People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed Khider University - Biskra
Faculty of Exact Sciences and Sciences of Nature and Life
Computer Science Department

Order Number: ....

THESIS

In Candidacy for the Degree of
DOCTOR 3^{rd} CYCLE IN COMPUTER SCIENCE
Option : Artificial Intelligence

TITLE

ROS-based Solution for Robotic Services in Cloud Computing

Presented by **Radhia BOUZIANE**

Defended on: 07/06/2022

In front of the jury composed of:

Mr. Soheyb AYAD	Associate Professor at University of Biskra	President
Mr. Labib Sadek TERRISSA	Professor at University of Biskra	Supervisor
Mr. Jean-François BRETHE	Professor at University of le Havre, France	Co-supervisor
Mr. Okba TIBERMACINE	Associate Professor at University of Biskra	Examiner
Mr. Saber BENHARZALLAH	Professor at the University of Batna 2	Examiner
Mr. Rachid SEGHIR	Professor at the University of Batna 2	Examiner

Academic year : **2021/2022**

Dedicated to my dear parents and all my family.

Acknowledgement

First and foremost, praises and thanks to *Allah*.

I would like to express my gratitude to my supervisor *Pr. Terrissa Sadek Labib* for his patience, continuous support, and guidance throughout this work.

I also thank all the members of the jury *Dr. Ayad Soheyb*, *Dr. Tibermacine Okba*, *Pr. Benharzallah Saber*, and *Pr. Seghir Rachid* for accepting to review and evaluate this work. Special thanks to *Dr. Ayad* for his advice, help, and knowledge.

I would also like to thank *Pr. Brethé Jean-François* for the discussions and helpful comments about this work.

I thank the LINFI laboratory members and all those who helped me in some way for realizing this work.

Finally, many thanks go to my parents and family for their support and encouragement.

Abstract

Robot Operating System (ROS) is becoming a widely-used environment for developing robot software systems. It provides unique features such as message-passing between processes and code reuse between robots. The new trend in ROS-based robotic systems is facing the development and delivery of effective services by combining the advantages of both cloud robotics and web services.

Cloud robotics is the way that allows robots to overcome their limitations of processing and knowledge by boosting computational and cognitive capabilities. On the other hand, as an implementation of Service-Oriented Architecture (SOA), web services allow mainly different ROS codes to be discovered over the internet for their reuse. However, the characterization, description, and discovery of the ROS service capability for the offered robotic functionality are still issues that are not fully addressed.

In this context, we focus in this thesis on developing an architecture for robotic software provisioning to both software developers and robots by exploiting the opportunities of ROS, web services, and cloud robotics. We propose a complete SOA approach for cloud robotics, in which ROS-based robotic tasks are defined as web services. The approach focuses on defining the service cycle process of describing, discovering, and selecting services. Two characterizations for ROS web services are proposed. The service characterizations describe the semantic representation of the robot task from ROS itself. In each case, we present a strategy that allows users to discover the relevant robotic service that can match their queries and robots.

Keywords: *Robot Operating System (ROS), Cloud robotics, Service-Oriented Architecture (SOA), Web services, Semantic web services, Robotic service discovery.*

Résumé

Robot Operating System (ROS) devient de plus en plus l'environnement le plus utilisé pour le développement de systèmes logiciels de robots. Il fournit des fonctionnalités uniques telles que la transmission de messages entre les processus et la réutilisation de code entre robots. La nouvelle tendance des systèmes robotiques basés sur ROS est confrontée au développement et provision de services efficaces en combinant les avantages de Cloud Robotics et des services Web.

Le Cloud Robotics désigne la manière qui permet aux robots de surmonter leurs limites de traitement et de connaissances en promouvant les capacités de calcul et cognitives. D'autre part, en tant qu'implémentation de l'Architecture-Orientée Services (SOA), les services Web sont principalement destinés à découvrir des codes ROS sur Internet pour leur réutilisation. Cependant, la caractérisation, la description, et la découverte de capacité d'un service ROS sur la fonctionnalité robotique offerte ne sont pas complètement adressées.

Dans ce contexte, nous nous concentrons dans cette thèse sur le développement d'une architecture pour la fourniture de logiciels robotiques aux développeurs de logiciels et aux robots en exploitant la technologie et le concept de ROS, services Web et de Cloud Robotics. Nous proposons une approche SOA complète pour le Cloud Robotics, dans laquelle les tâches robotiques basées sur ROS sont définies comme des services Web. L'approche se concentre sur la définition du processus de cycle de service pour la description, la découverte, et la sélection des services. Deux caractérisations des services web ROS sont proposées. Les caractérisations de service décrivent la représentation sémantique de la tâche du robot à partir de ROS lui-même. Dans chaque cas, nous présentons une stratégie qui permet aux utilisateurs de découvrir le service robotique pertinent qui peut correspondre à leurs requêtes et robots.

Keywords: *Robot Operating System (ROS), Cloud Robotics, Architecture-Orientée Services (SOA), services Web, services Web sémantiques, Découverte de services robotiques.*

ملخص

أصبح نظام تشغيل الروبوت (ROS) بيئة مستخدمة على نطاق واسع لتطوير أنظمة برامج الروبوت. فهو يوفر ميزات فريدة مثل تمرير الرسائل بين العمليات وإعادة استخدام الكود بين الروبوتات. تواجه الأنظمة الروبوتية القائمة على ROS من خلال اتجاهها الجديد تطوير وتقديم خدمات فعالة من خلال الجمع بين مزايا كل من الروبوتات السحابية وخدمات الويب. تعتبر الروبوتات السحابية الطريقة التي تسمح للروبوتات بالتغلب على قيود المعالجة والمعرفة من خلال تعزيز القدرات الحسابية والمعرفية. من ناحية أخرى، تتيح خدمات الويب بشكل أساسي، والتي تعتبر كتنفيذ للبنية الخدمية (SOA)، اكتشاف مختلف البرامج الخاصة ب ROS عبر الإنترنت من أجل إعادة استخدامها. ومع ذلك، فإن تحديد خصائص ووصف واكتشاف خدمة ROS المقدمة وقدرتها التي تعبر عن الوظيفة الروبوتية لا تزال من المشكلات التي لم تتم معالجتها بشكل كامل.

في هذا السياق، نركز في هذه الأطروحة على تطوير بنية لتوفير البرامج الروبوتية لكل من مطوري البرامج والروبوتات من خلال استغلال مزايا ROS وخدمات الويب والروبوتات السحابية. نقوم باقتراح نهج كامل مبني على SOA للروبوتات السحابية، حيث يتم تعريف المهام الروبوتية القائمة على ROS على أنها خدمات ويب. يركز النهج على تحديد عملية دورة خدمة الويب المتمثلة في كل من وصف واكتشاف واختيار الخدمات. تم اقتراح توصيفين لخدمات الويب ROS. تقوم توصيفات الخدمة بوصف التمثيل الدلالي لمهمة الروبوت من خلال ROS نفسه. نقدم في كل حالة إستراتيجية تسمح للمستخدمين باكتشاف الخدمة الروبوتية التي يمكن أن تتطابق مع استفساراتهم وروبوتاتهم.

الكلمات المفتاحية: نظام تشغيل الروبوت (ROS)، الروبوتات السحابية، البنية الخدمية (SOA)، خدمات الويب، خدمات الويب الدلالية، اكتشاف الخدمة الروبوتية.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AI	Artificial Intelligence
Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CAS	Circular Area Search
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRS	Cloud-enabled Robotic Services
DaaS	Discovery as a Service
FIL	Federated Imitation Learning
HTTP	HyperText Transfert Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IFR	International Federation of Robotics
IOPE	Inputs Outputs Preconditions and Effects
IP	Internet Protocol
ISO	International Organization for Standardization
Java EE	Java Enterprise Edition
JDK	Java Development Kit
KIT	Karlsruhe Institute of Technology
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
MQTT	Message Queuing Telemetry Transport
MRDS	Microsoft Robotics Developer Studio
MRS	Multi-Robot-based Services
MySQL	My- Structured Query Language

NIST	National Institute of Standards and Technology
NLI	Natural Language Inference
OpenRAVE	Open Robotics Automation Virtual Environment
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PaaS	Platform as a Service
PHM	Prognostic and Health Management
QoS	Quality of Service
RAaaS	Robotics and Automation as a Service
RaaS	Robot as a Service
RAM	Random Access Memory
REST	Representational State Transfer
RILaaS	Robot Inference and Learning as a Service
ROS	Robot Operating System
ROS-SWS	ROS Semantic Web Service
ROS-VMs	ROS Virtual Machines
ROS-WS	ROS Web Service
RPC	Remote Procedure Call
RSaaS	Robotic Services as a Service
RSCM	Robotic Service Composition Middlewares
RSi-Cloud	Robot Service initiative -Cloud
RSNP	Robot Service Network Protocol
SaaS	Software as a Service
SBERT	Sentence-BERT
SDK	Software Development Kit
SE	Sentence Embedding
SLA	Service Level Agreement
SLAM	Simultaneous Localization and Mapping
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SRDF	Semantic Robot Description Format
SSH	Secure Shell

STS	Semantic Textual Similarity
STSb	STS benchmark
tf-idf	term frequency - inverse document frequency
UAV	Unmanned Arial Vehicles
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
UNR-PF	Ubiquitous Network Robot Platform
UPnP	Universal Plug and Play
URDF	Unified Robot Description Format
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XaaS	Everything as a Service
XML	Extensible Markup Language
YARP	Yet Another Robot Platform

Chapter1
Introduction

1.1 Context

Cloud computing has brought a significant shift in the accessibility and utilization of robots because of technological advancements in processing, storage, and communication. This relevant innovation in robotics field is known as “Cloud Robotics”. Cloud robotics allows robots to overcome their processing constraints, share information, or acquire new skills [1]. It describes a new generation of robotics that use cloud computing to improve task performance, by boosting computational and cognitive capabilities and enabling knowledge sharing.

The development of robot software systems within this context has been widely built recently using “Robot Operating System” (ROS). This is due to its open-source robotic system that provides various features, including low-level device control, abstraction of hardware, and message passing between processes [2]. Nevertheless, as robotic systems development may require challenges of implementation complexity [3,4], understanding ROS requires thorough learning for achieving desired robot tasks. Thus, designing software that deliver reusable components in robotic systems has been receiving much attention using Service-Oriented Architecture (SOA). SOA presents an architectural style of designing software systems that can be accomplished using web service technologies. It provides encapsulated, discoverable, reusable, and loosely coupled application functions distributed in a network. By encapsulating ROS implementation details and offering loosely coupled application functions, SOA and web services provide a step to the response of ROS manipulation issue [5].

However, implementing SOA in robotics in general is influenced by the developers’ background [6], which can restrict the scope of SOA. By studying the state-of-the-art of service-oriented solutions in robotics, a diversity of research proposals can be observed in many levels of both system conception and development. Most studies have focused on SOA as a mechanism that adapts in each case study of the various robotic fields, rather than building an architectural style. This has affected the SOA life cycle process of dynamic service discovery.

1.2 Problem statements

The problem addressed in this thesis targets the development of service-oriented robotic services for ROS-based robots and cloud robotics. At first, we seek to study SOA adaptation for robotic services development and delivery. This includes SOA-based robotic evolution of research directions, trends, and service delivery models in cloud robotics. Thus, this first part can be determined by the following research question:

- RQ1. *How can SOA be applied to the development of robotic services, and particularly, to the delivery of robotic services in cloud-based systems?*

By tackling the first problem, the second issue of this thesis has been arisen. It studies the whole life cycle process of applying SOA by calling the "robotic services" concept into question. This issue involves the description, discovery and selection mechanisms of such services. Therefore, the second research question addressed in this thesis is:

- RQ2. *What are the key characteristics of robotic services?*

Indeed, this main research question is divided into the two following questions:

- RQ2.1. *What is a robotic service? How the robotic service is represented or described?*
- RQ2.2. *How can a robotic service be discovered and consumed to complete a robot task?*

1.3 Contributions

In this thesis, we carried out a comparative analysis that targets the service-oriented solutions in robotics, and we provide a literature categorization. Based on the review results, the thesis's main contributions may be summarized into two principal axes. In the first contribution, a complete SOA-based architecture for cloud robotic service provisioning has been proposed. The architecture addresses the idea of Robotic Services as a Service (RSaaS) using ROS. It supports the automatic search of services that deliver various robotic tasks, which allows users to assign different on-demand skills to their robots. We consider the ROS services as web services and aim to improve the task performance of services by boosting computational capabilities. The system architecture is structured as a layered architecture that is inspired from classical cloud computing architectures.

The second contribution focuses on defining the elements and cycle process of ROS-based robotic services. Two solutions are proposed for this definition, which presents a characterization of these services that describes the robot task representation from ROS itself. In the first one, we define the functional meta data of ROS Web Service (ROS-WS), by considering ROS-WSs as SOAP (Simple Object Access Protocol) web services due to this protocol's completeness of architecture elements. The architecture system allows also users to obtain and access the relevant ROS-WS for their robots, according to the score assigned to a service-query match using similarity measures. To that end, the discovery engine uses the computationally efficient sentence-BERT [7] to generate sentence embeddings. This allows the system to estimate the most suitable service of a desired task according to the user's query by calculating the similarity between their embeddings. In this context, a reinforcement of training dataset has been proposed by distinguishing the relation between ROS requirements and robot tasks. The second one exploits the opportunities of semantic web services using the Ontology Web Language for Services (OWL-S), and brings a semantic layer to ROS-WSs. We define a ROS Semantic Web Service (ROS-SWS) description that expresses itself via a ROS domain ontology of capabilities and properties to handle the dynamic discovery of services.

1.4 Thesis structure

The remainder of this thesis is structured as follows:

- Chapter 2 introduces the fundamental basics of ROS, SOA, web services and cloud robotics. It summarizes as well an overview about architectures and projects in cloud robotics.

- In Chapter 3, we present a comparative analysis about service-oriented solutions in robotics that examine the SOA-based modelling architectures, delivery models and discovery of robotic services.

- Chapter 4 presents our first contribution for Robotic Services as a Service approach. We present the overall cloud architecture, actors and system modules.

- Chapter 5 presents the second contribution for the cycle of ROS web services. It is drawn on two main parts. The first part defines the ROS-WS requirements and proposed solution for their service discovery. The second part introduces the ROS-SWS description and ROS domain ontology of capabilities and properties for the discovery process. The chapter outlines also the case study and experiments.

- Chapter 6 concludes this thesis and summarizes some potential challenges and future research.

Chapter2

Fundamentals

2.1 Introduction

This chapter presents the fundamental concepts linked to our work and an overview of cloud robotics architectures. It begins by summarizing the world of robots in section 2.2 and outlining the features of this work's core element, the Robot Operating System (ROS), in section 2.3. Next, we briefly introduce section 2.4, the scopes of Service-Oriented Architecture (SOA) and Web services. Section2.5 presents the basics of cloud computing and its relation with SOA. Finally, section 2.6 gives an overview of cloud robotics projects and architectures.

2.2 World of robots

Robotics is an interdisciplinary field that interests in designing and applying robots. It benefits from many disciplines including mechanical engineering, electrical and electronic engineering, computer science and others [8]. From a historical point of view, the origin of robotics goes back to science fiction. In 1921, the term robot was coined through the czech word 'robota' meaning 'labor' in the play *Rossum's Universal Robots*. A number of stories that popularize the idea of robotics was subsequently published. By the late 1950s, the concrete transition to reality and development of first industrial robot has been occurred [9], [10].

As reported by the ISO 8373:2012 definition [11,12], a robot is an "*actuated mechanism programmable in two or more axes with a degree of autonomy*[1]*, moving within its environment, to perform intended tasks*". It consists of an integrated set of hardware and software components that build the whole machine. Figure 2.1 summarizes the seven main components of a robot as described by author of [8]. Robots can be fixed or mobile. Thus, various kinds of mobile robots can be distinguished due to their motion mechanism in the three environments: terrestrial, aquatic and aerial [13].

The evolution of robots showed an enormous expansion in several areas of modern human society. This is because robots provide a set of positive effects such as improved worker safety, and increase of manufacturing productivity and quality, which has been shown especially during the COVID-19 pandemic [14]. Robots are classified by their intended application into *industrial* and *service* robots [11] [14]:

[1]Autonomy is the robot capability of performing tasks, on the basis of sensing and current state, without the intervention of humans.

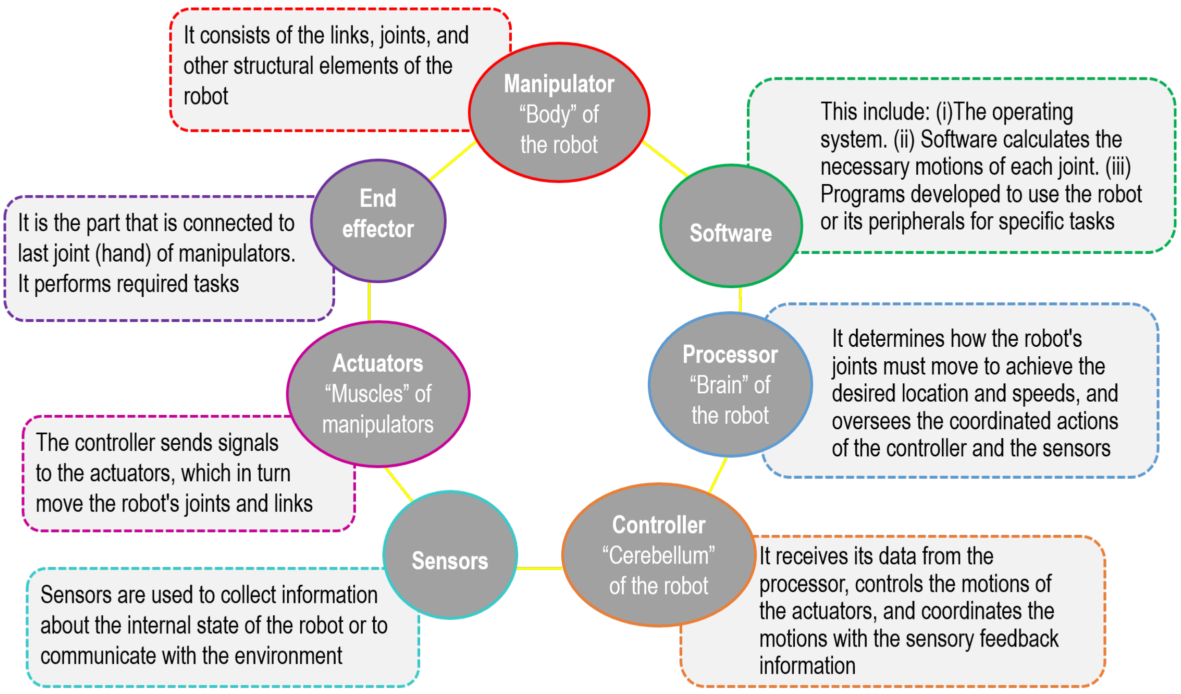

It consists of the links, joints, and other structural elements of the robot
Manipulator "Body" of the robot
This include: (i)The operating system. (ii) Software calculates the necessary motions of each joint. (iii) Programs developed to use the robot or its peripherals for specific tasks
Software
End effector
It is the part that is connected to last joint (hand) of manipulators. It performs required tasks
Actuators "Muscles" of manipulators
Processor "Brain" of the robot
It determines how the robot's joints must move to achieve the desired location and speeds, and oversees the coordinated actions of the controller and the sensors
The controller sends signals to the actuators, which in turn move the robot's joints and links
Sensors
Controller "Cerebellum" of the robot
It receives its data from the processor, controls the motions of the actuators, and coordinates the motions with the sensory feedback information
Sensors are used to collect information about the internal state of the robot or to communicate with the environment

Figure 2.1: Components of a robot.

Industrial robots: Industrial robots were the first kind of invented robots, and have known a large use through the years. Unimate, the first industrial robot with magnetic drum and six degrees of freedom[2], has been put on an assembly line in 1961 [10]. At present, industrial robot applications comprise numerous processes and manufacturing activities such as handling, welding, assembly, painting, and processing[3] [15].

Service robots: Service robotics encompass a broad range of professional and domestic applications for humans, excluding industrial automation applications [11, 16]. The degree of autonomy for service robots can be distinguished to partial, including human robot interaction, or full in which human intervention is not required [14]. The practical need of using service robots is growing. In accordance with the last statistics of the International Federation of Robotics (IFR), sales of service robots have been increased considerably in 2020 [16]. This increase has known an additional demand for some robot applications due to the global pandemic including medical, hospitality, professional cleaning, and logistics.

[2]Degrees of freedom are the set of coordinates used to describe a pose of a mobile robot or an end effector. Further information can be found in [13].

[3]This includes material removal processes like grinding, deburring, milling, and drilling.

2.3 Robot Operating System

Robot Operating System (ROS) is considered as the main axis of this thesis. The work does not use ROS just as a tool for system implementation, since it relies principally on it. We present in this section the main features about ROS.

2.3.1 Overview

ROS [2] is an open-source meta-operating system that is widely used in robot application developments nowadays. It was initially developed by the *Stanford AI Laboratory* in 2007, and its development was continued at the robotics research institute "*Willow Garage*" in collaboration with more than 20 institutions by the year 2008 [17].

ROS provides several features such as abstraction of hardware, low-level control of device, and message passing between processes[4]. Although the existing of some similar aspects between ROS and other robotic frameworks such as MRDS [18] and YARP [19], there are many reasons to prefer ROS [2], [20]. ROS is designed for supporting code reuse in robotics development and research. Primarily, a ROS code can be worked with many robots by changing the data related to every robot. Robots that we can use with ROS are multiple such as Nao, TurtleBot, PR2, and many other robots[5]. In addition, ROS provides a large set of tools for building, visualizing, and performing simulation. There are also many available and ready to use packages for some robots like SLAM that is used for performing autonomous navigation.

2.3.2 ROS concepts

We present below the three levels of ROS concepts.

Computation Graph level

The *Computation Graph* is the network of processes in ROS. This level's concepts are summarized in the following.

- *Nodes*: The main feature about ROS is the *Node*. Nodes are processes that carry out computation and constitute the ROS distributed framework. ROS

[4]ROS is officially supported on `Ubuntu`.
[5]Robots using ROS: `https://robots.ros.org/`.

libraries gives the ability to program the nodes in many languages including Python, C ++ and Java. It is possible to display information about nodes due to the `rosnode` command-line tool. For instance, the command `rosnode info node_name` gives the set of information about a particular node[6].

- *Master*: The *Master* in ROS could be seen as a manager node that allows nodes to find each other and exchange messages. This is done due to its XMLRPC-based API enabling nodes to store and retrieve each other information. We usually use the `roscore` command to run the master.

- *Parameter Server*: It is a part of the Master, which provides a central location to store data.

- *Messages*: The communication between nodes is done via *messages*. A message is data structure that comprises some field types. This includes standard primitive types such as integer, boolean, string, etc, and arrays of primitive types.

- *Topics*: *Topics* look like buses, in which the messages can be exchanged between nodes. They are based on "publish/subscribe" transport system. A node can publish a message through a topic, so that an interested node will subscribe to this topic. For interacting with topics, ROS provides the `rostopic` command-line tool. The command `rostopic pub topic_name msg_type data`, for example, publishes data to a given topic.

- *Services*: ROS provides also a "request/reply" transport system via *services*. A client can call a service by sending a request message, from a node that acts as s server, and awaits the reply. We can discover information about active services using the `rosservice` command-line tool.

- *Bags*: Message data in ROS are stored in a file called a *bag*, which can be used for recording and playback.

Filesystem level

The *Filesystem* level comprises the resources stored on the disk, which are organized as follows:

[6]To run a node, we generally use the command `rosrun package_name node_name`.

- *Packages*: The main unit of ROS software organization is the *packages*, which contain runtime nodes, a ROS library, configuration files, etc. The main directories and files in a ROS package are: *src, scripts, msg, srv, package.xml, CMakeLists.txt*.
- *Package Manifests*: A *package manifest* is a file in XML format called "package.xml". It describes the package metadata like its name, version, authors, and dependencies.
- *Message types*: The ROS message descriptions are stored in files with `.msg` extension in the *msg* directory of a package.
- *Service types*: The definition of ROS service types is stored in files with `.srv` extension in the *srv* directory of a package.
- *Metapackages*: A *metapackage* in ROS is used to reference related packages. It contains only the package manifest.
- *Repositories*: A *repository* is a collection of packages.

Community level

The ROS *Community* Level covers the set of sources that enable the knowledge and software to be exchanged and reused between communities. This is mainly given by the ROS Wiki pages [2] and its other forums and sites such as ROS repositories, and ROS Answers[7].

2.3.3 Difficulty in learning ROS

The main drawback about ROS is the difficulty in learning and in starting with simulation and robot modeling [20], which can be noticed by ROS users. Indeed, even with the ROS huge command-line tools and wiki pages, it is difficult to learn ROS and use it. A user will spend a lot of time in learning and discovering new skills, or in testing existing examples to obtain desired actions for his robot, which make the possibility of developing robotic application quickly very hard. Therefore, this is the main reason why SOA is introduced towards improving the ROS software use.

[7]ROS Answers: `https://answers.ros.org/questions/`.

2.4 Service-Oriented Architecture and Web services

Web services were regarded as an acceptable manner for implementing the architectural style "Service-Oriented Architecture" over the last twenty years. These two concepts are briefly presented in the following.

2.4.1 Service-Oriented Architecture (SOA)

Service-Oriented Architecture, or SOA, is an approach that provides distributed systems, based on usage of software components called "services". It allows mainly these services to be discovered over the internet, through published interfaces. As defined by the W3C [21], SOA is *"a set of components which can be invoked, and whose interface descriptions can be published and discovered"*[8].

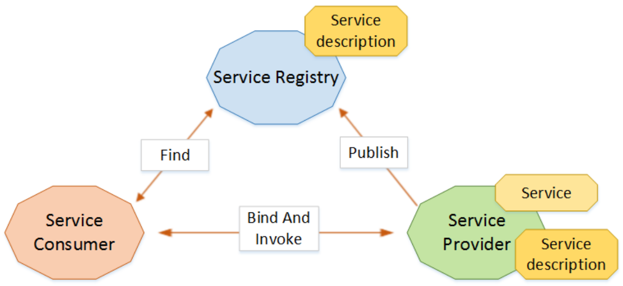


Figure 2.2: SOA Architecture.

The main integrated components in SOA architecture are illustrated in Figure 2.2 [23]. As shown in the figure, the service provider makes services available and accessible to service consumers[9], by publishing the interfaces of services via a service registry. The relation between consumers and providers of services may be specified and guaranteed by a formal agreement, known as Service Level Agreement (SLA), in order to ensure the Quality of Service (QoS) criteria [24, 25]. The QoS includes many characteristics about the service and performance criteria such as response time, availability, security, throughput, etc.

In the context of SOA, a service is *"a discrete unit of business functionality that is made available through a service contract, which specifies all interactions between the service consumer and service provider"* [24]. Services are described by many characteristics, like [24]:

[8]There are many other definitions for the term SOA. A set of additional definitions are listed in [22].
[9]In SOA, a service consumer is a software or other services that need a service.

- *Encapsulation*: All the internal implementation of service operations is hidden from its published interface.
- *Loose coupling*: This refers to the dependency degree between modules. There are few dependencies within loosely coupled services.
- *Autonomy*: Autonomy means that services are independent from each other in terms of deployment, modification, and maintenance.
- *Reuse*: Service reuse deals with the sharing of building blocks of software, which is the main SOA aspect for process construction.
- *Dynamic discovery and binding*: Services are discoverable entities. The service consumer can be dynamically routed and bound to the proper service provider, by inquiring the service registry for a service that matches its criteria.

2.4.2 Web services

Web services are considered as a way for SOA implementation. Independent of any programming language or platforms, web services offer a distributed computing technology for the integration of the heterogeneous applications over the Internet [23]. Thus, they offer loosely coupled application components enabling interoperability and reuse of services. In general, realization types of web services encompass two major ways of service implementation [24], [26]. The choice for the way of service implementation is depending on use aspects as presented in the comparison below [25].

The first type are the services that agree with Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL). SOAP web services are also referred to as WS-*. WS-* implementation has great support for QoS such as security and availability. Furthermore, it defines many standards including: (*i*) an infrastructure for service composition[10], (*ii*) transactions, (*iii*) service discovery, and (*iv*) reliability.

As a protocol, SOAP is a form of XML specification that relies on HTTP and RPC for the transmission of messages. A SOAP message is structured in an *envelope* and a *body* in XML format. SOAP offers an interoperability between applications due

[10]SOAP can use the Business Process Execution Language (BPEL) which allows, among others, to compose web services.

to this standardization of messages in which the implementation features are not mandated.

The second type are those described as Representational State Transfer (ReST or REST) style services. REST is a client-server based architectural style, where the information network is identified and addressed by a URI scheme. The communication between an HTTP client and server is accomplished using a small operations collection known as: *read*, *create*, *update*, and *delete* operations. REST is a stateless protocol and meant to be self-descriptive, in which REST interfaces gives only a syntactic interoperability between services.

On top of HTTP, only few characters are used for the exchange of messages in REST contrary to SOAP. Therefore, REST performs better than the SOAP structured messages in case of systems that exchange lots of messages. However, unlike SOAP, REST is more suitable for read-only functionality that requires minimum QoS requirements. Ultimately, SOAP expresses "*completeness*" while REST expresses "*simplicity*".

2.4.3 Semantic Web services

Existing Web service description standards such as WSDL that employ XML structures to describe the services' functionality are syntactic structures. This might provide a certain level of ambiguity, in which the same structure may be read in different ways by various users. To address this issue, semantic web services bring the application of semantic web technologies to the description and use of web services. A semantic Web service is a web service that is described using semantic annotations in a well-defined language such as ontologies[11]. It enables the services to have an interpretable interface and to facilitate the automation of specific tasks such as discovery, selection, invocation and composition [27]. There are several ontology description languages for semantic Web services have been developed. The well-known languages are Ontology Web Language for Services (OWL-S) [28], which is based on the OWL language, and WSMO [29].

[11]An ontology provides a way of defining a specific domain's core concepts and features.

2.5 Opportunities of cloud computing

Basically, from a technological development perspective, cloud computing is a combination of different research axes. It combines a set of technologies such as grid computing[12], virtualization[13], and even SOA [30–32]. Thus, cloud computing is a logical continuation of a number of computer technologies [33]. The final cloud computing definition and its main assets have been published by the NIST in 2011, in [34].

2.5.1 Cloud computing basics

As given in [34], cloud computing is *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*. The NIST determines three major elements (characteristics, deployment models, service models) that compose the cloud model as presented in Figure 2.3.

Primarily, cloud computing is distinguished by five basic characteristics that describe its scope. For enabling pooling of computing resources, consideration had been given to multi-tenant model[14] to serve the numerous consumers as often as needed. This is made using different physical and virtual resources. The provision of resources is elastically built to rapidly scale inward and outward proportionate. Thus, from point of view of a consumer, the capabilities appear to be unlimited. Resource usage can be measured by the consumption of storage, processing, bandwidth, or active user accounts based on "pay-per-use" basis that determine the consumer charges.

As shown in Figure 2.3, there exist four deployment models for cloud infrastructure functioning: private, community, public, and hybrid. As their names indicate, each model is distinguished depending on how a cloud consumer may be

[12]Grid computing is a distributed computing form that combines networked computers acting with each other to perform very large tasks.

[13]Generally, virtualization refers to the process of creating multiple Virtual Machines (VMs) on a single physical device. This allows to run multiple instances of operating systems on one computer system concurrently. For more information see section 4.2.2 of chapter 4.

[14]Multitenancy enables sharing of resources among consumers to support many concurrent users at once.

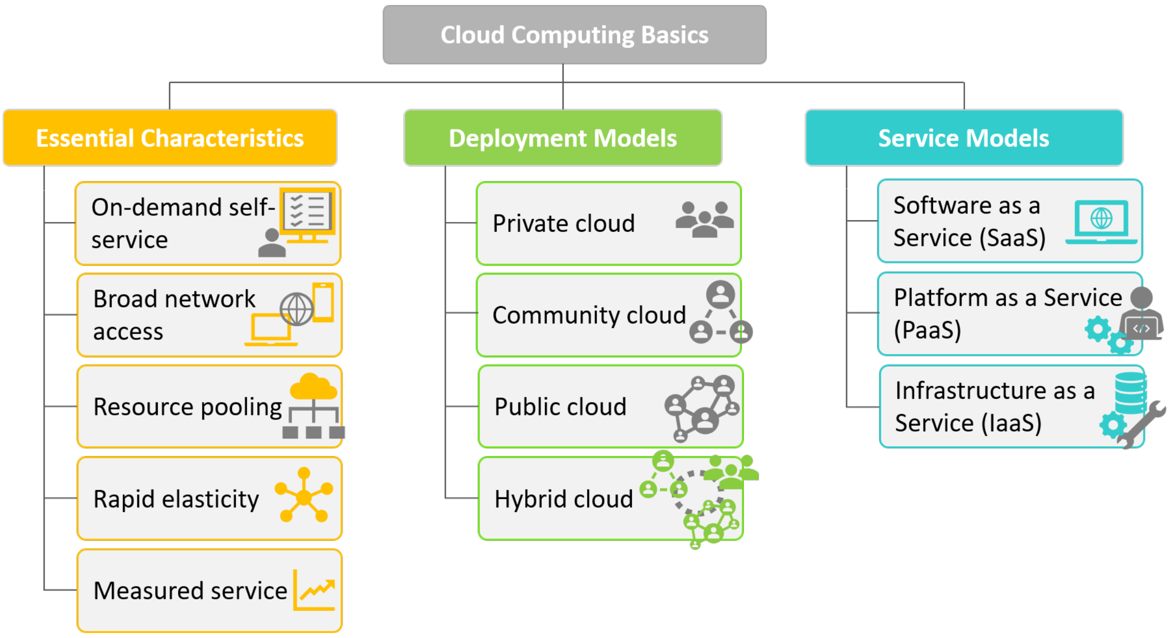


Figure 2.3: NIST basics of cloud computing.

allowed to access the computing resources. The NIST definition comprises three models for service provisioning: (*i*) Software as a Service (SaaS), (*ii*) Platform as a Service (PaaS), and (*iii*) Infrastructure as a Service (IaaS), in which providers and consumers of cloud systems share the control of available resources, as illustrated in Figure 2.4 [35]. However, Everything as a Service (XaaS) [36] has also been introduced as an exhaustive denotation to express other emerged service models. These models are briefly highlighted in the following.

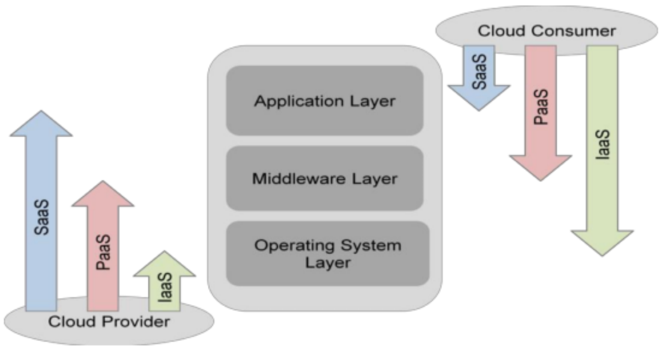


Figure 2.4: Scope of controls between providers and consumers in a cloud system. For instance, a middleware layer enables PaaS consumers to develop their application software. On the other hand, this layer is hidden from SaaS consumers and managed by IaaS consumers or PaaS providers [35].

Software as a Service

The services provided to users with SaaS encompass *applications* and *data* as Web-based applications. In a SaaS environment, the provider is basically responsible for everything about storage, system monitoring, and so on [37]. There are many SaaS providers such as *Outlook.com*, *Google Drive*, and *Salesforce.com*.

Platform as a Service

PaaS provides an operating system with platforms of development and database to the customers. Examples of PaaS providers can be given by *Windows Azure* and *Google App Engine*.

Infrastructure as a Service

IaaS offers basic services such as computing power, storage, networking, and operating systems, where the customer can construct his environment above. A well-known IaaS provider is *Amazon EC2* [37].

Everything as a Service

There are various emerging delivery models that has been identified, in the course of time, to provide anything "*as a Service*" via the internet. XaaS, the acronym for Everything as a Service [36], covers not only the above three known models but also any kind of provided computing services. This encompass a wide variety of computing resources or capabilities including applications, tools, data, communication and so on. For instance, Discovery as a Service (DaaS) [38], Robot as a Service or RaaS (see section 3.2.1 of chapter 3), and PHM (Prognostic and Health Management) as a Service [39] are illustrations of XaaS concepts.

2.5.2 SOA and cloud computing

Indeed, one of the key elements in architecting and delivering cloud services is SOA [40]. SOA constitutes one of the major assets of the cloud technology. As described in [41], "*SOA is to offer services which are based on open standard Internet services and virtualisation technology and have been running in a different environment, grid offers services from multiple environments and virtualisation, and cloud combines both*".

SOA guides business process management for creating, organizing, and reusing the computing components. It provides independent, reusable application functions as services. On the other side, cloud computing is more technical model that addresses many technical details, which services a bigger and more flexible platform [32, 42]. The five characteristics defined by the NIST for cloud computing [34] must be satisfied in a cloud computing service, but they considered as optional factors in SOA services [32].

Thus, the relationship between cloud computing and SOA can be compared with the relationship between web services and SOA [32, 43]. SOA can be used without or with web services. Similarly, cloud computing uses SOA, however, it is possible to implement cloud computing without having an SOA [43].

2.6 Cloud Robotics: An overview

With the rise of the World Wide Web and Internet, many initiatives have been revolved around the connection of robots to the global network. Robot teleoperation via internet browsers, remote computing for robot control, and networked robotics[15] received much attention since the 1990's [1]. Over the last decade, the focus of robotic systems development is oriented to the integration of cloud computing properties, where robots can meet their needs as an on-demand solution. This refers to "Cloud Robotics" term that was announced for the first time in 2010, in [44], by James Kuffner [1].

Various projects (e.g. the European Union funded "RoboEarth" [45] and "RAPP" [46] projects) and architectures (e.g. [47–57]) have been designed and developed around the cloud robotics topic. Indeed, two fundamental goals were addressed: (*i*) knowledge sharing among robots, and/or (*ii*) offloading robotic computational tasks to the cloud. However, diversity of proposals can be observed from one work to another. This is mainly related to the various robotic fields and case studies on one hand, and to the variety of used technology innovation, including architecture styles, protocols, and frameworks on the other hand. We summarize the scope of a sample of these different works in Table 2.1.

As stated in [62, 63], there are some challenges and issues in cloud robotics that should be addressed. This includes: (*i*) Resource allocation and scheduling, (*ii*)

[15]Networked robotics is the study that aims to communicate the robots with each other.

Table 2.1: Overview of cloud-based projects and architectures in robotics

Work	**Address the idea of**	**Overview**	**Applications/Case study**
RoboEarth project [45]	"World Wide Web" (WWW) for robots	RoboEarth enables robots to share and reuse data, across a distributed database. This includes data about objects, maps, task knowledge and object recognition models. Also, it allows the robots to offload heavy computation by providing an open-source cloud robotics platform called "Rapyuta" [58].	Demonstrations of sharing knowledge of RoboEarth are oriented to service robots that assist humans in their daily activities. Knowledge representation is based on the knowledge processing system "KnowRob" [59]. It provides vocabulary about objects and actions as an ontology for robot task execution.
RAPP project [46]	"Robots as Platforms"	RAPP provides an open-source software platform to support the creation and delivery of robotic applications. The cloud RAPP platform can be utilized by the client robots across HOP Web services and rosbridge [60]. Further information about RAPP architecture can be found on [61].	RAPP applications are intended to meet the needs of people at exclusion risk, especially those of old age, using robots. Connecting this category with family and friends, giving information about news, etc, are examples of these applications.
[48]	IaaS for robotic applications	The work presents an IaaS architecture, which supports network-level virtualization and delegation of tasks to robots that belong to multiple clouds. Each task is divided into a set of sub-tasks to be assigned to selected robots by IaaS via gateways.	The system prototype was implemented using multi robots with different capabilities in a wildfire suppression scenario.
[50]	Layered cloud robotics architecture	The authors proposed a new cloud robotics architecture that is inspired by classical cloud computing architecture. A cloud-based Virtual Robot Layer was defined to offer an abstraction of robots features and hardware.	Private cloud for both virtual robotic and management systems.
[52]	"Robot Control as a Service "	The authors proposed a cloud-based control for industrial robot. The system parts run on interconnected computers or virtual machines using a VPN independently.	Algorithms like trajectory-planning and collision detection services have been experimentally verified in the Pick & Place scenario.
[53]	Cloud manufacturing	Wang et al. developed a cloud robotics application for ubiquitous manufacturing. The coordination and supervision of the whole production system is handled by a local server, which connects a physical resource layer with the cloud layer.	(*i*) Machining service, (*ii*) On-line robot control in assembly tasks case (for providing safe environment for human operators), (*iii*) Minimization for energy consumption of an industrial robots movements.
FIL [56]	Imitation Learning	An imitation learning framework for cloud robotics was introduced in this work. It enables the cloud to fuse heterogeneous knowledge from local robots and produce learning guide models for them, using a federated learning algorithm.	The framework and algorithms are verified with a self-driving example.

Data heterogeneity and their exchange between robots and cloud platforms, (*iii*) Data privacy and security, and (*iv*) Network latency.

2.7 Conclusion

This chapter outlined our thesis's main axes: ROS, SOA, web services, and cloud robotics. We have presented specifically the fundamental concepts and definitions that are related to these concepts. We have also introduced the cloud robotics domain by presenting a set of its architectures. The use of ROS will be presented more in chapters 4 and 5. The following chapter aims to present a review that underline the Service-Oriented Robotic Architectures.

Chapter3

Service-Oriented Robotic Architectures

3.1 Introduction

In this chapter, we investigate the works discussing the use of SOA, Web services and cloud computing concepts, as main sources of robotic services. We carried out a comparative analysis that targets their objective, scope, architecture, robotic service design, application and evaluation criteria. The review process starts with keywords such as (*"Robotic"* OR *"Robot"* OR *"Cloud Robotics"*) AND (*"Services"* OR *"Service-Oriented"* OR *"Discovery"*) AND (*"Architecture"* OR *"System"* OR *"Software"* OR *"Approach"* OR *"Applications"* OR *"Middleware"*). The identified and selected articles for this review shows that the works in this area are published with different research proposals that are diverse in many levels of both system conception and development.

Through a comparative analysis, this chapter is drawn on three main axes. In section 3.2, we propose a classification of the literature of service-oriented robotic architectures into four main models. Then, we conduct a comparison about the description, discovery, and applications of robotic services for exhaustive state of the art of each proposed model in section 3.3. Furthermore, the main issues of the reviewed works are discussed in section 3.4.

3.2 Service-oriented robotic models

The ongoing efforts to define and improve the research of SOA and cloud-based robotic solutions result in the appearance of various studies. Preliminary work in this field focused primarily on designing flexible software for robotic systems, and it grows vaster for further improving existing problems and developing new approaches. However, a huge diversity can be observed from one study to another. Indeed, it is difficult for readers to identify and follow the comprehensive analysis of the domain due to the heterogeneous scopes, architectures, and used technologies.

To address this issue, we conducted in this chapter a global overview on the emergence and evolution of research directions. We classify the existing service-oriented approaches and architectures for robotic service delivery based on the scope of each proposal. Therefore, we identify four major service models as illustrated in Figure 3.1. The diversity is revealed in the significant difference between the propositions with regard to the provision of robotic services, which is introduced through the as-

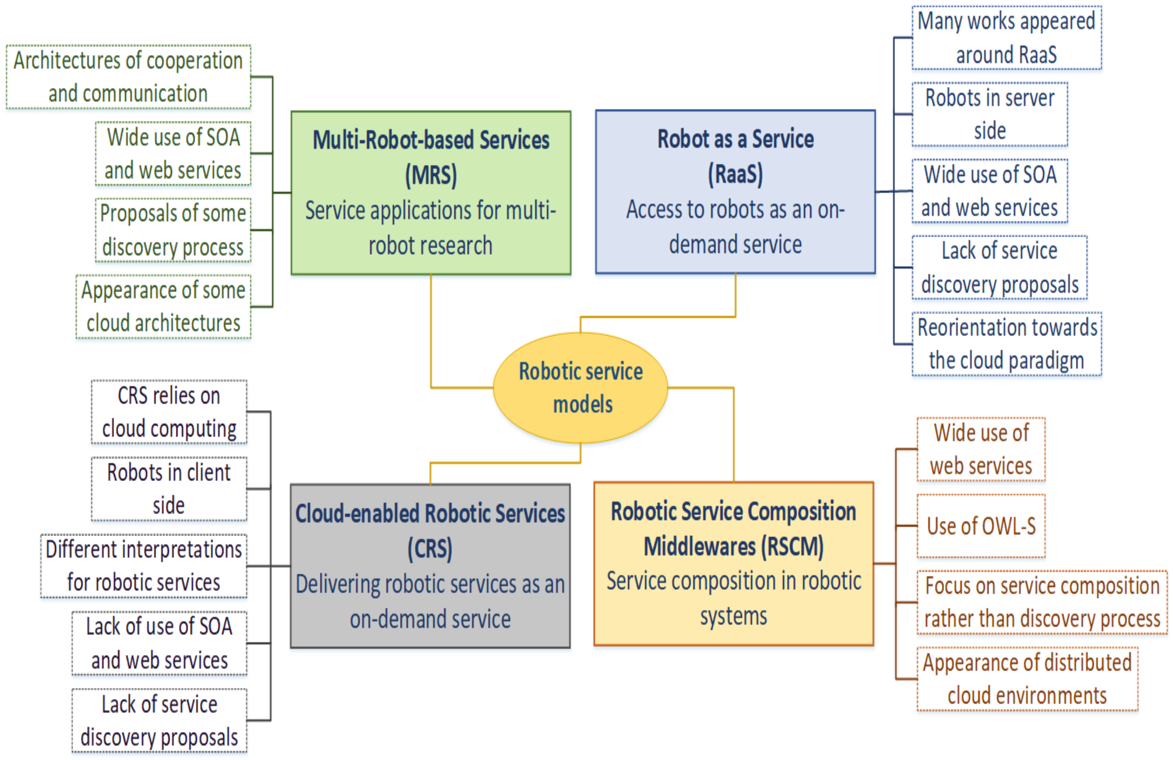

Architectures of cooperation and communication
Wide use of SOA and web services
Proposals of some discovery process
Appearance of some cloud architectures
Multi-Robot-based Services (MRS)
Service applications for multi-robot research
Robot as a Service (RaaS)
Access to robots as an on-demand service
Many works appeared around RaaS
Robots in server side
Wide use of SOA and web services
Lack of service discovery proposals
Reorientation towards the cloud paradigm
Robotic service models
CRS relies on cloud computing
Robots in client side
Different interpretations for robotic services
Lack of use of SOA and web services
Lack of service discovery proposals
Cloud-enabled Robotic Services (CRS)
Delivering robotic services as an on-demand service
Robotic Service Composition Middlewares (RSCM)
Service composition in robotic systems
Wide use of web services
Use of OWL-S
Focus on service composition rather than discovery process
Appearance of distributed cloud environments

Figure 3.1: Service-oriented robotic models.

pects of each category. We review the key concept, contribution and related works of each model as follows.

3.2.1 Robot as a Service (RaaS)

As the “Robot as a Service (RaaS)” term indicates, this concept reflects on the use, and the access to the robots as an on-demand service throughout the Internet. Users at the client side can manipulate their server-side robots by handling robotic software such as algorithms of robot navigation remotely. SOA and web services technologies have played a significant role in building the background fundamentals of RaaS, which has known a significant growth in its implementation.

RaaS concept was invented for the first time by Chen et al. in [64] with the intention of building a kind of robot that has the function of the SOA architecture as cloud units, as shown in Figure 3.2. As a proposed implementation of this idea, users were able to manipulate robotic services and applications remotely, using the robotic environment MRDS. Subsequently, various initiatives were taken around the design of RaaS.

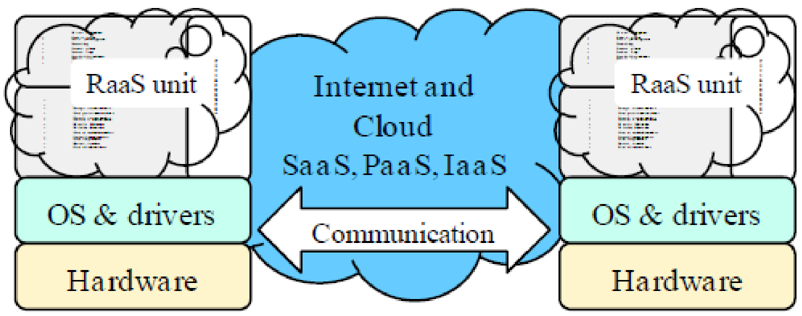


Figure 3.2: RaaS in cloud environment [64].

The concrete application of this model was given initially through the remote laboratories and remote monitoring of robots [5, 64–69]. However, quite recently, RaaS implementation was oriented to provide more flexible services by offloading computational capabilities, storage, and communication of various applications to the cloud using SOA [54,70–72], or without it [73–76].

In [54] for instance, Du et al. introduced the idea of "Robot Cloud" that integrates RaaS into the cloud platform. As the authors pointed, unlike James Kuffner's cloud-enabled robotics (see section 2.6 of chapter 2), the work's objective is to form robots as an integral element of the cloud computing service. Robots can communicate with one another, instead of acting as separate entities capable only of exchanging data with remote servers. The authors develop a prototype, using the popular Google App, and a mechanism of scheduling robot services that adopts the benefits of SOA.

Recent study [72] uses RaaS to facilitate the seamless integration of robots into a cloud robotic system that enables the switching between several tasks using shared data. The mechanism is conducted by transmitting the surrounding environment information sensed by the robot to a fog[1] service node in order to receive the analysis result and the control information. More recent studies have also focused on network communication improvement for the effective control of robots by emphasizing on several protocols including ROSLink and MAVLink [71], MQTT and CoAP [78], REST [79], and WebSocket protocol for implementing a 'plug-and-play' solution [80].

[1]Fog computing [77] is considered as a form of computing that enables computing, storage, networking closer to the users.

3.2.2 Cloud-enabled Robotic Services (CRS)

Unlike RaaS model, some of the architecture designs push the vision of obtaining and consuming hosted services for robots, rather than accessing and controlling them. We refer to this group as the "Cloud-enabled Robotic Services (CRS)". Similarly to works that support the cloud robotics paradigm, this approach adopts particularly on the same process in data exchange and acquisition between robots and clouds. The cloud storage provides a service for robots where the data about tasks, environments, users, etc., are available and remotely accessible over a network, usually the internet.

Works based on this approach are quite diverse. Heterogeneity of the proposals is mainly related to robotic services representation and how these services can be used or accessed. For instance, the interpretations of the robotic service concept are outlined by the following contributions:

RSi-Cloud

For providing more useful and attractive services, Kato et al. proposed "RSi-Cloud" in [81], in order to combine current internet services with robot services. RSi defines robot services as "*information services and physical services provided via computer networks*". This definition consists of providing robot services on the cloud, and robot applications on the robots using RSNP protocol. Numerous services may be provided to different robots due to service profiles classification. Another implementation of this proposition can be found in [82].

Cloud Networked Robotics

This research [49] is intended to overcome the limitations that cannot be met alone or in conjunction with networked robotic services. As reported in [49], robotic services were considered as "*systems, devices, and robots with three functions: sensation, actuation, and control*". The work's attention was to gather logically such devices to create a cloud of robots via network connectivity in order to achieve an integrated system for supporting daily activity using the available resources on demand. Using the Ubiquitous Network Robot Platform (UNR-PF) [83], as a layer of intermediary between the service application and robotic components, a case study in a shopping mall was released in order to enable multi-location robotic services in daily activities.

Robotic Service as a Service (RSaaS)

Under the name of "Robotic Service as a Service" (RSaaS), authors in [84] distinguish the model in which "*the robotic system deliver high-order services to the end-users to complete a robot task*". Examples of this model are given by social robots that interact with users who need news, information, etc., by connecting to the cloud. RSaaS model can be given by the RAPP project [46] (see Table 2.1 of chapter 2).

Robotics and Automation as a Service (RAaaS)

An implementation of this concept is presented in [85]. "*RAaaS was introduced in [1], and it is the robotics equivalent model of SaaS. In RAaaS the software modules are stored on a central cloud server and provided to the users/robots over the internet*" [85]. A SaaS-based architecture was also proposed in [86] who intend additionally to make the source code publically available for users as a PaaS Implementation. The proposed framework is applicable to systems with a single or multi robots, where robots are linked to a JavaScript-based cloud server via use of web sockets.

Robot Inference and Learning as a Service (RILaaS)

Authors in [87] introduced a RILaaS platform for offering user-based inference in order to deploy models of deep learning. It distributes the queries for trained models either over cloud or edge[2] based on their resource consumption. RILaaS experiments were presented for serving deep models of both grasp planning and object recognition as a service.

SOA-based architecture

Initiatives around providing reusable robotic services in CRS with SOA can be found in [88], [89], [90], [91]. In [91], a web service-based layered cloud robotics architecture that is inspired by [50] was proposed. The approach integrates web services technologies to describe robot packages as an on-demand solution. In [92], Oliveira et al. developed a tool that enhances service cataloging and discovery for ROS. The search strategy is built on the basis of a semantic taxonomy with common vocabularies that organize knowledge about the domain. The work is presented without

[2]Edge and fog computing share the concept of computation to the edge of the network, but [77] provides a distinction, in which fog computing further involves computing, networking, storage, and control.

integration of cloud computing technologies, nevertheless, it can be considered as a step for cloud services search.

By developing respectively context-aware and emotion-aware dialogues for the development of robot dialoguing services to achieve natural human-robot interactions and enhance user experiences, a cloud-based SOA is adopted in [93] and [94]. The proposed frameworks target two types of dialogue services: (*i*) domain-specific dialogues that provide knowledge services of a certain domain through a question-answering manner between the user and the robot, and (*ii*) task-specific dialogues that accomplish a given task's objective by iteratively performing human–robot dialogues to adapt to the user's purpose or preference in relation to the task's goal.

3.2.3 Multi-Robot-based Services (MRS)

Multi-robot research is the study that focuses on the cooperation of robots with each other to perform tasks, which would be difficult or impossible to be accomplished by individual robots. We refer to service-oriented applications in such systems as "Multi-Robot-based Services (MRS)". The main addressed aspects in MRS are as follows:

Cooperation architecture in MRS:

One of the challenges in MRS is to design an architecture that has the ability to provide a common way to deal with the heterogeneous hardware and software of robots. Thus, this is an essence of bringing SOA into this sector in diverse application domains such as collaboration of mobile service robots (e.g. [95], [96]), robotic swarm system (e.g. [97], [98]), and UAV (Unmanned Arial Vehicles) applications (e.g. [99–101]).

A layered multirobots cooperative architecture with SOA was introduced by Cai et al. [102], as shown in Figure 3.3. At the start of the multi-robot cooperative serving process, a service applicant provides its service descriptions to the layer of service translation in order to get standard descriptions. Therefore, this layer is in charge of translating dissimilar communication language into corresponding service description with a uniform format, which allows multi-robots to interact with one another. Then, the standard descriptions are sent to SOA interface layer, where the descriptions and additional application data will be packaged and submitted to service registry center. When the application is received by the service registry center, it will

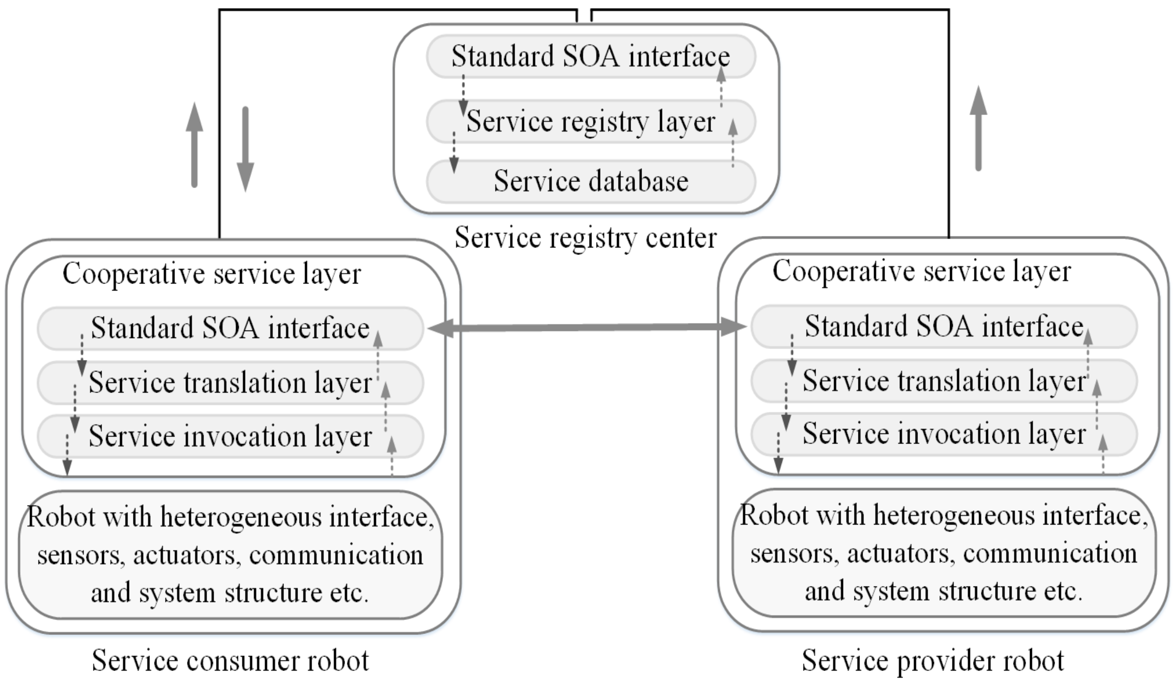

Standard SOA interface
Service registry layer
Service database
Service registry center
Cooperative service layer
Standard SOA interface
Service translation layer
Service invocation layer
Robot with heterogeneous interface, sensors, actuators, communication and system structure etc.
Service consumer robot
Cooperative service layer
Standard SOA interface
Service translation layer
Service invocation layer
Robot with heterogeneous interface, sensors, actuators, communication and system structure etc.
Service provider robot

Figure 3.3: Layered multi-robots cooperative architecture [102].

send the information and service licenses of the most appropriate providers to the service applicant to connect with.

In [103] and [104], authors aim to develop a web-based entity that uses service robots represented as collections of web services to create and execute plans for completing tasks provided by a user. A centralized control architecture, used for the allocation in the shortest possible time of numerous robots, was presented. To represent the task knowledge, OWL-S was combined with Prolog and ROS Python using different ontology-based knowledge representations.

Communication and Service scheduling in MRS:

The communication in MRS refers to applications of networking that allow robots to dynamically discover each other. The framework of distributed coordination of a robot swarm [105], and the middleware for mobile robots inside a fleet in an ad-hoc network [106] are examples of this application that exploit the Universal Plug and Play (UPnP) for service discovery protocol.

The robots' communication with the cloud was addressed in [107]. The aim of the study was to introduce a SOA architecture that is distributed on cloud to address the issue of localisation in cooperative multi-robot systems. Differently, Zhou et al.

in [108] proposed an algorithm of Circular Area Search (CAS) for the scheduling problem of regional service in SOA cloud platform for multi-robot service.

Some authors, as in [109] and [110], address the issue of cloud-based multi-robotic services in real-time applications such as fire emergency management, which necessitate instantaneous responses and transmission of data between robots and the cloud. The solutions exploit the concept of Edge Cloud [77] to address mainly the communication latency constraint, via the execution of the latency sensitive and services requiring a high level of computation closer to the work environment.

3.2.4 Robotic Service Composition Middlewares (RSCM)

Robotic Service Composition Middlewares (RSCM) are the set of development platforms that address the issue of service composition for robotic systems. Service composition "*consists of creating new complex services by combining the existing atomic services that cannot satisfy the needs of complex robot tasks*" [111].

Multiple works have been designed around RSCM by exploiting the web service architecture, using OWL-S to describe knowledge about services semantically. This includes development platforms for robot action sequences [97, 98, 112–114], [103, 104], and those that enable robots to utilize heterogeneous resources in ubiquitous computing [115–117] and ambient intelligence [118–120] environments.

In a different context, cloud-based architecture for RSCM have focused on presenting architecture for distributed and virtualized environments [121], [122], [123], [124], [125]. This involves different context of applications including scheduling [121], resource allocation [125], integration of robots in ambient [120] and cyber-physical [124] systems, and others [103, 104].

3.3 Description, discovery, and applications of robotic services

In the context of the service reuse principle, discovery is critical [26]. Reuse of services requires us to find the services that exist, and to examine them to decide if they perform the functions required, provide the appropriate qualities of service, are reliable, and so on [24]. Therefore, the first step toward service reuse and consumption is discovery [26]. For this purpose, The service must be published along with a collection of descriptive data to be effectively searched in response to criteria-driven queries [26,126]. This information consists of (*i*) functional meta data that describes what the service is capable of, and (*ii*) QoS meta data that encompasses behavioral characteristics, limitations, and interaction requirements [126].

Generally, there are two forms of discovery: manual process and automatic process called "runtime discovery" [126]. Runtime discovery provides programmatic interfaces into service registry repositories that build programs and services capable of issuing dynamic discovery queries [126]. However, Rosen et al. [24] pointed out that service discovery does not necessarily have to be based on a repository, but should provide mainly:

- A catalog of available services.
- Ability of identifying potential services through sophisticated search capabilities.
- Capabilities for examining a service in order to identify if it is appropriate for the desired usage.
- Metrics on service usage.

In robotics, there is only few researches that have addressed the issue of service discovery as a research axis, such as in [105] and [92]. However, most of the works of RSCM model that have focused on web service composition have indicated a search process, by using ontologies for identifying the services to compose. Also, service discovery for robots has been announced in some works of MRS model to dynamically discover each other. On the other hand, there is a lack of service discovery proposals in both RaaS and CRS models.

To highlight this issue, a comparative analysis with the reviewed papers of different models is given in the initial and continued parts of Table 3.1. The comparison is made according to:

1. Model: That targets the model of each work according to the four identified service models to define the scope and focus of works.
2. Software process: The software process encompasses the details of software proposals and system design. This includes the service representation, its requirements and meta data, service description language, communication protocol and the discovery process, in which we have indicated if the similarity

criteria between services and compatibility of services with different robots are discussed or no.

3. Cloud deployment: Respectively, cloud deployment outlines: (*i*) the cloud service model, i.e. SaaS, PaaS or IaaS, (*ii*) the way of realizing the cloud solution, and (*iii*) the robot-cloud communication mechanism.

4. Case study and Robotic tools: Which summarize the experimental robotic settings, including the robotic system, case studies and the used robots.

5. Experimental analysis: That describe how the results are evaluated according to the evaluation criteria and comparison with other works.

As we can observe from the parts of Table 3.1, each solution is addressed with different aspects, thus, there is no standard use in the the software process from the conceptual perspective, among others, the representation of robotic services (e.g. [92], [122]). Additionally, each study targets specific scope factors and it has its own data and conditions of implementation for building the service-based middleware, in which the experimental results are discussed. Therefore, this is the reason behind the absence of experimental comparison criteria, contrary to domains that provide benchmarks and standard data for comparison and validation. Nevertheless, it is essential to emphasize that ROS is among the most widely used frameworks, in particular, for service robots, which provides a standard development framework for robotic software, and allows reuse of codes between.

Table 3.1: Comparison with state-of-the-art frameworks

Work	Model	Software process						
		Service representation	Service description	Requirements/ Meta data	Service knowledge storage	Communication protocol	Discovery process	Selection
[71]	RaaS	Web service	WSDL/–	–	–	SOAP/REST	–	–
[54]	RaaS	Robots themselves	WSDL	–	–	–	Residual energy of robots	State of robots
[76]	RaaS	IaaS modules	–	Designed Unified Description Model for Robots	Robots Repository, Service Marketplace	REST	Robots' presence information	Coalition formation algorithm
[79]	RaaS	Mission: series of tasks	–	–	–	REST	–	–
[80]	RaaS	–	–	–	–	–	–	–
[46]	CRS	Hosted applications (RApps)	–	Description	Knowledge ontology	–	User interpretation and keywords	User interpretation
[85]	CRS	Hosted remote software	URDF/ SRDF	–	–	–	–	–
[92]	CRS	On-line robot codes	–	Name, description, type, provider, quality attributes and others	On-line repository with proposed taxonomy	–	On-line search and search Plug-In based on Task type criteria	User interpretation, QoS (Reputation)
[91]	CRS	Web service	WSDL	Web services interfaces	Robotic UDDI (R UDDI)	SOAP	User interpretation	User interpretation
[94]	CRS	Domain/task-specific dialogues	–	–	Dataset	–	Deep learning/–	Similarity score/–

Table 3.1: Comparison with state-of-the-art frameworks (continued)

Work	Cloud deployment			Case study and Robotic tools		Experimental analysis	
	Cloud model	Cloud realization	Robot-cloud communication	Robotic system	Case study	Evaluation criteria	Comparison criteria
[71]	SaaS	Real-time performance and networking	MAVLink / ROSLink	ROS	Drone-based realtime tracking	Real-time performance, Tracking accuracy	–
[54]	PaaS	Prototype using Google App Engine	–	–	–	Performance of proposed scheduling vs exclusive scheduling, Impacts on scheduling	–
[76]	IaaS	LAN-based machines to host IaaS	–	–	Search and rescue prototype (Fire suppression)	Cost of federating several IaaSs (delay factor)	–
[79]	–	–	–	ROS	Tasks for drone manipulation	–	–
[80]	–	Public central server	WebSocket, SSH, Rosbridge	ROS	Teleoperation of speed commands	Reliability, quality of simultaneous teleoperations	Spiral trajectories performance (simulation)
[46]	PaaS	Open source software platform	Rosbridge	ROS	Social applications for people at risk of exclusion	–	–
[85]	SaaS	Real-time performance over remote cloud server	VPN	YARP/ ROS	Tasks of perception and robot control	Cloud network analysis	–
[92]	–	–	–	ROS	Navigation functionalities	Discovery results	–
[91]	SaaS	Private cloud	–	ROS	Tasks for robot manipulation	–	–
[94]	–	Private cloud parallel computing VM	–	ROS	Use of datasets for question-answer/ and restaurant recommendation dialogues	Performance metrics, Loss (root mean squared error)	Distance and performance/ Loss

Table 3.1: Comparison with state-of-the-art frameworks (continued)

Work	Model	Software process						
		Service representation	Service description	Requirements/ Meta data	Service knowledge storage	Communication protocol	Discovery process	Selection
[105]	MRS	Semantic web service	OWL-S	Context information	Robot Ontology	UPnP	IOPE-matching (context based-bipartite graph matching)	Matching scores
[107]	MRS	Robots themselves/ Localization service as web service	–/WSDL	IP address, velocity, name, status	Database/ UDDI	SOAP	Robot name	–
[109]	MRS	Robotic tasks	–	Data dependency delay, QoS	Edge Cloud	–	–	–
[112, 114]	RSCM	Semantic web service	OWL-S	Description, QoS	Task ontology	SOAP/REST	IOPE-matching	QoS (Reputation)
[121]	RSCM	Web service	WSDL	Keywords, description, ROS packages, nodes and topics	Function and task models (Non-relational database)	SOAP	Keywords	ROS node name
[122]	RSCM	Semantic web service/ Image (matrix of cells)	OWL-S/–	Description, QoS	Task ontology/ Database	–	IOPE-matching/–	QoS (Reputation) /–
[123]	RSCM	Microservice	–	–	Registry	REST	–	–
[125]	RSCM	Web service	WSDL	QoS: availability, reliability, duration and execution price	–	SOAP	–	QoS score

Table 3.1: Comparison with state-of-the-art frameworks (continued)

Work	Cloud deployment			Case study and Robotic tools		Experimental analysis	
	Cloud model	Cloud realization	Robot-cloud communication	Robotic system	Case study	Evaluation criteria	Comparison criteria
[105]	–	–	–	–	Search and rescue (searching survivors) using swarm of robots	Matchmaking performance, Impacts on matchmaking	–
[107]	SaaS	Private cloud and VMware Workstation	–	–	Localization of a remote mobile robot	Processing according to the number of clients, Performance of the service for localization	–
[109]	–	Edge Cloud	–	–	Emergency management service in smart factory (simulation)	Impact of: edge cloud, varying number of tasks, varying number of computing resources	Performance (optimization of objectives)
[112, 114]	–	–	–	Simulation (OpenRAVE)	Daily activities in home environment	–	–
[121]	SaaS	Private cloud built by OpenStack	–	ROS	Map building (Navigation)	Discussion on network	–
[122]	SaaS	Private cloud	–	–/ROS	Activities in home environment/Map Building, Vision-based recognition	Map merging and face recognition performance	–
[123]	SaaS	Private cloud built by OpenStack	Socket	–	SLAM	SLAM system performance and results	Deployment and Development criteria, running time
[125]	SaaS	Private cloud	–	ROS	Object recognition services	Composition method metrics, QoS in workflow processing	–

3.4 Discussion

As illustrated in Figure 3.4, the trend towards introducing service-oriented solutions in robotics has been receiving much attention in the literature, and has been evolved a lot in the last five years. In the remainder of this section, we discuss the review findings and research issues of the conducted analysis on both cloud and SOA-related aspects.

3.4.1 Deployment model and Service Level Agreements

Although there has been numerous research around the design of cloud-based SOAs in robotics, the integration of the cloud and its service models in such systems from the aspects of architecture modeling, resource virtualization and implementation process was not highlighted until 2017 [54], [121], [91], [125], [109], [71]. Instead of implementing a remote intercommunication with robots via Internet under the name of cloud, the set of studies that has been conducted accordingly has turned the "robotic service" concept into more powerful service models by providing Software, Platforms and Infrastructures as an on-demand service through virtualized computing resources.

As presented in the continued parts of Table 3.1, realizing the cloud solution in each study was conducted following the private deployment model [34, 35]. The exclusive access and usage of the computational resources are given to local users in order to establish the set of local case studies. The concern of researchers is the validation of the used technology innovation without any consideration of shared concerns between different cloud consumers that belongs to different organizations or the general public. The cloud architecture implemented through the public network shown in Figure 3.4 are presented in a real-time performance and networking over internet privately, which is not related to the cloud deployment model. The approach proposed in [91] addresses the issue of collaborative multi-clouds system that is comprised of many providers, but its implemented solution needs an empirical validation in this context.

During the review, we found that this way of system architecting among the majority of researchers has influenced significantly the SLA [35] contract definition and use. At present, there is a lack of SLA management in SOA-based cloud robotics systems. Generally, the set of QoS requirements of the on-demand service offerings is

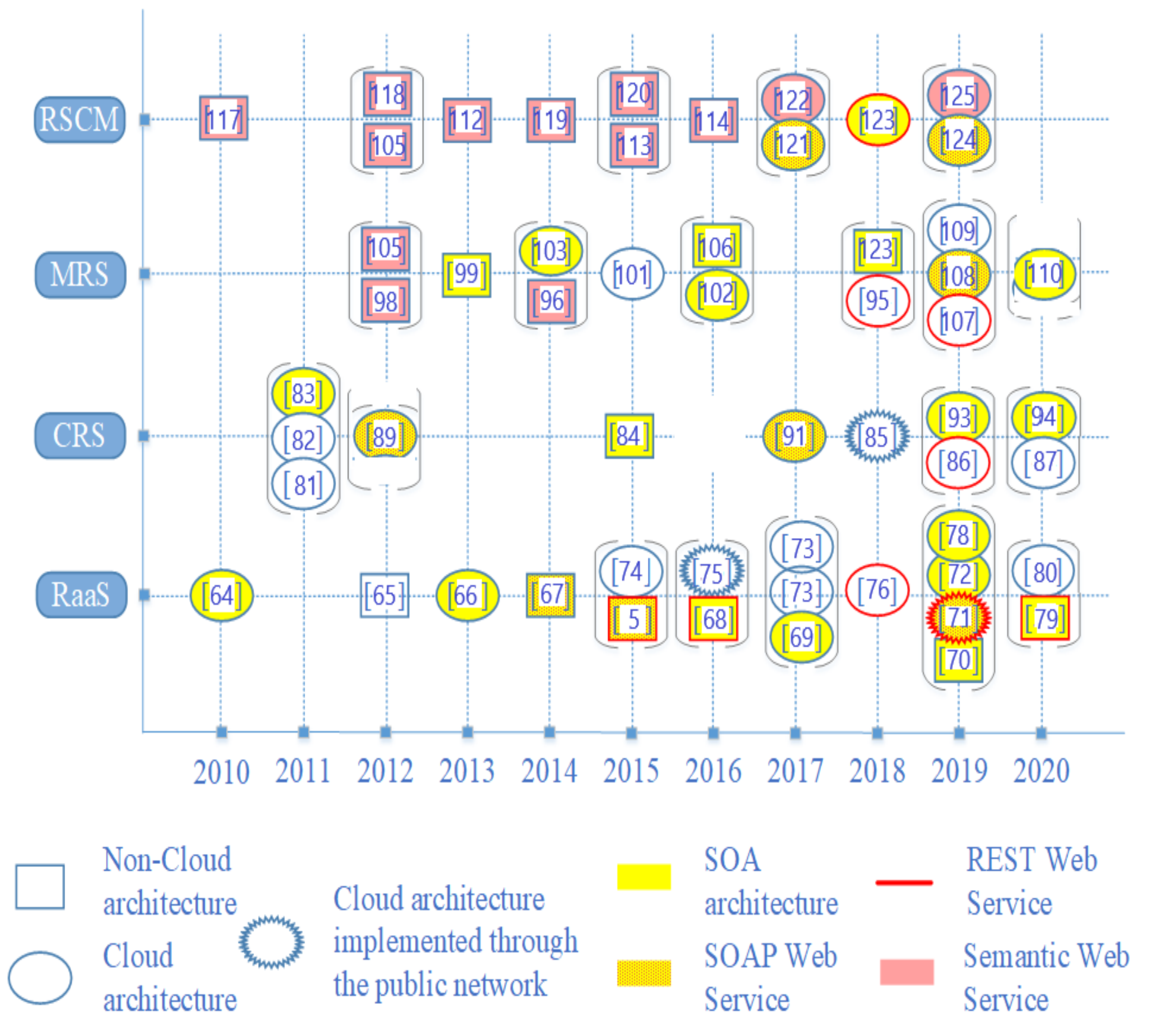


Figure 3.4: Historical overview of the reviewed works of the service-oriented robotic models.

defined in a given SLA, which leads to a regulated relationship between the service provider and consumers [91], [125]. Thus, SLAs are highly required and need to be addressed to specify the terms fulfilled by cloud providers, especially in multi-cloud environments.

3.4.2 Representation and description of robotic services

Robotic services are varied. Robotic service denotation is used in PaaS [54] and IaaS [76] to refer to the computing resource offered by the service model. In SaaS, it is generally understood to mean software components [71], [85], [91], [123], [125] such as algorithms and applications of navigation, object and voice recognition, or it can refer to entities like images [122], dialogue services [93], [94], or deep learning models [87]. Different technology styles have been considered for service deployment implementation including SOAP [5], [71], [91], [121], REST [5], [71], [79], microservices[3] [78], [123], OWL-S [103,104], and others, as outlined in Table 3.1.

The most used standard format for describing the service is WSDL. WSDL provides a description that includes the name, address, operations, inputs and outputs of the web service. Nevertheless, there is a need to describe and standardize more the content of the service, and keeps the aspect of the encapsulation. Robotic services should provide more information about their capability, which characterize their functionality and applicability on the different types of robots. In that regard, exploiting the semantic descriptions and strategies that have been widely used in RSCM frameworks (see Table 3.1) may offer an improvement to the definition and description of cloud robotic services (e.g. [124]).

3.4.3 Deficiencies in SOA implementation: Robotic service discovery gap

Service implementation in the matter of SOA life cycle process in robotics still has several deficiencies. Most of the studies have focused on SOA as a mechanism that provides service encapsulation and loose coupling between system modules. However, there is less focus in the literature on the dynamic discovery for enabling the reuse of services. Despite the interest of introducing SOA, this drawback affects the implementation of services as discoverable entities, which is the fundamental scope and focus of service-oriented architecture style.

Research on service discovery has been investigated in some earlier work of MRS (e.g. [105], [96]), RSCM (e.g. [120], [114]), and in [92]. It has taken many initiatives and modeling techniques that involve semantic technologies and ontologies to cover the underlying background of MRS and RSCM models. But the research has known a lack in the recent and cloud-based systems as shown in Table 3.1. For instance, authors of [54] proposed a scheduling algorithm of robots based on the customer's request. In [76], a discovery mechanism that targets the use of IaaS for robots when needed was proposed. [93] and [94] adopted deep learning for the answer searching and selection mechanisms in dialogue services between the user and the robot.

[3]Microservices are small and independent processes that can communicate together to build complex applications.

As presented in 3.3, service discovery is the phase that enables the service consumers to find relevant services by offering a list of candidate services, through matching the set of descriptive information of published services with the consumers' criteria. Hence, there is a lack of this scope in the cloud-based service-oriented architectures. Some authors have discussed the impact that service discovery offers to use the available robotic services, especially, for the future issue of service composition [123], [111], [125]. Nevertheless, there is a gap that lies in the absence of sophisticated search capabilities for identifying potential cloud robotic services provided by different service providers that can help users to find relevant services. This involves the capability of examining the published robotic services, which can be even similar, depending on: (*i*) the needs of service consumers, and (*ii*) the compatibility between the service and the various kinds of robots. Indeed, the problem of finding relevant cloud services that can respond the needs of robots remains a challenge that is not fully addressed.

3.4.4 Robot Operating System and case studies

As shown in the continued parts of Table 3.1, ROS is the most used robotic framework. This is due to the set of ROS benefits including its compatibility with many robots due to its development support for software reuse.

The set of case studies and used robots (see Table 3.1) in cloud-based SOA robotic solutions show that they are implemented to support service robotics in a variety of environments where mobile robots are employed, and less use in the industry [124], [127].

3.5 Conclusion

SOA is considered as a key element for providing services over the internet in robotics applications, which have known a growing shift into cloud-based architectures. In this chapter, we have proposed a classification of research proposals in the literature of this area. We propose the following service model: Robot as a Service (RaaS), Cloud-enabled Robotic Services (CRS), Multi-Robot-based Services (MRS), and Robotic Service Composition Middlewares (RSCM). This classification is made according to the provisioning and delivery of robotic services as well as the architecture of different approaches. In addition, we have conducted a comparative analysis that examines the software modelling and experimental deployment of the presented works.

The result of the study shows that the works are diverse in many levels of both system conception and development. From a conceptual point of view, we have noticed that the web services technology have been widely used in robotic services representation. However, the standardization of representation and robotic services discovery have not received enough attention. The next chapter will present the thesis's contribution axes.

Chapter4
Robotic Services as a Service approach

4.1 Introduction

The development of on demand software for robotic service provisioning is growing. By the encapsulation of implementation details and offering loosely coupled application functions, the use of SOA provides a solution to the response of software development problem for robots. However, as presented in section 3.4 of chapter 3, there are a set of issues about service-oriented solutions.

In this chapter, we present the first contribution addressed in this thesis based on the defined issues in the previous chapter. We present our proposed architecture as "Robotic Services as a Service" (RSaaS) approach. To that end, we highlight firstly the scope of RSaaS in section 4.2. Next, we present the actors, architectural elements and modules that build the RSaaS architecture in section 4.3. The experimental settings that have been used to develop the system are summarized in section 4.4.

4.2 Highlights

Unlike traditional robots that use a large number of devices for storage, calculation and processing, they can use less expensive and more effective hardware due to cloud robotics. Based on that, our proposed solution addresses the drawback about SOA implementation in cloud robotics systems for service provisioning. We present in this section the main items that are addressed by our proposals.

4.2.1 Robotic Services as a Service scope

The change in the form of service distribution, which has been introduced by cloud computing into robotics, motivated our proposed solution for robotic services delivery. Our work aims to propose an architecture for delivering "Robotic Services as a Service" (RSaaS) to the robots as a CRS model (see subsection 3.2.2 of chapter 3), by leveraging the advantages of cloud robotics and web services.

Indeed, we notice in the first place that there is a lack of using classical cloud computing architecture, which is based mainly on virtualization concept, in SOA-based works. The studies that has been conducted accordingly have designed the cloud solutions as client-server model for remote intercommunication with robots via Internet. This influenced the real aspect of cloud computing that provides powerful service models by providing Software, Platforms and Infrastructures as an on-demand service. For this reason, we addressed this issue and we proposed an architecture firstly in [91], by extending the cloud robotics solution of [50] with web services technologies.

We proposed subsequently the detailed architecture of this work in [128] that focus on providing a complete SOA-based architecture for service provisioning. The proposed approach extends the architecture [91] that has discussed the service searching topic mainly without a particular mechanism.

The proposed approach [128] presents a full SOA-based approach, in which robotic ROS tasks are considered as web services, according to a defined representation, that are hosted, virtualsed, and delivered over a cloud infrastructure. The idea of hosting robotic software is also addressed in [85]. However, the work does not consider robotic services as web services, and does not discuss how the software can be virtualsed and discovered. The addressed aspects of CRS-based works are different from our contribution, which is more focused on the SOA context, technologies, methods and tools. We denote this approach by "Robotic Services as a Service" or "RSaaS". Although this denotation is given by the taxonomy proposed in [84], our definition presents another approach for RSaaS. Indeed, according to [84], RSaaS refers to user service that are obtained from the cloud using robots, which can be given by the RAPP project [46] or dialogue-based systems [93,94] (see CRS model in subsection 3.2.2 of chapter 3).

4.2.2 Virtualization concept

Cloud computing has becoming increasingly used paradigm. This is mainly shown by the numerous cloud providers that are engaged in the creation and delivery of various computing services. Indeed, "*virtualization*" is the engine that enables this paradigm change in computing, and in particular, "*machine virtualization*" [129]. Virtualization abstracts some physical component into a logical object, in which we can acquire a higher level of utility from the resource. [130]. It creates the artificial view that many computers are one computing resource or that a single machine is many individual computers [131].

Instead of owning the computing resources, cloud computing applies virtualization to enable consumers to access them as a service using cloud data centers. Virtualization allows the computer systems' hardware resources to be divided into a number of different execution environments known as *Virtual Machines* (VMs). VMs can be classified based on how much functionality they implement of the targeted machines[1] [132]. Each VM can act as a complete system to execute the user applications in isolation with other VMs. A VM is defined in [133] as "*a complete compute environment with its own isolated processing capabilities, memory, and communication channels*". Accordingly, there are several benefits of virtualization for enabling cloud computing, including execution isolation, easier management, and enhancing reliability [129].

For hosting a VM, everything a VM needs in terms of CPU, memory, storage and network bandwidth is provided by a physical machine or a server. The VMs are managed within the physical machine by a layer of software anointed VM Monitor (VMM), or as it is commonly called a *hypervisor*. A hypervisor resides below the virtual machines (*guests*) and above the hardware (*host*) [130, 132]. There are two categories of hypervisors that are generally named [129, 130]:

Type 1: A Type 1 hypervisor does not require an operating system since it runs directly on the physical hardware. It has direct access to hardware resources, and the guest does not affect the hypervisor on which it is running. VMWare ESX, Xen, and XtratuM are some examples of Type 1.

Type 2: Type 2 hypervisors run atop a traditional operating system. They are straightforward to install and deploy because the operating system handles most hardware configuration tasks, such as networking and storage. Examples of Type 2 can be given by the KVM, VMware Workstation, VirtualBox, etc.

4.3 Robotic Services as a Service architecture

The RSaaS solution provides both of robotic services, which are delivered to be consumed "as a service", and virtualized computing resources that enable to run these services, on different instances, according to desired requests. The description of the architecture will be presented in the following subsections.

[1]There are four major levels of virtualization that can be distinguished: full virtualization, para virtualization, hardware assisted virtualization, and resource virtualization.

4.3.1 Overall system

The overall architecture of RSaaS is shown in Figure 4.1. In our design, a robot benefits from the computational resources, storage, and necessary software to perform its required task. The main features of the architecture are summarized as follows:

- Different robotic software or services as navigation, object or voice recognition algorithms can be used "as a service" for robots. Unlike the pre-programming of robots that addresses limited use cases, robotic services can be discovered and invoked dynamically to respond to the various needs of robots, which allows them to adapt to their situations and be more autonomous.
- Robotic services can be obtained and accessed following the search of available services over the cloud.
- Robotic services are proposed as Web services to benefit from all their advantages as (*i*) data encapsulation, (*ii*) service reuse, (*iii*) and interoperability between services.
- The solution improves task performance of services by boosting computational capabilities over the cloud infrastructure.
- The architecture provides full abstraction of system construction. Users are invited to consume robotic services without knowing how the provided solutions are built.
- From a cloud provider's point of view, the system architecture is structured as a layered architecture that is inspired from classical cloud computing architectures [35].
- The cloud layers are hidden from robotic consumers, as it is the case in a cloud system for computational resources delivery between providers and consumers [35].

There are three levels in the cloud side:

1. The *Physical* level: It represents the set of physical resources, such as storage servers, network equipments...etc.
2. The *Virtual* level: It contains virtual machines, which enables to provide multiple applications on multiple instances to serve a large number of customers.

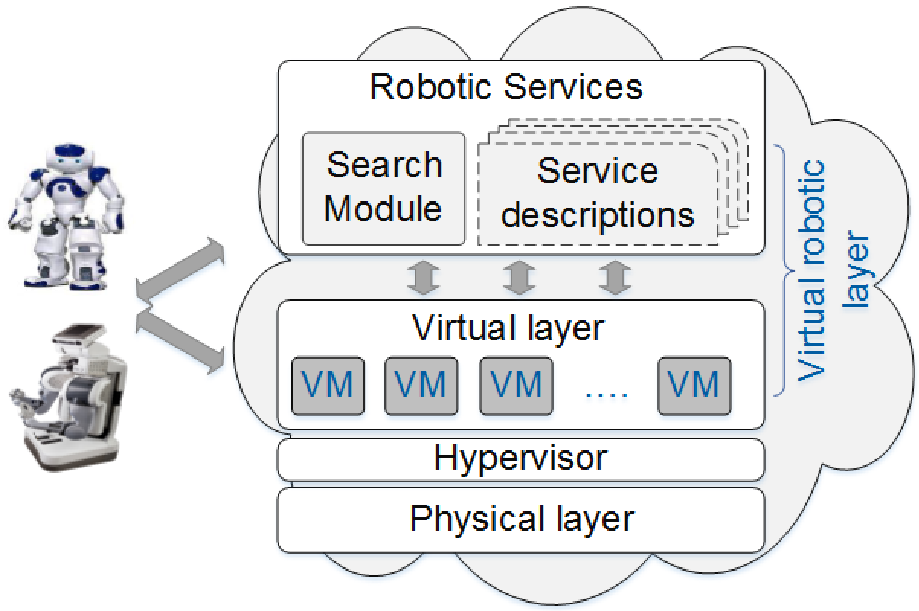


Figure 4.1: RSaaS System overview.

3. The *Robotic Services* level: In this layer, the robotic software are exposed as Web services. Both of virtual and robotic services layers constitute the *Virtual robotic layer*.

The collaboration of RSaaS architecture's components is presented in Figure 4.2. We distinguish a set of modules to define the RSaaS architecture. These modules are highlighted in the following.

4.3.2 RSaaS virtualization and service model

In order to help robots to perform their required tasks, users can interact with the virtual robotic resources of the cloud, through a Web interface, similarly to SaaS offerings. The cloud provider offers this virtual layer to customers that guarantee the promised SLA. This can be based on pay-per-use mode that covers duration of use, quantity and quality of resources. The cloud provider takes the responsibility of installation, control, maintenance, etc, in which all the computational resources are hidden from consumers.

The cloud virtual layer is constituted by a set of ROS Virtual Machines (ROS-VMs) as shown in Figure 4.2. ROS-VMs are similar to virtual machines but with specific tools. This provides a robotic operating system that is virtually executed in the cloud environment. They are considered as robotic platforms where each one

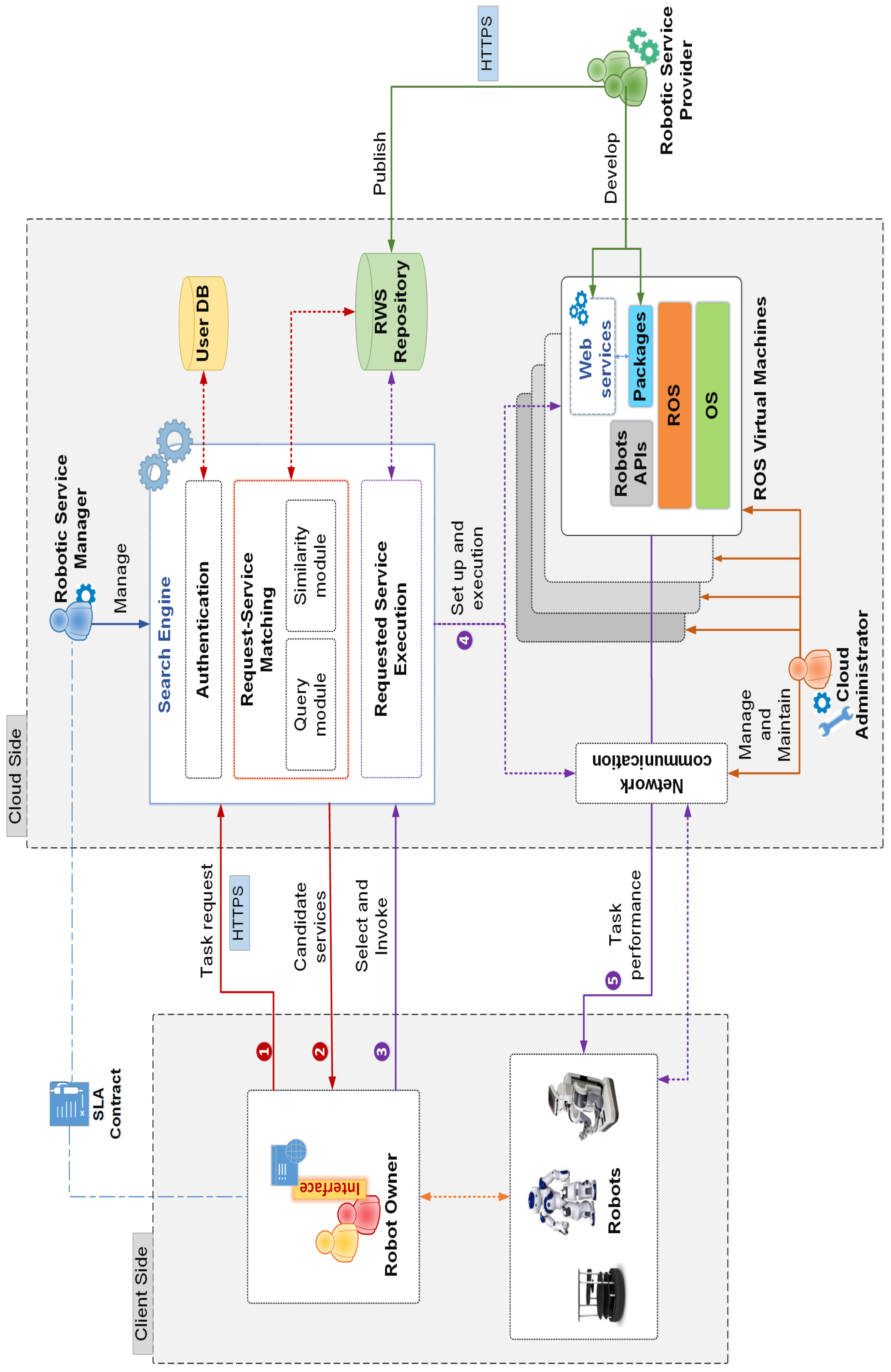


Figure 4.2: The proposed RSaaS architecture.

collects multiple robotic services. Such services enable to run corresponding ROS codes that perform robot tasks. The system applies a dynamic attribution of ROS-VMs according to service demand requirements.

Principally, a ROS-VM is composed by:

- An operating system (Ubuntu).
- Robot Operating System (ROS).
- Robots APIs.
- A set of Packages: A package, which is developed for robotic tasks performance, contains source code for ROS nodes. These codes are encapsulated, reused, and interoperable robotic services. They are proposed as web services.

4.3.3 RSaaS cloud actors

Cloud actors are intended to use, manage, and monitor the provided cloud resources for keeping the system in good operating conditions, and to prevent any possible failures. Mainly, these actors and the main features of their responsibilities are presented as follows.

RSaaS Client

At the client side, users can interact with the robotic resources of the cloud, through a Web interface, to help their robots performing their required tasks. Hence, there are two kinds of clients:

- **Robot Owner**: A Robot Owner is the end user that wants his connected robot to accomplish a specific task. The robot owner makes an agreement with the service provider by signing a service contract SLA that describes the Cloud-Client features. The service contract is based on the achievement of the following tasks:
  - Search for candidate services: Authorized users are invited to access the external web application for searching services remotely. According to their requests, a list of candidate services will be displayed by the search engine.

  - Select and execute best web service: Selecting and executing the best web service among the search results by the user enables the performance of desired robotic tasks.
  - Establish Robot-Cloud connection: At the client side, a private network must be configured for service invocation that requires an establishment of robot connection to the cloud, across the public network, in order to enable robots to receive action orders.

- **Robots**: This is the main actor. By being connected to the cloud, robots would be able to use the suitable service to perform their required tasks of locomotion, grasping, object recognition, etc.

RSaaS Cloud Administrator

The RSaaS Cloud Administrator plays an effective role in providing, and managing the cloud computing resources for users and service providers. He manages the entire cloud infrastructures and platforms. The responsibilities of a Cloud Administrator are:

- Resources Provisioning: This includes the resource allocation of ROS virtual machines for certified customers, with a dynamic attribution according to the demand.

- Resources Monitoring: This comprises the supervision of the system behavior during execution including the platforms, the network management and security.

- Network Management: Encompasses the configuration of the network and its security. This comprises:

  - User-Cloud connection: Includes the network configuration about the locations of hardware and the client settings.
  - Robot-Cloud connection: Which enables the connection of robots to the ROS-Cloud platform for executing the desired services.

RSaaS Robotic Service Provider

Different web services can be available for use according to the multiple robotic tasks. As presented in Figure 4.2, each service is created and provided by a Robotic Service Provider. Its responsibilities can be summarized as follows:

- Setting pre-required tools: This step covers the installation, and the configuration of all the required development tools and APIs on Ubuntu-based ROS-VMs.
- Developing ROS programs: This operation includes the creation of packages, nodes, compilation of codes to ensure the proper functioning of ROS codes.
- Publishing services: This procedure enables the services to be available and dynamically discovered through their service interface and requirements.

RSaaS Robotic Service Manager

The RSaaS Robotic Service Manager is the supervisor and manager of the cloud service provided to end users, and has the following responsibilities.

- Establishing SLA contracts: In which responsibilities and features about the provided cloud service are identified.
- Managing the RSaaS Engine: This is assured through the data flow monitoring with the robot owner. It comprises also web service data extraction and setting up of the computing environment for service invocation.

4.3.4 RSaaS life cycle process for service provisioning

The main focus addressed in this thesis targets the development of full SOA life cycle process for ROS-based service provisioning. This requires the definition of service requirements for the publication and discovery phases.

Service description and publication

Defining the requirements of the robotic service meta data is the first important issue that needs to be considered for response to criteria-driven queries. From a software engineering point of view [6], these requirements are viewed as key elements for robotic application characterization and architecture modelling of service-oriented systems in robotics. Thus, in the first step in the life cycle process, we define the ROS-based service requirements. This definition expresses the key features that characterize the ROS web service from ROS itself. We consider ROS web services as SOAP web services due to their completeness in service implementation, as presented in subsection 2.4.3 of chapter 2. The Robotic Service Provider makes services available and accessible by publishing the WSDLs of services. We also add a semantic description to these services by exploiting the opportunities of semantic web services using OWL-S. The description leverages the semantic notations for expressing ROS web service specifications based on the OWL language and ontologies.

Dynamic service discovery

Finding the appropriate service that can match the user request will provide a major impact on clients' satisfaction. Robots will be able to perform different tasks by the search of available services. This will offer, for both software developers and users, the opportunity to provide an abstraction of any platform requirements or robotic software. This is due to the service discovery process that is intended to carry out the appropriate service for robots, according to the matching of the requested task with robotic service features.

To cope with dynamic discovery queries of services in RSaaS system, runtime discovery is needed to assign different tasks to robots through accessing the suitable service. ROS web services can have similar or different functionalities for different kinds of robots. In such cases, the RSaaS search engine focus on matching strategy that intended to retrieve the most satisfied ROS web services. Finally, the list of candidate services is ranked according to their score of similarity.

Dynamic service invocations

By selecting the best service, the system allows the invocation and access to the web service remotely. It invokes the web service dynamically according to the extracted data by parsing WSDL files (web service name and its operations with the inputs). The communication between client codes and web services is ensured through SOAP messages.

In addition, the service invocation requires the configuration of the network to enable the connection of the robot, and the launching of robot's ROS nodes, which ensure the proper functioning of the service.

4.4 Experimental settings

The list of tools and ROS technical requirements that have been used to develop the system are summarized below.

4.4.1 Technical robotic tools

We hosted our robotic requirements in the cloud environment of *Synchromedia* [134], in which we use virtual resources hosted on cloud servers across the internet. A number of VMs of different usages (applications, authentication, web) were created for our research team. The experiments of this work were performed on a VM that has 3 GB of RAM memory, 4 processors, 100 GB of disk space, and Ubuntu server 14.04.

For a remote access to the graphical desktop of the VM, we used the open source `X2Go`[2]Client under `Windows` and `Ubuntu`. All session configuration details (including the Host, Session type, etc.) have been identified carefully at the first time of creating a new session with X2Go.

The NAO robot

The case study for developed packages was performed using real NAO robot [135] (see Figure 4.3), with the potential of solution reuse using simulated robots. NAO is the first humanoid robot built by "Aldebaran"[3] society. Currently, it is used in different countries around the world as a particular platform for the fields of research and education. The NAO robot was designed to have many senses for natural interaction to reproduce human behaviors. It can perform different functionalities such as moving, speaking, thinking, etc.

NAO with ROS

On the `Ubuntu 14.04`, we installed the full desktop installation of ROS `Indigo` distribution. In addition, to use NAO with ROS[4], it is necessary that:

[2]X2Go Website: `https://wiki.x2go.org`.
[3]In 2016, Aldebaran brand name became "SoftBank Robotics".
[4]NAO with ROS: `http://wiki.ros.org/nao`.

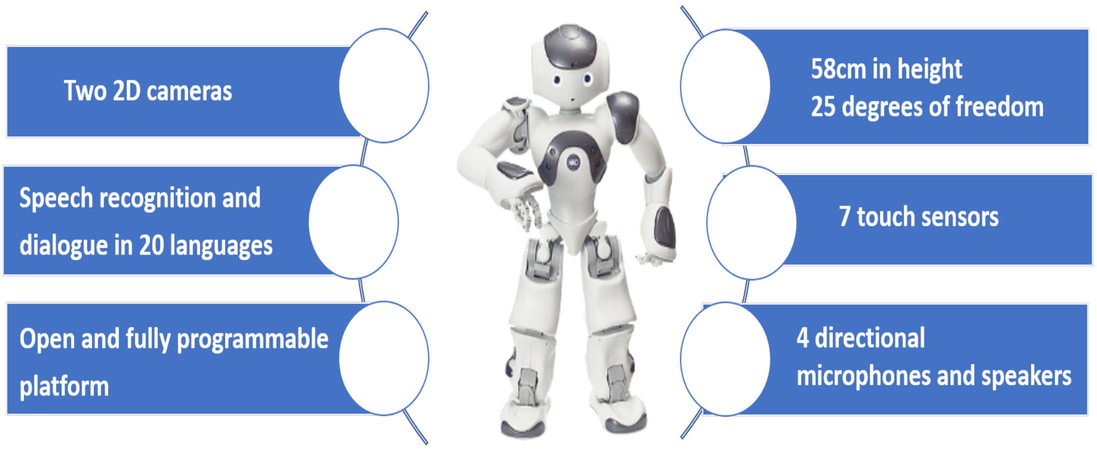

Two 2D cameras
Speech recognition and dialogue in 20 languages
Open and fully programmable platform
58cm in height
25 degrees of freedom
7 touch sensors
4 directional microphones and speakers

Figure 4.3: The NAO robot.

- The Nao packages should be installed to have the required components for getting started with the robot. Hence, the needful commands that meet our ROS version were applied.
- The `NAOqi` SDK should be set up as well. We used SDK version `2.1.4.13-linux64`[5].

All basic actuators and sensor publishers for NAO become up into a running state, after launching `nao_full_py.launch`[6] file of `nao_bringup` package by the following command:

- `roslaunch nao_bringup nao_full_py.launch`

Moreover, the NAO's IP (`NAO_IP`) and the IP of roscore[7] (`ROS_MASTER_URI`) must be exported, when starting the robot bringup command, to establish a correct network connection between the robot and ROS's computer.

To launch NAOqi, the following command should be run in a another terminal:

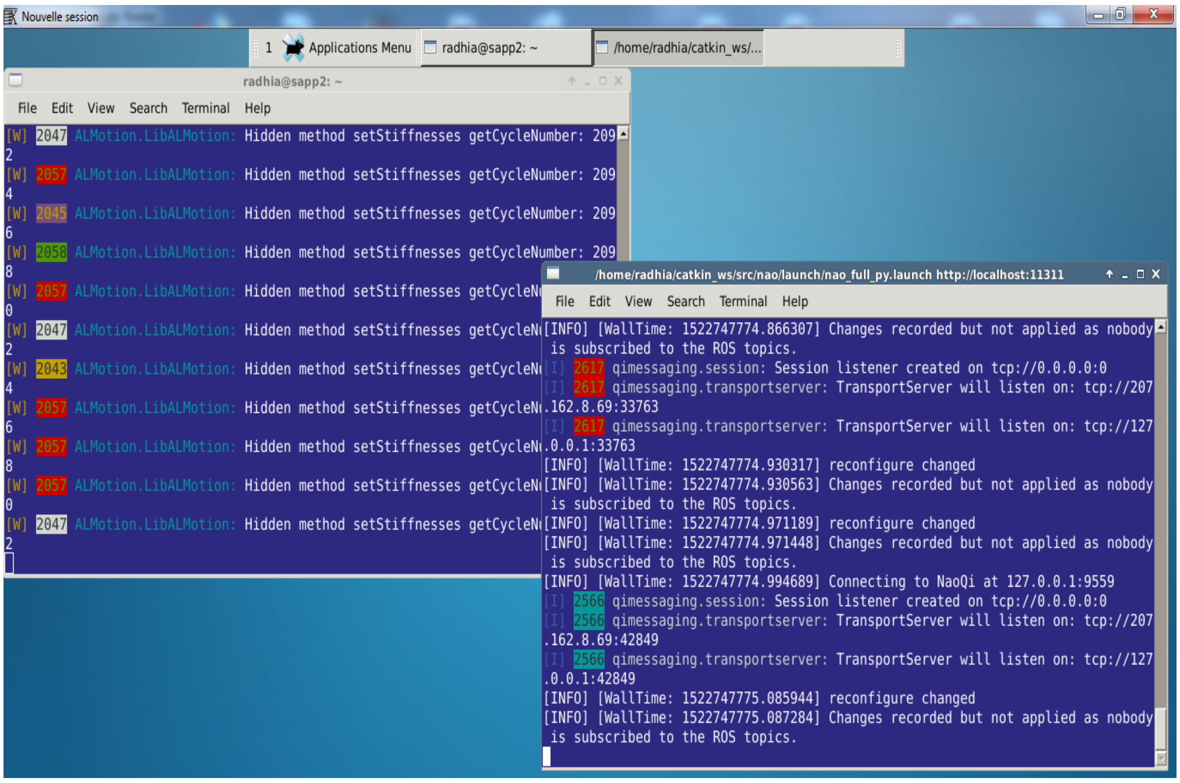
- `~/naoqi/naoqi-sdk-2.1.4.13-linux64/naoqi`

To ensure its proper functioning, the NAOqi library path should be added to PYTHONPATH with the following command:

[5]Logged in and Downloaded from: `https://community.ald.softbankrobotics.com/`.

[6]With the ROS distributions, the launch file name can be different. For instance, in ROS hydro we use `nao_full.launch` instead of `nao_full_py.launch`

[7]The roscore IP: is the IP of the computer where the ROS Master is running.


Nouvelle session
1 Applications Menu
radhia@sapp2: ~
/home/radhia/catkin_ws/...
radhia@sapp2: ~
File Edit View Search Terminal Help
/home/radhia/catkin_ws/src/nao/launch/nao_full_py.launch http://localhost:11311
File Edit View Search Terminal Help

```
[W] 2047 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleNumber: 209
2
[W] 2057 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleNumber: 209
4
[W] 2045 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleNumber: 209
6
[W] 2058 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleNumber: 209
8
[W] 2057 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
0
[W] 2047 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
2
[W] 2043 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
4
[W] 2057 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
6
[W] 2057 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
8
[W] 2057 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
0
[W] 2047 ALMotion.LibALMotion: Hidden method setStiffnesses getCycleN
2
```

```
[INFO] [WallTime: 1522747774.866307] Changes recorded but not applied as nobody
 is subscribed to the ROS topics.
[I] 2617 qimessaging.session: Session listener created on tcp://0.0.0.0:0
[I] 2617 qimessaging.transportserver: TransportServer will listen on: tcp://207
.162.8.69:33763
[I] 2617 qimessaging.transportserver: TransportServer will listen on: tcp://127
.0.0.1:33763
[INFO] [WallTime: 1522747774.930317] reconfigure changed
[INFO] [WallTime: 1522747774.930563] Changes recorded but not applied as nobody
 is subscribed to the ROS topics.
[INFO] [WallTime: 1522747774.971189] reconfigure changed
[INFO] [WallTime: 1522747774.971448] Changes recorded but not applied as nobody
 is subscribed to the ROS topics.
[INFO] [WallTime: 1522747774.994689] Connecting to NaoQi at 127.0.0.1:9559
[I] 2566 qimessaging.session: Session listener created on tcp://0.0.0.0:0
[I] 2566 qimessaging.transportserver: TransportServer will listen on: tcp://207
.162.8.69:42849
[I] 2566 qimessaging.transportserver: TransportServer will listen on: tcp://127
.0.0.1:42849
[INFO] [WallTime: 1522747775.085944] reconfigure changed
[INFO] [WallTime: 1522747775.087284] Changes recorded but not applied as nobody
 is subscribed to the ROS topics.
```

Figure 4.4: The Output of running NAOqi and NAO bringup package under ROS via X2Go.

- `export PYTHONPATH=/TheNaoqiLibraryPath/lib:${PYTHONPATH}`

Figure 4.4 displays the output of the command launching the NAO (`nao_bringup`) and the NAOqi as well, which is needed to be running in order for the launch file to work.

4.4.2 APIs and ROS packages

We develop the NAO codes using `rosjava` library, which implements ROS with Java language, after its installation.

To start working with rosjava packages, we need to to create and build `catkin packages`. This can be built as a standalone project, however, it is recommended to use a `catkin workspace` in which multiple packages can be built together. We used the following commands to create and build a `catkin workspace`:

- `mkdir -p ~/name_ws/src`
- `cd ~/name_ws/src`
- `source /opt/ros/indigo/setup.bash`

- `catkin_init_workspace`
- `cd ..`
- `catkin_make`

In this workspace, we create and build each `catkin package` using the following commands:

- `cd ~/name_ws/src`
- `catkin_create_rosjava_pkg name_pkg`
- `cd ..`
- `catkin_make`
- `source devel/setup.bash`

Instead of running execution commands from the terminal and manipulating ROS codes using the text editor, we used the rosjava library under `Eclipse`[8] IDE environment. We used Eclipse for `Java EE` Developers (`jee-oxygen` version) with `JDK 8`. The tutorial [136] is a helpful source to start working with rosjava under Eclipse, which is familiar to us as java developers.

To use rosjava in Eclipse, we have to create firstly in Eclipse a *"Java Project"* or *"Dynamic Web Project"* as needed. Then, we need to *"Configure Build Path"* in order to *"Add External JARs"* of rosjava that are found in the package created in: `name_ws/src/name_pkg/rosjava_project/build/install/rosjava_project/lib`[9].

The execution of rosjava class from Eclipse is done using the toolbar of Eclipse by the following instructions:

- Run → Run Configuration, then, we choose Java Application → New → [in the tab "Main" we choose our rosjava project for `Project`, and ROSRun - org.ros for `Main class`] then [in the tab "Arguments " we write `name_pkg.name_class`] → Apply → Run.

[8]Eclipse Desktop IDEs: `https://www.eclipse.org/ide/`.

[9]In case of using a *"Dynamic Web Project"*, we must copy these JARs also in `WebContent/WEB-INF/lib` of the project.

Now, by launching your robot, you can see it executes the desired action of code.

We translate the codes to SOAP Web services automatically, using the server `Apache Tomcat`[10] (`apache-tomcat-7.0.85` version), to keep instances of WSDL files in the database and to avoid their recreation. To develop the ROS domain ontology and ROS service ontology, we used the `Protégé` editor [137].

In addition, the following java APIs were used for system development:

- `WSDL4J`: Used for parsing WSDL documents.
- `std_srvs` and `naoqi_msgs`[11]: Used respectively for defining "std_srvs" and "naoqi_bridge_msgs" rosjava messages of appropriate web services. We rewrite the jar file "naoqi_msgs" into "naoqi_bridge_msgs" with the necessary modifications to make the version compatible with ROS Indigo version. Information about ROS messages of developed web services are listed in subsection 5.5.1 of chapter 5.

4.5 Conclusion

In this chapter, we presented our proposed approach for Robotic Services as a service (RSaaS) concept, in which the solution aims to offer a full abstraction of any platform requirements or software. The RSaaS combines the benefits of both cloud computing and web services. A user can search for an appropriate service that respond to his robotic request, which is assured by the service discovery process. The following chapter aims to introduce the definition of this life cycle process.

[10]Apache Tomcat: `http://tomcat.apache.org/`.
[11]rosjava_messages: `https://github.com/rosjava/rosjava_mvn_repo/tree/master/org/ros/rosjava_messages`.

Chapter5

ROS Web service description and discovery

5.1 Introduction

In this chapter, we present in details the two contributions that address the life cycle process of ROS-based web service. The first contribution of ROS Web Service (ROS-WS) relies on SOAP-based services and defines a set of characterization requirements. The second contribution adds a semantic layer to ROS-WS on the basis of OWL-S ontology and designs the ROS Semantic Web Service (ROS-SWS).

The motivation as well as related works of the context of our work are described in section 5.2. In section 5.3, we introduce the ROS-WS definition and ROS-WS discovery process with its flow of actions. Section 5.4 presents the cycle process proposed for ROS-SWSs. This includes the ROS-SWS description, domain ontology, and ROS-SWS discovery engine. Finally, we present in section 5.5 the ROS-WS/ROS-SWS implementation, case study, and obtained results.

5.2 General scope

The main aim of ROS-WS/ROS-SWS contributions is the development of full cycle process for ROS Web Services. We highlight in this section the scope about ROS-WS and ROS-SWS.

5.2.1 Motivation and related works

The general scope of our work targets the drawback of service-oriented solutions, in particular for ROS-based systems, by filling the gap between ROS Web services and their discovery process.

In that regard, we import in the first place a definition of functional meta data to ROS web services [5], [91], [121], which was proposed in [128]. The ROS-WS definition describes the robot task representation from ROS requirements to present a characterization of such web services. For finding these services, we compute the similarity score between sentence embeddings of each service and user query using sentence-BERT [7], by reinforcing the training dataset.

Secondly, we propose a semantic description ROS-SWS that build the ROS Web Service as single-semantic unit, which expresses its ability through a ROS domain ontology of properties and capabilities.

The decription and discovery of ROS-WS and ROS-SWS are based mainly on high-level semantic concepts of ROS messages [138] and independent of any case study or robots.

5.2.2 Message and service types in ROS

The type of messages and services in ROS are data structures that are stored in "`msg`" and "`srv`" ROS packages respectively. Both of these packages combine a set of common message and service types that share a common context of use.

ROS provides a standard naming manner for the type of messages: `pkg_name/msg_file_name` and services type: `pkg_name/srv_file_name`. For instance, `geometry_msgs/Twist` refers to `Twist` message type that is defined in the package `geometry_msgs`, which provides messages for geometric primitives. Each ROS message or service can contain one or more fields[1].

Tiddi et al. highlighted in [138] the high-level semantic concepts of ROS messages and their fields, which improve the accessibility to ROS. Each ROS message definition has illustrative names of fields and a description about their use. This allows to extract capabilities and to identify how to parametrize this capability to achieve the robot behavior, as shown in Figure 5.1. For example, it is possible to derive the capability of "*Movement*" from data fields such as velocity or acceleration, in which these are the parameters of this robot behavior [138].

This proposition gives an abstraction to the previous technical realization of ontology capabilities that relies on hardware/software architectures of robots such as [103,104] and [139,140].

We consider accordingly the message and service ROS types as key aspect that define the characterization of ROS-WSs and ROS-SWSs.

5.2.3 Distinguishing Robot-Service compatibility

In both ROS-WS and ROS-SWS search engines, we display the search findings after distinguishing the robot-service compatibility and availability. This is due to the convenient property of code reuse between robots offered by ROS. Therefore, we

[1]See geometry_msgs/Twist Message: `http://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html`.

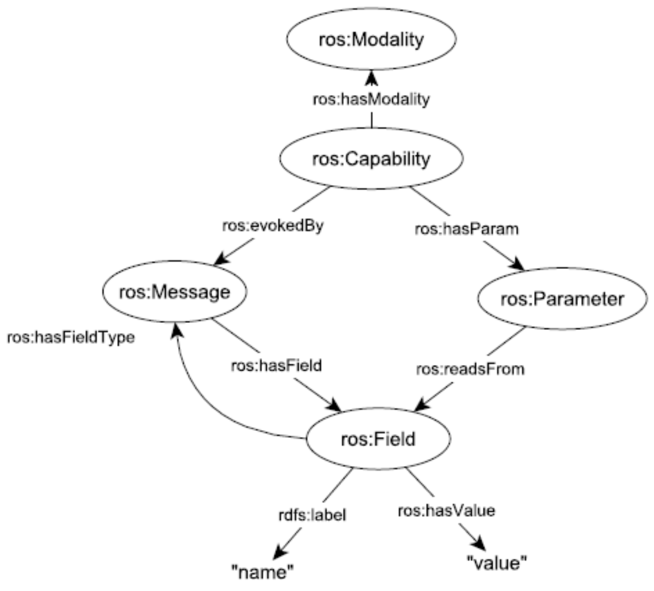


Figure 5.1: Mapping ROS components to capabilities [138].

collect the compatible service according to the ROS requirements. Each ROS-based robot has its own communication mechanisms through its list of own topics and services. Thus, we can determine the possibility of using codes by different robots across their communication mechanisms.

To that end, both of the robot name and task query must be indicated by the user. By specifying the name of the selected robot, a matching of the registered Topic/Service (T/S) of robot with those of web services is carried out. As shown in Figure 5.2, we consider ROS topics (T) and services (S) as the condition factor of proper choice of candidate services related to each robot registered in the database. A published web service that provides the same name of topics (T) or services (S) of the selected robot is considered as compatible service.

5.3 ROS Web Service (ROS-WS): Requirements and discovery

Like any traditional web service, a ROS Web Service (ROS-WS) is an interoperable software that can be accessed using standard Internet technologies. It could be seen as a function that requires a set of inputs and provide an action as output. As a result of web service execution, a robot will be able to perform different tasks. We present in this section the set of requirements that characterize the ROS-WS based mainly on

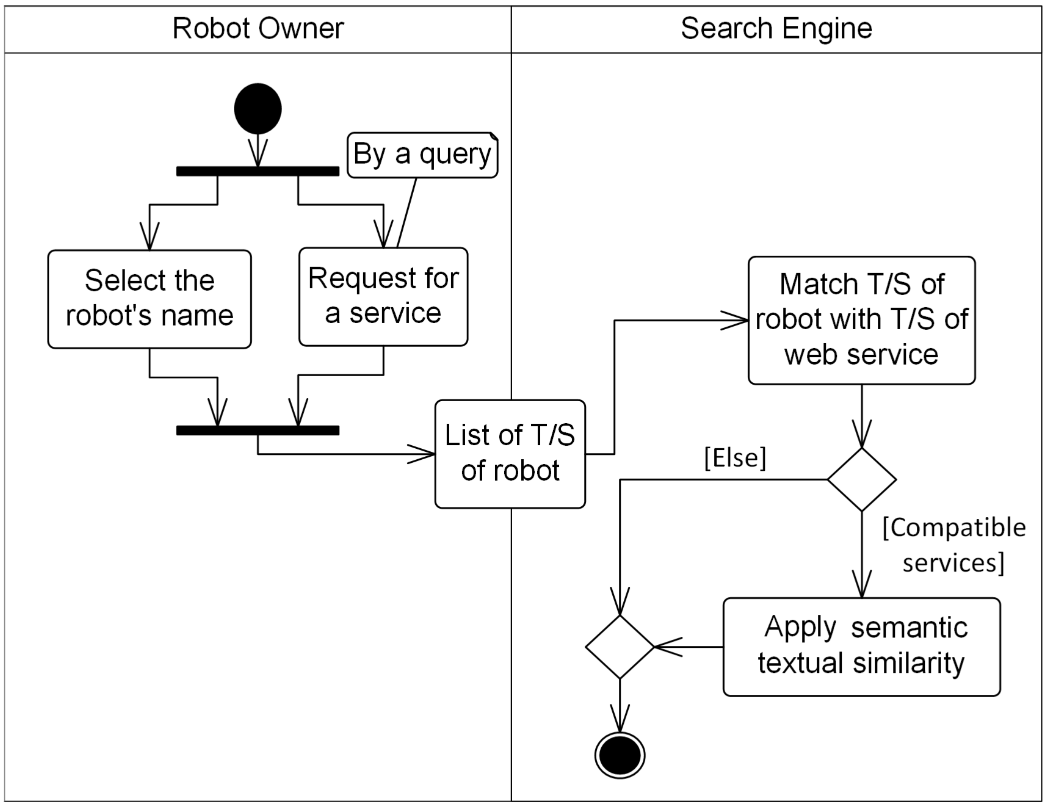


Figure 5.2: UML activity diagram for distinguishing Robot-Service compatibility.

types of messages and services in ROS, which provide semantic information about ROS use.

5.3.1 ROS-WS requirements

We define the representation of ROS-WS, in SOAP format, as a set of "*Functional Requirements*". The specification of ROS-WS enables the service search and their use depending on the users' needs. We present in Figure 5.3 the ROS-WS metamodel. Its core structure comprises three main parts: *ROS Requirements*, *WSDL Requirements*, and *Registry Requirements*, which are composed of the following elements:

ROS Requirements

The *ROS Requirements* are the several ROS data that are used to generate a robotic task of the web service. This includes:

- *Node*: Which refers to the ROS process that performs a robotic task. Generally the name is explicitly referring to the task.

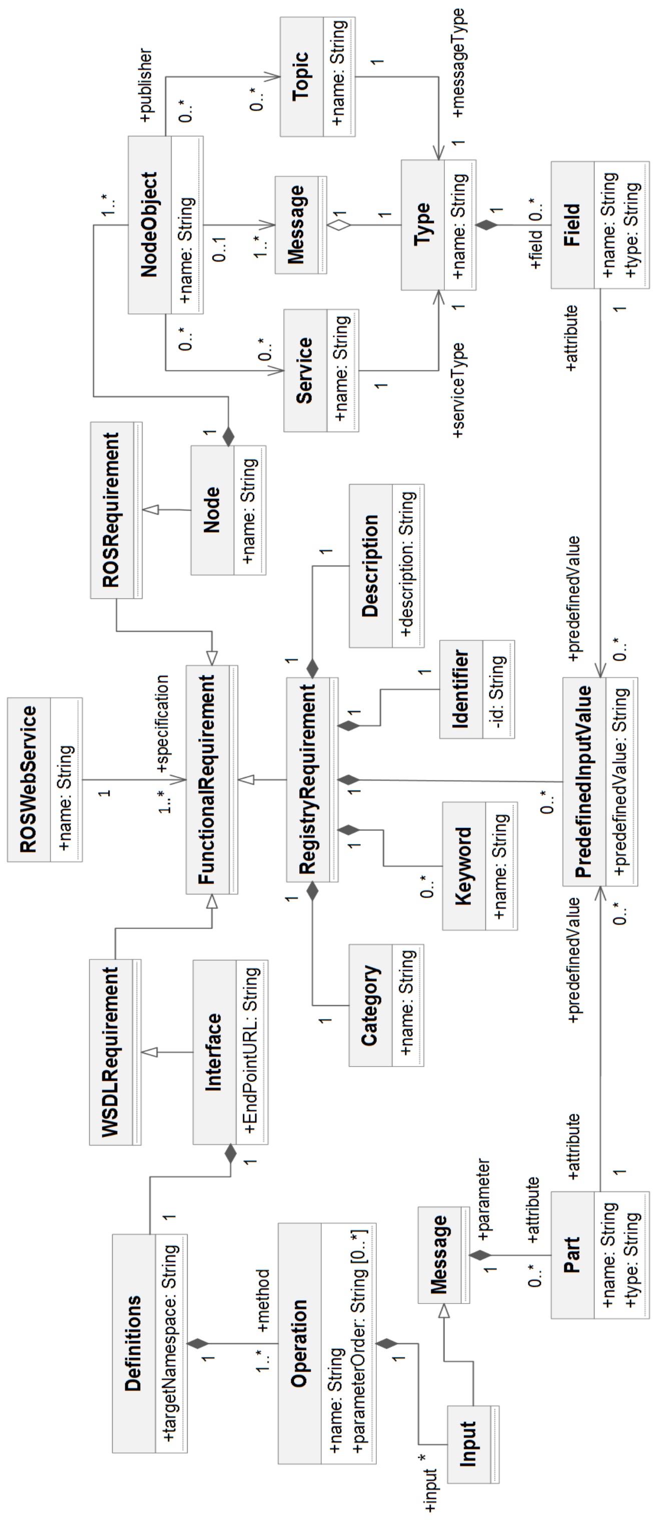


Figure 5.3: ROS Web service (ROS-WS) metamodel.

- *Message type*: *Messages* describes the communication of nodes. Their *message types* are important data structure that characterize the robot action.
- *Topic*: Which express the transport mechanism for message publication.
- *Service*: This refers to possible synchronous communication between nodes through request/reply mechanism, which is similar to the notion of remote procedure call.
- *Service Type*: Each service has an associated *service type*.

WSDL Requirements

The *WSDL Requirements* are the set of web service data that are obtained by the web services' WSDL description, which enables the communication with client applications. Mainly, these requirements are:

- *Service Name*.
- *WSDL location*: We can access WSDL documents due to their links of URLs.
- *Operations*: The *operations* are the public methods of services that client applications can invoke. An operation refers to the ROS method that enables a node to perform a robot behavior.
- *Inputs*: The *inputs* are the parameters used in each web service operation. They represent the set of ROS message fields that parametrize the desired task of the robot.
- *Output*: The *output* is the action of a ROS capability, which can be derived from message fields.

Registry Requirements

The *Registry Requirements* are the set of requirements that are indicated in the service repository by the providers of services, in order to describe their services.

- *Category*: The *category* of services is a kind of robotic tasks that share the same context of use. Thus, it classifies the set of services according to such context. For example, "object recognition" category includes different services from the category of "grasping", which collects the services that enable the robot to grasp objects.

- *Description*: A *description* is intended to provide everything relative to the use of services for users. It is obtained from the description and names of fields of ROS messages and services.
- *Keywords*.
- *Predefined Input Values*: In some cases, a list of predefined values must be determined in input fields to ensure the correct use of web services. These values depend on ROS specification of messages fields. For instance, the field `joint_names[]` of `naoqi_bridge_msgs/JointAnglesWithSpeed`[2] message, which controls the defined NAO joints, requires a set of predefined values for the joints of *head*, *left arm*, *left leg*, *right leg*, and *right arm* as follows[3]: `HeadYaw`, `HeadPitch`, `LShoulderPitch`, `LShoulderRoll`, `LElbowYaw`, `LElbowRoll`, `LWristYaw`, `LHand`, `RShoulderPitch`, `RShoulderRoll`, `RElbowYaw`, `RElbowRoll`, `RWristYaw`, `RHand`, `LHipYawPitch`, `LHipRoll`, `LHipPitch`, `LKneePitch`, `LAnklePitch`, `RAnkleRoll`, `RHipYawPitch`, `RHipRoll`, `RHipPitch`, `RKneePitch`, `RAnklePitch`, `LAnkleRoll`.

In addition, to define the services among others, an *Identifier* (*ID*) is a unique information that is generated for this purpose.

5.3.2 ROS-WS discovery

The request-service matching is applied according to the user query that denotes his needs. We briefly present in the following the SBERT model that we have used and the training dataset.

Overview of semantic textual similarity with Sentence-BERT

Semantic Textual Similarity (STS) is a fundamental task for modeling and understanding the meaning in numerous research applications such as question answering, semantic search, and machine translation. STS estimates the semantic equivalence of two sentences by measuring their meaning similarity [141]. The STS research area has been evolved from earlier methods of lexical semantics and basic syntactic similarity to deep learning models [141]. One of more recent and performed models is Sentence-BERT (SBERT) [7], which is designed based on BERT

[2]See the case study in section 5.5.
[3]NAO - Joints: `http://doc.aldebaran.com/2-1/family/robots/joints_robot.html`.

(Bidirectional Encoder Representations from Transformers) model [142]. BERT was introduced to pretrain deep bidirectional representations by conditioning all layers on both the right and left environment.

BERT shows that the need for task-specific architectures which are heavily-engineered is reduced [142], however, it requires that sentences are fed into the network [7]. To overcome the massive computational overhead of this case, SBERT imported a modification to the pretrained BERT network by using siamese and triplet networks to create embeddings of sentences that are semantically meaningful [7]. To derive a fixed sized sentence embedding in SBERT model, a pooling operation is added to the output of BERT. On the other hand, siamese and triplet networks were created for fine-tuning BERT for updating the weights so that the embeddings produced are semantically meaningful. Given two sentence embeddings *u* and *v*, their comparison can be computed with cosine-similarity as illustrated in Figure 5.4.

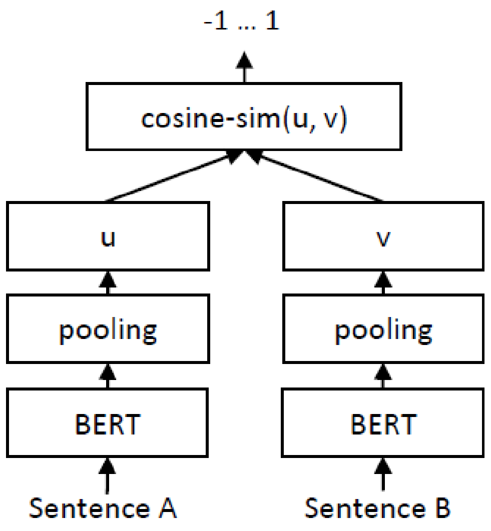


Figure 5.4: Similarity score computation and SBERT architecture at inference [7].

Applying Sentence-BERT and continue training with ROS-kit reinforcement

The service discovery process is relying on a similarity study for determining the set of suitable services. In order to illustrate the flow of actions of the service discovery process, we used the UML activity diagram, as shown by Figure 5.5. We compute the similarity score between sentence embeddings of each service and user query using sentence-BERT, by reinforcing the training dataset.

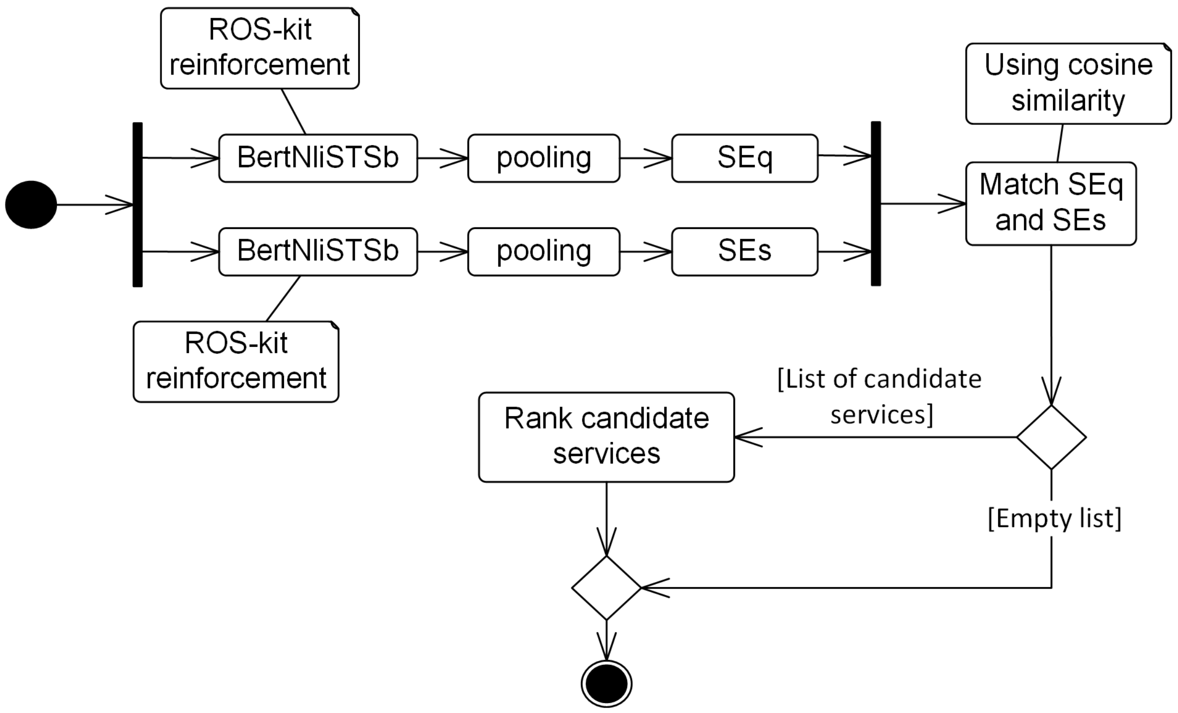


Figure 5.5: UML activity diagram for applying Sentence-BERT.

We used the pre-trained model `bert-base-nli-stsb-mean-tokens` [7,143] for generating sentence embeddings for the descriptions of each query and web services. The model was first fine-tuned on NLI dataset, then fine-tuned on STS benchmark (STSb). This strategy has improved the performance of BERT, which is shown by Spearman's rank correlation between the cosine-similarity of the sentence embeddings [7].

To continue the training in our case, we propose to continue training on this fine-tuned model according to ROS messages (and services) and KIT dataset, by following the sentence pairs in STSb.

STS benchmark (STSb) dataset

STSb[4] dataset [7,141] is a collection of 8,628 sentence pairs that is generally used to evaluate supervised STS systems. The sentence pairs are obtained from three categories: news, captions, and forums. Each pair of sentences in the dataset is annotated with a score that denotes the semantic meaning of sentences. The score ranges from 0 which indicates that the two sentences are completely different to 5, which denotes

[4]STSbenchmark: `http://ixa2.si.ehu.eus/stswiki/index.php/STSbenchmark`.

the opposite.

KIT dataset

The KIT Motion-Language Dataset [144] combines human motion and descriptions thereof in natural language, and was used in human-robot communication applications (e.g. [145]). Although this different context, the set of its terms of descriptions match significantly the robot tasks. Thus, we used a set of sentences of the KIT dataset to describe the robotic tasks because we did not find a robot dataset of sentence pairs for semantic robot tasks similarity.

ROS-kit reinforcement

The main purpose of ROS-kit reinforcement phase is to determine the relation between ROS descriptions and robot tasks. We extend the kit dataset with ROS as training data to fine-tune our network. By following the scores of sentence pairs in STSb, we generate a set of sentence pairs with scores as follows:

$$(Score, ROS\_description, Robot\_task)$$

For *`ROS descriptions`*, we involve a set of most common descriptions given by ROS messages and services including some `common_msgs`[5] and others. For *`robot tasks`* on the other side, we extend kit dataset using "`robot`" word in each sentence instead of words like "`person`" or "`human`".

We annotated each sentence pair with a score of their semantic relation. Table 5.1 illustrates an example of ROS-kit dataset. The two given ROS descriptions are obtained from `geometry_msgs/Twist` message and `naoqi_bridge_msgs/CmdPoseService` service respectively. We generate 370 sentence pairs that were splitted into `train`, `dev` and `test`.

Matching queries with services

We compute the similarity score of sentence pair of service-query by calculating the similarity between their embeddings. For each pair, we compute the degree of similarity between sentence embeddings of the service *SEs* and sentence embeddings of the query *SEq* using the common measure *cosine similarity* [7], as shown in Fig.

[5]`http://wiki.ros.org/common_msgs`.

Table 5.1: A sample of ROS-kit dataset

Score	Sentence 1	Sentence 2	Clarification
5	Twist expresses velocity in free space broken into its linear and angular parts	robot moves forward a brief distance in a fast walk and stops	There is a semantic equivalence. Twist enables the control of walking speed
0	Twist expresses velocity in free space broken into its linear and angular parts	A robot sits down on a low platform	Twist does not enable a sitting task. Meaning completely different
5	Command pose as service	A robot walks forward	There is a semantic equivalence. CmdPoseService enables the walking control
3	Command pose as service	robot moves forward a brief distance in a fast walk and stops	CmdPoseService enables the walking control, however, it does not enable the control of velocity

5.5. The measure ranges between 0, which indicates the total dissimilarity, and 1 that denotes the opposite.

5.4 ROS Semantic Web Service (ROS-SWS): Description and discovery

The ROS Semantic Web Service (ROS-SWS) contribution is designed to enhance the syntactic description of ROS-WS and to add a semantic layer to these services. We present in the following the ROS-SWS description and discovery.

5.4.1 OWL-S Profile extension for ROS-SWS

OWL-S is an ontology that makes a service semantically described. It consists of three parts: the `service profile`, the `process model`, and the `grounding` [28]. The "*service profile*" is used to describe the services properties and capabilities for automating web service discovery. As it is is the scope of this paper, the focus of our work is to make the ROS capabilities as a part of the `OWL-S profile ontology`.

Indeed, OWL-S Profile enables adding additional attributes and parameters via the "*ServiceParameter*" class [28] as an expandable list of properties that may accompany a profile description (e.g. [146], [147]). Thus, we used this class to define the

OWL-S profile extension for ROS-SWS as illustrated in Figure 5.6. It defines the "*ROSParameter*" class as a kind of *ServiceParameter*, in which the `rosParameter` property points to the value of a parameter within the proposed "*ROSCharacteristic*" ontology. The core structure of *ROSCharacteristic* ontology defines the following properties:

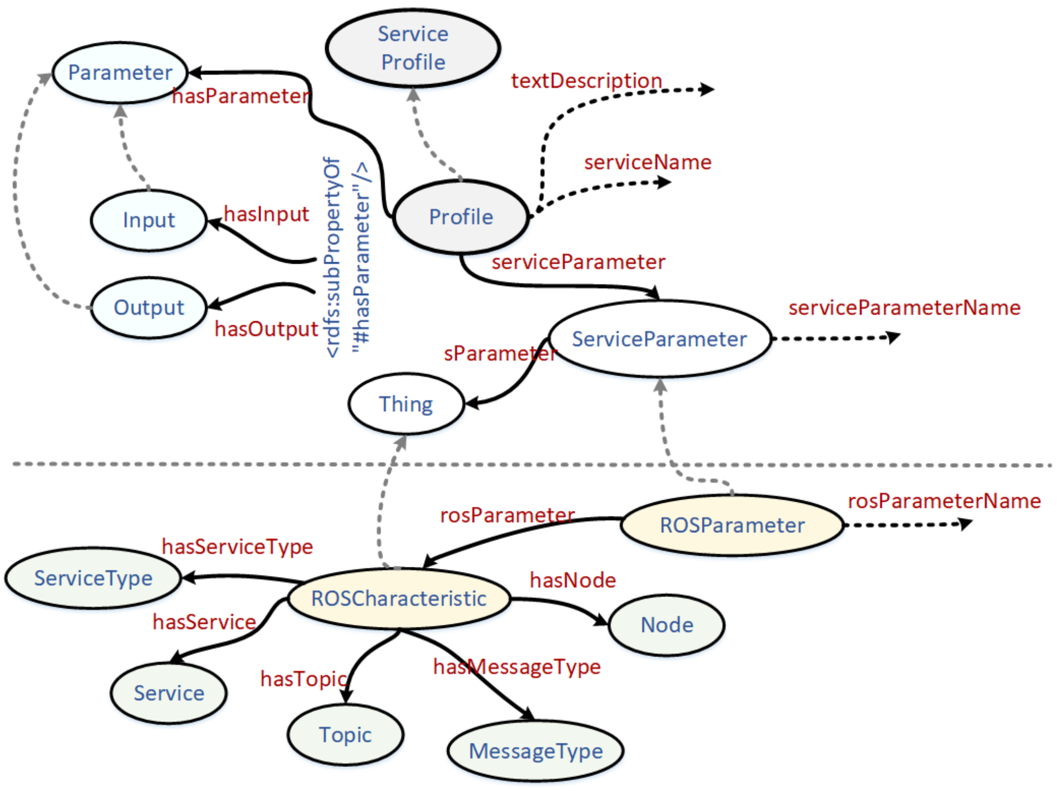


Figure 5.6: Proposed OWL-S profile extension for ROS-SWS.

- *hasNode*: ranges over instance of the ROS node.
- *hasMessageType*: ranges over instances of message types.
- *hasTopic*: ranges over instances of the used topics.
- *hasService*: ranges over instances of the used services.
- *hasServiceType*: ranges over instances of service types.

By including these `ROS Requirements`, everything relative to the use of ROS web services will be provided by their service description. This enables to determine the ROS process and its communication mechanisms of each robotic task offered by the service.

5.4.2 Mapping ROS messages to Inputs/Outputs

In our proposal, we use the convenient property of ROS messages (as highlighted in subsection 5.2.2 of section 5.2) to define the `capabilities` and `properties` (parameters) of the robotic task and mapping them to service outputs and inputs respectively. This is because the ROS messages and services control the flow of web services layer. Every ROS web service is built principally using them as a function that requires a set of inputs and provide action as output. However, it is not necessary use the whole fields as properties, which depends on the use context.

5.4.3 ROS capabilities and properties: Domain ontology

To design the domain ontology, we have based on most common components of ROS messages and services including common_msgs [6] and others. Initially, we have defined main concepts (classes) and relations to identify the ontology hierarchy of both capabilities and properties.

In parallel, during the research review of the ontology modelling of previous studies, we found that the existing RoboEarth ontology of [148] covers the semantic interpretation of ROS capabilities. The robot capabilities ontology [148] is described in Semantic Robot Description Language (SRDL) [149], which is mainly based on robot components (sensors, actuators and control programs). Although this different context of implementation, the set of its concepts and terms match significantly the ROS messages definition. Hence, we extend this ontology to fulfill the outputs.

On the other hand, we did not find any existing ontology that responds the set of ROS properties for service inputs, therefore, we built our ontology of properties. The implementation of the full ontology is constructed using OWL language. Figure 5.7 illustrates an overall fragment of the developed ontology using Protégé editor.

5.4.4 Search engine

The RaaS search engine displays the search findings relying on the task query of inputs and output. According to the user query that denotes his needs, the request-service matching is applied through the domain ontology. To that end, we used the four degrees of concept match between Output/Input of request (R) and Output/Input of service (S), which are identified as follows [150], [151]:

[6]`http://wiki.ros.org/common_msgs`.

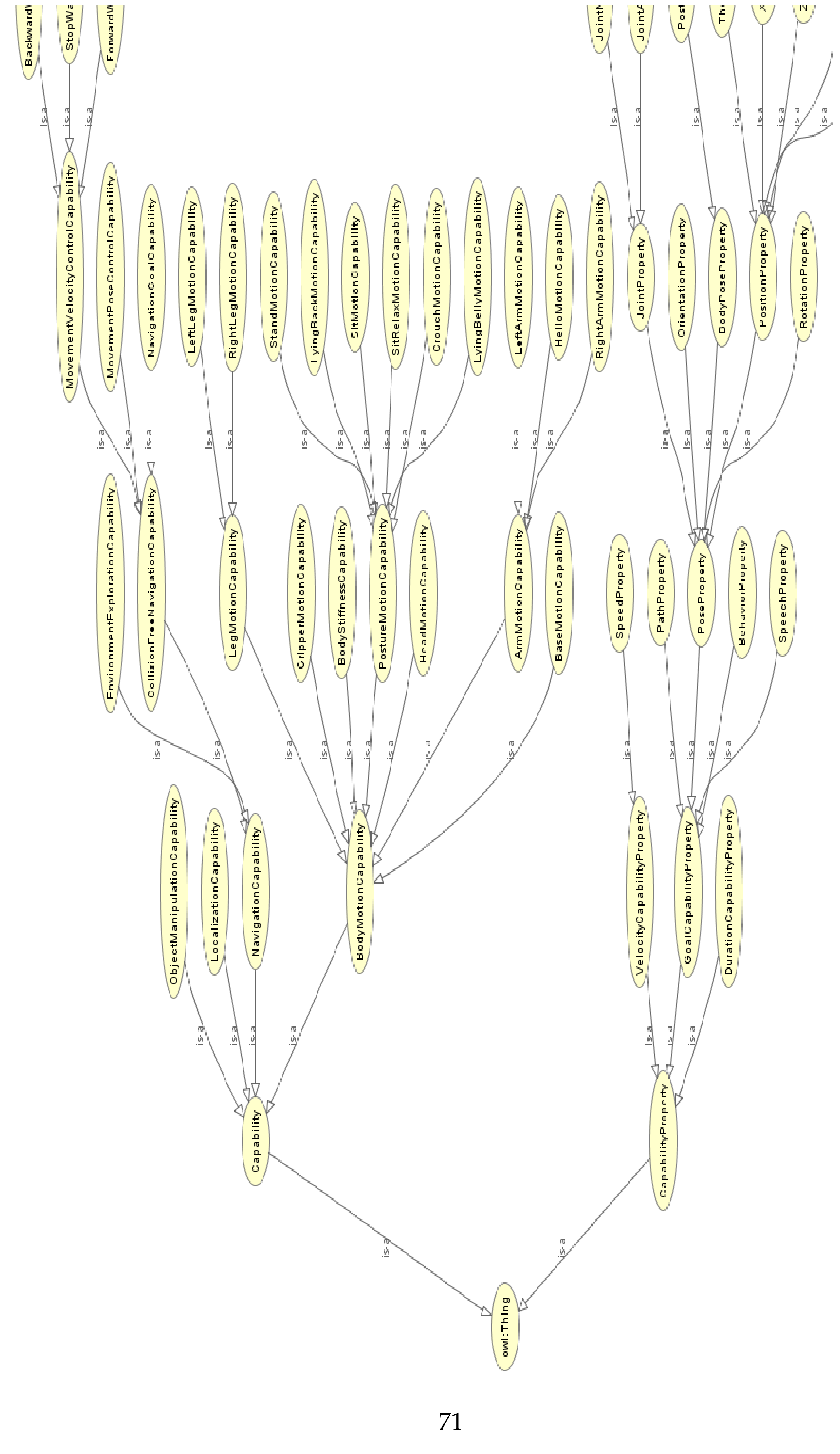


Figure 5.7: The ROS Domain Ontology.

- `exact`: if R of and S are same or if R is an immediate subclass of S.
- `plug in`: If S subsumes R.
- `subsume`: if R subsumes S.
- `fail`: a match is a fail when no subsumption relation between R and S.

The search is applied through two phases successively [150], [151]. Firstly, it distinguishes the matching between the outputs of the request and those of the services. In the second step, it applies an input matching of the request and services matched during the output phase.

5.5 Case study

We outline in this section the proposed solution for the cycle process that was applied to ROS-WS and ROS-SWS implementation. This includes ROS-WS/ROS-SWS requirements, ROS-WS/ROS-SWS functioning scenario, and the results obtained.

5.5.1 ROS Web Service experimentation

The experiments were performed over several web services that are generated from ROS codes on following functional requirements representation. This is made with the intention of using different services in a real scenario using NAO. We consider two categories of services:

- Locomotion: Collect the services that enable the robot to move or navigate around its environment.
- Joints motion and Positions: Encompasses the group of services responsible for the control and the movements of robot joints to constitute new positions.

All the web services in the database were considered in the list of compatible services for NAO. This is due to the matching of their ROS topics (T) and services (S) with those of registered (T) and (S) of the NAO. In case of `TurtleBot` robot for example, services like S2, S4, S6 are considered as non compatible services because their topics and services are not specific to `TurtleBot`. Thus, these web service can not be consumed by this robot.

ROS experimentation

The ROS requirements of the different services are presented in Tables 5.2 and 5.3, in which we outline the information of each ROS node[7], responsible of an action, according to the communication mode.

Table 5.2: ROS requirements of Topic-based services

Id	Node	Topic	Message Type
S1	nao_move	/cmd_vel	geometry_msgs/Twist
S2	nao_poses	/body_pose/goal	naoqi_bridge_msgs/BodyPoseActionGoal
S3	nao_postures	/body_pose_naoqi/goal	naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal
S5	nao_walk	/cmd_vel	geometry_msgs/Twist
S7	nao_navigate	/move_base_simple/goal	geometry_msgs/PoseStamped
S8	nao_move_pose	/cmd_pose	geometry_msgs/Pose2D
S11	nao_move_joint	/joint_angles	naoqi_bridge_msgs/JointAnglesWithSpeed

Table 5.3: ROS requirements of Service-based services

Id	Node	Service	Service Type
S4	nao_move	/cmd_vel_srv	naoqi_bridge_msgs/CmdVelService
S6	nao_move_pose	/cmd_pose_srv	naoqi_bridge_msgs/CmdPoseService
S9	stop_walk_node	/stop_walk_srv	std_srvs/Empty
S10	move_node	/cmd_pose_srv	naoqi_bridge_msgs/CmdPoseService
S12	stiffness_setting_nao	/body_stiffness/enable	std_srvs/Empty
		/body_stiffness/disable	std_srvs/Empty

Each ROS message provides a set of fields. As an example, the message *naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal* of S3 has has the following components[8]:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
actionlib_msgs/GoalID goal_id
  time stamp
  string id
naoqi_bridge_msgs/BodyPoseWithSpeedGoal goal
```

[7]Given during program creation by proposition unlike the names of Topics, services and Messages/Services Type that are obtained from ROS.

[8]Displayed by `rosmsg show naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal` command.

```
string posture_name
float32 speed
```

By determining the `posture_name` and the `speed` as inputs[9] (as shown in the structure of the message), the robot will perform a predefined posture with the speed that has been specified.

The postures are the list of predefined values of `posture_name` field. The predefined values of this input are: {`Stand`, `Sit`, `StandInit`, `StandZero`, `Crouch`, `SitRelax`, `LyingBelly`, `LyingBack`}[10,11].

We started with [136,152] to implement these services using rosjava. We present in the following the rosjava classes for implementing the S3 service as an example of all ROS-WSs implementation. The organization of rosjava classes is made through three main classes. First, we implement the `NaoPostures.java` class that inherits from `Properties.java` class as shown in Listing 5.1 and 5.2 respectively. The `NaoPostures.java` class has been realized to implement the node that publishes the *naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal* message through the */body_pose_naoqi/goal* topic so that the NAO can achieve one of the postures. Any rosjava class should extend the `AbstractNodeMain` class that contains the necessary predefined methods. We describe the main instructions of a rosjava code using a set of comments in the presented listings.

Listing 5.1: Publishing *naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal* of S3.

```
package posture;

import org.ros.concurrent.CancellableLoop;
import org.ros.node.ConnectedNode;
import org.ros.node.topic.Publisher;

public class NaoPostures extends Properties {

// onStart refers to the entry point of node
// ConnectedNode is used for defined methodes for publishers and subscribers
  public void onStart ( final ConnectedNode connectedNode ) {

// create a publisher for naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal using the topic
    body_pose_naoqi/goal
    final Publisher<naoqi_bridge_msgs.BodyPoseWithSpeedActionGoal> publisher =
        connectedNode.newPublisher("body_pose_naoqi/goal" , naoqi_bridge_msgs.
        BodyPoseWithSpeedActionGoal._TYPE ) ;
```

[9]It is not necessary to use all the message fields in some cases.
[10]NAO's Predefined postures: `http://doc.aldebaran.com/2-1/family/robots/postures_robot.html`.
[11]We identified this list through the related ROS packages that have been installed.

```
// CancellableLoop is used to publish the message and sleep in a loop
     connectedNode.executeCancellableLoop(new CancellableLoop() {
          protected void setup() {
          }
          protected void loop() throws InterruptedException {
      // create a new message
              naoqi_bridge_msgs.BodyPoseWithSpeedActionGoal pose = publisher.newMessage();
                if ( getPosture_name().equals("Crouch" )||
               getPosture_name().equals("LyingBack" )||
               getPosture_name().equals("LyingBelly" )||
               getPosture_name().equals("Sit" )||
               getPosture_name().equals("SitRelax" )||
               getPosture_name().equals("Stand" )||
               getPosture_name().equals("StandInit" )||
               getPosture_name().equals("StandZero" )
              ) {
         // set the parameters of the message by the indicated posture and speed
                pose.getGoal().setPostureName(getPosture_name()) ;
                pose.getGoal().setSpeed(getSpeed());
                // publish the message
                publisher.publish(pose);
              }
         else
                System.out.println("the posture is not defined");
     Thread.sleep(1000); }
          });
        }
      }
```

Listing 5.2: Properties.java class of S3.

```
package posture;

import org.ros.node.AbstractNodeMain;
import org.ros.namespace.GraphName;

// any rosjava class should extend the AbstractNodeMain class
public class Properties extends AbstractNodeMain{

  protected String posture_name ;
  protected float speed ;

  public String getPosture_name() {
    return posture_name;
  }
  public void setPosture_name(String posture_name) {
    this.posture_name = posture_name ;
  }
  public float getSpeed() {
    return speed;
  }
  public void setSpeed(float speed) {
    this.speed = speed ;
  }
  public GraphName getDefaultNodeName ( ) {
     // return the default name of the node
    return GraphName.of( "nao_postures" ) ;
    }
}
```

To execute the postures and see the results on our robot, we used the `PosturesNAO.java` class that utilizes the method `execute` of the rosjava `NodeMainExecutor` class, as presented in Listing 5.3.

Listing 5.3: Execute the postures of S3 on Nao robot.

```
package posture;

import org.ros.node.DefaultNodeMainExecutor;
import org.ros.node.NodeConfiguration;
import org.ros.node.NodeMainExecutor;
import com.google.common.base.Preconditions;
import java.net.URI;

public class PosturesNAO {

  public void applyPosture(String posture, float speed) {

    NaoPostures postureNode = new NaoPostures();
    postureNode.setPosture_name(posture);
    postureNode.setSpeed(speed);

    NodeMainExecutor nodeMainExecutor = DefaultNodeMainExecutor.newDefault();
     // indicate the ROS_MASTER_URI
    URI masterUri = URI.create("http://sapp2:11311");
     // indicate the NAO_IP, in case of simulation NAO_IP = 127.0.0.1
    String robotIP = "xxx.xxx.xxx.xxx";
    NodeConfiguration postureNodeConfiguration = NodeConfiguration.newPublic(robotIP ,
        masterUri);
    Preconditions.checkState(postureNode != null);
    nodeMainExecutor.execute(postureNode , postureNodeConfiguration);
    }
}
```

In case of Service-based services as presented in Table 5.3, we implement client classes instead of publishing messages. We give an example of S9 client class in Listing 5.4. There is no exchanged data between the client and the service with the ROS service type `std_srvs/Empty` of S9. The `Empty` service type of the service package `std_srvs` does not contain fields. Executing the code will stop your walked robot, by enabling the communication with the robot like Listing 5.3.

Listing 5.4: S9 example of client class in case of service communication.

```
package stopService;

import org.ros.node.AbstractNodeMain;
import org.ros.namespace.GraphName;
import org.ros.node.ConnectedNode;
import org.ros.node.service.ServiceClient;
import org.ros.node.service.ServiceResponseListener;
import std_srvs.EmptyResponse;

public class CallStopService extends AbstractNodeMain{

  public GraphName getDefaultNodeName () {
    return GraphName.of( "stop_walk_node" ) ;
    }

  public void onStart(final ConnectedNode connectedNode) {

  ServiceClient<std_srvs.EmptyRequest, std_srvs.EmptyResponse> serviceClient = null;
  try {
  serviceClient = connectedNode.newServiceClient("stop_walk_srv", std_srvs.Empty._TYPE);
} catch (org.ros.exception.ServiceNotFoundException e1) { e1.printStackTrace(); }

  std_srvs.EmptyRequest pose =  serviceClient.newMessage();

  serviceClient.call(pose, new ServiceResponseListener<std_srvs.EmptyResponse>() {
  public void onFailure(org.ros.exception.RemoteException arg0) {
  }
  public void onSuccess(EmptyResponse arg0) {
  }
  });
}
}
```

ROS-WS experimentation

All the implementation details are encapsulated by the Web Services' WSDL description, which enables the communication with client applications using the host address of each Web service. The following WSDL document (See Listing 5.5) describes the web service that transported the previous presented ROS message of S3, using the topic */body_pose_naoqi/goal* as stated in Table 5.2. We generate it automatically from the class `PosturesNAO.java` as provided by Eclipse for generating a web service using Apache Tomcat. The service requires *posture* and *speed* as inputs.

Listing 5.5: WSDL File of S3

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://posture" xmlns:apachesoap="http://xml.apache.org/
    xml-soap" xmlns:impl="http://posture" xmlns:intf="http://posture" xmlns:soapenc="http:
    //schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org
    /2001/XMLSchema">
  <wsdl:message name="applyPostureResponse">
  </wsdl:message>
  <wsdl:message name="applyPostureRequest">
    <wsdl:part name="posture" type="xsd:string">
    </wsdl:part>
    <wsdl:part name="speed" type="xsd:float">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="PosturesNAO">
    <wsdl:operation name="applyPosture" parameterOrder="posture speed">
      <wsdl:input message="impl:applyPostureRequest" name="applyPostureRequest">
     </wsdl:input>
      <wsdl:output message="impl:applyPostureResponse" name="applyPostureResponse">
     </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="PosturesNAOSoapBinding" type="impl:PosturesNAO">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="applyPosture">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="applyPostureRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://posture" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="applyPostureResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://posture" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="PosturesNAOService">
    <wsdl:port binding="impl:PosturesNAOSoapBinding" name="PosturesNAO">
      <wsdlsoap:address location="http://localhost:8080/ROS_WserviceNAO/services/
          PosturesNAO"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

For accessing the needed Web service, client applications can parse the WSDL documents to determine the following data: (*i*) the Web service name and its operations, (*ii*) the input message of each operation, (*iii*) the part of the message, which indicate its structure (the name and the type of parameters).

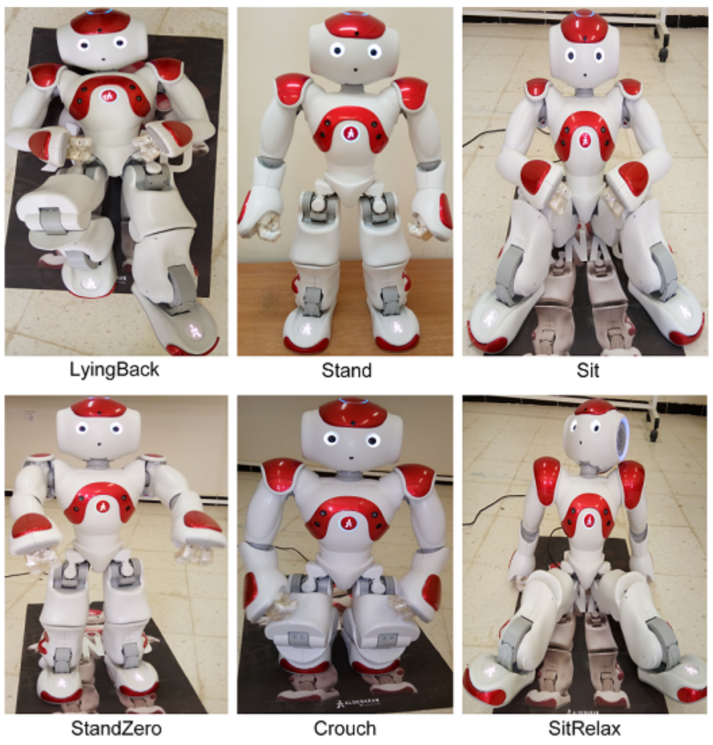


Figure 5.8: Some NAO's postures execution when invoking the Web service.

Listing 5.6 shows the Request Envelope in case of invoking the S3 Web service, which meets the needs of the user, with `posture` = `Sit` and `speed` = `1` as parameters.

Listing 5.6: SOAP Request Envelope of S3 service invocation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns0="
    http://posture" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns0:applyPosture>
      <posture soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="
          xsd:string">Sit</posture>
      <speed soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="
          xsd:float">1</speed>
    </ns0:applyPosture>
  </soapenv:Body>
</soapenv:Envelope>
```

The execution of the different postures when invoking this Web service is illustrated with NAO in Figure 5.8. We can see the running node[12] of the invoked service in Figure 5.9 thanks to the `rqt_graph` command line.

[12]The node is called `nao_postures` as named in the class `Properties.java`.

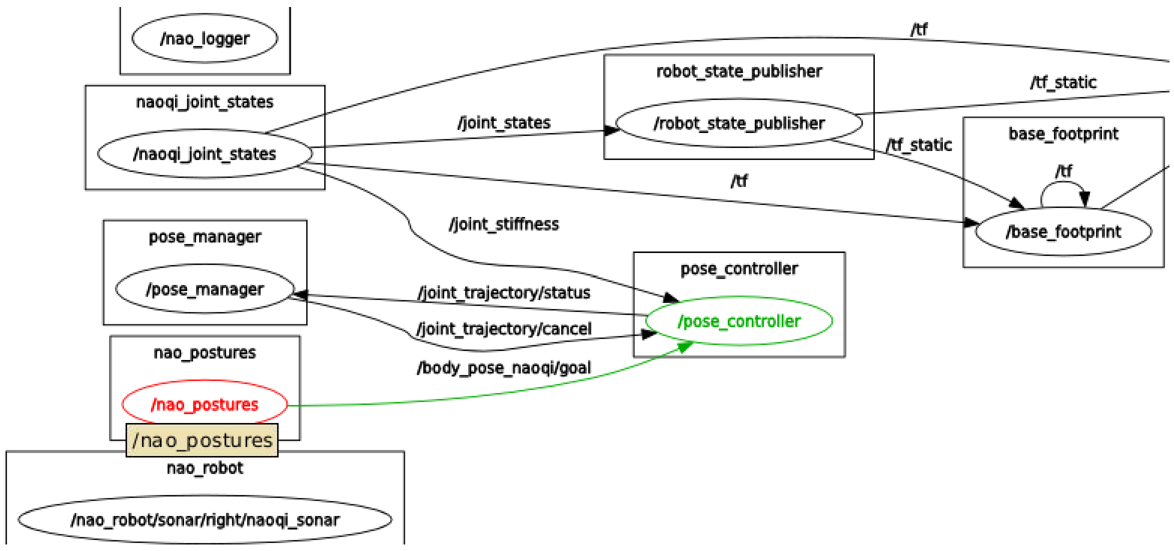


Figure 5.9: The running node "nao_postures" of the invoked Web service.

ROS-SWS experimentation

Table 5.4 summarizes the set of Inputs and Outputs of the different developed services. We have added S13, S14, S15, and S16 for ROS-SWSs (that require `naoqi_bridge_msgs/BodyPoseActionGoal`, `naoqi_bridge_msgs/JointAnglesWithSpeed`, `naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal` and `geometry_msgs/Twist` respectively) in comparison with the services presented in Tables 5.2 and 5.3.

Mapping ROS components to Inputs/Outputs was accomplished following the definition of each used ROS message or service and their fields. For example, we have outlined for S3 "*PostureProperty*" and "*SpeedProperty*" as inputs, based on its message fields, and "*BodyPostureMotionCapability*" as output, by linking them to the ontology concepts as shown in Figure 5.7. The following OWL-S document (See Listing 5.7) describes the ROS-SWS of S3.

Listing 5.7: OWL-S profile for ROS-SWS of S3

```
<profile:Profile rdf:about="http://www.owl-ontologies.com/Ontology1613388520.owl#
    ProfilePosturesNAOService">
  <profile:textDescription rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Different motions like standing up and sitting down, with a speed, are provided by
      this service</profile:textDescription>
  <profile:hasInput>
    <process:Input rdf:about="http://www.semanticweb.org/hp/ontologies/2021/0/
        DomainOntology.owl#PostureProperty"/>
  </profile:hasInput>
  <profile:hasInput>
    <process:Input rdf:about="http://www.semanticweb.org/hp/ontologies/2021/0/
```

Table 5.4: Inputs and Outputs of ROS-SWSs

Id	Input	Output	Id	Input	Output
S1	OrientationProperty	NavigationCapability	S9	–	StopWalkingCapability
S2	PoseProperty	PostureMotionCapability	S10	–	NavigationCapability
S3	PostureProperty, SpeedProperty	PostureMotionCapability	S11	JointNameProperty, JointAngleProperty, SpeedProperty	BodyMotionCapability
S4	SpeedProperty	ForwardWalkingCapability	S12	–	BodyStiffnessCapability
S5	XProperty, YProperty, ZProperty	MovementVelocityControlCapability	S13	HelloProperty	HelloMotionCapability
S6	PositionProperty	MovementPoseControlCapability	S14	JointAngleProperty	LeftArmMotionCapability
S7	SpeedProperty, OrientationProperty	NavigationGoalCapability	S15	SpeedProperty, SitProperty	SitRelaxMotionCapability
S8	XProperty, YProperty, ThetaProperty	MovementPoseControlCapability	S16	VelocityCapabilityProperty	BackwardWalkingCapability

```
          DomainOntology.owl#SpeedProperty"/>
  </profile:hasInput>
  <profile:hasOutput>
    <process:Output rdf:about="http://www.semanticweb.org/hp/ontologies/2021/0/
          DomainOntology.owl#BodyPostureMotionCapability"/>
  </profile:hasOutput>
  <profile:serviceName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        PosturesNAOService</profile:serviceName>
  <profile:serviceParameter>
    <ROSParameter rdf:ID="ROSParameterPosturesNAOService">
      <rosParameterName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >ParameterPosturesNAOService</rosParameterName>
      <rosParameter>
        <ROSCharacteristic rdf:ID="ROSCharacteristicPosturesNAOService">
          <hasNode>
            <Node rdf:ID="nao_postures"/>
          </hasNode>
          <hasTopic>
            <Topic rdf:ID="/body_pose_naoqi/goal"/>
          </hasTopic>
          <hasMessageType>
            <MessageType rdf:ID="naoqi_bridge_msgs/BodyPoseWithSpeedActionGoal"/>
          </hasMessageType>
        </ROSCharacteristic>
      </rosParameter>
    </ROSParameter>
  </profile:serviceParameter>
</profile:Profile>
```

5.5.2 Search evaluation metrics: Precision and recall

To measure the search performance, the precision and recall of the services returned were considered. These evaluation metrics are denoted as [153]:

- *Precision*: It refers the ability of identifying the most precise services. It is defined as the number of relevant services retrieved (RsR) divided by the total number of relevant (RsR) and irrelevant services retrieved (IsR).

$$\frac{\text{Number of RsR}}{\text{Number of RsR} + \text{Number of IsR}} \times 100$$

- *Recall*: It refers to the capability of retrieving the maximum number of services that match or are relevant to a query. It is defined as the relevant services retrieved (RsR) divided by the total number of existing relevant services (Rs).

$$\frac{\text{Number of RsR}}{\text{Number of RsR} + \text{Number of Rs not retrieved}} \times 100$$

5.5.3 ROS-WS search results

In addition to the URL of the service, the available services are provided with a description and a set of keywords to describe their features. We give descriptions for the services of the case study, according to the description and names of fields of their ROS messages and services, as summarized in Table 5.5.

The computed similarity score of sentence pair embeddings for search results was presented for the used pre-trained and continue-trained model (presented in section 5.3.2) to evaluate their performance. In addition, an evaluation of *tf-idf* performance was also carried out. *tf-idf* is briefly introduced in the following.

tf-idf (term frequency - inverse document frequency)

tf-idf is a numerical statistics that is designed to estimate how significant a word is to documenting a collection or corpus. It is widely applied to generate corresponding weight vectors of each service's content and user query, or between a set of items as text documents. The tf-idf weighting is given by the following equation [154]:

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$$

Table 5.5: Given descriptions for developed services

Class	Category	Id	Description of the service
Body Motions	**Locomotion**	S1	Move the base of the robot by sending velocity commands
		S4	The service allows you to send a velocity to the walking controller
		S5	The service enables the NAO to walk with linear and angular velocities
		S6	The service controls the position and orientation that makes the robot pass from one place to another
		S7	Through this service, the NAO will move to a given goal
		S8	Command the pose (x, y, theta) of the robot
		S9	Stop walking immediately
		S10	The service provides a change in the location by changing the coordinates
	Joints motion and Positions	S2	The service allows the robot to perform some body poses like hello and stand postures
		S3	Different predefined motions like standing up and sitting down, with a speed, are provided by this service
		S11	This service allows you to change the values of NAO joints: joints of the head, the arms or the legs
		S12	Setting NAO stiffness by turning on/off all motors

Where:

- $tf_{t,d}$ (term frequency) calculates the number of occurrences of term *t* in document *d*.

- idf_t (inverse document frequency) is calculated as follows:

$$idf_t \ = \ log(\frac{N}{df_t})$$

  Where *N* denotes the total number of documents in the collection, and df_t is the number of documents that contain the term *t*.

To evaluate the tf-idf weighting, we applied a keyword extraction of user query and services. Then, the tf-idf vectors of each query and services were generated by assigning a weight for each query term in the service vector. Using these vectors, the common cosine similarity was computed as follows. We denote by $Q = (q_1, q_2, ..., q_n)$ the tf-idf vector of the query, and $S_j = (s_{1j}, s_{2j}, ..., s_{nj})$ the tf-idf vector of the service$_j$ of the collection of services.

$$\begin{aligned} C(Q, S_j) &= \frac{Q \cdot S_j}{\|Q\| \cdot \|S_j\|} \\ &= \frac{\sum_{i=1}^{n} q_i s_{ij}}{\sqrt{\sum_{i=1}^{n} q_i^2} \cdot \sqrt{\sum_{i=1}^{n} s_{ij}^2}} \end{aligned}$$

Queries

The experiments of the discovery process were tested using multiple queries. We tested a set of queries that target the two categories of services. We present a sample of 9 used queries for evaluation in the following.

- *Query1*. A robot moves straight forward.
- *Query2*. A robot sits on an approximately 1m high platform.
- *Query3*. A robot moves his left arm.
- *Query4*. robot moves forward in a fast jog.
- *Query5*. A robot is standing relaxed.
- *Query6*. A robot doing a waving movement with the right hand.
- *Query7*. a robot walks slowly and stops.
- *Query8*. A sitting robot is standing up.
- *Query9*. a robot stands and then walks quickly a few steps forward.

Model-Similarity findings

Figure 5.10 presents the experimental results of cosine similarity measure, according to the different outlined queries with each web service for NAO.

The computed similarity score of sentence pair embeddings was presented for both pre-trained and continue-trained model to evaluate their performance, as well as *tf-idf*. We denote the used pre-trained model as `bertNliSTSb` and the continue fine-tuned model as `--ROSkit`.

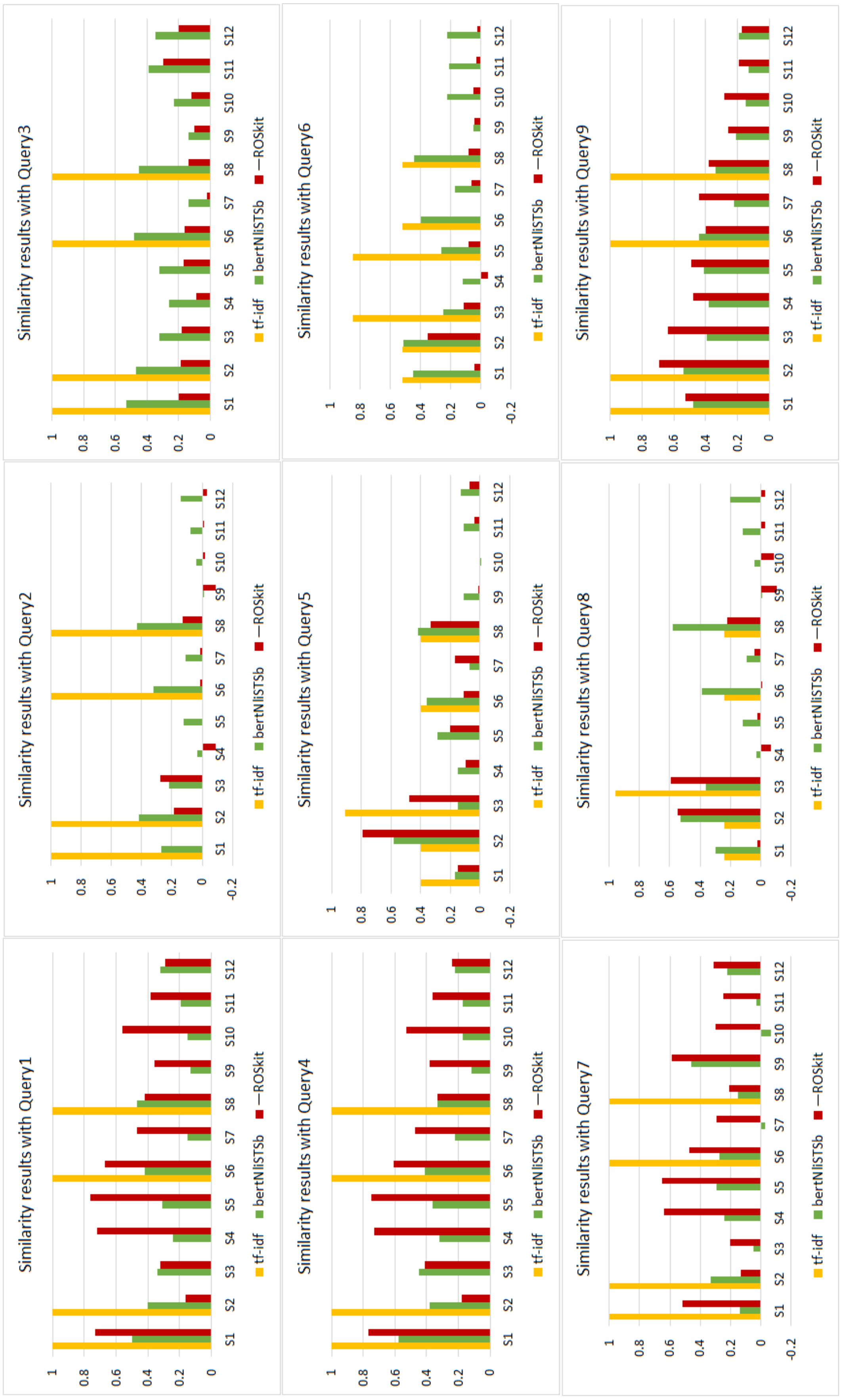


Figure 5.10: Similarity results with search queries for ROS-WS.

Figure 5.11 gives the precision and recall results for every query according to each model. We consider the significance of similarity scores that are equal or less than 0.2 negligible. Therefore, we compute the precision and recall measures for similarity findings that are equal or higher than 0.3 and 0.4 respectively.

Performance discussion

As we can observe from the finding averages in Figure 5.12, *tf-idf* performs similar findings with the precision of 47.96% and the recall of 51.96% compared to the two similarity cases (equal or higher than 0.3 and 0.4). This is because *tf-idf* retrieves the same set of candidate services without taking into consideration the semantic meaning. Most of their similarity scores are obtained due to the "robot" term that appears in the query and services containing the same word. Almost the rest of the words in each query are considered as ambiguous words, therefore, it is expected to perform significantly worse without this term in the query.

On the other hand, `bertNliSTSb` model provides a level of semantic. In the similarity case where the scores are equal or higher than 0.3 and 0.4 respectively, the average of precision is 43.27% and 51.85%, while the recall is 63.21% and 36.49%. The decrease of recall is due to topic of sentence pairs in the dataset, which is not able to give best similarity scores.

Furthermore, the results show that the ROS-kit reinforcement enhances significantly the similarity scores, which improves the performance of the discovery process. As we can observe, the proposed training enables to: (*i*) increase the similarity score values of services, and (*ii*) indicate the similitude between relevant services and user requests. The achievement of `--ROSkit` results in the improvement of the average of precision of recall compared to `bertNliSTSb`.

This is because the ROS-kit reinforcement distinguishes the relation between services and each query. Thus, the system assigns a score for semantic equivalence between ROS web services and robotic tasks according to ROS messages functionalities. In the similarity case where the scores are equal or higher than 0.3, the average of precision and recall values is 77.86% and 84.88% respectively. However, their measures were decreased to 65.08% and 58.44% respectively in the case where similarity scores are equal or higher than 0.4. Indeed, the `--ROSkit` model is still able to distinguish candidate services, however, it does not increase the similarity score of retrieved services in some cases. This is due to ROS-kit dataset that need to add

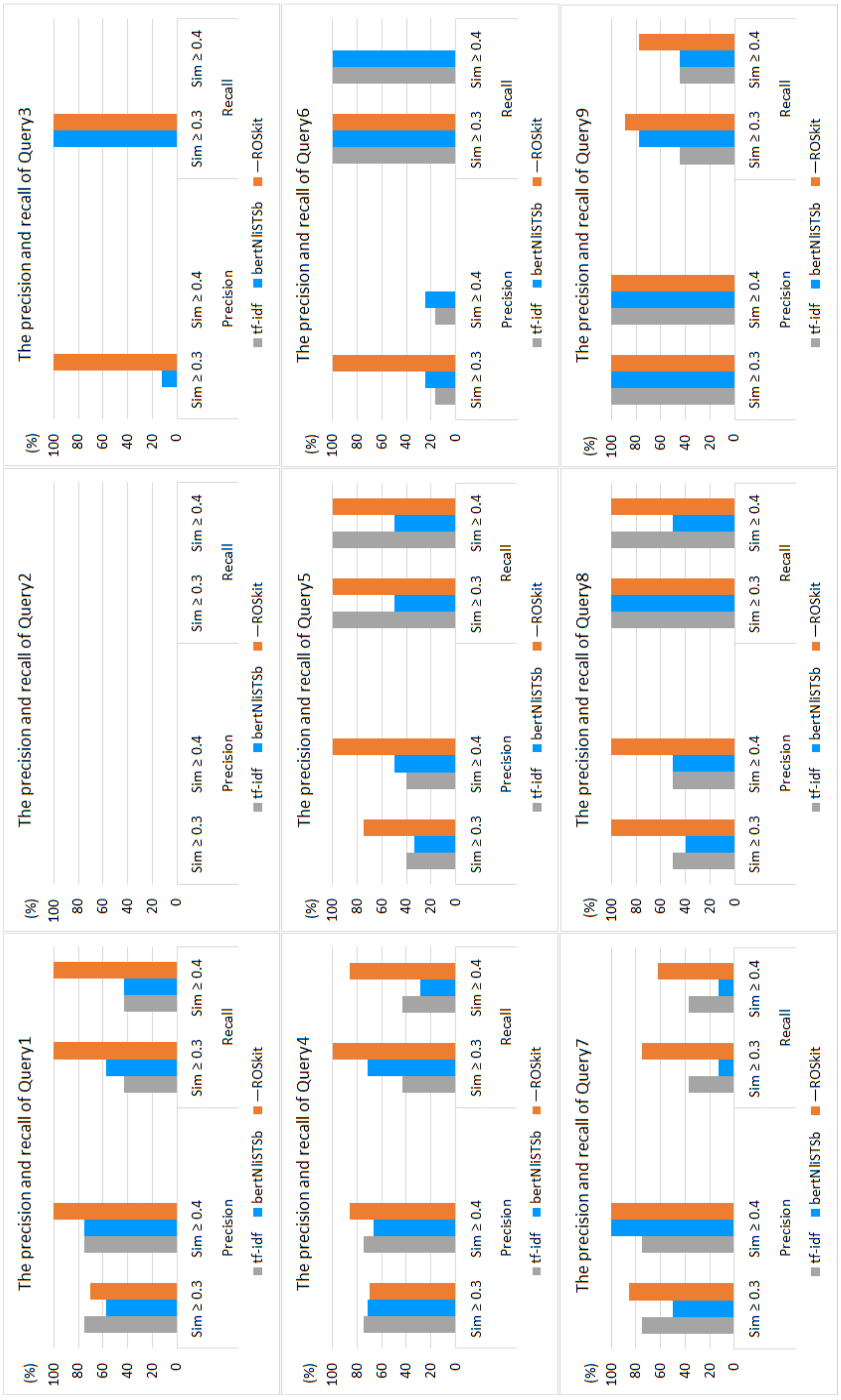


Figure 5.11: The precision and recall of search queries for ROS-WS.

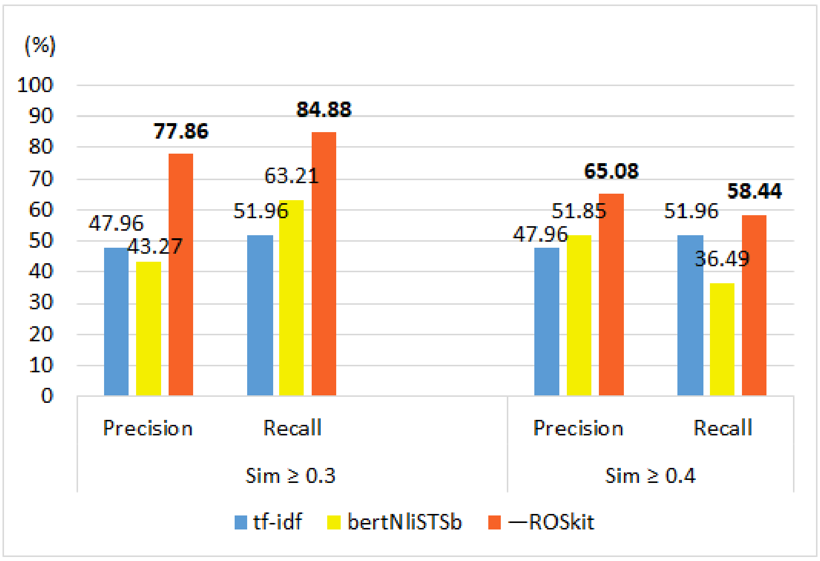


Figure 5.12: The average of precision and recall of search results for ROS-WS.

not only the specified definition given by ROS messages, but also those terms that indicate the same meaning in order to be more exhaustive in the future.

5.5.4 ROS-SWS search results

We present in the following the search results for ROS-SWSs.

Queries

The experiments of the semantic search process were tested using multiple queries. We tested 21 queries of inputs and outputs as presented in Table 5.6.

Search discussion

Figure 5.13 describes the recall and precision of the three matching degrees of search and their total for every query. Both of the obtained precision and recall of exact degree are better than the plug in and subsume degrees, which is reflected on their average.

The average of precision and recall of search results for ROS-SWS is given in Figure 5.14. The recall average of total candidate services based on the ROS domain ontology is 71.9%, while the precision is 63.75%.

Table 5.6: Search queries for ROS-SWS discovery.

Query	Input	Output	Query	Input	Output
Q1	PoseProperty	MovementPoseControlCapability	Q12	GoalCapabilityProperty	NavigationGoalCapability
Q2	JointProperty, SpeedProperty	GripperMotionCapability	Q13	PostureProperty	PostureMotionCapability
Q3	XProperty, YProperty, ZProperty	ForwardWalkingCapability	Q14	PathProperty	Collision-FreeNavigationCapability
Q4	SitProperty	SitMotionCapability	Q15	GoalCapabilityProperty, VelocityCapabilityProperty, DurationCapabilityProperty	localizationCapability
Q5	PoseProperty, PathProperty	BodyMotionCapability	Q16	OrientationProperty	BackwardWalkingCapability
Q6	BehaviorProperty	BodyMotionCapability	Q17	VelocityCapabilityProperty	HelloMotionCapability
Q7	SpeedProperty, JointAngleProperty	LeftLegMotionCapability	Q18	JointNameProperty	ArmMotionCapability
Q8	PathProperty	NavigationCapability	Q19	SpeedProperty	HeadMotionCapability
Q9	OrientationProperty	NavigationCapability	Q20	GoalCapabilityProperty	ForwardWalkingCapability
Q10	XProperty	StopWalkingCapability	Q21	XProperty, YProperty	MovementVelocityControlCapability
Q11	PathProperty, SpeedProperty, OrientationProperty	EnvironmentExplorationCapability			

5.6 Conclusion

This chapter focused on the designed approaches that enable to automate the ROS web services use. The work exploits firstly the opportunities of SOAP web services technologies to provide a service description for ROS-WS. Users can obtain the suitable robotic service by displays the search findings, according to a service-query matching score using SBERT. On the other hand, we designed a second contribution that brings a semantic layer to ROS-WS and defines the ROS-SWS. ROS-SWS characterizes the scope and capability of each service by expressing itself through a ROS domain ontology of capabilities and properties.

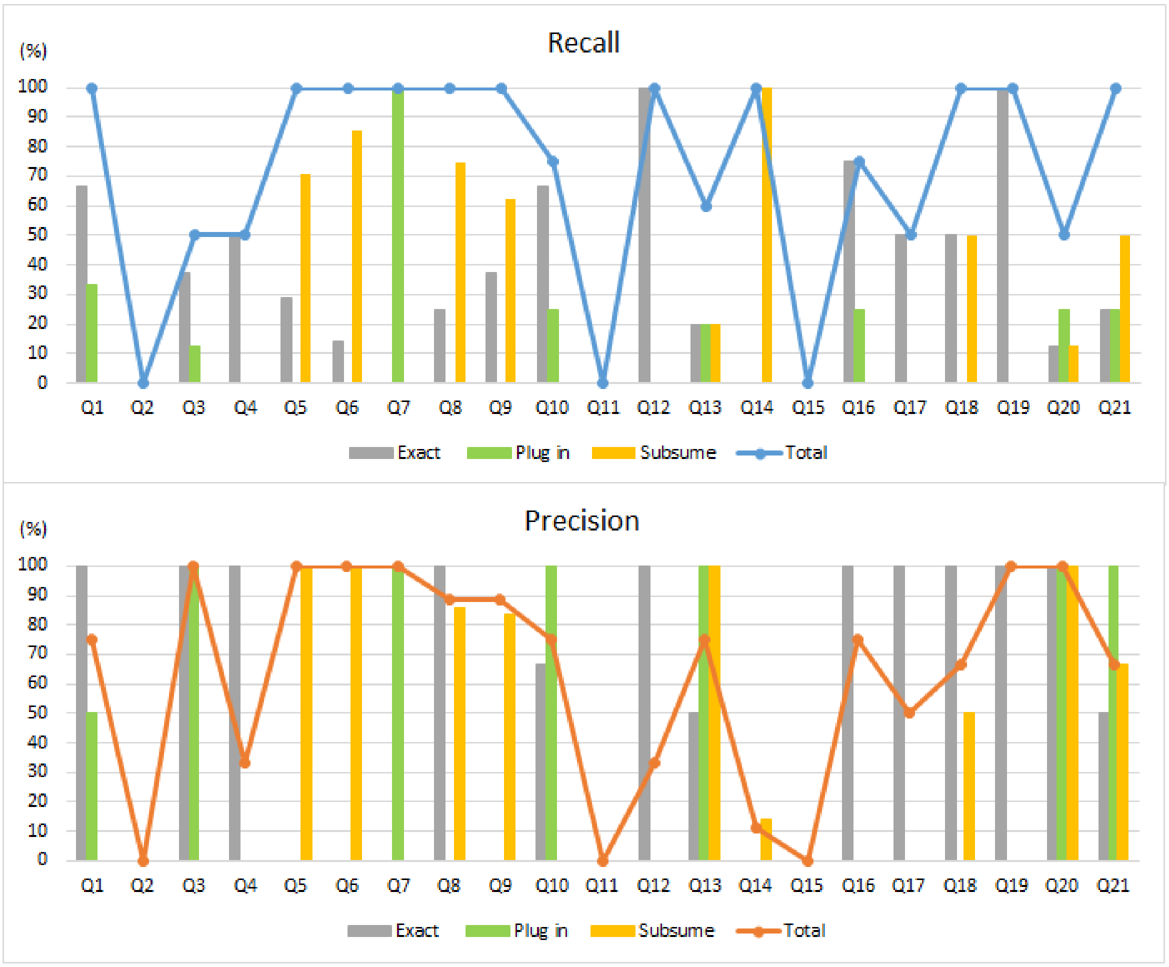


Figure 5.13: The recall and precision of search queries for ROS-SWS.

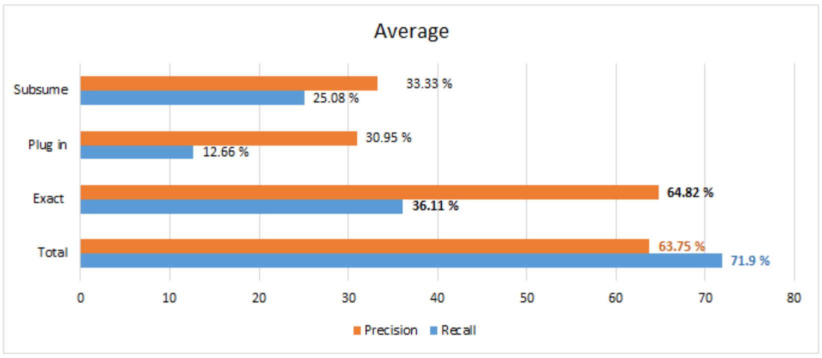


Figure 5.14: The average of precision and recall of search results for ROS-SWS.

Chapter6

Conclusion and Future Work

6.1 Conclusion

The trend towards shifting robotic applications into service-oriented solutions is growing. Various service delivery concepts and models have been proposed. We have presented a model categorization of the research literature on this topic into: Robot as a Service (RaaS), Cloud-enabled Robotic Services (CRS), Multi-Robot-based Services (MRS), and Robotic Service Composition Middlewares (RSCM). However, considerable diversity can be observed between proposals. The majority of works regard web services as a technique for building software components, and concentrate on how these services may be utilized in each case study, instead of developing an architectural style. As a result, there are two key points to consider. First, heterogeneity of robotic service representations and accessibility, in which previous research do not fully describe the service capability and characterization of the offered robotic functionality. Second, a lack of service discovery methods.

This thesis exploits the opportunities of web services technologies in terms of the life cycle process towards ROS-based robotic service provisioning. It is drawn on the foundation of two major contributions. The first entails presenting an approach for robotic services delivery in a cloud environment, as described in chapter 4. According to a defined representation, on-demand robotic tasks are expressed as ROS web services and delivered over a cloud infrastructure as a CRS solution. The second contribution consists of proposing a solution to define and locate such services. It focuses on designing an approach that enables to automate the ROS web services discovery and selection on the basis of their definition. By displays the search findings, users may find the most appropriate service for their robots based on the result of a service-query matching score. As stated in chapter 5, this contribution is divided into two main contributions. The initial contribution of ROS Web Service (ROS-WS) is based on SOAP web services and defines a set of ROS-WS characterization requirements. The ROS-WS search engine uses sentence-BERT to generate sentence embeddings in order to estimate the most suitable service of a desired task according to the user's query. In this context, we reinforce the training dataset by distinguishing the relation between ROS and robot tasks. The second contribution extends ROS-WS with a semantic layer based on the OWL-S and designs the ROS Semantic Web Service (ROS-SWS). The description of ROS-SWS expresses itself through a ROS domain ontology of capabilities and properties to handle service discovery requests.

6.2 FutureWork

There several potential challenges and perspectives for future research. We listed some of them in the following.

6.2.1 Quality of Service and ROS2

QoS aspects are main requirements for the characterization of robotic applications in service-oriented systems [6]. In the future, we aim to improve the service description and discovery by considering the QoS requirements that can meet the user's criteria. We plan to study and implement the solution in ROS 2 [155] that supports QoS policies.

6.2.2 Resource allocation in robotic service composition

As outlined by authors of [111], service discovery and selection are the key functionalities that should be extended within the robotic middleware of service composition. This can offer a flexible mechanism to respond the user's needs of complex tasks leading to knowledge sharing. In this regard, the resource allocation in such systems (e.g. [125]) is a fundamental challenge. The dynamic allocation of needed computing resources that offload the computation intensive tasks of robots requires QoS issues to be resolved. Future work should involve a comprehensive study for validating the service representation and the strategy of service discovery, within the service composition process, by taking resource allocation aspect into account.

6.2.3 Fog computing in robotic service provisioning: Fog Robotics

Robotic service provisioning over the cloud faces the challenge of network latency [62,63] and ensuring QoS [156] requirements. Network latency, which refers to the delay of cloud-robot communication that affect the response time of robot tasks, can be caused by QoS of the performance and network criteria such as cost and loss in the transmission of data packets. Indeed, this can cause serious problems especially for real-time applications such as search and rescue applications (e.g. [76]). One of the relevant computing forms that can be used to improve these issues is fog computing [77]. This emerging concept was introduced in robotics recently and named "*Fog Robotics*" [157]. Tanwani et al. define it as "*an extension of Cloud Robotics that distributes storage, compute and networking resources between the Cloud and the Edge in* *a federated manner*" [158]. Fog robotics represents an architecture in which storage, networking, control and decentralized computing are closer to robots [157]. Due to this proximity, it can improve the real-time performance of data processing and networking. Fog robotics need to be addressed in the future of SOA-based solutions (e.g. [72]).

List of Publications

- Radhia Bouziane, Labib Sadek Terrissa, Soheyb Ayad, Jean-François Brethé. Towards an architecture for cloud robotic services. International Journal of Computers and Applications. p. 1-12. 2021.
- Radhia Bouziane, Labib Sadek Terrissa, Soheyb Ayad, Jean-François Brethé, Okba Kazar. A Web services based solution for the NAO Robot in Cloud Robotics environment. The 4th International Conference On Control, Decision And Information Technologies (Codit 2017), April 5-7, 2017, Barcelona, Spain.
- Labib Sadek Terrissa, Bouziane Radhia, Soheyb Ayad, Jean-François Brethé. ROS-Based Approach for robot as a service in cloud computing. 2nd Conference on Computing Systems and Applications, December 13-14, 2016, Algiers, Algeria. At Ecole Polytechnique Militaire, Algiers, Algeria.
- Labib Sadek Terrissa, Bouziane Radhia, Jean-François Brethé. Towards a new approach of Robot as a Service (RaaS) in Cloud Computing paradigm. 5th. International Symposium ISKO-Maghreb (Knowledge Organization in the perspective of Digital Humanities: Researches and Applications), Nov 2015, Hammamet Tunisia.

Bibliography

[1] Kehoe, B., Patil, S., Abbeel, P., Goldberg, K.: A survey of research on cloud robotics and automation. IEEE Transactions on automation science and engineering, 12 (2) 398-409 (2015)

[2] ROS Website, `http://wiki.ros.org`.

[3] Ahmad, A., Babar, M.A.: Software architectures for robotic systems: A systematic mapping study. Journal of Systems and Software. 122 16-39 (2016)

[4] Oliveira, L.B.R., Osório, F.S., Nakagawa, E.Y.: An Investigation into the Development of Service-Oriented Robotic Systems. Proceedings of the 28th annual ACM symposium on applied computing, pp. 223-228 (2013)

[5] Koubâa, A.: ROS As a Service: Web Services for Robot Operating System. Journal of Software Engineering for Robotics 6(1) 1-14 (2015)

[6] Oliveira, L. B. R., Leroux, E., Felizardo, K. R., Oquendo, F., Nakagawa, E. Y. : ArchSORS: A Software Process for Designing Software Architectures of Service-Oriented Robotic Systems. The Computer Journal 60(9) 1363-1381 (2017)

[7] Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019)

[8] Niku, S.B.: Introduction to robotics: analysis, control, applications. Third Edition. John Wiley & Sons (2020)

[9] Hockstein, N. G., Gourin, C. G., Faust, R. A., Terris, D. J.: A history of robots: from science fiction to surgical robotics. Journal of robotic surgery, 1(2) 113-118 (2007)

[10] Kalan, S., et al.: History of robotic surgery. Journal of Robotic Surgery 4.3 141-147 (2010)

[11] The International Organization for Standardization (ISO) website: `https://www.iso.org/home.html`.

[12] ISO 8373:2012(en) Robots and robotic devices — Vocabulary: `https://www.iso.org/obp/ui/#iso:std:55890:en`.

[13] Ben-Ari, M., Mondada, F.: Elements of robotics. Springer Nature (2017)

[14] The International Federation of Robotics (IFR) website: https://ifr.org/.

[15] Hägele, M., Nilsson, K., Pires, J.N., Bischoff, R.: Industrial robotics. In : Springer handbook of robotics. Springer, Cham, p. 1385-1422 (2016)

[16] The International Federation of Robotics: Executive Summary World Robotics 2021 - Service Robots. Available online: https://ifr.org/img/worldrobotics/Executive_Summary_WR_Service_Robots_2021.pdf.

[17] Martinez, A., Fernández, E.: Learning ROS for robotics programming: A practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework. Packt Publishing Ltd (2013)

[18] Johns, K., Taylor, T.: Professional Microsoft Robotics Developer Studio. Wiley Publishing, Inc (2009)

[19] YARP - Yet Another Robot Platform, https://www.yarp.it/latest/.

[20] Lentin, J.: Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities. Packt Publishing (2015)

[21] W3C: Web Services Glossary - service-oriented architecture. W3C Working Group Note 11 February 2004. URL https://www.w3.org/TR/ws-gloss/#defs.

[22] Zernadji, T.: Pattern-based Approach for Quality Integration in Service-based Systems (Doctoral dissertation, Université de Biskra) (2016)

[23] Endrei, M., et al.: Patterns: Service-Oriented Architecture and Web Services. IBM Corporation, International Technical Support Organization (2004)

[24] Rosen, M., Lublinsky, B., Smith, K. T., Balcer, M. J.: Applied SOA: service-oriented architecture and design strategies. John Wiley & Sons (2012)

[25] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, third edit ed (2013)

[26] Bean, J.: SOA and web services interface design: principles, techniques, and standards. Morgan Kaufmann (2009)

[27] Ayad, S.: Une approche pour la découverte sémantique des services Web dans les réseaux mobiles ad-hoc. Doctoral dissertation (2016)

[28] OWL-S: Semantic Markup for Web Services: https://www.w3.org/Submission/OWL-S/.

[29] Web Service Modeling Ontology (WSMO): https://www.w3.org/Submission/WSMO/.

[30] Buyya, R., Broberg, J., Goscinski, A.: Cloud computing: Principles and paradigms. John Wiley & Sons (2010)

[31] Mahmood, Z.: Cloud Computing for Enterprise Architectures: Concepts, Principles and Approaches. In: Mahmood Z., Hill R. (eds) Cloud Computing for Enterprise Architectures. Computer Communications and Networks, Springer, London (2011)

[32] Hamdaqa, M., Tahvildari, L.: Cloud computing uncovered: a research landscape. In Advances in Computers. Vol. 86. Elsevier, 41-85 (2012)

[33] Srinivasan, S.: Cloud computing basics. Springer (2014)

[34] Mell, P., Grance, T.: The NIST definition of cloud computing (2011)

[35] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D.: NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292) (2012)

[36] Duan, Y., et al.: Everything as a service (XaaS) on the cloud: origins, current and future trends. 2015 IEEE 8th International Conference on Cloud Computing. IEEE (2015)

[37] Rountree, D., Castrillo, I.: The basics of cloud computing: Understanding the fundamentals of cloud computing in theory and practice. Newnes (2013)

[38] Elgazzar, K., Hassanein, H. S., Martin, P.: DaaS: Cloud-based mobile Web service discovery. Pervasive and Mobile Computing 13 67-84 (2014)

[39] Terrissa, L. S., Meraghni, S., Bouzidi, Z., Zerhouni, N.: A new approach of PHM as a service in cloud computing. 2016 4th IEEE international colloquium on information science and technology (CiSt). IEEE (2016)

[40] Rittinghouse, J.W., Ransome, J.F.: Cloud computing: implementation, management, and security. CRC press (2017)

[41] Ramachandran, M.: Component-Based Development for Cloud Computing Architectures. In: Mahmood Z., Hill R. (eds) Cloud Computing for Enterprise Architectures. Computer Communications and Networks. Springer, London (2011)

[42] Tsai, W.T., Sun, X., Balasooriya, J.: Service-Oriented Cloud Computing Architecture. 2010 Seventh International Conference on Information Technology: new generations. IEEE, pp 684-689 (2010)

[43] Barry, D.K., Dick, D.: Web Services, Service-Oriented Architectures, and Cloud Computing (2nd Edition). Morgan Kaufmann (2013)

[44] Kuffner, J.: Cloud-enabled robots, IEEE-RAS International Conference on Humanoid Robotics (2010)

[45] Waibel, M., et al.: RoboEarth. IEEE Robotics & Automation Magazine. 18(2) 69-82 (2011)

[46] Reppou, S.E., Tsardoulias, E.G., Kintsakis, A.M., et al.: RAPP: A Robotic-Oriented Ecosystem for Delivering Smart User Empowering Applications for Older People. Int J of Soc Robotics 8(4) 539-552 (2016) https://doi.org/10.1007/s12369-016-0361-z.

[47] Hu, G., Tay, W.P., Wen, Y.: Cloud Robotics: Architecture, Challenges and Applications. IEEE Network 26(3) 21-28 (2012)

[48] Mouradian, C., Errounda, F.Z., Belqasmi, F., Glitho, R.: An infrastructure for robotic applications as cloud computing services. IEEE World Forum on Internet of Things (WF-IoT) IEEE, pp. 377–382 (2014)

[49] Kamei, K., Nishio, S., Hagita, N., Sato, M.: Cloud Networked Robotics. IEEE Network 26(3) 28-34 (2012)

[50] Terrissa, L.S., Ayad, S.: Towards a new cloud robotics approach. 10th International Symposium on Mechatronics and its Applications (ISMA) IEEE (2015)

[51] Terrissa, L.S., Bouziane, R., Ayad, S., Brethé, J.F.: ROS-Based Approach for robot as a service in cloud computing. 2nd Conference on Computing Systems and Applications, December 13-14, Algiers, Algeria. At Ecole Polytechnique Militaire, Algiers, Algeria (2016)

[52] Vick, A., Vonásek, V., Pěnička, R., Krüger, J.: Robot control as a service - towards cloud based motion planning and control for industrial robots. 10th International Workshop on Robot Motion and Control (RoMoCo). IEEE, pp. 33–39 (2015)

[53] Wang, X.V., et al.: Ubiquitous manufacturing system based on Cloud: A robotics application. Robotics and Computer Integrated Manufacturing 45 (2017): 116-125. http://dx.doi.org/10.1016/j.rcim.2016.01.007.

[54] Du, Z., et al.: Robot Cloud: Bridging the power of robotics and cloud computing. Future Generation Computer Systems 74: 337-348 (2017) https://doi.org/10.1016/j.future.2016.01.002.

[55] Huang, J.Y., Lee, W.P.: Enabling vision-based services with a cloud robotic system. 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS) IEEE, pp. 84-88 (2016)

[56] Liu, B., Wang, L., Liu, M., Xu, C.Z.: Federated Imitation Learning: A Novel Framework for Cloud Robotic Systems With Heterogeneous Sensor Data. IEEE Robotics and Automation Letters 5(2) 2020 3509-3516.

[57] Manikanda Kumaran, K., Chinnadurai, M.: Cloud-based robotic system for crowd control in smart cities using hybrid intelligent generic algorithm. J Ambient Intell Human Comput (2020). https://doi.org/10.1007/s12652-020-01758-w.

[58] Mohanarajah, G., Hunziker, D., Waibel, M., D'Andrea, R.: Rapyuta: A Cloud Robotics Platform. IEEE Transactions on Automation Science and Engineering 12(2) 481-493 (2014)

[59] Tenorth, M., Beetz, M.: KnowRob: A knowledge processing infrastructure for cognition-enabled robots. The International Journal of Robotics Research 32(5) 566-590 (2013)

[60] Crick, C., Jay, G., Osentoski, S., Pitzer, B., Jenkins, O.C.: Rosbridge: Ros for non-ros users. In: Christensen H., Khatib O. (eds) Robotics Research. Springer Tracts in Advanced Robotics, vol 100. Springer, Cham (2017)

[61] RAPP project, https://rapp-project.github.io/.

[62] Wan, J., Tang, S., Yan, H., Li, D., Wang, S., Vasilakos, A.V.: Cloud robotics: Current status and open issues. IEEE Access. 4 2797-2807 (2016)

[63] Saha, O., Dasgupt, P.: A Comprehensive Survey of Recent Trends in Cloud Robotics Architectures and Applications. Robotics 7(3) 47 (2018)

[64] Chen, Y., Du, Z., Garcia-Acosta, M.: Robot as a Service in Cloud Computing. 2010 Fifth IEEE International Symposium on Service Oriented System Engineering. IEEE, pp.151-158 (2010)

[65] Osentoski, S., Pitzer, B., Crick, C., Jay, G., Dong, S., Grollman, D., Suay, H.B., Jenkins, O.C.: Remote robotic laboratories for learning from demonstration. International Journal of Social Robotics 4(4) 449-461 (2012)

[66] Chen, Y., HU, H.: Internet of intelligent things and robot as a service. Simulation Modelling Practice and Theory 34: 159-171 (2013)

[67] Koubâa, A.: A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds. International Conference on Architecture of Computing Systems. Springer, Cham, pp. 196-208 (2014)

[68] Costa, L.F., Gonçalves, L.M.G.: RoboServ: A ROS Based Approach Towards Providing Heterogeneous Robots as a Service. 2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), IEEE, pp. 169-174 (2016)

[69] Vanelli, B., Rodrigues, M., Silva, M. P. D., Pinto, A., Dantas, M. A. R.: A Proposed Architecture for Robots as a Service. In: Branco K., Pinto A., Pigatto D. (eds) Communication in Critical Embedded Systems. WoCCES 2013, WoCCES 2014, WoCCES 2015, WoCCES 2016. Communications in Computer and Information Science, vol 702. Springer, Cham, (2017)

[70] Di Napoli,C., Rossi,S.: A Layered Architecture for Socially Assistive Robotics as a Service. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), IEEE. pp. 352-357 (2019)

[71] Koubâa, A., Qureshi, B., Sriti, M. F., Allouch, A., Javed, Y., Alajlan, M., Cheikhrouhou, O., Khalgui, M., Tovar, E.: Dronemap Planner: A Service-Oriented Cloud-Based Management System for the Internet-of-Drones. Ad Hoc Networks 86 46-62 (2019) `https://doi.org/10.1016/j.adhoc.2018.09.013`.

[72] Qian, K., Liu, Y., Song, A., Li, J.: A Control System Framework Model for Cloud Robots Based on Service-Oriented Architecture. In: H.Yu et al. (Eds.): ICIRA, vol 11741. pp. 579–588 (2019)

[73] Merle, P., Gourdin, C., Mitton, N.: Mobile Cloud Robotics as a Service with OCCIware. 2017 IEEE International Congress on Internet of Things (ICIOT). IEEE, pp. 50–57 (2017)

[74] Bonaccorsi, M., Fiorini, L., Cavallo, F., Esposito, R., Dario, P.: Design of Cloud Robotic Services for Senior Citizens to Improve Independent Living and Personal Health Management. In: Andò B., Siciliano P., Marletta V., Monteriù A. (eds) Ambient Assisted Living. Biosystems & Biorobotics, vol 11. Springer, Cham (2015)

[75] Bonaccorsi, M., Fiorini, L., Cavallo, F., Saffiotti, A., Dario, P.: A Cloud Robotics Solution to Improve Social Assistive Robots for Active and Healthy Aging. Int J of Soc Robotics 8(3) 393-408 (2016) `https://doi.org/10.1007/s12369-016-0351-1`.

[76] Mouradian, C., Yangui, S., Glitho, R.H.: Robots as-a-Service in Cloud Computing: Search and Rescue in Large-scale Disasters Case Study. 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE (2018)

[77] Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J.P.: All one needs to know about fog computing and related edge computing paradigms: A complete survey. Journal of Systems Architecture 98 289-330 (2019)

[78] Bhavsar, P.C., Patel, S.H., Sobh, T.M.: Hybrid Robot-as-a-Service (RaaS) Platform (Using MQTT and CoAP). In 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE. pp. 974-979 (2019)

[79] Gupta, S., Durak, U.: RESTful Software Architecture for ROS-based Onboard Mission System for Drones. AIAA SciTech 2020 Forum. 2020.

[80] Sorrentino, A., Cavallo, F., Fiorini, L.: A Plug and Play Transparent Communication Layer for Cloud Robotics Architectures. Robotics 9(1) 17 (2020)

[81] Kato, Y., Izui, T., Tsuchiya, Y., Narita, M., Ueki, M., Murakawa, Y., Okabayashi, K.: RSi-Cloud for Integrating Robot Services with Internet Services. IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society. IEEE, pp. 2158–2163 (2011)

[82] Kato, Y., Izui, T., Murakawa, Y., Okabayashi, K., Ueki, M., Tsuchiya, Y., Narita, M.: Research and development environments for robot services and its implementation. 2011 IEEE/SICE International Symposium on System Integration (SII). IEEE, pp. 306-311 (2011)

[83] Sato, M., Kamei, K., Nishio, S., Hagita, N.: The ubiquitous network robot platform: common platform for continuous daily robotic services. 2011 IEEE/SICE International Symposium on System Integration (SII). IEEE, pp. 318-323 (2011)

[84] Dyumin, A.A., Puzikov, L.A., Rovnyagin, M.M., Urvanov, G.A., Chugunkov, L.V.: Cloud computing architectures for mobile robotics. 2015 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW). IEEE, pp. 65-70 (2015)

[85] Muratore, L., Lennox, B., Tsagarakis, N.G.: XBot-Cloud: a Scalable Cloud Computing Infrastructure for XBot Powered Robots. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 7781-7788 (2018)

[86] Penmetcha, M., Kannan, S.S., Min, B.C.: Smart Cloud: Scalable Cloud Robotic Architecture for Web-powered Multi-Robot Applications. arXiv preprint arXiv:1912.02927. (2019)

[87] Tanwani, A.K., Anand, R., Gonzalez, J.E., Goldberg, K.: RILaaS: Robot Inference and Learning as a Service. IEEE Robotics and Automation Letters 5(3) 4423-4430 (2020)

[88] Quintas, J., Menezes, P., Dias, J.: Cloud robotics: Towards context aware robotic networks. International Conference on Robotics. pp. 420-427 (2011)

[89] Doriya, R., Chakraborty, P., Nandi, G.C.: Robotic Services in Cloud Computing Paradigm. 2012 International Symposium on Cloud and Services Computing. IEEE, pp. 80-83 (2012)

[90] Terrissa, L.S., Bouziane, R., Brethé, J.F.: Towards a new approach of Robot as a Service (RaaS) in Cloud Computing paradigm. 5th. International Symposium ISKO-Maghreb (Knowledge Organization in the perspective of Digital Humanities: Researches and Applications), Hammamet Tunisia (2015)

[91] Bouziane, R., Terrissa, L.S., Ayad, S., Brethéé, J.F., Kazar, O.: A Web services based solution for the NAO Robot in Cloud Robotics environment. 2017 4th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE, pp. 0809-0814 (2017)

[92] 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. In: Osório F., Wolf D., Castelo Branco K., Grassi Jr. V., Becker M., Romero R. (eds) Robotics. SBR 2014 2014, ROBOCONTROL 2014, LARS 2014. Communications in Computer and Information Science, vol 507. Springer, Berlin, Heidelberg (2015)

[93] Huang, J.Y., Lee, W.P., Lin, T.A.: Developing Context-Aware Dialoguing Services for a Cloud-Based Robotic System. IEEE Access 7 44293-44306 (2019)

[94] Huang,J.Y., Lee,W.P., Chen,C.C., Dong.B.W.: Developing Emotion-Aware Human–Robot Dialogues for Domain-Specific and Goal-Oriented Tasks. Robotics 9(2) 31 (2020)

[95] Skarzynski, K., Stepniak, M., Bartyna, W., Ambroszkiewicz, S.: SO-MRS: A Multi-robot System Architecture Based on the SOA Paradigm and Ontology. In: Giuliani M., Assaf T., Giannaccini M. (eds) Towards Autonomous Robotic Systems. TAROS 2018. Lecture Notes in Computer Science, vol 10965. Springer, Cham. pp. 330-342 (2018)

[96] Mori, Y., Ogawa, Y., Hikawa, A., Yamaguchi, T.: Multi-robot Coordination Based on Ontologies and Semantic Web Service. In: Kim Y.S., Kang B.H., Richards D. (eds) Knowledge Management and Acquisition for Smart Systems and Services. PKAW 2014. Lecture Notes in Computer Science, vol 8863. Springer, Cham. pp. 150-164 (2014)

[97] Mokarizadeh, S., Grosso, A., Matskin, M.n Kungas, P., Haseeb, A.: Applying Semantic Web Service Composition for Action Planning in Multi-Robot Systems. 2009 Fourth International Conference on Internet and Web Applications and Services. IEEE, pp. 370-376 (2009)

[98] Zhou, G., Zhang, Y., Bastani, F., Yen, I.L.: Service-oriented robotic swarm systems: Model and structuring algorithms. 2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE, pp. 95-102 (2012)

[99] Mohamed, N., Al-Jaroodi, J.: Service-oriented middleware for collaborative UAVs. 2013 IEEE 14th International Conference on Information Reuse & Integration (IRI). IEEE, pp. 185-192 (2013)

[100] Mahmoud, S., Mohamed, N.: Collaborative uavs cloud. 2014 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, pp. 365-373 (2014)

[101] Mahmoud, S., Mohamed, N.: Broker architecture for collaborative uavs cloud computing. 2015 International Conference on Collaboration Technologies and Systems (CTS). IEEE, pp. 212-219 (2015)

[102] Cai, Y., Tang, Z., Ding, Y., Qian, B.: Theory and application of multi-robot service-oriented architecture. IEEE/CAA Journal of Automatica Sinica 3(1) 15-25 (2016)

[103] van Gastel, P.J.G.: A planning module for a ROS-Based ubiquitous robot control system. MS thesis (2014)

[104] Janssen, R., van de Molengraft, R., Bruyninckx, H., Steinbuch, M.: Cloud based centralized task control for human domain multi-robot operations. Intelligent Service Robotics, 9(1), 63-77 (2016)

[105] Bouten, N., Hristoskova, A., Ongenae, F., Nelis, J., De Turck, F.: Ontology-Driven Dynamic Discovery and Distributed Coordination of a Robot Swarm. In: Sadre R., Novotný J., Čeleda P., Waldburger M., Stiller B. (eds) Dependable Networks and Services. AIMS 2012. Lecture Notes in Computer Science, vol 7279. Springer, Berlin, Heidelberg (2012)

[106] Chitic, S.G., Ponge, J., Simonin, O.: SDfR-Service discovery for multi-robot systems (2016)

[107] Hayet, T., Knani, J.: SOAP-Based Web Service for Localization of Multi-robot System in Cloud. In: Arai K., Kapoor S., Bhatia R. (eds) Intelligent Computing. SAI 2018. Advances in Intelligent Systems and Computing, vol 857. Springer, Cham (2019)

[108] Zhou, H., Zhang, J., Liu, Z., et al.: Research on Circular Area Search algorithm of multi-robot service based on SOA cloud platform. Applied Soft Computing Journal (2019) 105816. `https://doi.org/10.1016/j.asoc.2019.105816`.

[109] Afrin, M., Jin, J., Rahman, A., Tian, Y.C., Kulkarni, A.: Multi-objective resource allocation for Edge Cloud based robotic workflow in smart factory. Future Generation Computer Systems 97 119-130. `https://doi.org/10.1016/j.future.2019.02.062` (2019)

[110] Queralta, J.P., Qingqing, L., Gia, T.N., Truong, H.L., Westerlund, T.: End-to-End Design for Self-Reconfigurable Heterogeneous Robotic Swarms. arXiv preprint arXiv:2004.13997. 2020.

[111] Chibani, A., Amirat, Y., Mohammed, S., Matson, E., Hagita, N., Barreto, M.: Ubiquitous robotics: Recent challenges and future trends. Robotics and Autonomous Systems. 61(11) (2013) 1162-1172. `https://doi.org/10.1016/j.robot.2013.04.003`.

[112] Yang, T.H., Lee, W.P.: A Service-Oriented Framework for the Development of Home Robots. International Journal of Advanced Robotic Systems 10(2) 122 (2013)

[113] Sugawara, Y., Morita, T., Saito, S., Yamaguchi, T.: An Intelligent Application Development Platform for Service Robots. MuSRobS@ IROS. pp. 16-20 (2015)

[114] Yang, T.H., Lee, W.P.: Intelligent service reconfiguration for home robots. In: Ding X., Kong X., Dai J. (eds) Advances in Reconfigurable Mechanisms and Robots II. Mechanisms and Machine Science, vol 36. Springer, Cham. pp. 735-745 (2016)

[115] Ha, Y.G., Sohn, J.C., Cho, Y.J., Yoon, H.: A robotic service framework supporting automated integration of ubiquitous sensors and devices. Information Sciences 177(3) 657-679 (2007)

[116] Yachir, A., Tari, K., Amirat, Y., Chibani, A., Badache, N.: QoS based framework for ubiquitous robotic services composition. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE pp. 2019-2026 (2009)

[117] Tari, K., Amirat, Y., Chibani, A., Yachir, A., Mellouk, A.: Context-aware dynamic service composition in ubiquitous environment. 2010 IEEE International Conference on Communications. IEEE (2010)

[118] Qian, K., Ma, X., Dai, X., Fang, F.: Flexible ambient service discovery and composition for component-based robotic system. Journal of Ambient Intelligence and Smart Environments 4(6) 547-562 (2012)

[119] Qian, K., Ma, X., Dai, X., Fang, F.: Knowledge-enabled decision making for robotic system utilizing ambient service components. Journal of Ambient Intelligence and Smart Environments 6(1) 5-19 (2014)

[120] Qian, K., Ma, X., Dai, X., Fang, F., Zhou, B.: A utilization framework of ubiquitous resources for service robots using semantic matchmaking. International Journal of Advanced Robotic Systems 12(4) 41 (2015)

[121] Luo, J., Zhang, L., Zhang, H.Y.: Design of a cloud robotics middleware based on web service technology. 2017 18th International Conference on Advanced Robotics (ICAR), IEEE, pp. 487-492 (2017)

[122] Huang, J.Y., Lee, W.P., Yang, T.H., Ko, C.S.: Resource sharing for cloud robots: Service reuse and collective map building. In : Advanced Robotics (ICAR), 18th International Conference on. IEEE, pp. 303-309 (2017)

[123] Xia, C., Zhang, Y., Wang, L., Coleman, S., Liu, Y.: Microservice-based cloud robotics system for intelligent space. Robotics and Autonomous Systems 110 139-150 (2018) `https://doi.org/10.1016/j.robot.2018.10.001`.

[124] Puttonen, J., Lobov, A., Soto, M.A.C., Lastra, J.L.M.: Cloud computing as a facilitator for web service composition in factory automation. J Intell Manuf 30 687–700. `https://doi.org/10.1007/s10845-016-1277-z` (2019)

[125] Xie, Y., Guo, Y., Mi, Z., Yang, Y., Obaidat, M.S.: Loosely Coupled Cloud Robotic Framework for QoS-Driven Resource Allocation-Based Web Service Composition. IEEE Systems Journal (2019)

[126] Erl, T.: SOA Principles of Service Design (paperback). Prentice Hall Press (2016)

[127] Aissam, M., Benbrahim, M., Kabbaj, M.N.: Cloud robotic: Opening a new road to the industry 4.0. In: N.Derbel et al. (Eds.) New Developments and Advances in Robot Control. Studies in Systems, Decision and Control, vol 175. Springer, Singapore. pp. 1-20 (2019)

[128] Bouziane, R., Terrissa, L.S., Ayad, S., Brethéé, J.F.: Towards an architecture for cloud robotic services. International Journal of Computers and Applications p. 1-12 (2021)

[129] García-Valls, M., Cucinotta, T., Lu, C.: Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture, 60(9), 726-740 (2014)

[130] Portnoy, M.: Virtualization essentials. second edition. Vol. 19. John Wiley & Sons, (2016)

[131] De, D.: Mobile cloud computing: architectures, algorithms and applications. CRC Press Taylor & Francis Group (2016)

[132] Masdari, M., Nabavi, S. S., Ahmadi, V.: An overview of virtual machine placement schemes in cloud computing. Journal of Network and Computer Applications, 66, 106-127 (2016)

[133] Bugnion, E., Nieh, J., Tsafrir, D.: Hardware and Software Support for Virtualization. Synthesis Lectures on Computer Architecture 12.1: 1-206 (2017)

[134] Synchromedia, `http://www.synchromedia.ca/`.

[135] SoftBank Robotics, `https://www.ald.softbankrobotics.com/en`.

[136] Ellouze, F., Koubâa, A., Youssef, H.: ROSWeb Services: A Tutorial. In: Koubaa A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence, vol 625, pp 463-490. Springer, Cham (2016)

[137] Protégé. url: `https://protege.stanford.edu/`.

[138] Tiddi, I., Bastianelli, E., Bardaro, G., d'Aquin, M., Motta, E.: An ontology-based approach to improve the accessibility of ROS-based robotic systems. Proceedings of the Knowledge Capture Conference (2017)

[139] Zander, S., et al.: A model-driven engineering approach for ros using ontological semantics. arXiv preprint arXiv:1601.03998 (2016)

[140] Awad, R., et al.: ROS engineering workbench based on semantically enriched app models for improved reusability. 2016 IEEE 21st international conference on emerging technologies and factory automation (ETFA). IEEE (2016)

[141] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., Specia, L.: SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Cross-lingual Focused Evaluation. arXiv preprint arXiv:1708.00055 (2017)

[142] Devlin, J., Chang, M. W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[143] SBERT.net: Sentence-Transformers: `https://www.sbert.net/`.

[144] Plappert, M., Mandery, C., Asfour, T.: The KIT motion-language dataset. Big data, 4(4), 236-252 (2016)

[145] Plappert, M., Mandery, C., Asfour, T.:Learning a bidirectional mapping between human whole-body motion and natural language using deep recurrent neural networks. Robotics and Autonomous Systems, 109, 13-26 (2018)

[146] Aier, S., Offermann, P., Schönherr, M., Schröpfer, C.: Implementing non-functional service descriptions in soas. In: International Conference on Trends in Enterprise Application Architecture. Springer. Berlin, Heidelberg (2006)

[147] Baklouti, N., Gargouri, B., Jmaiel, M.: Semantic-based approach to improve the description and the discovery of Linguistic Web Services. In: Engineering Applications of Artificial Intelligence 46, pp. 154-165 (2015)

[148] Riazuelo, L., et al.: RoboEarth semantic mapping: A cloud enabled knowledge-based approach. In: IEEE Transactions on Automation Science and Engineering 12.2, pp. 432-443 (2015)

[149] Kunze, L., Roehm, T., Beetz, M.: Towards semantic robot description languages. 2011 IEEE International Conference on Robotics and Automation. IEEE (2011)

[150] Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. International semantic web conference. Springer, Berlin, Heidelberg (2002)

[151] Srinivasan, N., Paolucci M., Sycara K.: An efficient algorithm for OWL-S based semantic search in UDDI. International Workshop on Semantic Web Services and Web Process Composition. Springer, Berlin, Heidelberg (2004)

[152] rosjava_core documentation, `http://rosjava.github.io/rosjava_core/0.1.6/index.html`.

[153] Paulraj, D., Swamynathan, S., Madhaiyan, M.: Process model-based atomic service discovery and composition of composite semantic web services using web ontology language for services (OWL-S). Enterprise Information Systems 6.4 (2012): 445-471.

[154] Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)

[155] ROS 2 Documentation, url: `https://docs.ros.org/en/foxy/index.html`.

[156] Hu, B., Wang, H., Zhang, P., Ding, B., Che, H.: Cloudroid: A Cloud Framework for Transparent and QoSaware Robotic Computation Outsourcing. 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), IEEE. pp. 114-121 (2017)

[157] Gudi, S.L.K.C., Ojha, S., Johnston, B., Clark, J., Williams, M.A.: Fog Robotics: An Introduction. In IEEE/RSJ International Conference on Intelligent Robots and Systems (2017)

[158] Tanwani, A.K., Mor, N., Kubiatowicz, J., Gonzalez, J.E., Goldberg, K.: A Fog Robotics Approach to Deep Robot Learning: Application to Object Recognition and Grasp Planning in Surface Decluttering. In 2019 International Conference on Robotics and Automation (ICRA). IEEE, pp. 4559-4566 (2019)