J Intell Robot Syst (2017) 87:487–506
DOI 10.1007/s10846-017-0504-y

SOM4R: a Middleware for Robotic Applications Based on the Resource-Oriented Architecture

Marcus V. D. Veloso · José Tarcísio C. Filho · Guilherme A. Barreto

Received: 23 September 2015 / Accepted: 26 January 2017 / Published online: 23 February 2017

Abstract This paper relies on the resource-oriented architecture (ROA) to propose a middleware that shares resources (sensors, actuators and services) of one or more robots through the TCP/IP network, providing greater efficiency in the development of software applications for robotics. The proposed middleware consists of a set of web services that provides access to representational state of resources through simple and high-level interfaces to implement a software architecture for autonomous robots. The benefits of the proposed approach are manifold: i) full abstraction of complexity and heterogeneity of robotic devices through web services and uniform interfaces, ii) scalability and independence of the operating system and programming language, iii) secure control of resources for local or remote applications through the TCP/IP network, iv) the adoption of the Resource Description Framework (RDF), XML language and HTTP protocol, and v) dynamic configuration of the connections between services at runtime. The middleware was developed using the Linux operating system (Ubuntu), with some applications built as proofs of concept for the Android operating system. The architecture specification and the open source implementation of the proposed middleware are detailed in this article, as well as applications for robot remote control via wireless networks, voice command functionality, and obstacle detection and avoidance.

Keywords Middleware · Resource-oriented architecture · Mobile robotics · Subsumption architecture · BPMN diagrams

1 Introduction

With the rapid pace of evolution of mobile devices (e.g. smartphones, tablets, ultrabooks) it has been observed the birth of a novel paradigm for the construction of robots, especially mobile ones, but not restricted to them: from robots built using special-purpose hardware (e.g. PIC - Programmable Interface Controller, dsPIC) with reduced computational resources, to robots built using general-purpose mobile devices (e.g. notebooks, netbooks, tablets or smartphone). Currently available mobile devices possess much higher computational power than an average desktop PC of a decade ago, adding to the robot parallel processing capability, and providing various options of wireless network with broadband connection, multiple choice for operating systems and programming languages. This new robot construction

M. V. D. Veloso · J. T. C. Filho · G. A. Barreto (✉)
Department of Teleinformatics Engineering,
Federal University of Ceará Center of Technology,
Campus of Pici, Fortaleza, Ceará, Brazil
e-mail: gbarreto@ufc.br

M. V. D. Veloso
e-mail: veloso@fisica.ufc.br

J. T. C. Filho
e-mail: jtcosta@ufc.br

paradigm has popularized robotics beyond academic and industrial borders and triggered the development of applications requiring integration of resources from one or more networked robots. Indeed, integration of robotic resources over network is within the core of several open and non-open source projects available in the robotics community.

Despite mind-blowing advances in robotic systems in recent years, transparent integration and sharing of resources from one or more robots through TCP/IP network is still a challenge. This results from the fact that resources e.g. laser, sonar, camera, accelerometer, gyroscope, among others, have different data formats, drivers and communications APIs,[1] with most of these APIs being either proprietary, which complicates the debugging process at all levels of the software stack, or are designed for a specific operating system, limiting the efforts for integration of resources.

Furthermore, rapid technological changes and constant alterations in the requirements of applications demand periodic reviews of the software projects in order to prevent the robotic middleware to become obsolete in the short term. The smaller the changes in software resulting from changes in hardware, the smaller are the implementation and maintenance costs for the robotic company. Bearing this in mind, a well-designed middleware should allow application developers not to get concerned with the details of each data source and the complexity of computing environments (including distributed scenarios). As a consequence, for developing and maintaining software systems, as in the case of middleware systems, suitable software engineering methodologies must be used to facilitate the design and to reduce implementation and maintenance time. Reuse and integration of codes from existing robotic projects then become mandatory.

Intensive research has been carried out to develop, analyze and compare different software projects with regard to the integration of robotic resources over distributed systems. For example, Kramer and Scheutz [27], in a survey paper on robotic development environments (RDEs), describe and evaluate nine open source projects, suggesting potential areas of improvement for the maintainers of these RDEs based on the demands of multi-agent systems (MAS). Mohamed et al. [31] present a brief review of middleware for networked robots. They examine the current limitations of ten middleware platforms and identify several open issues that need to be addressed, including security and other advanced integration features, automatic discovery and high-level abstractions. Elkady and Sobh [19] present a survey of middleware projects for robotics aiming at discussing their architectures and important features of middleware platforms for robotics. Finally, Elkady et al. [18] propose their own robotic middleware (RISCWare), based on the desired features identified in [19].

It should be noted that the development of a software architecture able to integrate and share the multitude of heterogeneous software and hardware components, either of a single robot, or of networked robots, has been a great challenge for robotic researchers along the last two decades in particular [6, 14, 15, 18, 19, 23, 25–27, 30–34, 37, 38]. The main difficulties in building such a complex software architecture for robotics involve basically the issues of hardware abstraction, communication reliability and security.

Hardware Abstraction/Modularity A common challenge for robotics researchers is the reuse of software modules developed by the existing robot projects, since in most of them there is absolutely no compatibility between hardware architectures and operating systems. Thus, reuse of available software modules by new robot projects is minimal, because the software architecture of the former projects is designed to meet the requirements of their specific robot hardware. As a consequence, the software is often completely redesigned and built for each new robot hardware [33].

A middleware common to multiple robots is then highly desirable and should not only define how the components can interact with each other by means of communication and synchronization mechanisms, but also provide infrastructure and functionality to the system construction. A modularized software architecture allows change in one module to have less impact on the others and makes the incorporation of new hardware devices easier [23, 31]. This stimulates other researchers to use the available modules across different hardware architectures helping to make hardware integration painless.

Communication Reliability This is an issue of great impact in networked, swarm and cloud robotics. Poor

[1] Application Programming Interface.

signal availability and small bandwidth of current wireless networks (eg, 3G, Wi-Fi 802.11b) can either slow down considerably or even restart the communication channels between the processes (running programs or modules) distributed in the network. In some cases this may lead to a degradation of response time due to communication, impairing the system integration. Thus, the choice of a suitable communication technology is of paramount relevance in networked robotics.

Concurrence is also an issue that has great impact in communication reliability. Disordered accesses to resources shared by concurrent processes generate unpredictable behavior of the distributed computing system. The management of competition between two or more processes for an access to the same resource in the network is a source of difficulty in the phase of software development. There are a few different methodologies for implementing mechanisms for managing shared resources. For example, Gerkey et al. [23] associated a command and a data buffer for each device, but as the server does not implement any locking device, customers must implement their own arbitration mechanism. Metta et al. [30] implemented synchronization mechanisms (no-wait, wait-after and wait-before) on the server side and behavior mechanisms (triple-buffer, double-buffer and single-buffer) on the client side. Nevertheless, these mechanisms still allow resource updates to be either lost, because the server never send them, or discarded on the client side, generating unnecessary data traffic on the network.

Security Issues Software integration through the Internet always raises concerns about security. It is necessary, therefore, to include such mechanisms to ensure that only robots authorized by the middleware can communicate with each other and that only authenticated users can access and control networked robots. Obviously, in order to penalize minimally real-time operations mediated by the robotic middleware it is important to minimize network overhead due to security protocols (for example, in handshaking). In fact, it is worth mentioning that there is a trade-off between network overhead due to security and efficient real-time operation. For this reason, the middleware can be deliberately designed to disregard any security mechanism, restricting its use to an isolated or firewall-protected network [30]. As will be shown later on this paper, it is possible to ensure network security integrated with the Internet without degrading the "real time" operation of the middleware.

From the exposed, we propose an easy-to-use middleware for intelligent integration and sharing of robotic resources (sensors, actuators and services) identified by URIs (Uniform Resource Identifiers), using the TCP/IP network, employing protocols with minor restrictions on firewalls and a resource description language that can be extended to express propositions about numerous subjects. With this middleware architecture, it is possible to ensure security of access to resources, abstraction of heterogeneous robotic hardware, reuse of software infrastructure for robots across multiple search efforts, reduction of the coupling among multiple applications, faster code portability and scalability of the architecture. The proposed software architecture is general enough in the sense it can take advantage of the software resources provided by existing robotic middlewares, such as the *Robot Operating System* (ROS) [37], managing and integrating them in a transparent and seamless way to the user.

The remainder of this paper is organized as follows. In Section 2 we introduce the proposed middleware architecture and discuss its features with respect to the state-of-the-art in open-source robotic middlewares. In Section 3 we describe applications of the proposed middleware to robot control via voice commands and robot remote control through smartphones/tablets. We conclude the paper in Section 4.

2 SOM4R: Simple and Open Middleware for Robotics

The proposed middleware, henceforth referred to as SOM4R, is a service layer interfacing with the base layer of sensors and actuators and with the upper layer of users applications (see Fig. 1). The base layer is comprised of sensors and actuators devices directly connected to the robot computer or microcontroller and recognized by their operating system drivers (or firmware modules). The upper layer is comprised of the software applications that are responsible for functions, such as voice command, obstacle detection, navigation system, people identification, among other functions utilized by the user through a web-based

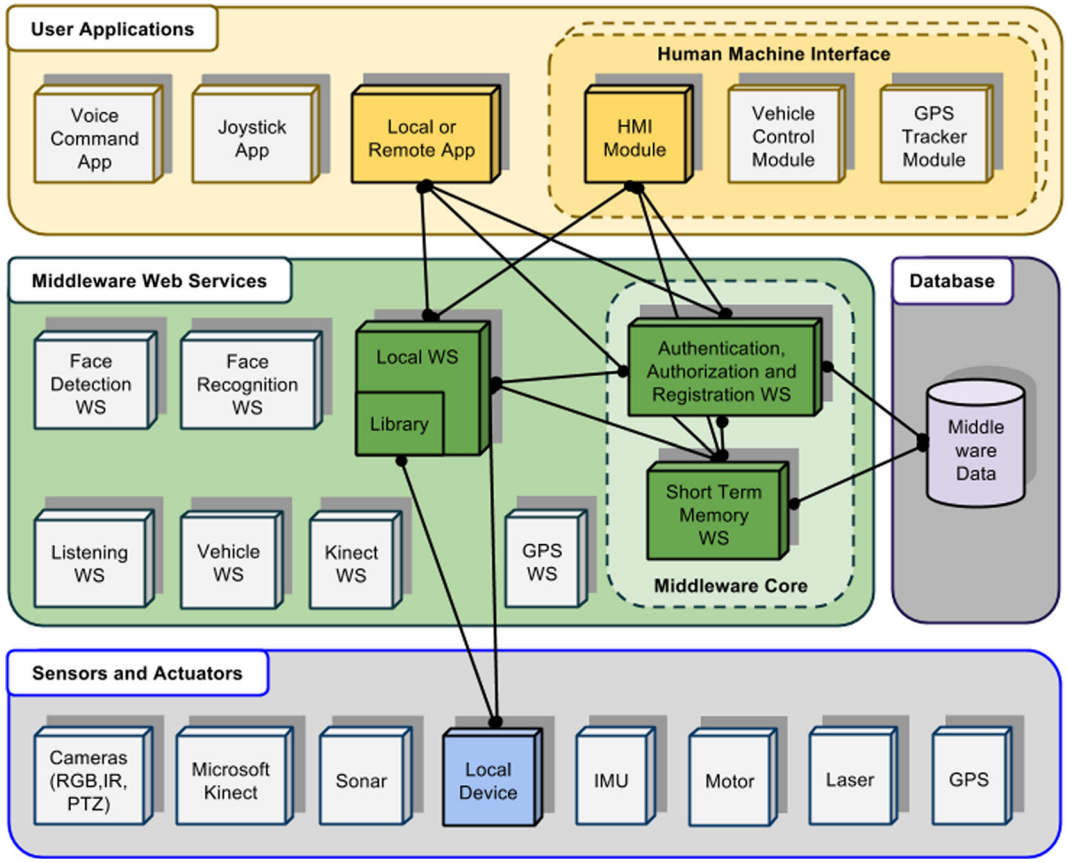

Fig. 1 Model of data and control flows between applications (local or remote), local services, middleware core and database server

Human-Machine Interface (HMI). These applications must use the middleware services in order to interact with the robots capabilities to achieve their goals and fulfill the tasks under their responsibilities.

More specifically, the proposed middleware consists of a set of web services identified by URIs and organized in a modular way, following the resource oriented architecture (ROA) [35], in order to implement a software structure for autonomous robots. In this sense, the SOM4R has abstract interfaces for all hardware systems and their functionalities, independently of the distributed computing platform and language implementation of web services. Thus, web service changes cause minimal impact on the remaining services. The middleware web services specify the interfaces of abstraction, as defined and described using XML, to access the resources of robot(s), such as sensors, actuators and services, and transfer the representational state (REST) of these resources at multiple levels of services granularity using uniform interfaces (classification system of the request message) of the HTTP protocol.

REST is a manifestation of ROA [35], which comprises an additional set of constraints over the service orientation architecture (SOA). The term REST was introduced and defined in [20] and represents a new software architectural style for distributed systems. A REST web service is supplied for consumption independently of the technology used in client-server communication.

Each web service member of the SOM4R has one or more abstraction interfaces (see Section 2.1). Interfaces are sets of attributes (i.e. data structures) defined by the web service to describe the representational state of resources (sensors, actuators and services) using RDF/XML (see Section 2.2). They allow greater decoupling between the sensor or service associated with the resource and the client applications of the middleware, as the resource cannot be handled directly by client applications of the middleware. In practice,

several resource representational states are prepared and transferred either by the web service to the client application, or by the client to the web service.

In the proposed middleware, any information or concept that can be presented by a name qualifies itself as a potential resource. A resource is a conceptual mapping to an entity set [7]. The resource, identified by a URI, is the information provided by the sensor, or the information sent to the actuator, or also the information obtained from a web service. As an example, a resource may be either the information of the status of a robotic device (e.g. sonar, laser, GPS, sensor/actuator hardware), or the information arising from the processing of resources provided by other services (e.g. voice command, face detection, landmark detection). It is possible to find several resources of different granularities, according to the scenario of each client application. Granularity refers to the level of detail or summary contained in the existing data units.

In SOM4R, it is assured that data exchange between client and server is done in atomic way, without coupling between software applications and middleware. It should be noted that REST has become an alternative to RPC/SOAP-like approaches to web services architecture by making full use of web resources. In [29], it is performed a battery of tests to measure the performance of REST web services, and it was concluded that it is a more suitable solution for the integration of data distributed across world wide web. To implement REST architectural style in SOM4R, we use HTTP uniform interfaces to form a data CRUD (Create, Read, Update and Delete), where the GET method is used to retrieve a representation of a resource or perform a query, POST is used to create a new resource, dynamically named, PUT is used to update a known resource, and DELETE is used to remove a resource.

Finally, in the proposed middleware, the representation of a resource is transferred only on demand, where the frequency is determined by the client application at runtime according to their needs or processing capability, reducing the traffic of data in the distributed computing system.

2.1 Implementation

In this section, we describe in more detail the implementation of the SOM4R core middleware web services and security. We also present additional services built specifically for an autonomous wheeled mobile robot (to be described in Section 3) using the proposed middleware in order to integrate functionalities, such as voice recognition, face detection, landmark detection, vehicle locomotion, TTS, GPS (Global Positioning System) positioning, battery charge monitoring and 3D obstacle detection. Emphasis is given to the structure of control, support for swarm robotics, and the functionality of the core and additional services.

All project codes were implemented as open source software and are publicly available for download at http://som4r.net.

Security Issues To support a basic level of secure access to web services we adopted an approach inspired by the OAuth 2.0 security protocol [4] and based on the HTTP Digest (RFC2617) access authentication method [21, 36]. The rationale for this approach stems from the following facts: i) HTTP Digest is the authentication method used by HTTP servers to validate the authentication of clients, and ii) OAuth is an open protocol currently adopted by major IT companies. It is simple and standard, which allows secure authorization between web and desktop applications.

OAuth 2.0 defines four roles in the process of authentication and authorization: a) resource owner, b) authorization server, c) resource server, and d) client. In this scenario, the SOM4R core performs simultaneously the roles of *resource owner* - responsible for allowing access to a protected resource, and *authorization server* - issuing tokens to clients after authentication and authorization of the resource owner. The other middleware web services perform similarly to a resource server that allows access to a protected resource through requests with token, while the applications perform the role of a client issuing resource requests on behalf of the resource owner and with its authorization.

The SOM4R uses security methods with MD5 encryption (HTTP Digest and tokens) whose one way hash algorithm of 128 bits is described in RFC1321, allowing secure use of the middleware through the Internet. Thus, in order to gain access to resources protected by the middleware a client needs to be authenticated and authorized in this environment, according to the steps described in Figs. 2 and 3 using Business

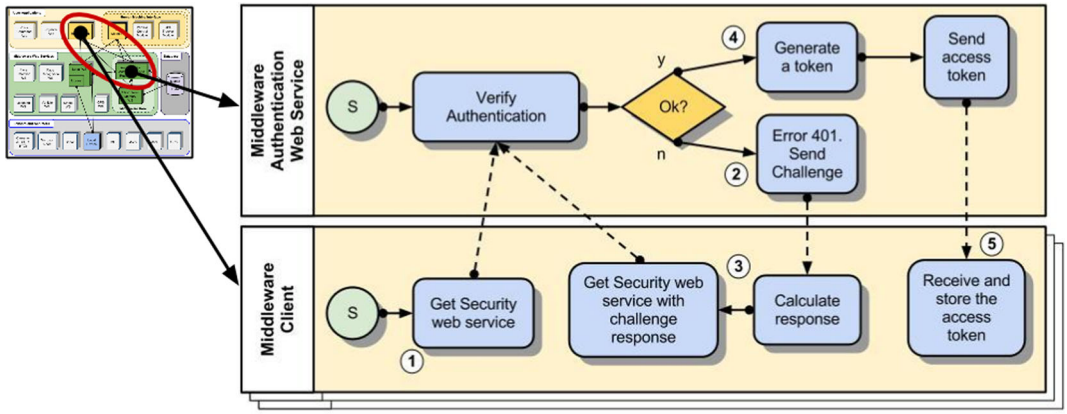

Fig. 2 Diagram representing the SOM4R authentication process

Process Modeling Notation (BPMN) [1] graphical representation. To increase security, a client can make a secure connection (HTTPS) with middleware services, because the incapacity of the client on confirming a server's identity is a weakness of HTTP Digest authentication method.

For the authentication process, illustrated in Fig. 2, the client needs to perform the request (Fig. 2, step 1) and inform (Fig. 2, step 3) the appropriate response to the challenge sent by SOM4R (Fig. 2, step 2). The answer to the challenge is calculated based on the login and password data known only by the client and the server, and also based on a random number (nonce) generated by the middleware and embedded in the challenge. When the client informs the appropriate response (Fig. 2, step 4), SOM4R registers the client and it receives an access authorization token (Fig. 2, step 5) from SOM4R.

Figure 3 describes this authorization process using BPMN. When a middleware web service receives the request (Fig. 3, step 1) made by the client, it makes a request (Fig. 3, step 2) containing the access authorization tokens for the SOM4R authorization service. This authorization service is responsible for validating tokens and checking access authorization to resources between services and applications, returning (Fig. 3, step 3) to the requester web service the expiration time (timeout) of the tokens received. Thus, the web service provides the resource (Fig. 3, step 4) requested by the client once the middleware responds with the access

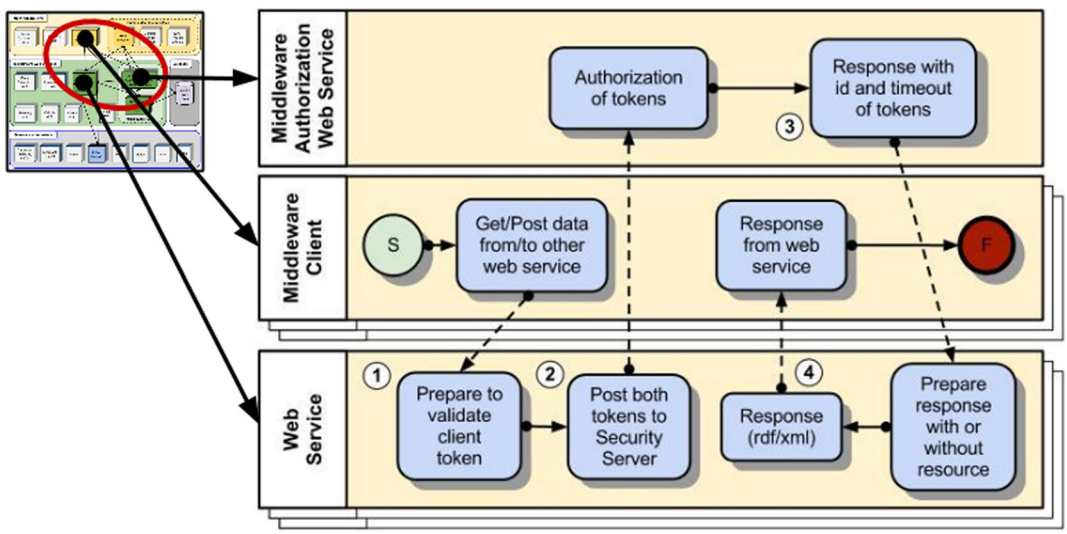

Fig. 3 Diagram representing the SOM4R authorization process

permission (i.e. $timeout > 0$). Otherwise ($timeout = 0$), it responds with HTTP 401 (Unauthorized) error.

It is worth mentioning that SOM4R employs tokens with a validity period (timeout) that allow dynamic configuration of communication between processes at runtime, providing security on identification of services and making data and control flows safer and more configurable.

Middleware Core Web Services The SOM4R core web services are responsible for the authentication and registration, the authorization, and the short-term memory of the proposed middleware, namely:

(i) Authentication and Registration Web Service - In SOM4R, this service is responsible for the authentication and security of the middleware, controlling access to resources using HTTP Digest Authentication method [21] and tokens [36]. In order to gain access to resources protected by the middleware environment, all services and applications have to initiate the connection, an authentication process and a registration by interacting with this service (see Fig. 2). The P2P (Peer-To-Peer[2]) network requires some kind of search engine to allow processes to meet each other at run time [37]. In this sense, the registration web service maintains an updated list of services and applications authenticated by SOM4R with their respective addresses.

(ii) Authorization Web Service - All communications between applications and services connected to the proposed middleware are validated using this authorization service (see Fig. 3). The SOM4R web services can keep in memory the expiration time (timeout) of the client token, reducing data traffic through the network over the next accesses of this client.

(iii) Short-term Memory (STM) Web Service - Responsible for maintaining a database with the recent history of events and activities related to usage of middleware modules. This web service also allows queries about previous events and activities for use in decision-making of the proposed middleware applications and services. It uses a database server (MySQL) to ensure the integrity and availability of the increasing volume of messages exchanged between local and remote software processes interconnected by the middleware. This web service is of crucial importance because it helps the debugging process at middleware level, in order to collect information about what is happening with all the robot processes. For example, if an application for "Greeting a Person" is developed, it can perform a search in the STM web service in order to avoid greeting the same person more than once in a certain interval of time. Thus, the STM mechanism simplifies the software modules, since they do not need to maintain an internal data structure to store their own activities neither those of the others software modules. In addition, this service keeps a maximum limit of data per application or service, eliminating older and less used records to avoid compromising the storage capacity and speed of queries.

Additional Services In the following we briefly describe additional services that were implemented in SOM4R for allowing access of the clients to various resources of a mobile robot with wheels (more on the robot in Section 3).

An important feature of the proposed middleware is its ability to easily incorporate resources from existing open-source libraries for specific tasks or sensors (e.g. OpenCV [11], ARToolKit [9, 10], OpenKinect [5], PyFaces [6]). Some services to be described below were developed using such libraries. These particular web services were implemented in Python.

(i) Vehicle Web Service - responsible for direct communication with the firmware that controls the motors of a robotic vehicle connected via a USB port, providing the reading of the current status of the resource (e.g. speed and direction) and receiving commands to be sent to the hardware which controls robot actuators (Fig. 4). In this case, communication between the firmware and the web service uses USB Human Interface Device Class (HID). The advantage of using HID interface is that for most needs, the existing support for HID devices can typically be adapted much faster than having to create an entirely new protocol, and most operating systems will recognize standard HID devices without needing a specialized driver (e.g. mouse, webcam).

[2]Each computer in the network can act as a client or server for the other computers in the network.

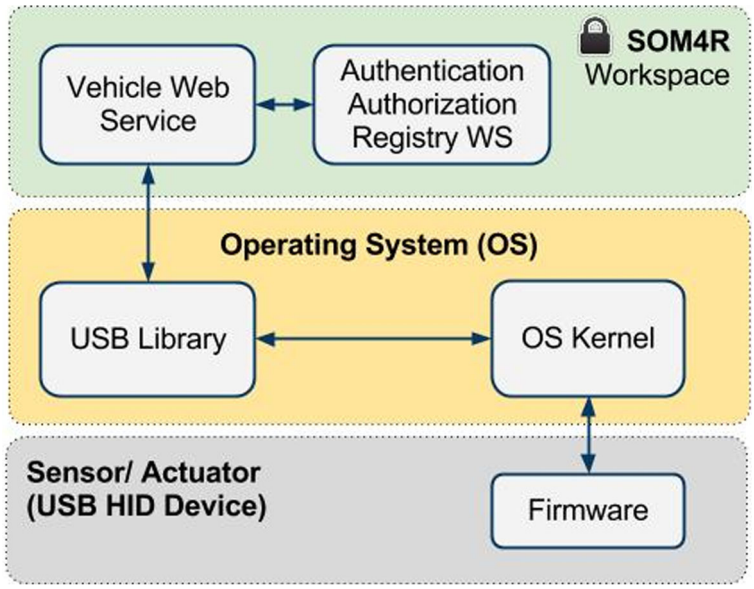

Fig. 4 Communication between vehicle web service and firmware that controls the actuators

(ii) Voice Recognition Web Service - Responsible for the recognition of human voice commands and the provision of the result to other services and applications. This service uses the CMU Sphinx library [2], an open source toolkit for speech recognition developed at Carnegie Mellon University. This software implementation was based on the examples of CMU Pocketsphinx project.

(iii) Web Service for Text-To-Speech (TTS) - Responsible for receiving a word or sentence and send it to the voice synthesizer of the robot, enabling the robot with a voice interface with the user. This service uses the *eSpeak* software [3], a compact open source software speech synthesizer with support for more than 30 languages.

(iv) Web Service for Global Positioning System (GPS) - Responsible for reading the GPS position of the robot, directly from hardware (or through APIs), providing the result to other services and applications. For the mobile robot with wheels built along this research, the GPS device is connected to the Arduino hardware that is connected to the notebook through a USB-to-Serial Com Port Adapter. In this case, communication between the firmware and the web service uses a standard RS232 serial interface.

(v) Face Detection Web Service - Responsible for face detection using the *OpenCV* library [11], providing the results to other services and applications. This software implementation was based on OpenCV code samples and uses classifiers obtained from the OpenCV library (e.g. the cascade classifier [28]). This software implementation was based on the examples of OpenCV project.

(vi) Face Recognition Web Service - Responsible for identifying people by face using libraries of the PyFaces project [6], providing the results to other services and applications. PyFaces project is a facial recognition system that uses eigenfaces algorithm [8]. This software implementation was based on the examples of PyFaces project.

(vii) Landmark Detection Web Service - Responsible for detecting landmarks using the ARToolKit API [10], a software library for building augmented reality applications. In SOM4R this service can be used to guide navigation. For example, a battery recharge application could detect when the battery reaches a critical low level and then drive the robot to the recharging base landmark. This software implementation was based on the examples of ARToolKit project.

(viii) Web Service for the Microsoft Kinect Sensor - Responsible for handling data from the Kinect sensors (RGB and infrared cameras, accelerometer and microphone) using the library OpenKinect [5]. Data from the Microsoft Xbox Kinect color (640 × 480, 32 bit) and depth (320 × 240, 16 bit) cameras are collected at a rate of 30 fps. In SOM4R, such data

can be used for face detection, obstacle avoidance and navigation, and can also be used for mapping the robot environment.

(ix) Battery Charge Level Web Service - Responsible for reading the battery charge level directly from the robot's hardware or through APIs, providing the result to other services and applications (e.g. recharge application).

In Section 3.1 we provide additional information about these web services and report experimental results using them.

Control Structure: Lessons from Subsumption Architecture The subsumption architecture was introduced by Brooks [12, 13], aiming at implementing a general, high-level control architecture for autonomous robots. Based on concepts from the subsumption architecture, the middleware web services support methods of suppression and inhibition of behaviors. That is, it is possible to suppress the entry of a service for a short period of time (50-200 ms), within which this service only accepts commands sent by the client that suppressed it, ignoring all the others clients' commands. It is also possible to inhibit the output of a service for a short period of time or temporarily keep the input or output values of these services unchanged.

Such an approach enables the implementation of a flexible control architecture, adapting the behavior of services and applications to the dynamic context of the robot environment. For instance, when a robot has a service or application (e.g. Runaway-App) running to avoid obstacles, but it has another local or remote application (e.g. Push-App) whose mission is to push an object, such application may act in two ways to ensure the movement of this robot in the direction of the object and push it to achieve your goal: i) inhibition of the output of the Runaway-App that avoids obstacles, or ii) suppression of the input of the vehicle web service. However, the first option (i.e. inhibition of the output of the Runaway-App) does not prevent other applications or services from sending movement commands. Whereas, by suppressing the input of the web service of the vehicle, only the movement commands of the Push-App will be considered.

Support for Network and Swarm Robotics The web-oriented architecture of the SOM4R enables integration between software processes that are embedded in robotic hardware (firmware) and those that operate on computers in a network. This is of particular interest for robots with limited computational resources (e.g. microcontrollers) because it allows them to benefit from powerful resources and services of cloud computing [24, 25].

The SOM4R allows robots to communicate along the network aiming at the formation of swarms (swarm robotics). For example, let us assume a swarm of quad-rotors[3] controlled by firmware embedded in their microcontrollers which support Ethernet and Wi-Fi technologies (e.g. Microchip PIC18F97J60 with MRF24WB Wi-Fi module, Arduino Nano and a Wi-Fi module). Using the web services of the proposed middleware, they can directly connect to each other securely in a peer-to-peer-based service model through the wireless network. This allows a given quad-rotor device to provide secure access to its resources over the network and consume the resources of other robots shared by SOM4R, allowing the swarm of quad-rotors to be remotely monitored and controlled by one or several software processes distributed along the network.

Integration with Existing Middlewares The service-oriented architecture of SOM4R allows easy integration with modules already developed by other middleware, such as ROS [37], YARP [30], CARMEN [32] and Player [23]. For example, in Fig. 5 it is shown an integration scenario between SOM4R and ROS using Rosbridge API [16]. In this case we have a web service (named Rosbridge-WS) (Fig. 5, center) that will translate from the RDF/XML message format, sent by SOM4R-WS (Fig. 5, left, bottom) to Rosbridge Protocol (based on JavaScript Object Notation - JSON). The Rosbridge-WS uses Rosbridge API Implementation (rosbridge_library and/or rosapi and/or rosbridge_server) for mounting the JSON string equivalent to RDF/XML message of SOM4R-WS message and sending commands to the nodes and services of ROS (Fig. 5, right, bottom) and vice versa.

2.2 Further Discussion

The need for establishing an appropriate set of evaluation criteria that serve as a common basis for comparison of robotic projects [14, 23, 26, 30, 32–34, 37, 38]

[3] Aircrafts driven by four motors.

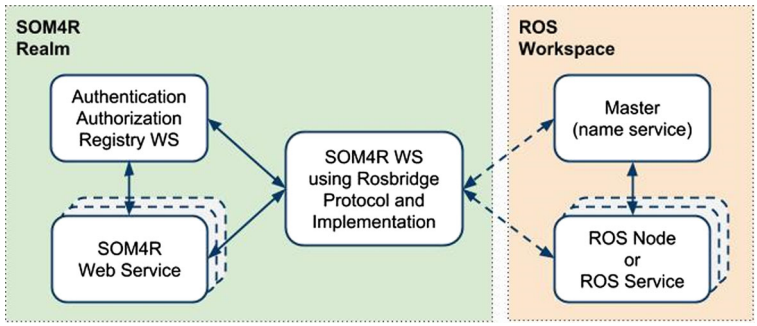

Fig. 5 SOM4R and ROS integration scenario. The SOM4R can communicate with ROS using Rosbrigde API, this allows the reuse of software (node or service) developed with the ROS API

is a common point of agreement among all previous works on this issue. More recently, some authors have been engaged in such endeavor [18, 19, 27, 31].

For instance, Kramer and Scheutz [27] proposed four criteria for comparing RDEs: specification, platform support, infrastructure and implementation. Mohamed et al. [31] proposed six criteria for evaluating the characteristics of middleware for networked robots: communication model (standard / nonstandard); interoperability flexibility; automatic discovery, configuration, and integration support; type of middleware services (specific/expandable); communication services; and embedded components and low-resources devices support. Sobh and Elkady [19] proposed nine criteria for evaluation: architecture, simulation environment, standards and technologies, support for a distributed environment, security for accessing modules, fault detection and recovery, real-time and behavior coordination capabilities, open-source and dynamic wiring. Elkady et al. [18] proposed a structured approach for modular design in robotics and automation environments, named RISCWare. They implemented a "greeting a person" application to evaluate the RISCWare framework with respect to applicability. For this purpose, a series of stress tests has been performed using different message sizes to measure the end-to-end data packet latency.

It is worth mentioning, however, that each set of evaluation criteria used to compare pros and cons of robotic projects is directly correlated with the application scenario of interest, hence a certain criterion may be relevant to a specific need or context [27]. We list in Table 1 a number of important attributes of existing middleware for robotics in order to compare their functionalities with the ones provided by SOM4R. Among the myriad of available attributes, we have chosen the following ones: technologies, architectures, protocols and layers of communications, the format used in the description of the message, supported operating systems and programming languages, HMI, security, and date of last update (until 2015).

Related to communication, the SOM4R project uses the client-server model based on the request-response paradigm [20], where the state of a resource is transferred only upon customer demand through the implementation of a web service server. The client process always has the initiative to establish communication with the server process to send or request the representational state transfer (REST) of the resource. This paradigm has the advantage of reducing the data traffic to a minimum explicitly requested to servers according to the need and capacity of processing of each client at run-time. Therefore, there is no need for the client to specify the frequency with which data should be sent by the server as in the case of Player (10Hz default), or configure synchronization and buffer mechanisms as can be the case with YARP, nor subscribe to threads posted on the server (topic-based publish-subscribe model) as can be the case with ROS.

Regarding the communication protocol, unlike the projects CLARAty, Player, CARMEN, YARP and ROS, the SOM4R employs the HTTP applications layer protocol for distributed and collaborative systems. We choose this stateless[4] protocol for the following reasons: a) it presents minor restrictions on firewalls which facilitates the interoperability between different network domains, b) it has a high compatibility with different platforms through native libraries available in several programming languages and operating systems, which facilitates the integration,

[4] A communications protocol that treats each request as an independent transaction that is unrelated to any previous request. It provides no means of storing a user's data between requests.

Table 1 Common features of open source projects to integrate robotic resources over net- work for comparison purposes with the SOM4R project

Features/ Projects	Player	CLARAty	CARMEN	YARP	ROS	SOM4R
Architecture/ Software design	Client-Server	Client-Server	Client-Server	Client-Server	SOA	ROA (REST)
Communication protocols	TCP (Socket)	TCP and UDP (Socket)	IPC (TCP/IP sockets)	TCP, UDP, Multicast and QNet	XML-RPC	HTTP
TCP/IP Layer	Transport	Transport	Transport	Transport	Application	Application
Message description format	Own protocol	Own protocol	Own protocol	Binary	IDL	RDF/XML
Operating system	Linux, Solaris and Windows	Linux, Solaris, MAC OSX and Cygwin	Linux	Windows, Linux, QNX 6 and MAC OSX	Linux and Windows (partial functions)	Linux, Windows and Android (partial functions)
Supported programming languages	C/C++, Java, Matlab, Python, Perl, Tcl/Tk	C++	C++, Java,	C++, Python, Matlab, Java, Tcl, Lisp, Ruby	C++, Python, Octave, Lisp	C/C++, C#, Python, PHP, Java Ruby, Flex/Flash
Human-Machine Interface	Client software	Client software	Client software	Client software	Client software	Web portal
Middleware security	None	None	None	None	Possible	Inherent
Last update	2015	2007	2008	2014	2015	2015

reducing significantly the complexity of inter-process communication, and c) it is available in many embedded devices (e.g. PIC microcontrollers with Ethernet module, Arduino Nano with ethernet module), following the current trend of employing Ethernet network in devices from various manufacturers.

In SOM4R, the messages exchanged between software processes are formatted in RDF/XML and describe the representational state (REST) of resources in a simple way, as shown in Fig. 6. The projects Player, CLARAty and CARMEN use proprietary message formats, while the YARP employs a binary format and ROS uses Interface Description Language[5] (IDL) to describe the content of messages exchanged between the software modules. However, in [25], it is shown that an image (RGBA, 842x595) that takes 18 kB in bandwidth if transported as PNG (lossless data compression), uses approximately 2MB when transported as a serialized ROS message. We opted for the description of resources using RDF by making data more portable and interoperable for different computers, operating systems and programming languages. It should be noted that in the evolutionary stage of web contexts, describing and representing information resources using RDF language is quite attractive and promising, allowing these resources to be linked together, integrated and reused [17].

The choice for the TCP/IP network is a general consensus among all the aforementioned projects, basically for the following reasons: a) TCP/IP stack is a collection of various types of communication protocols that work together to perform network communication, recommended both for small networks and for huge networks as the Internet, b) it is compatible with a wide variety of hardware, and c) is included in versions of major operating systems.

From a more technical viewpoint, we aimed at the transport layer to guide our choice for the TCP protocol, rather than some other protocol (e.g. User Datagram Protocol - UDP), for the following reasons:

[5] A specification language used to describe a software component's interface in a language-independent way, commonly used in RPC software.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
      "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
      <rdf:Description  rdf:ID="TTS">
            <textToSpeech  rdf:datatype="&xsd;string">
                  Hello World</textToSpeech>
            <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
      </rdf:Description>
</rdf:RDF>
```

```
<< TTS interface >>
+ id: integer
+ textToSpeech: string
```

Fig. 6 Abstract interface of the TTS Web Service. A description of the representational state of a TTS resource using RDF/XML syntax

i) higher reliability: there is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent, with error checking and error recovery, *ii*) full-duplex communication: it allows both parties to send and receive data within the context of the single TCP connection, and *iii*) rate adaptation: the rate of data transfer is intended to adapt to the prevailing load conditions within the network and consider the processing capacity of the receiver, attempting to achieve the highest possible data-transfer rate.

3 Experimental Results

Aiming at evaluating the performance of the proposed middleware, we present a number of experiments with the developed applications, built specifically for an autonomous wheeled mobile robot, by integrating voice command functionality, obstacle avoidance, greeting a person, battery recharge, and remote control of the vehicle movement. The wheeled robot shown in Fig. 7 was designed and built along the development of this research for the sake of experimental validation of the SOM4R. The rationale for building a specific wheeled mobile robot, instead of using a commercially available one, arose from the need to test the hypotheses behind the proposal of the SOM4R, especially hardware and software abstraction, for a completely new robotic hardware project. As proof of concept, some experiments were replicated using a Pioneer P3DX robot available in our laboratory for the sake of evaluating the portability of SOM4R to existing robotic platforms.

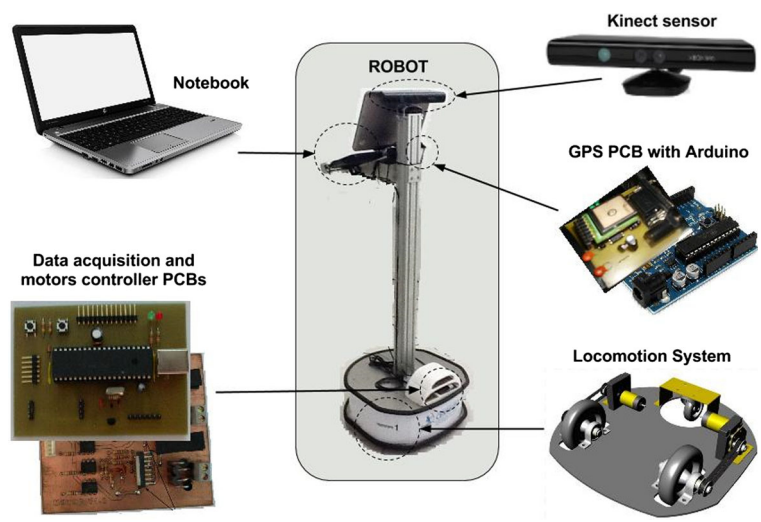

Fig. 7 A wheeled mobile robot built as a platform for research, development and experimental validation of the proposed middleware

The proposed middleware was installed on the personal computer of the aforementioned robot, namely, a notebook with Intel(R) Core2 Duo (TM) SU7300 1.3 GHz processor, 4GB DDR3 RAM, 320GB 7200rpm SATA hard drive, USB 2.0, LED 12.1" touchscreen, and connected to Wi-Fi (802.11g 54Mbps). The MS Kinect sensor, the Printed Circuit Board (PCB) that controls the robotic vehicle, and the Arduino board with integrated GPS sensor (via USB-Serial adapter) were connected to the USB ports on the notebook. All experimental tests were carried out using this notebook running Linux operating system (Ubuntu 11.04 32bit).

According to the request-response model of SOA where web services run only on demand, the consumption of CPU resources was minimal, as expected. During the experimental tests using the considered notebook, consumption was around 35 % when all services were running. When the user applications were running, it demanded a greater consumption of CPU resources because they were running with certain frequency, which increased the average CPU consumption to 55 %, fully justifying the innovations proposed by SOM4R.

A human machine interface was developed in order to be simultaneously accessed by multiple computers and tablets, including smartphones running Android (Google) and iOS (Apple) operating systems, and a solution was found to be adherent to the needs of a friendly user interface. The entire SOM4R open source project are available at www.som4r.net with documentation at github.com/som4r/som4r.

3.1 Applications

According to SOM4R's design philosophy, a single application can monitor and control the behavior of a set of robots. Furthermore, different aspects of the robot's behavior can be controlled by multiple applications running directly on the computer of this robot or on other computers and mobile devices distributed across the network (see Fig. 8). In HTTP Digest authentication method, Realm is a directive that defines the authentication territory, typically a description of the system or computer being accessed. The use of realms allows applications and web services, both local and remote, be partitioned into a set of protection spaces (e.g. `quadrotors@lab1.som4r.net`, `mobile_robots@lab2.som4r.net`).

The SOM4R allows the integration between applications and web services running on various robots

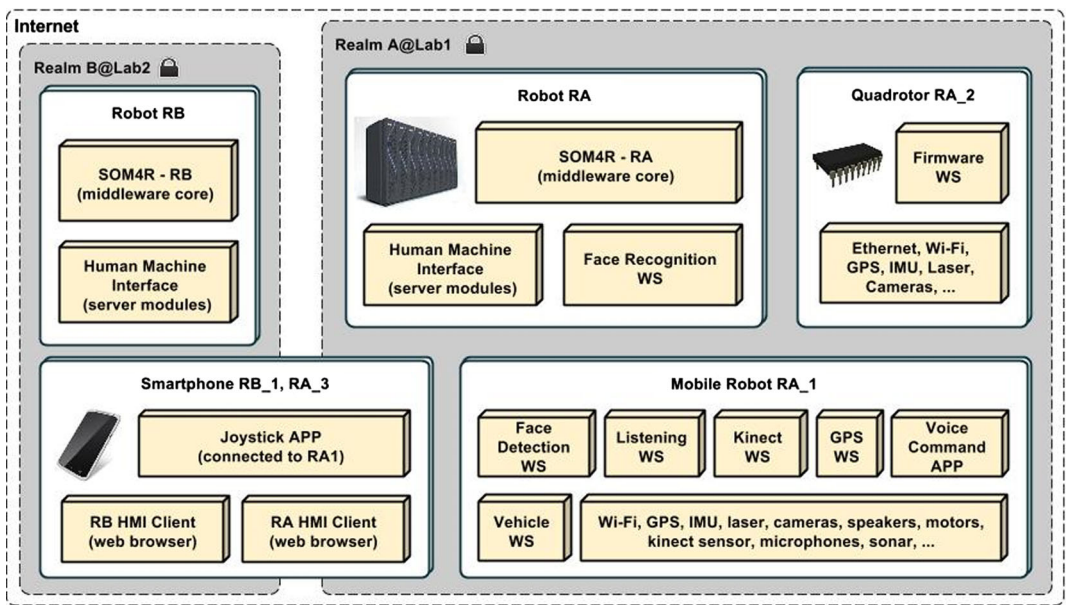

Fig. 8 Integration scenario between applications and web services (*yellow*) running on various robots (*white*) authenticated in different realms (*gray*) of SOM4R via internet, integrating microcontrollers (RA_2), mobile devices (RB_1) and supercomputers (RA)

and authenticated in different realms through the computer network. In Fig. 8, the five robots (in white) are distributed into two realms (dark gray) interconnected through the internet (light gray). There is no restriction to the physical location of the robots. Applications and services (in yellow) that are running on these robots can take part in one or more realms, interacting with several robots at the same time.

The applications and web services, interconnected through the authentication "Realm" of SOM4R, are developed using the native HTTP library, available in several programming languages on different operating system platforms (e.g. Linux, Android, MS Windows, OS X and iOS). This facilitates the choice of the most appropriate environment (software and hardware) to implement each robotic task. We describe in the following paragraphs three general-purpose applications that are useful for several robotic tasks.

Joystick Application It is responsible for controlling the movement of the robot using motion sensors (e.g. accelerometers) of a mobile device, such as smartphones or tablets, equipped with the Android operating system from version 2.2 or greater. The readings of accelerometers (Fig. 9, step 1) are transformed into commands (Fig. 9, step 2) and sent over the wireless network (Wi-Fi, 3G) to the robot vehicle web service (Fig. 9, step 3). The vehicle web service executes the command (Fig. 9, step 4), posts its action to STM web service (Fig. 9, step 5) and returns the status of the vehicle. The joystick application posts its action to STM web service (Fig. 9, step 6) and waits a few milliseconds (Fig. 9, step 7) before restarting its cycle.

The performance tests of the application were executed on a smartphone, with single-core processor of 600 MHz and 170MB of internal memory. Four snapshots taken along the execution of the task are illustrated in Fig. 10. The average response time of the application connected to the middleware via a protected Wi-Fi (WPA2 with TKIP+AES, up to 54 Mbps) was about 300 ms, which can be considered very acceptable for remote control applications of robotic vehicles. Additional tests of access to the robot were performed with this smartphone using the 3G cell phone network. In this case, due to the low speed of these networks (measured at about 512Kbps) in Brazil, the response time had a delay of up to seven seconds, compromising remote control of the movement of the vehicle. That is, a reduction in the speed of the wireless network of the order of 108:1 led to an increment on the response time of the order of 1:23. However, the current evolution scenario of the power of parallel processing of mobile devices, of transmission speed in broadband networks (e.g.

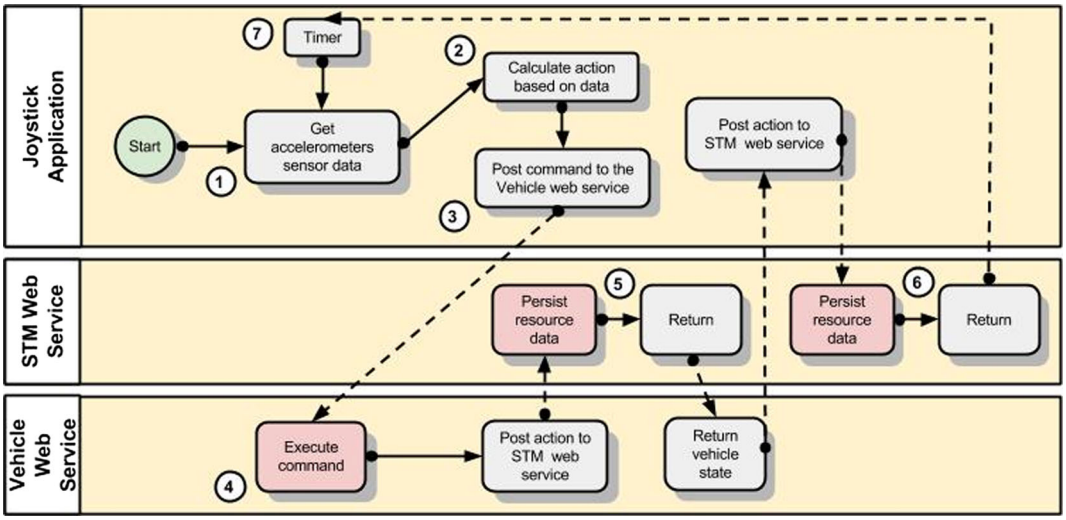

Fig. 9 BPMN diagram representing the communication process between web services for the joystick application

Fig. 10 Snapshots taken along the execution of the joystick application tests (full video available at youtu.be/G2iMuNAkWkE)

WiMAX, 4G) and of Wi-Fi technologies [22] (e.g. IEEE802.11ac, IEEE802.11ad), will certainly lead to better performance results.

For the sake of completeness, this application was replicated on a Pioneer P3DX-SH robot (snapshots are shown in Fig. 11). For this purpose, we developed a new Vehicle Web Service for this robot using the APIs of ARIA[6] (Advanced Robotics Interface for Applications). The response time of the joystick application was about the same of the previous test, what indicates that the replacement of Vehicle Web Service has minimum impact in the other modules of the SOM4R.

Runaway Application It is responsible for the obstacle avoidance function. This application uses the vehicle and Kinect web services. In Step 1 of Fig. 12, it constantly monitors the obstacles detected by the web service for the Kinect sensor and considers that each neighboring point has a "repulsive force" proportional to the inverse of the squared distance (Fig. 12, steps 1 and 2). We followed Brooks' implementation [12] for developing our runaway application. When the magnitude of this "resultant force" exceeds a certain empirically defined threshold (Fig. 12, step 3), the application acts by suppressing the input to the vehicle web service. If the robot is moving, the STOP command is sent to the robot (Fig. 12, step 4). Afterwards, the commands ROTATE and MOVE TOWARDS (the direction of the resultant vector) are executed in order to avoid the obstacle (Fig. 12, step 5), recording the actions taken and starting the cycle of obtacle monitoring.

Two types of experiments were performed with the Runaway application. For the first one, the robot's movement is being controlled by the joystick application. During the motion, when a "repulsive force" indicates the presence of a nearby object, the Runaway-app takes control by suppressing the input to the vehicle web service, stopping and positioning the robot in a posture where it is capable of deviating from the obstacle. Then, the vehicle web service

[6] Available at http://robots.mobilerobots.com/wiki/ARIA

Fig. 11 Snapshots taken along the execution of the Joystick application tests on Pioneer P3DX-SH robot (full video available at youtu.be/bSoOqbzGmYQ)

returns to a mode in which it accepts again commands from the Joystick-app. For the second experiment, the Runaway-app is configured with a different behavior. The robot's motion is being controlled by the web module via the HMI. When it detects the presence of a nearby object, the Runaway-app reads the current robotic vehicle status (speed and direction) before it takes control by suppressing the input to the vehicle web service, stopping the robot and appropriately positioning it in order to deviate from the obstacle. After that, the Runaway-app sends to the vehicle web service the motion command read prior to subsumption. As a result, when the robot is sent towards a wall, it adopts the wall-following navigation mode.

In Fig. 13, we show the boxplot of the time required for reading and processing of obstacle detection data received from Kinect sensor web service. We used 7,000 measurements collected within a 16-minute interval while conducting the tests. The minimum time was 27 ms and the maximum was 509 ms, the first quartile was 61 ms, the median was 83 ms, the third quartile was 134 ms and the interquartile range was 73 ms. The maximum time was recorded when the use of two cores processors was almost at 100 percent, being regarded as an outlier. The wheeled robot was able to avoid obstacles during the navigation (speed around 0.2 m/s), as shown in the video for this performance test in youtu.be/TMPayPo31qU.

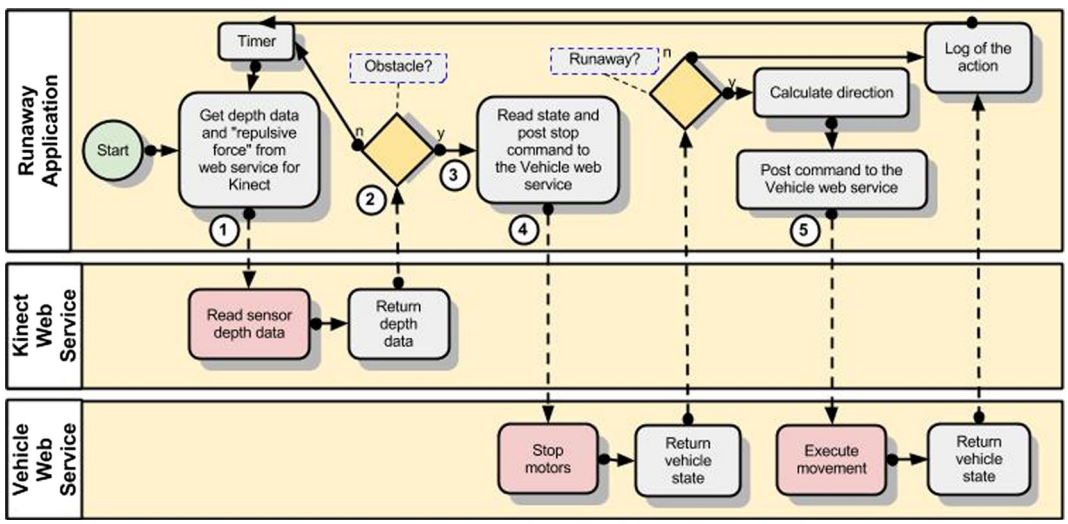

Fig. 12 BPMN diagram representing the communication process between web services for the Runaway application

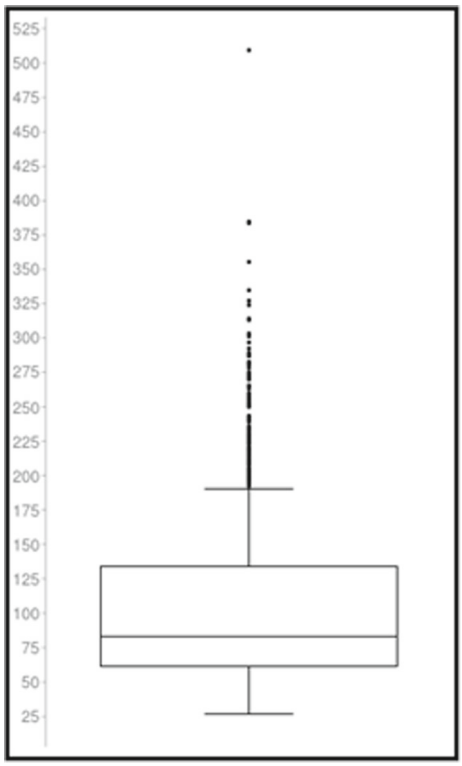

Fig. 13 Boxplot of the time (in ms) for reading and processing obstacle detection data received from Kinect sensor web service

Voice Command Application It is responsible for activating, running and disabling the voice command functions. The BPMN diagram of the communication links required by the application is shown in Fig. 14. It monitors the web service for the recognition of human voice as indicated by step 1. When the response of the web service is a word or statement previously defined as a command, as in step 2, the application selects the action to be performed. If the command is to turn on or turn off the voice command, it sends a statement about the current state of the voice command feature (enabled or disabled) using the web service for speech synthesis (TTS, text-to-speech), which emits the sound of the word/statement via the computer's sound synthesizer as step 3. If the command is for moving the robotic vehicle, initially defined as `right`, `left`, `ahead`, `back`, `faster`, `slower` and `stop`, the corresponding action to this command is performed (step 4). Finally, the commands received and events they triggered are registered by the application using the STM web service (step 5) and then the monitoring cycle is reinitiated.

Performance tests have shown a response time below 500 ms, a value that we considered satisfactory within the context defined for this project (surveillance, remote monitoring and control, autonomous and semi-autonomous navigation). In the tests we used the CMU Sphinx library [2], which achieved considerable high false negative rates (i.e. no recognition of spoken words). This means that the speaker had to repeat the verbal command several times for the voice command application to recognize the word correctly.

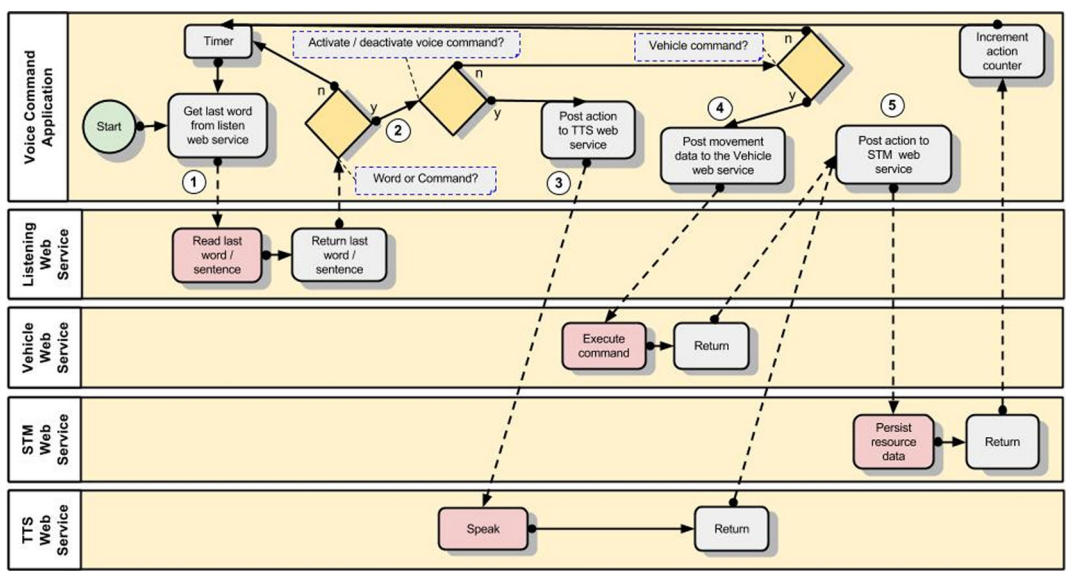

Fig. 14 BPMN diagram representing the communication process between web services for the Voice Command application

However, this has occurred because the experiments were conducted with the speaker located at different positions from the robot microphone and in a room with high background noise. Thus, we do not consider it a poor result, but rather a common situation to be faced in real-world scenarios.

The reported experiments also achieved a small false positive rate (recognition of the spoken command as another phonetically similar word). This result led us to search for a set of words that could represent well the desired commands (e.g. forward or ahead? Stop or brake?) aiming at reducing this figure of merit even further. The full video for this performance test can be found in youtu.be/TMPayPo31qU.

4 Conclusions and Future Work

In this paper we proposed a middleware with the goal of accessing robot resources from different computing devices over the local network or Internet. The software modules (services and applications) can be implemented in different languages, according to the performance needs of each user. In this work, for example, several programming languages, such as Python, PHP, Java, Javascript, Flash/Flex and C/C++, were used, giving rise to a more flexible, portable and modular implementation of new web services and applications, significantly reducing the restrictions for the collaboratively evolution of middleware. The modular nature of the proposed middleware architecture makes the inclusion of new robotic resources very simple and has proved to be very useful in other robotic applications in our laboratories.

The adopted security methodology opens up the possibility to integrate robotic systems through the Internet, because the use of HTTP Digest Authentication method combined with the HTTPS secure protocol increased the middleware's security access and reduced the weakness of the HTTP Digest about the inability of client to confirm the identity of the server. This improvement in security was also a result of the use of tokens, that allows control practically in real-time of cross access permissions between services and applications authenticated by the middleware. The adoption of the HTTP protocol and XML, which have fewer restrictions on firewalls, has facilitated the interoperability between different network domains. The middleware was developed using the Linux operating system, with some applications to the Android and Microsoft Windows operating systems. The use of web browsers as the interface for human-robot interaction is an innovative aspect of the proposed middleware that allows local and remote access (fixed and mobile) in a concurrently and safe way, reducing the need for software installation on the client and increases HMI portability between different operating systems, computers and mobile devices. The robot control structure was based on Brook's subsumption architecture, adapting the behavior of applications and services to the dynamic context of the robot's environment. As an arbitration method, the subsumption architecture turned out to be very efficient for resolving conflicting commands, as illustrated in Section 2.

Currently, we are developing a rosbridge [15] support for SOM4R. This will create a common interface for robots running ROS and SOM4R to send messages to each other, and for application developers to write software that can support robots running any operating system. We are also extending the features of the SOM4R in order to use it as the middleware for managing the energy resources of smart homes and buildings, in consonance with the Internet of Things (IoT) concept.

Acknowledgments The authors would like to thank the financial support of NUTEC (Núcleo de Tecnologia Industrial do Ceará) and CNPq (grant no. 309451/2015-9).

References

1. Business process model and notation (BPMN). http://www.omg.org/spec/BPMN/index.htm, a. Accessed: 02-03-2012
2. CMUSPhinx wiki. http://cmusphinx.sourceforge.net/wiki/, b. Accessed: 02-03-2012
3. eSpeak Text to speech. http://espeak.sourceforge.net, c. Accessed: 02-03-2012
4. The OAuth 2.0 authorization protocol v2-23. http://tools.ietf.org/html/draft-ietf-oauth-v2, d. Accessed: 03-02-2012
5. OpenKinect Project. http://openkinect.org/wiki/main_page, e. Accessed: 02-03-2012
6. Pyfaces: Face recognition system. https://code.google.com/p/pyfaces/, f. Accessed: 08-27-2011
7. Fielding, R.T.: Ph.D. thesis. http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm, g. Accessed: 11-10-2011
8. Belhumeur, P.N., Hespanha, J.P., Kriegman, D.: Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. IEEE Trans. Pattern Anal. Mach. Intell. **19**(7), 711–720 (1997)

9. Billinghurst, M., Cheok, A., Prince, S., Kato, H.: Real world teleconferencing. IEEE Comput. Graph. Appl. **22**(6), 11–13 (2002)
10. Billinghurst, M., Kato, H.: Collaborative augmented reality. Commun. ACM **45**(7), 64–70 (2002)
11. Bradski, G., Kaehler, A.: Learning OpenCV. O'Reilly Media, Inc (2008)
12. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(1), 14–23 (1986)
13. Brooks, R.A.: A robot that walks - emergent behaviors from a carefully evolved network. Neural Comput. **2**, 692–696 (1989)
14. Bruyninckx, H.: Open robot control software: the OROCOS project. In: Proceedings of the International Conference on Robotics and Automation (ICRA'2001), vol. 3, pp. 2523–2528 (2001)
15. Crick, C., Jay, G., Osentosiki, S., Pitzer, B., Jenkins, O.C.: Rosbridge: ROS for non-ROS users. In: Proceedings of the 15Th International Symposium on Robotics Research (ISRR'2011), pp. 1–12 (2011). www.isrr-2011.org/ISRR-2011//Program_files/Papers/Jenkins-ISRR-2011.pdf
16. Crick, C., Jay, G., Osentoski, S., Jenkins, O.: Ros and rosbridge: roboticists out of the loop. In: ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 493–494 (2012)
17. Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I.: The semantic web: the roles of XML and RDF. IEEE Internet Comput. **4**(5), 63–74 (2000)
18. Elkady, A., Joy, J., Sobh, T., Valavanis, K.: A structured approach for modular design in robotics and automation environments. J. Intell. Robot. Syst. **72**(1), 5–19 (2013)
19. Elkady, A., Sobh, T.: Robotics middleware: a comprehensive literature survey and attribute-based bibliography. Journal of Robotics **2012**(ID-959013), 1–15 (2012)
20. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. In: Proceedings of the International Conference on Software Engineering (ICSE'2000), pp. 407–416 (2000)
21. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Stewart, L.: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication. Technical Report, The Internet Engineering Task Force (1999). www.faqs.org/rfcs/rfc2617.html
22. Garber, L.: Wi- races into a faster future. Computer **45**(3), 13–16 (2012)
23. Gerkey, B.P., Vaughan, R.T., Stoy, K., Howard, A., Sukhatme, G.S., Mataric, M.J.: Most valuable player: a robot device server for distributed control. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2001), vol. 3, pp. 1226–1231 (2001)
24. Hu, G., Tay, W.P., Wen, Y.: Cloud robotics: architecture, challenges and applications. IEEE Netw. **26**(3), 21–28 (2012)
25. Hunziker, D., Gajamohan, M., Waibel, M., D'Andrea, R.: Rapyuta: the RoboEarth cloud engine. In: Proceedings of the International Conference on Robotics and Automation (ICRA'2013), pp. 438–444 (2013)
26. Jackson, J.: Microsoft robotics studio: a technical introduction. IEEE Robot. Autom. Mag. **14**(4), 82–87 (2007)
27. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: a survey. Auton. Robot. **22**(2), 101–132 (2007)
28. Lienhart, R., Kuranov, A., Pisarevsky, V.: Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In: Michaelis, B., Krell, G. (eds.) Pattern Recognition - Proceedings of the 25th DAGM Symposium. Springer (2003)
29. Meng, J., Mei, S., Yan, Z.: RESTful web services: a solution for distributed data integration. In: Proceedings of the IEEE International Conference on Computational Intelligence and Software Engineering (CiSE'2009), vol. 3, pp. 1–4 (2009)
30. Metta, G., Fitzpatrick, P., Natale, L.: YARP: yet another robot platform. Int. J. Adv. Robot. Syst. **3**(1), 43–48 (2006)
31. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: A review of middleware for networked robots. International Journal of Computer Science & Network Security **9**(5), 139–148 (2009)
32. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (CARMEN) toolkit. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2003), vol. 3, pp. 2436–2441 (2003). doi:10.1109/IROS.2003.1249235
33. Nesnas, I., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau, J., McHenry, M., Shu, I.-H., Apfelbaum, D.: CLARAty: challenges and steps toward reusable robotic software. Int. J. Adv. Robot. Syst. **3**(1), 23–30 (2006)
34. Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T.: CLARAty and challenges of developing interoperable robotic software. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2003), vol. 3, pp. 2428–2435 (2003)
35. Overdick, H.: The resource-oriented architecture. In: Proceedings of the 2007 IEEE Congress on Services, pp. 340–347 (2009)
36. Peng, D., Li, C., Huo, H.: An extended username: token-based approach for REST-style web service security authentication. In: Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT'2009), pp. 582–586 (2009)
37. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Proceedings of the Open-Source Software Workshop of the International Conference on Robotics and Automation (ICRA'2009) (2009)
38. Volpe, R., Nesnas, I.A.D., Estlin, T., Mutz, D., Petras, R., Das, H.: The CLARAty architecture for robotic autonomy. In: Proceedings of the IEEE Aerospace Conference, vol. 1, pp. 1121–1132 (2001)

Marcus V. D. Veloso was born in Fortaleza, Ceará, Brazil, in 1965. He received the B.S. (2002) and M.Sc. (2005) degrees in Physics from the Federal University of Ceará (UFC), located at the northeast coast of Brazil. In 2014, he got a Ph.D. degree in Teleinformatics Engineering from the same university. His current research interests are computational intelligence, pattern recognition, middleware for robotics and distributed systems for robotics and automation.

José Tarcísio C. Filho was born in Crato, Ceará, Brazil, in 1959. He received the B.S. degree in Electrical Engineering from the Federal University of Ceará (UFC) in 1985, and the M.Sc. and Ph.D. degrees in Electrical Engineering from the University of Campinas in 1988 and 1992, respectively. His research interests are robust Kalman filtering, game theory and applications, robotics and automation.

Guilherme A. Barreto was born in Fortaleza, Ceará, Brazil, in 1973. He received his B.S. degree in Electrical Engineering from the Federal University of Ceará in 1995, and both the M.Sc. and Ph.D. degrees in Electrical Engineering from the University of São Paulo in 1998 and 2003, respectively. Currently, he is associate professor of the Department of Teleinformatics Engineering, Federal University of Ceará (UFC), Fortaleza, Ceará, Brazil. At this institution, Prof. Guilherme Barreto leads the Group of Advanced Machine Learning (GRAMA), whose members pursue a variety of research topics, such as neural networks & computational intelligence, pattern recognition & machine learning, nonlinear system identification, time series prediction, and intelligent robotics. More recently, members of GRAMA have been collaborating extensively with outstanding research groups in Portugal (FEUP), Spain (Granada), Germany (Bielefeld), Finland (Aalto) and England (Sheffield). Prof. Barreto has been serving as reviewer for several international journals and conferences. He is also serving as the editor-in-chief of the journal Learning & Nonlinear Models (L&NLM) published by the Brazilian Computational Intelligence Society (SBIC) and as an associate editor of the Journal of Machine Learning and Cybernetics (Springer), International Journal of Innovative Computing and Applications (Inderscience), and Frontiers in Bioengineering and Biotechnology. He is the president of the SBIC for the period 2015–2017.