CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Journal Paper

MyBot: Cloud-Based Service Robot using Service-Oriented Architecture

Anis Koubâa*
Mohamed-Foued Sriti
Yasir Javed
Maram Alajlan
Basit Qureshi
Fatma Ellouze
Abdelrahman Mahmoud

*CISTER Research Centre

CISTER-TR-170306

2017

MyBot: Cloud-Based Service Robot using Service-Oriented Architecture

Anis Koubâa*, Mohamed-Foued Sriti, Yasir Javed, Maram Alajlan, Basit Qureshi, Fatma Ellouze, Abdelrahman Mahmoud

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: aska@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

This paper presents a viable solution for the development of service robots by leveraging cloud and Web services technologies, modular software architecture design, and Robot Operating System (ROS). The contributions of this paper are two- folded (1) Design of ROS Web services to provide new abstract interfaces to service robots that makes easier the interaction with and the development of service robots applications, and (2) Integration of the service robot to the cloud using the ROSLink protocol. We demonstrate through real-world implementation on the MyBot robot the effectiveness of these software abstraction layers in developing applications for service robots through the Internet and the cloud, and in accessing them through Internet. We believe that this work represents an important step towards a more popular use of service robots.

MyBot: Cloud-Based Service Robot using Service-Oriented Architecture

Anis Koubâa [*¶‡], **Mohamed-Foued Sriti** [¶], **Yasir Javed** [¶], **Maram Alajlan** [*¶†], **Basit Qureshi** [¶], **Fatma Ellouze** [¶], **Abdelrahman Mahmoud** [§†]
[*]Prince Sultan University, College of Computer and Information Sciences, Saudi Arabia. [¶]Gaitech International Ltd., China.
[‡]Al-Imam Mohammad Ibn Saud Islamic University, Saudi Arabia. [†]King Saud University, Riyadh, Saudi Arabia. [§] German University of Cairo, Egypt.
[+] CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Porto, Portugal. [+] National Engineering Institute of Sfax (ENIS), Tunisia.
akoubaa@coins-lab.org, mfsriti@ccis.imamu.edu.sa, yasirjaved@coins-lab.org, maram.ajlan@coins-lab.org, qureshi@psu.edu.sa, fatma.ellouze@coins-lab.org, abdelrahman.mahmoud@coins-lab.org

Abstract – This paper[1] presents a viable solution for the development of service robots by leveraging cloud and Web services technologies, modular software architecture design, and Robot Operating System (ROS). The contributions of this paper are two-folded (1) Design of ROS Web services to provide new abstract interfaces to service robots that makes easier the interaction with and the development of service robots applications, and (2) Integration of the service robot to the cloud using the ROSLink protocol. We demonstrate through real-world implementation on the MyBot robot the effectiveness of these software abstraction layers in developing applications for service robots through the Internet and the cloud, and in accessing them through Internet. We believe that this work represents an important step towards a more popular use of service robots.

I. INTRODUCTION

The tremendous growth in utilization of robots has brought numerous benefits for humans with application to manufacturing, healthcare, mining, deep excavation, space exploration, etc. Use of robots has been a significant factor in improvement of human safety, reduction in maintenance / production costs and improved productivity [1].

It is widely forecasted that service robots would inundate the market reaching record sales in the next 20 years. In its statistical report, the International Federation of Robotics reported sale of 3 million service robots for personal and domestic within 2012. This number represents 20% increase in sales from the previous year accounting to US$ 1.2 billion [2]. Nowadays, one of the major challenges in the development of service robots is the lack of software engineering frame-works to build complex service robots' applications that are modular, reusable, and easily extensible. Most of the available software for service robots are tightly coupled with the robotic platform and lack sufficient abstractions to remain generic for different platforms. Robot Operating System (ROS) is one of the widely used middleware to develop robotic applications, which represents an important milestone in the development of modular software for robots. In fact, it presents different abstractions to hardware, network and operating system such as navigation, motion planning, low-level device control, and message passing. However, the levels of abstractions are still not enough for developing complex and generic applications for mobile robots, in particular if those applications are distributed among several machines, requiring machine-to-machine communication. This paper addresses this gap, and proposes the design of a service-oriented software architecture that contains software abstractions. In particular, we designed and developed ROS Web Services, which are new interfaces that expose ROS ecosystem as Web services. Furthermore, we designed the ROSLink protocol that allow the service robot to be controlled and monitored through a cloud robotics system, namely Dronemap Planner [3], [4].

The contributions of this paper are as follow.

- Design of a low-cost service robot Based on the Turtlebot platform and Commercial off the Shelf (COTS) hardware.
- Design of software meta-models for the integration of Web services into ROS. To the best of our knowledge, the work presented is ground breaking as far as such integration is concerned.
- Integration of ROS-based robots into the cloud using the ROSLink protocol.
- Experimentation and deployment of the service robot for the validation of our architecture and discussion of experimental challenges.

The rest of this paper is organized as follows. Section II discusses the state-of-the-art with an emphasis on the contribution of this paper compared to similar works. Section III presents the mechanical design of the service robot. In Section IV, we present ROS Web services and the ROSLink protocol for cloud integration of the robot. In Section V, we present application deployments for the service robot. Section VI concludes the paper and outlines future works.

II. RELATED WORKS

Developing software architecture and frameworks for assistance and service robots has attracted a lot of attention in the literature. Authors in [5] developed an intelligent vehicle control architecture to allow multiple collaborating robots to accomplish missions. The proposed systems architecture is based on service oriented computing and agent software technology. The authors evaluate the proposed work using a very limited study involving multiple drones. In [6], authors presented Hyper-Flex tool-chain focusing on ROS metamodels and ROS-specific tools for supporting the process of exploiting reference architectures and demonstrated how reference architecture can be used for building complex software systems. The limitation of these

1 This paper is an extended version of the conference paper presented in IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2016)

works is the lack of concrete implementations demonstrating instantiation of these processes. In [7], the authors present an Event-Driven Architecture, which enhances the capabilities of robots to interpret events and react on those according to predefined functionality. The authors applied a service oriented architecture to a choreography engine to compose services without the need of an orchestration engine. The proposed system was applied to an automation system, although no implementation details are provided.

In [8], the authors proposed an architecture for a Domestic Robot targeting elderly users in assisting them to remain autonomous in their homes. The proposed architecture is based on the integration of three middleware frameworks PEIS, MIRA, and ROS. Most of the computation is performed by a large number of ROS nodes; the resulting robot services are exported to the PEIS middleware for seamless integration of the robot into the ambient assisted living system. rosjs and rosbridge [9] have been proposed to facilitate integration of Web services in ROS. Both these framework essentially cater to (1) allowing common web browsers to exploit users to interact with ROS enabled robots; (2) to provide Web developers lacking expertise in robotics with simple interfaces to develop client applications allowing control and manipulation of ROS-enabled robots. In [10], the authors proposed ROStful by extending rosbridge to support REST Web services and developed a lightweight Web server that exposes ROS topics, services and actions through RESTful Web services. However, the authors did not provide an architecture or meta-model for the integration of REST into ROS.

In this paper, we propose a Web service layer for ROS, in addition to the ROSLink protocol to integrate robots into the cloud and the Internet.

III. ROBOT DESIGN

A. Design requirements

The design requirements of MyBot service that were considered are:

- Cost-effectiveness: the robotic platform must be cost-effective to be affordable for the public use.
- ROS-enabled design: We focused our design on ROS-enabled robot, as ROS is attracting increasing interest in the robotics software developers communities. The reason is that ROS provides several layers of abstractions that make easier the development of robotics software through the use of open-source libraries such as navigation services (e.g. gmapping package), image processing (e.g. Open-CV and PointCloud), drivers for several robots and sensor platforms, etc.
- Commercial-off-the-shelf (COTS) hardware: To extend the robot capabilities with additional sensors and hardware, we consider the use of COTS hardware that is commonly available in the public market. This allows end-users to easily extend the robots with their custom requirements.

The Turtlebot 2 robot represents an appropriate base platform to meet the design requirement of the MyBot service robot. However, the software architecture that we propose in this paper can be applied to any type of ROS-enabled robot thanks to the abstraction layers that we designed for robot control and that will be presented in Section IV.

IV. SOFTWARE ARCHITECTURE

We designed a software architecture that provides two abstraction layers on top of ROS to make easier the development of distributed applications for service robots. It includes two major layers, namely: (1) COROS [11], which is a component-based software architecture that provides a first abstraction layer on top of ROS composed of modular components to develop cooperative and distributed applications, (2) ROS Web services is the second abstraction layer that allow client applications to seamlessly and transparently interact with the robot while hiding all implementation details. In what follows, we present the main features.

A. COROS

1) *Component-based Layer Architecture:* We reused and extended our COROS architecture defined in [11], by developing new modules for the service robot application logic, and also a new message serializer to effectively handle communication between heterogeneous platforms. In what follows, we describe the architecture and enhancements. COROS consists of five layers illustrated in Figure 1 that shows the component diagram of the software architecture. The software system is decomposed into five subsystems (or layers), each of which plays the role of a container of a set of components. These subsystems are:

- Communication: this subsystem was designed to ensure the interaction between the robot and other machines, which can be robots or user devices. It comprises extensible and modular client and server components that enable agents to exchange serialized messages through the network interface using sockets.

Figure 1. COROS Software Architecture.

- **ROS Interaction Layer:** this subsystem adds a lightweight layer on top of ROS allowing a seamless inter-process interaction between ROS nodes (processes) defined in the architecture.
- **Robot Control:** this subsystem adds another layer on top of ROS providing a bridge between the local software agents and the physical robots. The role of this layer is to manage the robot configuration and its state. The Robot Controller component provides an abstract model for any ROS-enabled robot.

- **Application Logic:** this subsystem addresses the problem solving requirements; it encapsulates all of the components needed to implement a complete service robot application. Any new application should reuse and configure the software components to define its proper behavior.
- **Knowledge Base Manipulation Layer:** This subsystem aims at satisfying knowledge base requirements and maintains up-to-date information about the robot status and its environment.

In the context of MyBot project, we have implemented four applications using COROS, including (1), Discovery application, (2) Courier Delivery application, (3) Coffee delivery application, and (4) people guidance application.

B. ROS Web Services

1) *Objectives:* The objective of designing ROS Web services is to expose ROS as a Service to the client applications, providing an additional abstraction layer of ROS resources including topics, services and actions for developers with no prior knowledge on robots or on ROS. There are three main benefits coming from exposing ROS as a service, namely:
   - *Fostering public usage of robots:* By exposing the complex ROS ecosystem through Web services interfaces to client applications, Web and mobile developers with no background on robotics can easily interact with the robots through the Internet through Web service invocation. This enables a wider usage of robots at public scale.
   - *Integration with the cloud*: Web services and Service Oriented Architecture (SOA) are major components of today's cloud as they allow virtualization of resources. Therefore, embedding Web services into ROS allows for the integration of ROS-enabled robots with the cloud so that users can virtually access the robots' resources through the cloud to either control or monitor the robots status.
   - *Standard interfaces*: Web services allows for providing standard interfaces to robotics resources so that it will be possible for client application to interact with heterogeneous robots if they are having the same Web services abstractions, independently from implementation details.

To address these objectives, we propose to use Web services as an additional abstraction layer on top of ROS. We develop a SOAP Web Service implementation (ros-ws) and a REST Web Service implementation (ros-rs), which represent the two fundamental architectural models for SOA. ROS Web Services allow any client application on any platform to interact with ROS simply by invoking the ROS Web Services in exactly the same way as invoking traditional Web Services.

2) *System Architecture:* Figure 2 depicts the deployment diagram of ROS Web services and illustrates the integration of the Web services' layers into the ROS-enabled service robot and the client device.

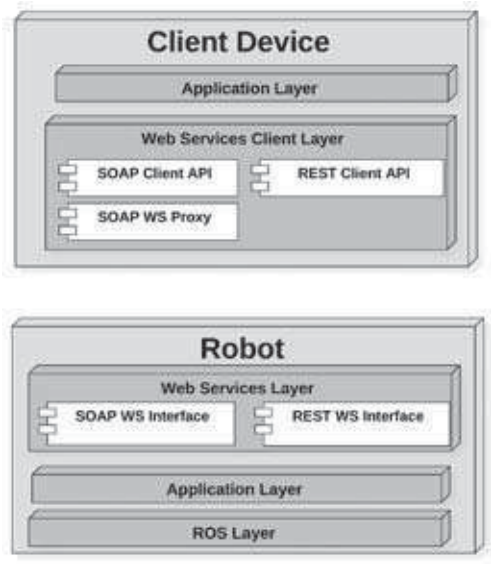

Figure 2. Deployment Diagram of ROS Web Services.

The Web services can be seen as a middleware that allows seamless interaction between client applications and ROS ecosystem in the service robot. Our architecture encompasses both SOAP and REST Web services to provide flexible alternative to client applications to interact with ROS ecosystem. In particular, the Web service layer allows a user to subscribe to or publish any ROS topic, action or service, and thus delivering ROS messages to client subscribed to a particular topic.

To integrate Web services into ROS, we faced the challenge of choosing the most appropriate technology to build the software system and design its architecture. We have opted for the use of Java as a Web service programming language, as it provides a native and advanced support of SOAP and REST Web services, although they are programming-language-independent and platform-independent. However, Java EE provides standard APIs for SOAP and REST Web Services, known as JAX-WS and JAX-RS specifications, respectively. Python also provides REST Web service support, but much less than Java for SOAP Web services.

V. INTEGRATION TO THE CLOUD

The main problem with the deployment of a service robot is to make accessible, controllable and monitored through the Internet. Some solutions like [9] proposed the ROSBridge with a Websockets server on the robot side. This approach enabled the effective integration of ROS with the Internet; however, the fact that the Websockets server is running on the robot machine requires the robot to have a public IP address to be accessible by Websockets clients, which is not possible for every robot, or being on the same local area networks. Network address translation (NAT) could also be used when the robot is behind a NAT domain, but still this option may be cumbersome in deployment. To address this issue, we proposed the ROSLink protocol [3] that overcomes the aforementioned limitations by (*i.*) implementing the client in the robot side, (*ii.*) manifestation of a proxy server located at a public IP server machine deployed in a cloud. The main objective of ROSLink is to control and monitor a ROS-enabled robot through the Internet. The general architecture of ROSLink is presented in Figure 3.

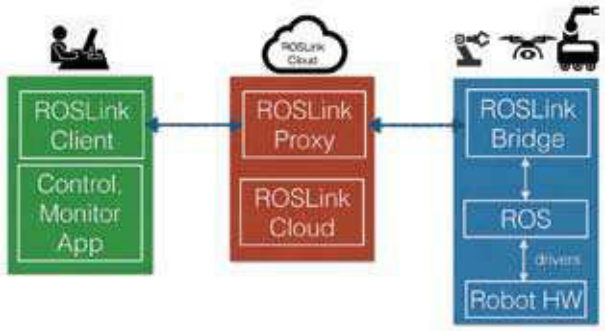

Figure 3. ROSLink Architecture [3].

The system is composed of three main parts:

- The ROSLink Bridge: This bridge provides the main interface between ROS and the ROSLink protocol. It subscribes to ROS topics/services to read data from, serializes them in JSON and send it to the cloud, server or user application. In addition, it receives commands in JSON format, and executes the corresponding action through ROS.
- The ROSLink Proxy and Cloud: it is a proxy server that connects user application with ROSLink Bridge in the robot. It acts as a mediator between the two ends, and forwards message between the user and the robot. Furthermore, it provide a complete management system for both robots and users, and their mapping in real-time.
- The ROSLink Client Application: This application is used to control and monitor the robot remotely through Internet. It provide status of the robot in real-time and allow to send commands to it, both using ROSLink Messages.

For more details about ROSLink communication and ROSLink messages, the reader if referred to [3].

The ROSLink communication protocol is based on the exchange of ROSLink messages. ROSLink messages are JSON formatted strings that contain information about the command and its parameters. To standardize the type of messages exchanged, we specified a set of ROSLink messages that are supported by the ROSLink Proxy. These message can be easily extended based on the requirements of the user and the application.

There are two main categories of ROSLink messages: (*i.*) *State messages*: these are message sent by the robot and carry out information about the internal state of the robot, including its position, orientation, battery level, etc. (*ii.*) *Command messages*: these are messages sent by the client application to the robot and carry out commands to make the robot execute some actions, like for example moving, executing a mission, going to a goal location, etc.

In what follows, we identify an example of messages and command types:

- Presence message: the robot should declare its presence regularly to declare itself and to be considered as active. Typically, Heartbeat messages sent at a certain frequency (typically one message per second) are used for this purpose.
- Motion messages: In robot mission, it is important to know the location and odometry motion parameters (i.e.linear and angular velocities) of the robot at a certain time. Thus, a motion message containing position information of the robot should be periodically broadcast.
- Sensor messages: The robot needs to broadcast its internal sensor data such as IMU, laser scanners, camera images, GPS coordinates, actuators states, etc. ROSLink also defines several sensor messages to exchange these data between the robot and the user.
- Motion commands: For the robot to navigate in ROS, certain commands are sent to it like Twist messages in ROS, and goal/waypoint locations. ROSLink also specifies different types of commands to make the robot moves as desired.

VI. EXPERIMENTATION AND DEPLOYMENT

To demonstrate the effectiveness of the proposed COROS architecture and ROS Web services, we used them to develop the applications and services of the MyBot service robot presented in Section 3. In addition, we deployed the MyBot service robot at Prince Sultan University to deliver courier between offices and also to bring coffee from the central cafe of the University in addition to other services and applications. In the remainder of this section, we present the experimental applications developed using the proposed architecture.

Figure 4. Weather Android Interface.

A. Climate Condition Application

The service robot provides the user with information about indoor climate conditions namely, the temperature, the light and humidity. Figure 4 shows the Android interface for the climate conditions. It presents the temperature, light and humidity of the operating environment of the robot extracted from the TelosB sensor node. The COROS architecture was used in this application. The integration process of the TelosB sensor into the robot has the following steps.

We developed the low-level driver for TelosB sensor node that we integrated it into ROS by developing a ROS package for the TelosB node that uses this driver to get the sensor values from the serial port, and then publishes the three sensor data as a new custom ROS topic /telosb_topic using a custom TelosBMsg message that we created for this purpose. It contains three fields for the three sensor data. In the Android interface, we have used the android_core and rosjava API available to create subscribers to the /telosb_topic, by creating an

Android activity that extends the ROSActivity class, and which allows publishing and subscribing to topics running on the robot.

The deployment of this application used the COROS architecture to deliver the sensor data to the mobile application within the same local area network, so it is displayed in the user mobile device, as illustrated in Figure 4. On the other hand, the deployment through the Internet is performed using the ROSLink protocol. The use of ROLink allows to easily send this sensor information to end-users through the cloud. It defines a custom ROSLink message that contains all the three sensor values. This message is firstly received by the cloud server, which identifies the user who is mapped to this service robot to forward this information to it. The mapping between a user and a service robot is done through the cloud, when the user registers to use a certain robot. It is clear that the ROSLink protocol contributes to the concept of Internet-of-Things through its platform-independent protocol specification that allows the transfer of any kind of data through the Internet, basically data collected from ROS ecosystem.

B. Delivery Application

The delivery application use case, illustrated in Figure 5, was developed so that the robot performs courier delivery mission between offices or also deliver coffee from the central Cafe to offices. The use case is illustrated in the following video demonstration [12].

The use case was implemented using two approaches: (1) using the COROS framework by instantiating the AgentOperator for the delivery mission and using JSON to ROS message serialization for platform-independent communication between the robot and the Android device, (2) using the ROS Web Services interfaces (ros-ws and ros-rs). A comprehensive demonstration on developing and using ROS Web services interfaces is presented in [13].

a) *Using ROS Web Services:* We also implemented the delivery mission using ROS Web services. The Web service interface is presented in Listing 2.

```
package org.ros-ws;
@WebService (serviceName="TurtlebotOfficeService",
    name="TurtlebotOffice",
            targetNamespace="http://ros-ws.org")
public interface TurtlebotPublishersInterface {
    @WebMethod (action="execute_delivery",
        operationName="executeDelivery")
    public void ExecuteDeliveryRequest(
        DeliveryRequestMessage msg);
}
```

Listing 1. TurtlebotPublishersInterface Interface.

The ROS Web service interface shows one method called ExecuteDeliveryRequest(DeliveryRequestMessage msg) that take a request message as parameter and send the request to the robot via Web services. Once the robot receives the coordinates of the user, it will execute the action accordingly, by invoking the back-end ROS delivery application that remains listening on the topic DeliveryRequestMsg/from_json.

b) *Using ROSLink and Cloud Services:* We developed a ROSLink Bridge for the MyBot robot. The ROSLink Bridge of the robot subscribe to ROS topics of the MyBot robot and sends Heartbeat messages to the cloud at regular time interval to maintain its active status in the cloud. It also sends its location in the map and motion parameters through the Global Motion ROSLink message. These messages are forwarded to the end user from the cloud server to keep track and monitor the status of the service robot. On the other hand, the user can execute the delivery service through the cloud by sending a custom ROSLink Command message ExecuteDelivery that takes as parameter the location of the user requesting the delivery service. Once this message is received by the cloud, it is forwarded to the ROSLink Bridge of the robot, which in turns forward it to the same back-end ROS delivery application that remains listening on the topic De-

Figure 5. Courier Delivery Scenario.

liveryRequestMsg/from_json, which receives ROS messages sent by the publisher of the dispatcher component.

The use the ROSLink protocol definitely solves the problem of accessing robots through the Internet. In addition, the software architecture that we proposed provide loosely coupled modules through front-end interfaces including Web service interfaces and ROSLink interfaces that allow to access the same back-end application in a seamless manner.

VII. CONCLUSIONS ACKNOWLEDGMENTS

This work is supported by the myBot project entitled "MyBot: A Personal Assistant Robot Case Study for Elderly People Care" under the grant from King AbdulAziz City for Science and Technology (KACST). In addition, the authors would like to thank the Robotics and Internet of Things (RIoT) Unit at Center of Excellence of Prince Sultan University for their support to this work. Furthermore, the authors thank Gaitech Robotics in China for their support to this work.

The authors would like to thank the Robotics and Internet of Things (RIoT) Unit at Center of Excellence of Prince Sultan University for their support to this work.

REFERENCES

[1] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *Robotics Automation Magazine, IEEE*, vol. 18, pp. 69– 82, June 2011;

[2] "World Robotics 2013 Service Robots," 2013;

[3] A. Koubaa, M. Aljlan, and B. Qureshi, "ROSLink: Bridging ROS with the Internet-of-Things for Cloud Robotics," *Springer Book on Robot Operating System (ROS), Volume 2*, vol. 2, May 2017;

[4] A. Koubaa, B. Qureshi, M.-F. Sriti, Y. Javed, and E. Tovar, "A Service-Oriented Cloud-Based Management System for the Internet-of-Drones," *17th IEEE Int. Conf. on Autonomous Robot Systems and Competitions* , April 2017;

[5] I. C. C., "Service-oriented agent architecture for unmanned air vehicles.," *33rd IEEE/AIAA Digital Avionics Systems Conference. Colorado Springs, CO*, 2014;

[6] L. Gherardi and D. Brugali, "Modeling and Reusing Robotic Software Architectures: The HyperFlex Toolchain," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 – June 7, 2014*, pp. 6414–6420, 2014;

[7] G. Starke, T. Kunkel, and D. Hahn, "Flexible collaboration and control of heterogeneous mechatronic devices and systems by means of an event-driven, soa-based automation concept," in *2013 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1982–1987, Feb 2013;

[8] N. Hendrich, H. Bistry, and J. Zhang, "PEIS, MIRA, and ROS: Three frameworks, one service robot – A tale of Integration," in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pp. 1749–1756, Dec 2014;

[9] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. C. Jenkins, "Robots as web services: Reproducible experimentation and application development using rosjs," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011;

[10] "Introducing ROStful: ROS over RESTful web services, http://www.ros.org/news/2014/02/introducing-rostful-ros-over-restful-web-services.html," 2015;

[11] A. Koubaa, M.-F. Sriti, H. Bennaceur, A. Ammar, Y. Javed, M. Ala-jlan, N. Al-Elaiwi, M. Tounsi, and E. Shakshuki, "COROS: A Multi-Agent Software Architecture for Cooperative and Autonomous Service Robots," in *Cooperative Robots and Sensor Networks 2015* (A. Koubaa and J. Martinez-de Dios, eds.), vol. 604 of *Studies in Computational Intelligence*, pp. 3–30, Springer International Publishing, 2015;

[12] "MyBot Courier Delivery Demonstrator, https://www.youtube.com/watch?v=oTLtmX2-ucA," 2015;

[13] "MyBot Cafe Delivery using Web Services Interfaces, https://www.youtube.com/watch?v=WvjY5XjAX7U," 2015.