiga, G.; Pires, J. N.; Nilsson, K. On the use of service oriented software platforms for industrial robotic cells. In: *Proceedings of the 8th IFAC International Workshop Intelligent Manufacturing Systems (IMS'07)*, Alicante, Spain, 2007, p. 109 – 115.

Veiga, G.; Pires, J. N.; Nilsson, K. Experiments with service-oriented architectures for industrial robotic cells programming. *Robotics and Computer-Integrated Manufacturing*, v. 25, p. 746–755, 2009.

Vinoski, S. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, v. 35, n. 2, p. 46–55, 1997.

W3C – World Wide Web Consortium, Extensible Markup Language (XML). Online, `http://www.w3.org/XML/` - Accessed in January 13th 2015, 2015a.

W3C – World Wide Web Consortium, SOAP (Version 1.2). Online, `http://www.w3.org/TR/soap/` - Accessed in January 13th 2015, 2015b.

W3C – World Wide Web Consortium, Web Service Description Language (WSDL). Online, `http://www.w3.org/TR/wsdl` - Accessed in January 13th 2015, 2015c.

W3C – World Wide Web Consortium, Web Services Choreography Description Language Version 1.0 (WS-CDL). Online, `http://www.w3.org/TR/ws-cdl-10/` - Accessed in January 13th 2015, 2015d.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; van de Molengraft, R. RoboEarth. *IEEE Robotics & Automation Magazine*, v. 18, n. 2, p. 69–82, 2011.

Waldo, J. The Jini architecture for network-centric computing. *Communications of the ACM*, v. 42, n. 7, p. 76–82, 1999.

Walter, C.; Schulenburg, E.; Beier, D.; Elkmann, N. Data acquisition and processing using a service oriented architecture for an automated inspection system. In: *Proceedings of the 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'07)*, Dortmund, Germany, 2007, p. 440–444.

Watkins, C.; Dayan, P. Q-learning. *Machine Learning*, v. 8, n. 3-4, p. 279–292, 1992.

Weyns, D.; Holvoet, T. A reference architecture for situated multiagent systems. In: *Proceedings of the 3rd International Conference on Environments for multi-agent systems (E4MAS'06)*, Hakodate, Japan: Springer-Verlag, 2006, p. 1–40 (LNCS v. 4389).

Wohlin, C.; Andrews, A. Prioritizing and assessing software project success factors and project characteristics using subjective data. *Empirical Software Engineering*, v. 8, n. 3, p. 285–308, 2003.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering.* Berlin, Germany: Springer, 2012.

Wolf, D. F.; Osório, F. S.; Simões, E.; Trindade Jr., O. *Intelligent robotics: From simulation to real world applications.* Rio de Janeiro, Brazil: JAI: Jornada de Atualização em Informática da SBC, (in Portuguese), 2009.

WSO2 WSO2 Enterprise Service Bus. Online, `http://wso2.com/products/enterprise-service-bus/` - Accessed in January 8th 2015, 2015.

Wu, B.; Xi, L.-f.; Zhou, B.-H. Service-oriented software architecture for flexible manufacturing control system. In: *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE'06)*, Shanghai, China, 2006, p. 425–430.

Wu, B.; Zhou, B.-H.; Xi, L.-F. Remote multi-robot monitoring and control system based on MMS and web services. *Industrial Robot: an International Journal*, v. 34, n. 3, p. 225–239, 2007.

Yang, T.-H.; Lee, W.-P. A service-oriented approach with neural networks and knowledge ontologies for robot control. In: *Proceedings of the 20th International Conference on Neural Information Processing (ICONIP'13)*, Daegu, Korea: Springer-Heidelberg, 2013a, p. 473–483 (LNCS v. 8227).

Yang, T.-H.; Lee, W.-P. A service-oriented framework for the development of home robots. *International Journal of Advanced Robotic Systems*, v. 10, p. 1–11, 2013b.

Yeom, G. Dynamic binding framework for open device services. In: *Proceedings of the 4th International Conference on Ubiquitous Intelligence and Computing (UIC'07)*, Hong Kong, China: Springer-Heidelberg, 2007, p. 73–82 (LNCS v. 4611).

Zeeb, E.; Bobek, A.; Bohn, H.; Golatowski, F. Service-oriented architectures for embedded systems using devices profile for web services. In: *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, Niagara Falls, Ontario, Canada, 2007, p. 956–963.

Zhou, G.; Zhang, Y.; Bastani, F.; Yen, I.-L. Service-oriented robotic swarm systems: Model and structuring algorithms. In: *Proceedings of the 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC'12)*, Shenzhen, China, 2012, p. 95–102.

Zimmermann, O.; Kopp, P.; Pappe, S. Architectural knowledge in an SOA infrastructure reference architecture. In: *Software Architecture Knowledge Management*, Berlin, Heidenberg: Springer-Verlag, p. 217–241, 2009.

Appendix

A

List of Included Primary Studies

This appendix lists, in Table A.1, the primary studies included in the systematic review presented in Section 2.4.5.

Table A.1: Title of the included primary studies

ID	Study title
S1	An intelligent service-based network architecture for wearable robots
S2	Service-oriented integration of networked robots with ubiquitous sensors and devices using the semantic Web services technology
S3	Web services based robot control platform for ubiquitous functions
S4	Reliable protocol for robot communication on web services
S5	UPnP SDK for Robot Development
S6	Distributed Robotic Architecture based on Smart Services
S7	A Service-Oriented Approach for Building Autonomic Peer-to-Peer Robot Systems
S8	A Web Lab for Mobile Robotics Education
S9	SENORA: A P2P Service-Oriented Framework for Collaborative Multirobot Sensor Networks
S10	Property service architecture for distributed robotic and sensor systems
S11	Data Acquisition and Processing using a Service Oriented Architecture for an Automated Inspection System
S12	Remote multi-robot monitoring and control system based on MMS and web services
S13	Dynamic Binding Framework for Open Device Services

Table A.1: Title of the included primary studies – *continued*

ID	Study title
S14	A Software System for Robotic Learning by Experimentation
S15	Developing a Security Robot in Service-Oriented Architecture
S16	On Robotics Applications in Service-Oriented Architecture
S17	A Middleware Based Control Architecture for Modular Robot Systems
S18	Utilizing semantic Web 2.0 for self-reconfiguration of SOA based agent applications in Intelligent Service Robots
S19	A Generic Service Oriented Architectural Model for Pervasive Applications: A Case Study in Internet-based Multiple Robot Control
S20	Dynamic control of a robotic swarm using a service-oriented architecture
S21	An ontology-based collaborative service-oriented simulation framework with Microsoft Robotics Studio
S22	ISROBOTNET: A testbed for sensor and robot network systems
S23	Design and Performance Evaluation of a Service-Oriented Robotics Application
S24	Applying semantic web service composition for action planning in multi-robot systems
S25	Adaptation of resource-oriented service technologies for industrial informatics
S26	Experiments with service-oriented architectures for industrial robotic cells programming
S27	Multirobot system architecture: environment representation and protocols
S28	DAvinCi: A cloud computing framework for service robots
S29	Model of integration and management for robotic functional components inspired by the human neuroregulatory system
S30	Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics
S31	A Service Oriented Architecture supporting an autonomous mobile robot for industrial application
S32	Robot as a Service in Cloud Computing
S33	Robopedia: Leveraging Sensorpedia for web-enabled robot control
S34	Towards a REST-style architecture for networked vehicles and sensors
S35	Service oriented soft real-time implementation of SLAM capability for mobile robots
S36	Robots on the Web
S37	RoboCoP: A Protocol for Service-Oriented Robot Control Systems
S38	Sea Robot-Assisted Inspection
S39	RoboEarth
S40	Towards a service-oriented architecture for teams of heterogeneous autonomous robots
S41	Cloud robotics: Towards context aware robotic networks
S42	Service Component Architectures in Robotics: The SCA-Orocos Integration
S43	Robotic Services in Cloud Computing Paradigm
S44	Robot-Cloud: A framework to assist heterogeneous low cost robots

Table A.1: Title of the included primary studies – *continued*

ID	Study title
S45	SOA4DERTS: A Service-Oriented UML profile for distributed embedded real-time systems
S46	Virtual planning for autonomous inspection of electromechanical products
S47	Service-oriented robotic swarm systems: Model and structuring algorithms
S48	A system integration approach for service-oriented robotics
S49	A service-oriented framework for the development of home robots
S50	Developing service oriented robot control system
S51	New layered SOA-Based architecture for multi-robots cooperative online SLAM
S52	Service Oriented Robotic Architecture for Space Robotics: Design, Testing, and Lessons Learned
S53	Capability-oriented robot architecture for maritime autonomy
S54	A service-oriented architecture for virtualizing robots in robot-as-a-service clouds
S55	Multi-robot data mapping simulation by using microsoft robotics developer studio
S56	Towards a Process to Design Architectures of Service-Oriented Robotic Systems
S57	Automating the Discovery of Services for Service-Oriented Robotic Systems

Appendix

B

Project Specification and Functional Requirements

This appendix shows both specification and functional requirements of an extended RobAFIS project used in the experiment reported in Section 4.3 and in the case study described in Section 5.6 of this thesis.

Objective

This document describes requirements of a robotic system that controls two handling robots that transport products in an industrial shop floor. In this document, the handling robots will be referred generically as robots.

Application Domain

This document is applicable to design a robotic system that will be developed and used in a comparative evaluation of several competing solutions. The robotic system will be evaluated in a simulated industrial shop floor.

Equipment Available

Items of equipment available for the competition are:

- Two robots that are capable of fulfilling the mission specified in Section 4;
- One computer used for managing the robotic application; and
- All network infrastructure needed to the communication among the devices (robots and computer).

Mission

The nominal mission of the robotic system is to transport products in a shop floor (schematically described in Section 6). Two robots must transport products manufactured in two different production units (PU1, PU2) and place them in their respective storage units (SU1, SU2) according to the type of the product (PT1, PT2).

Initially the robots are in standby mode on their rest areas. At the signal given by the operator, one of the two robots must:

- Pick up the product in the production unit;
- Bring it in the storage unit according to its type; and
- Return to standby mode and stop inside one of three rest areas (RA1, RA2, and RA3).

Every time the operator requests to the robotic system to transport a product, he informs the position (x,y) and the type of the product to be transported (PT1 or PT2). The robotic system must select the most adequate robot to perform the transport based on the overall distance between the rest area where the robot is, the production unit, and the respective storage unit. This distance must consider objects and walls blocking the path and not only the geographic distance.

During transportation, products must be carried by the robot (without ground contact). There is no orientation imposed on the robot in a rest area at the beginning and end of the mission. The positions of storage units, rest areas, and production units are well known and described according to simplified geographic coordinates (x,y). All coordinates will be informed in the day of the competition. Geographic coordinates of the four extremities of the shop floor will be also available in the day of the competition. There are no marks on the floor to guide the navigation.

As part of its mission, the robot must move autonomously (without remote control), except:

- If during its navigation the robot encounters an obstacle in its path. In this case, the robot must stop and wait until the obstacle is removed; and
- If the robot is in a non-operational state (stopped, damaged), an operator can perform corrective maintenance operations, repositioning and restarting the robot and then carrying it to the initial rest area using a remote control system in the computer.

Despite the positions of storage units, rest areas, and production units are known and fixed, the layout of the shop floor can change from a day to another. Walls surrounding the production units can be added, moved or removed after a working day (see Section 6). Therefore, before starting a work day (before perform the first mission of the day), the robots must create a new representation of the shop floor, which will be used to define the navigation path for each transportation request.

Robots Characterization

The two robots used in the mission are Pioneer P3-DX (Figure B.1), with the same hardware configuration. Each robot is 450mm length, 490mm width, and 500mm height (including the hardware devices on it). The base of Pioneer P3-DX platform can reach speeds of 1.6 meters per second and carry a payload of up to 23 kg. The robot is powered by three hot-swappable 9Ah sealed batteries. The robot's embedded motion controller performs velocity and direction control of the robot. In addition to the motion control, the Pioneer P3-DX platform also allows to perform control and information acquisition of an embedded GPS (Global Position System), eight sonar sensors in the front (with a maximum reach of 3.000mm) and the battery. The GPS provides absolute position and orientation of the robot (x, y, theta). Pioneer P3-DX moves using 2-wheel differential drive, with rear balancing caster. The two front wheels are independent and can have different speeds if needed.

The following additional sensors and actuators are attached the robot platform:

- Laser rangefinder sensor: it provides a range of 180 measures of distance in a single plan, with a maximum reach of 8.000mm (as represented in Figure B.2);

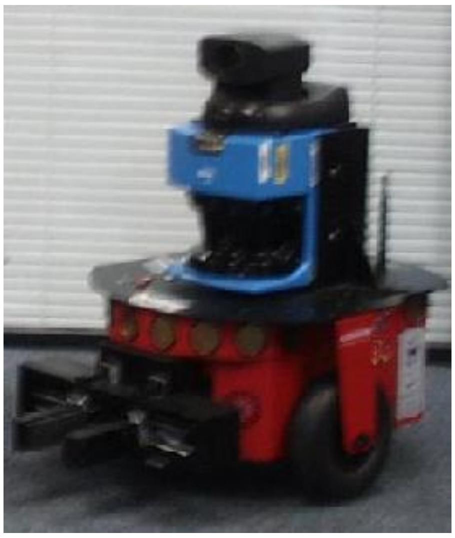


Figure B.1: Pioneer P3-DX robot

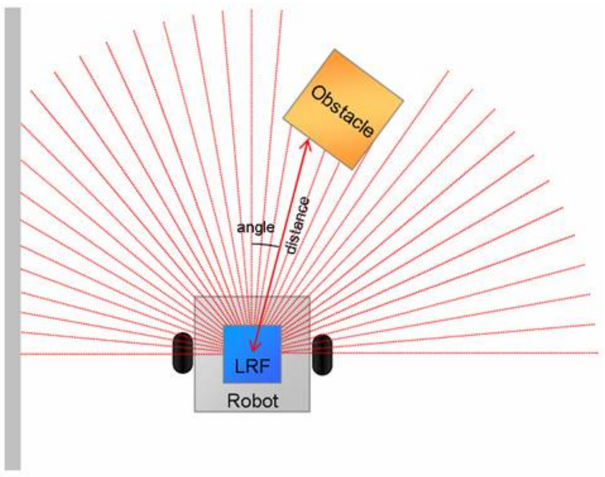


Figure B.2: Representation of the laser rangefinder measures

- Indoor pan-tilt-zoom (PTZ) camera: it features auto-focus and automatic brightness/gain control, and an image resolution of up to 704x576, making possible the robotic system to acquire and process images from the environment, including colored objects; and
- Gripper actuator: it is a 2-axis, 2 degree-of-freedom (DOF) robotic gripper that opens and closes horizontally and raises up to carry the grasped object off the floor.

Environment Characterization

The shop floor, defined schematically in Figure B.3, is subject to regular lighting (sun light passing through the acrylic roof of the room, without extra lighting) except shadows produced by objects present in the environment and the robots.

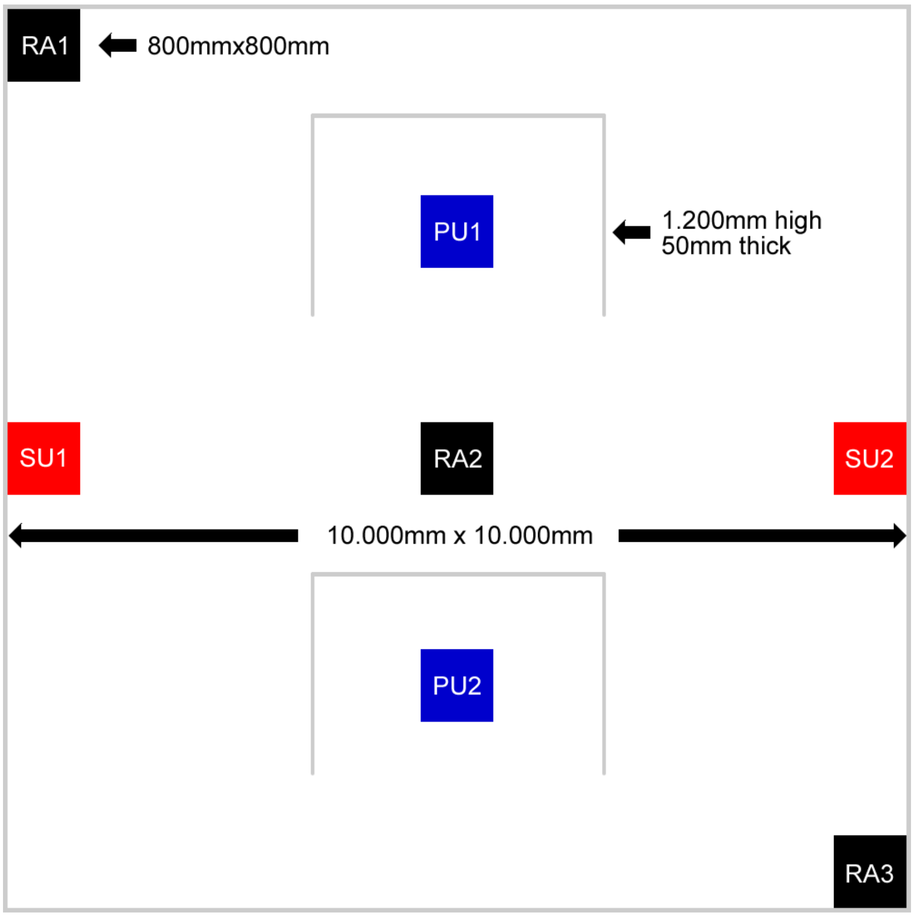


Figure B.3: Layout of the shop floor

The shop floor has a size of 10.000mm x 10.000mm. The floor is perfectly flat, horizontal, rigid with a hard surface, and uniform light color. Walls are also uniform light color, 1.200mm high and 50mm thick. The rest areas, production units, and storage units are indicated on the ground by 800mmx800mm colored squares (black for the rest area, blue for the production units, red for the storage units). As mentioned before, walls can be place in different layouts, according to the factory production requirements. Figure B.4 presents an alternative layout for the shop floor.

Product Characterization

Products transported by the robots, illustrated in Figure B.5, are rigid, green, and non-reflexives. Products of Type 1 and Type 2 are identical in design, color, and weight and are differentiated only by marks on the top (PT1, PT2). Products are cylinders 150mm high, with two different thicknesses: 50mm on the extremities, 20mm on the center. The thickest parts of the body (the extremities) are 10mm tall and the central part is 130mm tall.

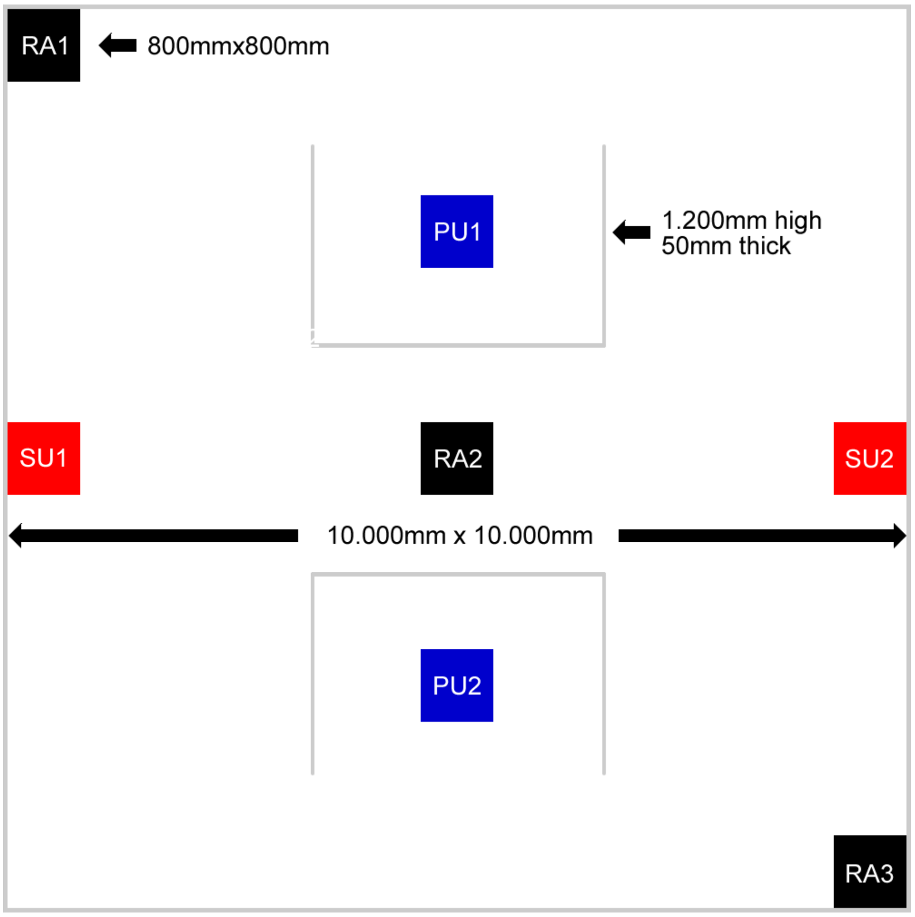


Figure B.4: Alternative layout of the shop floor

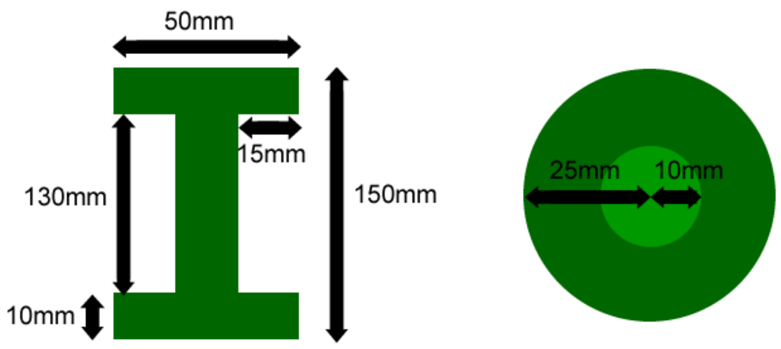


Figure B.5: Product specifications

Robotic System Functional Requirements

The robotic system to be designed must meet the following functional requirements:

FR 1: The robotic system must coordinate two mobile robots to autonomously transport products from production units to storage units.

FR 2: The robots coordinated by the robotic system must be in rest areas when they are not performing a transport request.

FR 3: At the beginning of the mission, one robot must be at RA1 and the other at RA3.

FR 4: The robotic system must use only one robot to transport each product.

FR 5: The robot controlled by the robotic system must transport only one product at each time.

FR 6: Once a robot lifts a product in a production unit, this product can only be released in the respective storage unit.

FR 7: The robotic system must be aware of the position and orientation of both robots.

FR 8: The robotic system must define the navigation path for both robots before choosing the robot that will perform the transport.

FR 9: The robotic system must choose the robot that can perform the mission using the shortest path.

FR 10: The robotic system should calculate the shortest path considering objects and walls blocking the path and not only the geographic distance.

FR 11: The robotic system must create a representation of the shop floor in the beginning of every working day, before realizing the first transport order.

FR 12: The robotic system should create the shop floor representation based on the sensory information of the robots.

FR 13: The representation of the shop floor must be built by both robots, cooperatively.

FR 14: After creating the representation of the shop floor, one robot must return to RA1 and the other to RA3.

FR 15: The current representation of the shop floor must be available for both robots at any time.

FR 16: The navigation paths must be defined according to the environment representation created by the robotic system.

FR 17: The navigation paths identified for the robots must be minimum paths.

FR 18: The robotic system must queue a transport order in case of none of the robots is available for the transport at the moment of a request.

FR 19: The transport order must be done by an operator on a remote computer.

FR 20: The transport order must inform the position of the product (in geographic coordinates) and its type.

FR 21: The robotic system must be able to identify the reference areas (rest areas, production unities, and storage units) in the shop floor.

FR 22: The robotic system must be able to transport products factored in both production units (PU1 and PU2).

FR 23: The robotic system must transport products PT1 to ST1 and PT2 to SU2.

FR 24: The robotic system must be able to autonomously localize and identify products inside the production unit.

FR 25: The robotic system must be able to autonomously pick and release products.

FR 26: The robotic system must release products in vertical position.

FR 27: After the transport mission, the robot must return to the closest Rest Area (RA1, RA2, or RA3) available in the shop floor.

FR 28: The robot system must be able to identify a rest area occupied by other robot as a not available rest area.

FR 29: The robotic system must notify the operator whenever it concludes a transport order.

FR 30: The robotic system must avoid collisions between the robots and the environment.

FR 31: The robotic system must avoid collisions between robots.

FR 32: The robotic system must detect objects blocking the path of a robot and stop it until the object is removed.

FR 33: If a robot is not able to autonomously navigate, the operator must remotely drive it to a rest area.

FR 34: The robots controlled by the robotic system must operate independently of each other. If one robot stops, the other must keep receiving transport requests.

FR 35: The robotic system must be able to be monitored remotely by the operator.

FR 36: The robotic system must be able to control the camera on the top of the robots.

FR 37: The robotic system must be able to process images obtained by the camera.

FR 38: The robotic system must be able to control the differential drive of the robots.

FR 39: The robotic system must be able to control the gripper actuator attached to the robots.

FR 40: The robotic system must be able to process data acquired by the laser rangefinder sensor on the robots.

FR 41: The robotic system must be able to process data acquired by the eight sonars in each robot.

FR 42: The robotic system must be able to process data acquired by the GPS sensor.

Appendix

C

Declaration of Original Authorship and List of Publications

I confirm that this doctoral dissertation has not been submitted in support of an application for another degree at this or any other teaching or research institution. It is the result of my own work and the use of all material from other sources has been properly and fully acknowledged. Research done in collaboration is also clearly indicated.

Excerpts of this dissertation have been either published or submitted for the appreciation of editorial boards of journals, conferences, and workshops, according to the list of publications presented below. My contributions to each publication are listed as well.

Publications Resulting from this Thesis

- **Oliveira, L. B. R.**; Amaral, F. A.; Martins, D. B.; Oquendo, F.; Nakagawa, E. Y.; *"RoboSeT: A Tool to Support Cataloging and Discovery of Services for Service-Oriented Robotic Systems"* (accepted for publication) (Oliveira et al., 2015).

  Journal: Communications in Computer and Information Systems.

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- **Oliveira, L. B. R.**; Osório, F. S; Oquendo, F.; Nakagawa, E. Y.; *"Towards a Taxonomy of Services for Developing Service-Oriented Robotic Systems"* (Oliveira et al., 2014c).

  Event: 26th International Conference on Software Engineering and Knowledge Engineering (SEKE'14).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- **Oliveira, L. B. R.**; Leroux, E.; Felizardo, K. R.; Oquendo, F.; Nakagawa, E. Y.; *"Towards a Process to Design Architectures of Service-Oriented Robotic Systems"* (Oliveira et al., 2014a).

  Event: 8th European Conference on Software Architecture (ECSA'14).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- **Oliveira, L. B. R.**; Martins, D. B.; Amaral, F. A.; Oquendo, F.; Nakagawa, E. Y.; *"Automating the Discovery of Services for Service-Oriented Robotic Systems"* (Oliveira et al., 2014b).

  Event: 11th IEEE Latin American Robotics Symposium (LARS'14).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- **Oliveira, L. B. R.**; Osório, F. S; Nakagawa, E. Y.; *"An investigation into the development of Service-Oriented Robotic Systems"* (Oliveira et al., 2013b).

  Event: 28th Annual ACM Symposium on Applied Computing (ACM/SAC'13).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- **Oliveira, L. B. R.**; Guessi, M.; Feitosa, D.; Manteufeel, C.; Galster, M.; Oquendo, F.; Nakagawa, E. Y.; *"An Investigation on Quality Models and Quality Attributes for Embedded Systems"* (Oliveira et al., 2013a).

  Event: 8th International Conference on Software Engineering Advances (ICSEA'13).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

Technical Report

- **Oliveira, L. B. R.**; Osório, F. S; Nakagawa, E. Y.; *"A Systematic Review on Service Oriented Robotic Systems Development"* (Oliveira et al., 2012).

  Institution: Institute of Mathematical and Computer Sciences, University of São Paulo (ICMC/USP).

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

Other Related Publications

- Guessi, M.; Cavalcante, E.; **Oliveira, L. B. R.**; *"Characterizing Architecture Description Languages for Software-Intensive Systems-of-Systems"* (accepted for publication) (Guessi et al., 2015a).

  Event: 3rd International Workshop on Software Engineering for Systems-of-Systems (SESoS'15) – held in conjunction with the 37th International Conference on Software Engineering (ICSE'15).

  Level of contribution: Medium – the PhD candidate helped in the paper writing.

- Guessi, M.; **Oliveira, L. B. R.**; Garcés-Rodríguez, L. M.; Oquendo, F.; *"Towards a Formal Description of Reference Architectures for Embedded Systems"* (accepted for publication) (Guessi et al., 2015b).

  Event: 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA'15) – held in conjunction with the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA'15)

  Level of contribution: Medium – the PhD candidate helped in writing and structuring the paper.

- Graciano Neto, V. V.; Garcés-Rodríguez, L. M.; Guessi, M.; **Oliveira, L. B. R.**; Oquendo, F.; *"On the Equivalence between Reference Architectures and Metamodels"* (accepted for publication) (Graciano Neto et al., 2015).

  Event: 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA'15) – held in conjunction with WICSA'15.

  Level of contribution: Low – the PhD candidate helped in the paper writing.

- Graciano Neto, V. V.; Guessi, M.; **Oliveira, L. B. R.**; Oquendo, F.; Nakagawa, E. Y.; *"Investigating the Model-Driven Development for Systems-of-Systems"* (Graciano Neto et al., 2014).

  Event: 2nd International Workshop on Software Engineering for System-of-Systems (SESoS'14) – held in conjunction with ECSA'14.

  Level of contribution: High – the PhD candidate helped in the planning and conduction of the review, as well as in the paper writing.

- Affonso, F. J.; Felizardo, K. R.; **Oliveira, L. B. R.**; Nakagawa, E. Y.; *"Reference Architectures for Self-Managed Software Systems: a Systematic Literature Review"* (Affonso et al., 2014).

  Event: 8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS'14)

  Level of contribution: Medium – the PhD candidate helped in the planning of the review, as well as in the paper writing.

- Oliveira, R. A. P.; **Oliveira, L. B. R.**; Cafeo, B. B. P.; Durelli, V. H. S.; *"On Using Mutation Testing for Teaching Programming to Novice Programmers"* (Oliveira et al., 2014d).

  Event: 22nd International Conference on Computers in Education (ICCE'14).

  Level of contribution: High – the PhD candidate helped in the planning and conduction of the experiment, as well as in the paper writing.

- Nakagawa, E. Y; Golçalves, M.; Guessi, M.; **Oliveira, L. B. R.**; Oquendo, F.; *"The State of the Art and Future Perspectives in Systems of Systems Software Architectures"* (Nakagawa et al., 2013)

  Event: 1st International Workshop on Software Engineering for Systems-of-Systems (SESoS'13) – held in conjunction with ECSA'13.

  Level of contribution: High – the PhD candidate helped in the planning and conduction of the review, as well as in the paper writing.

- **Oliveira, L. B. R.**; Nakagawa, E. Y.; *"A Service-Oriented Reference Architecture for Software Testing Tools"* (Oliveira and Nakagawa, 2011).

  Event: 5th European Conference on Software Architecture (ECSA'11).

Level of contribution: High – The PhD candidate is the main investigator and conducted the work together with his advisor.

- **Oliveira, L. B. R.**; Baldochi, L. A.; Nakagawa, E. Y.; *"Development of a Complex System Based on Architectural and Design Patterns"* (Oliveira et al., 2011).

  Event: 7th International Conference on Applied Computing (IADIS-AC'11)

  Level of contribution: High – the PhD candidate is the main investigator and conducted the work together with his contributors.

- Guessi, M.; **Oliveira, L. B. R.**; Nakagawa, E. Y.; *"Representation of Reference Architectures: A Systematic Review"* (Guessi et al., 2011).

  Event: 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE'11)

  Level of contribution: Medium – the PhD candidate helped in the planning of the review, as well as in the paper writing.