CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds

Anis Koubâa

CISTER-TR-140508
Version:
Date: 2/25/2014

A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds

Anis Koubâa

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: aska@isep.ipp.pt

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

Abstract

Exposing software and hardware computing resources as ser- vices through a cloud is increasingly emerging in the recent years. This comes as a result of extending the service-oriented architecture (SOA) paradigm to virtualize computing resources. In this paper, we extend the paradigm of the SOA approach to virtualize robotic hardware and software resources to expose them as services through the Web. This al- lows non-technical users to access, interact and manipulate robots simply through a Web browser. The proposed RoboWeb system is based on a SOAP-based Web service middleware that binds robots computing re- sources as services and publish them to the end-users. We consider robots that operates with the Robotic Operating System (ROS), as it provides hardware abstraction that makes easier applications development. We de- scribe the implementation of RoboWeb and demonstrate how researchers can use it to interact remotely with the robots. We believe that this work consistently contributes to enabling remote robotic labs using the cloud paradigm.

A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds

Anis Koubaa

Prince Sultan University, Riyadh, Saudi Arabia
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Porto, Portugal
COINS Research Group, Riyadh, Saudi Arabia
akoubaa@coins-lab.org

Abstract. Exposing software and hardware computing resources as services through a cloud is increasingly emerging in the recent years. This comes as a result of extending the service-oriented architecture (SOA) paradigm to virtualize computing resources. In this paper, we extend the paradigm of the SOA approach to virtualize robotic hardware and software resources to expose them as services through the Web. This allows non-technical users to access, interact and manipulate robots simply through a Web browser. The proposed RoboWeb system is based on a SOAP-based Web service middleware that binds robots computing resources as services and publish them to the end-users. We consider robots that operates with the Robotic Operating System (ROS), as it provides hardware abstraction that makes easier applications development. We describe the implementation of RoboWeb and demonstrate how researchers can use it to interact remotely with the robots. We believe that this work consistently contributes to enabling remote robotic labs using the cloud paradigm.

Keywords: Cloud Robotics, Service-Oriented Architecture, SOAP, Web Services, Robot Operating System (ROS), Remote Robotic Labs

1 Introduction

Cloud robotics have been attracting a lot of interest in the last three years [1–5]. In a general sense, this emerging paradigm consists in integrating cloud computing concepts and other Internet Web-centered technologies to leverage converged infrastructures and shared services for robotics [6]. Cloud robotics are very promising to be the most effective way to create and monitor robotic applications, in particular for service robots, in different fields including security and surveillance, remote robotic labs, and home and industrial automation. Indeed, on the one hand, robots will be able to go beyond their limited processing capabilities and take profit from Internet computing resources. On the other hand, robots can be accessed anywhere and anytime through Web interfaces. Several recent works have proposed different designs and implementations for cloud robotics [2, 4, 7, 6, 8]. With reference to these works [1], the robotic cloud

can play two different roles. The first role is to act as a *virtualization middleware*, where service-oriented technologies are used to build virtual environments of robotic ecosystem through Web services, which allow the users to access the robots through Web browsers and Internet utilities. The virtualization of robotic ecosystem through Web services contributed to offering the Robot as a Service (RaaS) model [1, 7]. For instance, in [1], the authors designed and implemented an service-oriented framework of RaaS model for both Windows and Linux operating systems using Web 2.0 technologies and complies with common service and development platforms standards. The second role that the robotic cloud plays is *computations offloading*, which consists in migrating intensive computations and processing tasks from the robot to the cloud computing infrastructure [9, 2]. This is particularly interesting for mobile robots that might have low computation and energy capabilities to perform computationally-intensive tasks, such as 3D localization and mapping, image processing, object recognition, etc. For instance, in [9], the authors proposed a cloud robotics system for recognizing and grasping common household objects by sending 2D images captured by the robot to the cloud, which returns semantic information about the object.

In this paper, we consider the design and implementation of a cloud robotics system of the first category, i.e. virtualization layer. Indeed, the idea of this work is triggered by our need to develop a remote robotic lab to allow different students and researchers outside the University and/or abroad to access and use our robotic platforms located in Saudi Arabia. Our main objective is to make our robots accessible through the Internet for authorized users through Web browsers. We would like to allow students and researchers to access, manipulate, interact and perform experiments with robots living behind the "cloud". For this purpose, we devised a service-oriented framework based on SOAP Web services for mapping hardware and software robotic resources as *services* and publish them to the end-users as Web services. We considered robots operating with the Robotic Operating System (ROS) [10], which also provides an abstraction layer, at the level of the operating system, of the hardware resources of the robots. The main advantage of ROS is that it allows to manipulate sensor data of the robot as a labeled abstract data stream, called topic, without having to deal with hardware drivers. Several previous works have also proposed different architecture for cloud robotics and remote robotic labs, which we present and discuss their advantages and limitations in details in Section 2, and we clarify the difference of our proposed system as compared to other existing systems.

The remainder of this paper is as follows. Section 2 surveys the most relevant works on cloud robotics and discusses their contributions to the field. Section 3 describes the system and software architecture of the RoboWeb system starting from requirements specification to system design. In Section 4, we present the implementation, deployment and experimentation with the RoboWeb system and we demonstrate its features. Section 5 concludes the paper and discusses future works.

2 Related Works

The concept of cloud robotics has been increasingly expanding since the last three years. Basically, the cloud robotics research trend can be roughly classified into two categories: (*i.*) using cloud for virtualizing robotic resources (e.g. [1, 7]), (*ii.*) using cloud for offloading heavy computations from the robot to the cloud (e.g. [9]). In what follows, we present the most relevant works over the past four years in the increasing chronological order of their publications dates.

In [1], the authors exploited the Service Oriented Architecture (SOA) technology to design and implement a prototype of the Robot as a Service (RaaS) cloud computing model. The design complies with the common service standards, development platforms, and execution infrastructure, following the Web 2.0 principles and participation. The authors also demonstrated through experiments that their system is effective, flexible, and portable.

DAvinCi was proposed in [2] as a cloud computing software framework for service robots. The goal of this system is to offload intensive workloads from the onboard robots' resources to a backend cluster system in the cloud. The idea was to investigate the possibility of parallelizing the execution some complex robotic algorithms, and applied it to the FastSLAM algorithm as a proof of concept. The DAvinCi architecture was implemented using the open source Hadoop cluster and ROS as messaging framwork for the robotic ecosystem. The deployment did not consider network latencies and delays, which turns the results limited to ideal operational conditions.

In [11], the authors designed a robot cloud center to overcome the limitation in capacity, versatility and extensibility of robotic applications, and to meet the diverse requirements of the end-users requesting robot resources according to their demand. They also designed a Robot Resources Scheduler to minimize the task execution cost while still meeting the end-users requirements. Robot scheduling simulation proved that robots, especially whose cost-capability density is low, can be used more efficiently with the scheduler. In [12], the authors have proposed the RSi Research Cloud (RSi-Cloud) that seamlessly integrates robotic services with the Internet.

In [4], the authors described their vision of cloud robotics and proposed different possible architectures to address the constraints faced by current networked robots. The motivation of the work was to allow the robots to share information and computation resources among each other and cooperate through the cloud to acquire new knowledge and behaviors. The cloud artchitecture design takes into account two types of communication paradigms namely the machine-to-machine (M2M) communications among participating robots, and the machine-to-cloud (M2C) communications between the robots and the cloud. The authors also proposed three elastic computing models for cloud robotics, namely the peer-to-peer model, proxy-based model and the clone-based model.

In [13] and [7], the authors made interesting extensions to the ROS middleware, namely *rosjs*, which is a JavaScript library for ROS that exposes the robot functionalities as web services, and *rosbridge*, which is a light weight protocol that exposes robot sensor data and controllers, through web sockets accessible any-

where over the Internet, and provides security mechanisms and runtime tools for remotely manipulating the robots. Similarly to one of the objective of our work, The rosjs and robridge were proved to enable remote laboratories, and a prototype was implemented and tested for monitoring iRobot Create and PR2 robots. The difference with our work is that our approach is based on a SOAP-based service oriented archirtcture, which represents a complementary solution to rosjs and rosbridge.

3 RoboWeb System Architecture

In this section, we describe the system architecture and the software development process of the RoboWeb system. We start by specifying the functional and non-functional requirements of RoboWeb, then, we describe the system architecture and software design. RoboWeb differs from existing systems in that it leverages SOAP-based Web services for building the virtualization layer of the cloud robotic infrastructure.

3.1 Requirements Specification

As a research group installed in Saudi Arabia at Al-Imam Mohamed bin Saud University and Prince Sultan University, we have several robotic platforms including four Turtlebot robots, two Wifibot Lab robots, two unmanned aerial vehicles (UAVs), namely the AscTec Pelican and AscTec FireFly, which are cutting-edge and expensive robotic technologies, and several other sensor and robotic devices. Our objective is to allow our students and researchers abroad or from outside the University during non-working hours to access and use the robots in a ubiquitous and seamless way, i.e. anywhere and anytime, through the Internet. This lead us to design a service-oriented cloud robotics system, namely RoboWeb.

Basically, the idea of RoboWeb is to develop a service-oriented middleware that plays the role of the virtualization layer. This layer binds software and hardware robotic resources as Web services allowing authorized users to subscribe to the published services of interests, through which they can "play" with the robots. Two Web services options were possible: SOAP approach or the RESTful approach. We had to make a milestone decision at this point. Finally, we have opted for the SOAP approach for several reasons. First, SOAP provides a well-structured transctional model between the client (service subscriber) and server (service publisher) that allows to define a contract between both ends. Indeed, in contrast to SOAP, REST is basically an architectural style based on the HTTP protocol rather than a SOA middleware as it is the case with SOAP Web services. Second, SOAP Web services enables the definition of composable and complex Web services in contrast to REST. This is an important requirement in the design of RoboWeb as we need to take advantage of the flexibility of the SOAP approach to define different service/abstraction layers, which help achieving virtulization more effectively.

With respect to robots to be supported by the system, we considered robotic platforms operating with ROS. The adoption of ROS has several advantages. First, ROS is a free and open-source middleware for robots that acts as a meta operating system and builds a hardware abstraction layer. This makes the programming of ROS-enabled robots much easier as software developers will not have to deal with hardware drivers and interfacing. In fact, ROS already provides comprehensive and well-structured libraries and drivers for several robots and sensor devices, and publishes sensor data (camera frames, laser range data, IMU data, motors speeds, etc.) simply as labeled data streams called *topics*. Second, the control of robots through Web services will be much easier when ROS is used as the Web server will only have to deal with topics rather than with hardware resources. Indeed, ROS provides another level of resources virtualization at the operating system level. Third, ROS complies with component-based software development, which makes ROS-based system modular, extensible and flexible. This is particularly important as architectural design since services can be mapped to software components making easier their composition, addition and removal.

We also derived the following four (most important) non-functional properties for the RoboWeb system:

- **Service-Orientation:** This is the most important requirement in our system as we need to map any robotic resource or operation as a service. The SOA approach allow to easily extends the capabilities and functionalities of the system by dynamically adding services. The users will be able to manipulate robots in the same way they use any Web service. In addition, robot software developers can reuse available services to design more complex composite services.
- **Reliability:** The system must be reliable in different perspective. First, it must be available such that it ensures continuous connectivity with users at anytime. In addition, it must provide consistent view of the robot status to the users. For example, the system should consistently report in real-time the list of connected robots and change the connectivity status each time a robot join or leave the cloud.
- **Modularity:** The system should be easily extensible by dynamically adding/removing components to/from the system. The modularity ensures the independence of the different modules which makes their integration more effective. This is very appropriate for service composition and orchestration to build more complex Web services for manipulating the robots. For instance, making experiments with a robot can be seen as a complex Web service composed of several other Web services including accessing robot, running a program, getting the list of nodes and topics, etc. The modularity has also the advantage of allowing software reuse.
- **Real-Time:** Once the user is connected to a particular robot, it is important that the system ensures small and controlled delays. Indeed, the user must be kept up-to-date with latest status updates of the robot for effective control and monitoring. Large delays and delay variations (jitter) will compromise

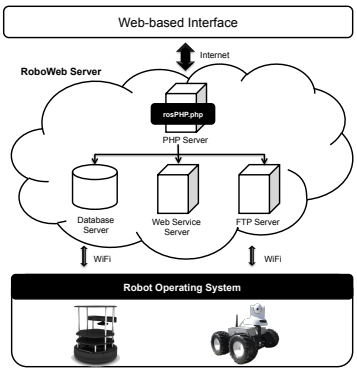

Fig. 1. High-Level System Architecture

the user experiments. Delays must be kept as low as possible to ensure interactivity between the user and the robot.

3.2 System Architecture Design

In this section, we describe the RoboWeb system architecture and discuss the design considerations. Figure 1 depicts a high-level overview of the system architecture. The bottom layer consists of the robotic ecosystem that comprises ROS-enabled robots, each of them runs its own ROS master node. Mobile robots are dotted with wireless communication capabilities allowing them to collaborate for performing certain missions on demand. the ROS platform is used for sensor data collection and streaming among the robot agents and the end-users (clients).

The top layer defines the Web interfaces for users to access and manipulate the robots remotely. We implemented a PHP library, called rosPHP, to act as an abstraction layer on top of ROS providing the required ROS functionalities to interact with ROS-enabled robots. The rosPHP layer allows the interaction between the end-users and the robots though SOAP Web services and provides several functionalities including connection to the Web server, getting the list of available ROS-enabled robots, getting ROS nodes and topics of selected robots, getting information about robots sensors, publishing and subscribing to a ROS topic, creating new ROS package, uploading, running and stopping ROS programs.

The core part of the system is the RoboWeb service broker. It basically include three main components: (1) the Web service server, which is required to deploy robotic Web services and respond to end-users requests. The Apache Axis Web server for development and deployment of SOAP Web services. (2) the back-end MySQL database that is used to store information about the whole cloud including robots, users, programs, reservations, experiments, etc. and (3) the FTP server, which is used to upload files, namely experiments output, user

programs, and robot description files. The robot description files are XML files that contain meta-data about the robots and their sensors.

Figure 2 presented a more detailed view of the system architecture and its subsystems.

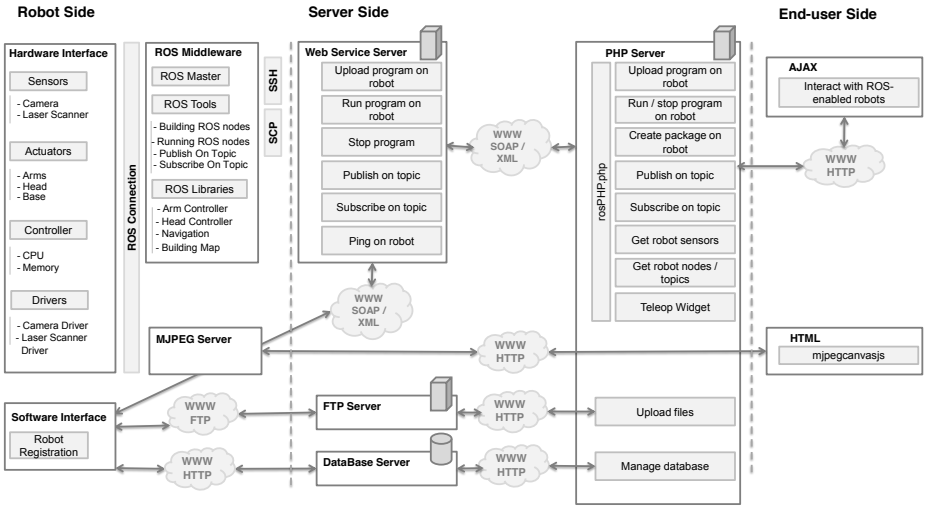

Fig. 2. Low-Level System Architecture: The figure presents four main parts of the RoboWeb System. The End-User side uses a AJAX interface to interact with ROS-enabled Robot through a PHP Server (rosPHP) the implements the core functionalities of ROS. The rosPHP server communication with the Web Service server that directly interfaces with ROS middleware installed on robots through SSH and SCP protocols to perform requested commands submitted by the user and ensure sending back responses to the end-user side.

The robot side of the system includes the hardware interface that consists of robot hardware resources (i.e. sensor, actuators, controllers) and their drivers. This interface is abstracted by the ROS middleware that provides a first level of virtualization to all robotic hardware resources. Indeed, any sensor or actuator data is provided by ROS as a a stream of data that can be manipulated by any client that subscribes to that data. ROS manages the hardware through ROS connection which provides tools and libraries required to control and manage hardware and software interfaces. The MJPEG server is defined as a ROS package that streams image topics captured from the robot camera ROS using the HTTP protocol, so that it can be displayed by any browser. The software interface is responsible for the robot auto-registration to the system. We developed a program that allows a robot to register to the RoboWeb system and publishes the services that it provides. In the server side, the Web service server interacts with the ROS middleware through the SSH secure communication protocol to execute commands or programs on the remote robot. On the other hand, it uses

the Session Control Protocol (SCP) to transfer computer files between the server and to run program on the remote robot. We also consider the PHP server which essentially includes three main parts. The first part consists of the rosPHP library, which represents a PHP layer that defines ROS and network commands to be executed on the robot such as upload program to robot, run a ROS node on the robot, stop a ROS node, publish a ROS topic, create a ROS package, get robot sensor information, subscribe to a ROS topic and ping a robot, and teleoperate the robot. the rosPHP layer communicates with the Web service server via SOAP/XML protocol, and also define methods to access FTP and database servers via FTP and HTTP protocols.

The end-user side consists of the user-interface which uses AJAX to interact with PHP server. It also uses the *mjpegcanvasjs*, a JavaScript tool that allows the user to easily display, manage and modify ROS image streams received from MJPEG sever via HTTP.

4 Implementation and Deployment

4.1 Hardware and Software Suits

To demonstrate the feasibility of our architecture, we developed a complete prototype of the RoboWeb application and tested on a wireless local area network. The Web Service server was implemented on a computer laptop with Intel(R) Core(TM) i3 CPU, 4.00 Go RAM, and Ubuntu 12.04 OS, running Apache Axis (Apache eXtensible Interaction System) Web service framework for generating and deploying Web service applications; Apache Tomcat, which provides a Java HTTP web server environment for Java code (including Servlets and JSP) to run in; and Eclipse IDE for software development.

The front-end user interface provides an easy-to-use and intuitive GUI to interact with the robots living behind the cloud. It was implemented using: (*i.*) AJAX client side scripting technology for ensuring asynchronous interaction with the rosPHP server library that we developed. It provides the benefit of asynchronous communication with the server seamlessly in the background without interference with the display and the behavior of the web page; (*ii.*) HTML5 Web Workers technology to take benefit from its multi-threading capability in particular for subscribing to ROS topics; indeed, Web Workers technology allows the execute and run multiple JavaScript scripts in the background of a web page independently of other user-defined scripts, and enables to perform parallel and computationally expensive tasks without interrupting the user interface. This is particularly useful in our RoboWeb system as a robot may independently subscribe to or publish several ROS topics that must be handled with different threads in the user-interface; (*iii.*) mySQL triggers and procedures to manage the information and reservations of the robots; and (*iv.*) JQuery and CSS for the dynamicity and the design of the interface.

The back-end database was also implemented using the mySQL 5.5 server. The FTP server was set-up on the same computer laptop using the vsftpd 3.0

server (Very Secure FTP Daemon), which is an FTP server for Unix-like systems, and represents the default FTP server for the Ubuntu OS. Regarding the PHP server, we have installed PHP5 on the same computer laptop.

As for the robotics hardware, we tested our RoboWeb prototype with two ROS-enabled robots, namely the TurtleBot 2.0 robot and the Wifibot Lab V2 robot. Any other ROS-enabled robots can easily be added to the RoboWeb system as will be explained in the deployment subsection.

4.2 Deployment

In this subsection, we provide step-by-step guidelines on the deployment of the RoboWeb system through illustrative examples and we demonstrate how to use it for accessing and manipulating ROS-enabled robots.

The first step in deployment consists in setting-up and configuring the back-end system of the RoboWeb cloud, that is configuring the robots, running their ROS middleware and setting-up their networking configurations including the IP addresses, the IP port numbers, the IP Addresses of the ROS Masters and its port numbers. These settings are crucial for the ensuring the communication between the ROS-enabled robots and the system. Second, at its startup, the Web Service server will check and discover existing robots in the cloud automatically. Actually, once a robot joins the cloud, it uploads its description file on the FTP server. Then, it registers itself in the back-end database, or updates its status and IP address if it has already been registered. At this instance, the robots are considered as *active* and accessible to the end-user interface through the RoboWeb cloud. Finally, the robot invokes a web service that periodically tests its connectivity. This web service tests the robot connectivity every 30 seconds. If the robot disconnects or fails, the web service will attempt three times testing the robot connectivity: 180, 300 and 600 seconds later. If the robot remains still disconnected, the web service will deactivate the robot by updating its status in the database to be *inactive*, notify the administrator by email, and stop testing the robot connectivity. In what follows, we present the main functionalities at the user side.

1. **Register and authenticate:** First, the end-user is required to create an account to be authorized to access the robotic resources. Once the registration request is submitted and approved by the administrator, the end-user must authenticate to use the system functionalities. Non-authenticated users are only able to get information about active robots in the cloud. They are not allowed to reserve robots or perform experiments.
2. **Browse the list of active robots:** Once authenticated, the end-user is allowed to obtain and browse the list of active robots (i.e. already connected with the Web Service server) with information including the list of available ROS topics and ROS nodes, robot sensors (camera, laser data), IP address, and status. The top half of Figure 3 depicts the list of robot categories (TurtleBot 2.0, Wifobot Lab V2) supported by the RoboWeb robotic prototype.

Fig. 3. Robot Description Interface

When the end-user selects a robot category, all active robots that belong to this category will be displayed. Then, the end-user can get information of a given robot including (1) robot description, (2) list of available ROS Nodes, and (3) list of available ROS Topics. The bottom half of Figure 3 shows the information about the selected robot. It is also possible for a user to look at the list of ROS topics and the list of ROS nodes of the selected robot, as illustrated in Figure 4. This interface allows the user to indentify the different ROS nodes and topics that he might need in his application and helps him choosing the most appropriate robot for his experiments before proceeding to the reservation.

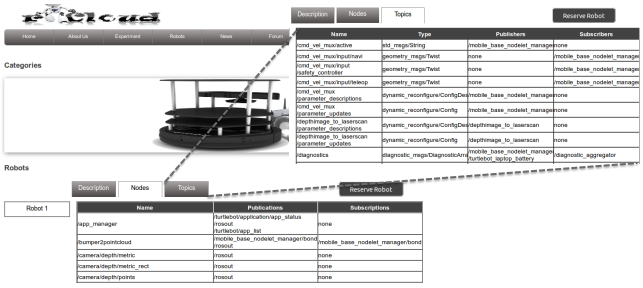

Fig. 4. List of Available ROS Nodes and ROS Topics in the Selected Robot

3. **Reserve Robot:** After browsing the list of robots and identifying currently active robots and their specifications, the end-user may reserve a robot to use for experimentation and/or remote manipulation. The reservation process

consists in booking the requested robotic platform for a particular reservation date and time. The RoboWeb reservation system is able to check possible booking conflicts and only propose the user with available dates/times for the available active robots.

4. **Perform Experiment:** Once an authenticated user has successfully booked a robot, he will be allowed to access and use that reserved robot for running his experiments at the allocated time slot. Authorized users are allowed to interact with and manipulate a given robot by uploading and running a ROS program, in addition to remotely controlling and monitoring it. For safety of execution, experiments should be run with the assistance of a local technical staff to avoid hazardous manipulation and control of the robot in the cyber-lab space. Figure 5 shows the ‘‘Robot Experiment Interface”, which provides information concerning the selected robot, the booked period, the robot camera video streams, and the list of ROS topics. The Robot Experiment Interface also shows buttons to upload, run, and stop ROS programs on reserved robot. The end-user is allowed to publish and subscribe to any of the available ROS topics. He also allowed to send rospy commands (ROS Python) to control the robot. Execution results are displayed in the output area in real-time.

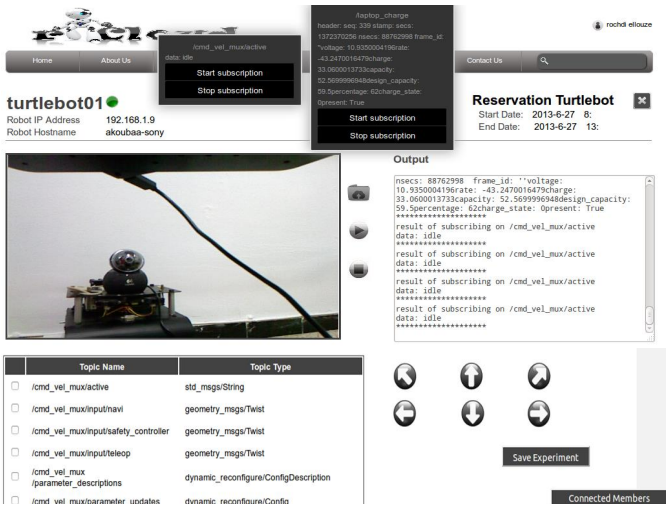

Fig. 5. Robot Experiment Interface

For maintaining connectivity with the server, the client system controls continuously whether the web server is working, and whether the reserved robot is still connected and active. The green light on the right side of the name of the reserved robot indicates that the robot is still active. When the connection is lost, the end-user will be updated by changing the connection color to the gray. Figure 6 displays the warning messages sent by the system to

alert the end-user when a problem, with the web server or the robot, occurs.

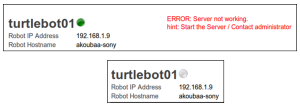

Fig. 6. System Fault-Tolerance: The figure shows an error message when the server stops working for any reason, warning the user and requesting him to contact the administrator

5 Conclusion

In this paper, we presented RoboWeb, a SOAP-based service-oriented architecture that virtualizes robotic hardware and software resources and exposes them as services through the Web, contributing to the evolving concept of cloud robotics. The major contribution of this paper lies in the integration of different Web services technologies with the Robot Operating System (ROS) middleware to allow for different levels of abstraction (multi-layer architecture), ensuring more modularity and flexibility of the deployment. We have also demonstrated the feasibility and the added value of RoboWeb through a complete prototypic implementation.

Although we believe that this work provides a consistent step towards the future cloud robotics paradigm, we are currently planning and working on extending the RoboWeb system design and deployment in several perspectives. First, we aim at extending the deployment to the Internet rather than on a local area network. Proxy servers can be used for that purpose. Furthermore, we aim at looking into more depth into security issues, in particular, investigating potential attacks and threats that might compromise the robots' cloud operation, and undertake appropriate preventive measures. Finally, the RoboWeb system should allow the user to reserve and use more than one robot to be able to deploy multi-robot applications.

Acknowledgment

This work is supported by the iroboapp project "Design and Analysis of Intelligent Algorithms for Robotic Problems and Applications"[14] under the grant of the National Plan for Sciences, Technology and Innovation (NPSTI), managed by the Science and Technology Unit of Al-Imam Mohamed bin Saud University and by King AbdulAziz Center for Science and Technology (KACST).

The author would like to thank Fatma Ellouze for her excellent performance in this work in the context of her graduation project and her outstanding work in the implementation of the RoboWeb system.

Also, the author would like to thank Rihab Chaari, Dr. Slim Kallel and Dr. Wajdi Louati for the technical support they provided.

References

1. Yinong Chen, Zhihui Du, and Marcos Garca-Acosta. Robot as a service in cloud computing. In *Fifth IEEE International Symposium on Service Oriented System Engineering*. IEEE, 2010.
2. Rajesh Arumugam, Vikas Reddy Enti, Liu Bingbing, Wu Xiaojun, Krishnamoorthy Baskaran Foong Foo Konga, Kang Dee Meng Senthil Kumar, and Goh Wai Kit. Davinci: A cloud computing framework for service robots. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3084–3089, 2010.
3. Markus Waibel, Michael Beetz, Javier Civera, Raffaello DAndrea, Jos Elfring, Dorian Galvez-Lopez, Kai Haussermann, Rob Janssen, J.M.M. Montiel, Alexander Perzylo, Bjorn Schiele, Moritz Tenorth, Oliver Zweigle, and Rene van de Molengraft. A world wide web of robots: Roboearth. *IEEE Robotics and Automation Magazine*, 2011.
4. Guoqiang Hu, Wee-Peng Tay, and Yonggang Wen. Cloud robotics: architecture, challenges and applications. *Network, IEEE*, 26(3):21–28, 2012.
5. K. Kamei, S. Nishio, N. Hagita, and M. Sato. Cloud networked robotics. *Network, IEEE*, 26(3):28–34, 2012.
6. Jihoon Lee. Project report:web applications for robots using rosbridge, 2012.
7. Osentoski S., Pitzer B., Crick C., Graylin J., Dong S., Grollman D., Suay H.B., and Jenkins O.C. Remote robotic laboratories for learning from demonstration. *International Journal of Social Robotics (SORO), special issue on Learning from Demonstration*, 2012.
8. Tsung-Hsien Yang and Wei-Po Lee. A service-oriented framework for the development of home robots. *International Journal of Advanced Robotic Systems*, 2013.
9. Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-based robot grasping with the google object recognition engine. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
10. Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
11. Zhihui Du, Weiqiang Yang, Yinong Chen, Xin Sun, Xiaoying Wang, and Chen Xu. Design of a robot cloud center. In *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, pages 269–275, 2011.
12. Y. Kato, T. Izui, Y. Murakawa, K. Okabayashi, M. Ueki, Y. Tsuchiya, and M. Narita. Research and development environments for robot services and its implementation. In *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 306–311, 2011.
13. Sarah Osentoski, Graylin Jay, Christopher Crick, Benjamin Pitzer, Charles DuHadway, and Odest Chadwicke Jenkins. Robots as web services: Reproducible experimentation and application development using rosjs, 2011.
14. iroboapp: Design and analysis of intelligent algorithms for robotic problems and applications, http://www.iroboapp.org.