A RESTful Architecture for the Development and Deployment of Companion Robots Applications

By

Razieh Safaripour

A Thesis

In

The Department of Electrical and Computer Engineering

Presented in Partial Fulfilment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

August, 2013

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Razieh Safaripour

Entitled: "A RESTful Architecture for the Development and Deployment of Companion Robots Applications"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. M. Z. Kabir

_____ Examiner, External
Dr. D. Goswami (CSE) To the Program

_____ Examiner
Dr. D. Qiu

_____ Supervisor
Dr. F. Khendek

_____ Supervisor
Dr. R. Glitho

Approved by: _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____20_____ _____
Dr. C. W. Trueman
Interim Dean, Faculty of Engineering and Computer Science

ABSTRACT

A RESTful Architecture for the Development and Deployment of Companion Robots Applications

Razieh Safaripour

Improving the quality of life particularly for the elderly and disabled persons is essential for society today. Despite their existing disabilities and limitations the elderly and the disabled still need and desire to be an integral part of society.

Statistics shows that the number of persons requiring home health care in the year 2040 will make up nearly 3.5 % of the population. The need for companion robots is growing with factors such as an aging population, limited infrastructure and social support. Robots capable of assisting people in daily tasks and providing various services represent part of the future solution.

The lack of a reusable platform is a significant obstacle to such a solution. In this thesis, we are proposing an architecture that will enable the development and deployment of companion robots applications. The architecture consists of a set of components at different layers, ranging from low-level robotics services to end user application components. The multiple layers interact through RESTful web services. This architecture enables the development of a range of applications, and can deal with robots that have varying capabilities and hardware.

Acknowledgments

It would not have been possible to write this master thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Foremost, I would like to express my appreciation to my supervisor Dr. Ferhat Khendek. This thesis would not have been possible without his help, support and patience. His useful comments and remarks especially during the writing process is something for that I will always be truly grateful to him. I also would like to give my sincere gratitude to my co-supervisor and professor, Dr.Roch Glitho, whose constructive comments, patience, insightful guidance and immense knowledge lightened my way through the learning process of this master thesis.

I am most grateful to Dr.Fatna Belqasmi for her continuing support and help, understanding and patience, great advices and guidance on the way, which without her help, fulfilling this thesis would have been almost impossible. I truly appreciate her valuable suggestions and ideas which made it an absolute pleasure working with her.

My sincere thanks go to my team fellow and friend, Majid Hormati, for being such a wonderful person, whose help, suggestions and ideas were so important to accomplish this thesis. His friendship during the hard times is what I am most delighted with.

My thanks and appreciations go to my supervisory committee members Dr. M.Zahangir Kabir, Dr. Dhrubajyoti Goswami and Dr. Dongyu Qiu for taking time to assess my thesis and for their helpful suggestions.

I would also like to thank all my friends in Florida and Montreal for their friendship which supported me through these years. My special thanks to Karen Assyag for her praiseworthy help in editing this thesis.

At last but not at least, I wish to thank my family specially my parents for their pure everlasting love and support which encouraged me to keep my hope and strengthened me to fulfill this thesis.

Thanks Mom, Thanks Dad for being such amazing and lovely parents! Thanks for being there!

Table of Contents

List of Figures

List of Tables

List of Abbreviations

REST	Representational State Transfer
SOAP	Simple Object Access Protocol
CORBA	Common Object Request Broker Architecture
XML	eXtensible Markup Language
XHTML	eXtensible HyperText Markup Language
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
WSDL	Web Services Description Language
WADL	Web Application Description Language
URI	Uniform Resource Identifier
FTP	File Transfer Protocol
WAIS	Wide Area Information Servers
API	Application Programming Interface
ROA	Resource Oriented Architecture
PDA	Personal Digital Assistant
IFR	International Federation of Robotics
MRDS	Microsoft Robotics Developer Studio
UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
UUV	Unmanned Undersea Vehicle

SOA	Service Oriented Architecture
XDR	eXternal Data Representation
TCP	Transmission Control Protocol
IIOP	Internet Inter-ORB Protocol
IETF	Internet Engineering Task Force
UML	Unified Modeling Language
RPC	Remote Procedure Call
DSS	Decentralized Software Service
CCR	Concurrency and Coordination Runtime
DLL	Dynamic Link Library
ER	End user – Robot
AC	Application – Camera
AD	Application – Drive
AS	Application – Sensor
ACD	Application – ColorDetection
CDC	ColorDetection – Camera

Chapter 1:
Introduction

Chapter one contains an introduction of the research domain, and proceeds to discuss motivation scenarios and the problem statement. The chapter concludes with the thesis contribution and the organization of the thesis.

1.1 Research Domain

Today, the need for companion robots is prevalent more than any time before. According to a study conducted in the USA, the American government can save three billion dollars a year if American senior citizens continue living in their own home only an additional three more months prior to moving into a senior residence [1]. The cost and the growing elderly and disabled population make it a necessity to come up with a technological solution to encourage, enable, and extend independent living.

Companion Robots, are targeted technology that aid people live self-sufficiently and longer in their own homes.

Figure 1-1 : Human-Robot Interaction

Due to the different needs, various companion robot applications are being developed. Some robots are used as guides and walking assistant to people in different places in their home, whereas other robots are programmed to do household chores such as vacuuming, getting things, etc.[1]. Robots can be manipulated locally or remotely through devices such as Personal Digital Assistant (PDA), smart phones, etc. (see Figure 1-1).

There are different works on designing robotics framework in industrial domain (e.g. [2]) as well as mobile robots (e.g. [3] and [4]). A majority of the existing works are proprietary solutions or require a deep knowledge of robot hardware from the application developers (e.g. [5] and [1]), however, there has been some efforts for robot applications using standards such as [6], [7] and [8] which employed Common Object Request Broker Architecture (CORBA) [9], Simple Object Access Protocol (SOAP) [10] and Representational State Transfer (REST) [11], respectively.

1.2 Problem Statement and Contributions

The wide variety of companion robots applications and robots technologies has raised the issue of reusing and extending existing systems effective for individual projects. Future elderly and disabled persons will require new and more complicated applications. It is very important that companion robots applications developers be able to integrate various software and robot systems to realize more sophisticated applications.

Below are two examples of people with different needs:

Sam is an elderly gentleman that needs to be reminded of his daily tasks such as taking medications, going to an event, etc. He also uses a walker so he needs his robot to carry things for him. For example, Sam puts his food dish on his robot's tray and the robot

follows him from the kitchen to the living room where he eats. The robot also has a storage compartment to store Sam's medications; this allows Sam's medications to be easily accessible to him. When it is time for Sam to take his medication, the robot reminds Sam to do so. The computer screen not only tells Sam what medication he is required to take but also shows him the dosage and instruction for said medication (Figure 1-2).

Figure 1-2: Motivating Scenario: Sam (taken from [63]**)**

Figure 1-3: Motivating Scenario: Lisa
(taken from [64]**)**

The second example is that of Lisa, Lisa is disabled and needs 24/7 help at home. Her robot has special capabilities (and unlike Sam's robot is equipped with human-like arms) to do her housework and help her complete her personal tasks. While she is out, using her smart phone, Lisa, can order her robot to clean the house and vacuum the floor (Figure 1-3). The robot also has the ability to hand her different objects, like the TV remote or books. Lisa has a respiratory problem, in an emergency situation, her robot can dial emergency personnel, and can alert them that Lisa is in danger and needs help.

Sam and Lisa both have different situations with different needs; both require the use of different robots with different capabilities. Taking into account the differences in both situations, and knowing there are various ailments, conditions and demands in this population, it is necessary to have different robots built for different purposes which poses a number of challenges.

The first challenge is to be responsive to different needs, so different applications are needed. The second challenge, is the robots heterogeneity, robots are manufactured from different companies (and therefore may have different firmware and protocols of communications).

The third challenge is resource-constrained devices which are devices with limited power and computation resource such as PDAs and Smart phones. These devices are used to complete daily tasks, manoeuvring the robots is one of the tasks needed to be completed by such devices.

There are several works in robotics that attempt to provide aid for the elderly and the disabled, however, many of these works propose proprietary methods that are not adaptable to different situations and are applicable only to specific robot or application. Standards solutions are being considered to develop companion robots applications. CORBA is a middleware that provides a platform- and language- independent environment which is suitable for distributed systems development; albeit, it has considerable limitations such as complexity, especially its Application Programming Interface (API)s (e.g. large and heavy-weight APIs), poorly designed APIs for naming, trading, and notification services, and compatibility [12]. SOAP is another standard that may be used to design development environment for companion robots applications. Despite its features such as platform- and language independency, it is not the best choice when light-weight and easy-to-use characteristics matter. Verbose eXtensible Markup Language (XML) [13] format and large-sized envelops are a few drawbacks that discourage using SOAP, these drawbacks will be discussed in details in Chapter 3. REST, nevertheless, due to its uniform, stateless and light-weight features is an appropriate solution to address different needs and will be looked at it in more detail in Chapter 2.

Besides addressing the demands of different applications using different robots, it is essential to consider future demands and easy modification to applications and robot

hardware. It is also necessary to provide companion robots applications developers with a flexible and maintainable platform that will facilitates the modification or extension of part of the system, without affecting the other parts.

Enabling rapid development and deployment of companion robots applications is still an open challenge and this thesis offers new contributions to deal with these challenges. The emphasis of this thesis is not only on designing a novel architecture that will provide a platform for the developers to enable and support different robots and applications but to facilitate development and deployment, hence, proposing a layered architecture. The proposed architecture enables developing applications through easy-to-use and unified interfaces. These interfaces are designed as RESTful Web services. Using RESTful Web services, the developers are further provided with light-weight interfaces that allows applications development for resource-constrained devices.

The main contributions of this thesis are as follow:

- A set of requirements for an architecture for the development and deployment of companion robots applications, categorized as *General* and *Interface specific* requirements.
- A novel architecture to enable development and deployment of companion robots applications: We propose a RESTful architecture that consists of a set of components at different layers. Layers ranging from low-level services to communicate with hardware devices, to application components for interacting with end user.

- Implementation of a prototype for a specific application: As a proof of concept we implement an application for which we use a LEGO robot and a webcam. Microsoft Robotics Studio Developer (MRDS) [14] is the tool we use for the development of our application. Performance evaluation is provided using CommView [15] network tool.

1.3 Thesis Organization

The thesis is organized as following: Chapter 2 presents the background information on robots and their applications particularly companion robots applications. In this chapter we also provide a background on REST and RESTful Web services which is crucial to understanding the rest of the thesis.

Chapter 3 discusses the derived requirements to design an architecture for the development and deployment of companion robots applications. It also provides a detailed review of the state of the art on companion robots applications and the solutions.

Chapter 4 proposes a RESTful architecture to enable companion robots applications developers to develop and deploy different application for various kinds of robots. The REST interfaces including resources modeling and HyperText Transfer Protocol (HTTP) actions on each resource, is also presented.

Chapter 5 is devoted to the implementation of our prototype for a specific application. We explain the environment of the application. We describe MRDS which we used for the implementation of the prototype. In Conclusion, this chapter presents and analyses our performance results.

Chapter 6 concludes the thesis with open issues that can be investigated in the future.

Chapter 2:
Background on Robots, Robot's Applications and RESTful Web Services

In this chapter we have an introduction to robots and robot's applications. We further discuss some applications necessary in the elderly and disabled population. We introduce REST architectural style and review REST definition, main characteristics, principles and advantages over SOAP-based Web services.

2.1 Robots

2.1.1 Introduction

According to [16] "a robot is a system that contains sensors, control systems, manipulators, power supplies and software all working together to perform a task or a set of tasks". They range from humanoids such as Advanced Step in Innovative MObility ASIMO [17] to nano robots [18] and industrial robots [19]. By imitating a life-like appearance or automating movements a robot may express a sense of intelligence of its own.

Robots are categorized in two main groups: Industrial Robots and Mobile robots. An industrial robot is defined by International Organization for Standardization (ISO) [20] as an "automatically controlled, reprogrammable, multipurpose manipulator that is programmable in three or more axes" [20]. In contrary to industrial robots there is no standard definition for mobile robots, however a well adopted definition is that a mobile

robot is a platform with a large mobility within its environment [21] [22]. It is a system with three functional characteristics: *mobility, a certain level of autonomy* and *perception ability* i.e. sensing and reacting in the environment [21].

Mobile robots can be categorized based on the environment they move [23] [23]:

- ✓ Unmanned Ground Vehicles (UGVs)
- ✓ Unmanned Aerial Vehicles (UAVs)
- ✓ Unmanned Undersea Vehicles (UUVs)
- ✓ Unmanned Surface Vehicles (USVs)

Ground robots are further classified based on their locomotion system [24]:

- ✓ Wheeled robots
- ✓ Tracked robots
- ✓ Legged robots

Robots come in all shapes and sizes. The purpose of each robot type is engineered to perform and carry out its given task/tasks. There are industrial robots which are robots with fixed arms with various axes of freedom and service robots. According to the International Federation of Robotics (IFR) [25] "A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations". A companion robot is a unique type of service robot that is specifically designed for personal use at home [26].

In the next section we briefly describe common applications for service robots in other sectors and examine in details common applications necessary for elderly and disabled population.

2.1.2 Applications

Robots have become more prevalent in our day to day lives. Different types of robots designs are used to enhance security, facilitate daily chores, aid in recalling medicine consumption and more.

Following is a list of common applications that are used in aiding people to complete different tasks:

- *Rescue applications:*

  There are situations where rescue personnel can use robot in aiding rescue staff in completing different missions in situations such as hostage taking , tunnel explosion, urban disasters, etc. (Figure 2-1).

Figure 2-1: Tokyo Fire Department's Robocue.
(taken from [65]**)**

- *Military Applications:*

There are dangerous military missions that soldiers need to complete such as walking through minefield, deactivating bombs, detecting and defusing landmines, etc. Figure 2-2 shows a robot called ACER that can handle tasks like clearing explosives and hauling cargo [27].

Figure 2-2: Military robot: ACER
(taken from [27]**)**

- *Medical Applications:*

Robotics technology has produced valuable tools and devices for rehabilitation,

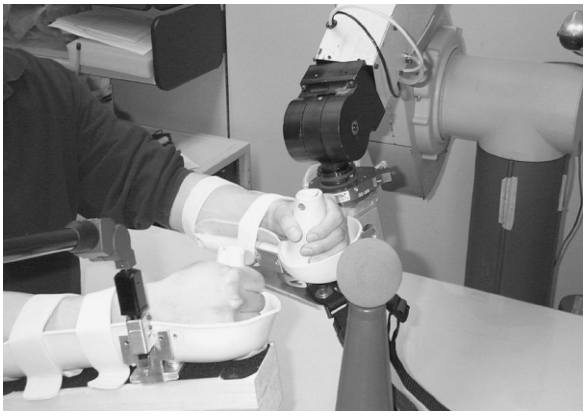


Figure 2-3: MIME: rehabilitation therapy robot
(taken from [28]**)**

surgery and medical training as well as new and improved prosthetics and assistive devices for people with disabilities, which are used to replace missing limbs, perform delicate surgical procedures, etc. Figure 2-3 shows an example of a medical robot used for delivering rehabilitation therapy to a patient with an arm impairment. Details on medical robotics applications specifically surgery applications can be found in [28].

- *Educational applications:*

South Korea is the first country which is currently using robots as English teachers in schools to address the shortage of English teachers in rural areas or remote islands. Figure 2-4 shows one of these robots. Till 2011 , Robotics teachers have been deployed in 500 preschools [29] [30].

Figure 2-4: Robot English Teacher (taken from [30]**)**

- Assisting Applications:

  There are robots which perform household tasks such as vacuuming [31], cleaning the table [32], etc. Companion robots are a type of robot that assists elderly and disabled persons to improve their quality of life. They help the disabled and the elderly to complete their daily tasks and routines. Figure 2-5shows a robot designed by the Robotics Research Institute at the University of Tokyo (Tokyo University's IRT) [32].

Figure 2-5: Tokyo University's IRT: assistant robot (taken from [32] **)**

As stated in [31], our society, especially elderly and disabled population, faces a critical challenge: how to increase and maintain their quality of life. Companion Robots will offer variety of ways in assisting this population to maintain and enhance their quality of life.

Companion robots can be viewed as a technology that integrates awareness, emotion, and action with a contextual knowledge of self, others, and the environment to provide a satisfactory welfare. They can get along with the smart homes to fulfil more complex

applications. Integrating robots with such technology provides a higher level of welfare specifically needed for people with restriction and disabilities.

There are some efforts by robotics researchers to aid elderly and disabled people to live their lives in a more comfortable and helpful environment. These efforts lead to design robots with different capabilities and hardware technologies. Some robots are simply surveillance robots that provide information about the environment they monitor. Other robots carry out more complex tasks like sensing abnormal changes in the environment such as gas licking or changes in the body conditions such as blood pressure, heart beats, etc. and execute appropriate tasks [33].

Decreased memory is common to age-related cognitive decline, which often leads to forgetfulness of daily routine activities (e.g. taking medications, attending appointments, eating, etc.). The need for a robot that can offer cognitive reminders is quite prevalent for the elderly. In addition, nursing staff in assisted living facilities frequently need to escort elderly people on walks to their exercise, to attend meals, appointments or social events. The fact that many elderly people move at extremely slow speeds (e.g. 5 cm/s) makes this one of the most labour-intensive tasks in assisted living facilities [34]. A robot can play an important role in completing this task while still providing verbal interaction with the patient once it is equipped with speech recognition technology.

Another application for which robots are significantly useful is household tasks. Some robots are able to move objects, for example carrying a food tray from the living room to the kitchen, or delivering the TV remote, the phonebook or the medication bag to the individual in need. Some robots take out the garbage or replace the dishes [35]. These robots are equipped with a camera to recognize the goal object. Robots with these kinds

of facilities play the role of a full-time house keeper; without the cost of having a full-time house keeper.

Robot design is crucial, it is an important issue that the robot be designed in a manner that is acceptable for the population it serves most importantly the elderly and the disabled. People using these robots need to feel comfortable to interact with robots. These needs have led to the design of humanoid robots [36]. The appearance of these robots is more human-like and they are even able to carry out conversation with people.

There are many more applications from robotics science that have helped people especially those with restricted capabilities. Robots with various capabilities are getting more and more involved in elderly and disabled people's daily lives not only by assisting them but also by providing companionship. This encourages researchers in different fields from sociology to robotics and computer science to further perfect their research to help the needs of these people.

2.2 REpresentational State Transfer (REST)

In this section we provide some background information on REST and RESTful Web services.

2.2.1 Introduction

REST is *an architecture style* for designing distributed applications [37]. The idea, was first, presented by Roy Fielding in his PhD dissertation [11]. REST is a lightweight alternative to mechanisms such as CORBA and Remote Procedure Call (RPC) [38] or complex Web services such as SOAP, Web Services Description Language (WSDL) [39], etc. Despite its simplicity, much like Web services, a RSET service offers the following features [40]:

- It is platform-independent: It does not matter if the Operating System (OS) of the server is UNIX, Windows, etc. or the OS of the client is Mac, Windows or anything else.
- It is language-independent: Client and server programmed in different programming languages can connect to each other using REST as the interface.
- It is standards-based: It runs on top of HTTP.
- Like other Web services, it goes through firewalls.

According to [11], REST is not limited to a specific protocol. Although the Web's primary transfer protocol is HTTP, it also includes seamless access to resources that originate on pre-existing network servers including File Transfer Protocol (FTP) [41], Gopher [42] and Wide Area Information Servers (WAIS) [43].

RESTful Web services are different from traditional ones (also known as Big Web services [44]) in various aspects:

- It is lighter weight than Big Web services. It does not require either XML parsing or message header to and from service provider [37].
- It is easy to build: unlike SOAP, no toolkit is required.
- Due to its light-weight characteristic wider range of devices from Personal Computers (PCs) to constrained devices are supported [37].
- REST is more scalable since operations are self-contained and each request transfers all the information (state) that the server needs in order to respond to it [11].
- REST results are intended to be human readable (e.g. HyperText Markup Language (HTML)).

A RESTful web service is a web service implemented using the principles of REST. RESTful web services can be described using the Web Application Description Language (WADL) [45]. A WADL file describes the requests that can be addressed to a service, including the Uniform Resource Identifier (URI) of the service and the data that the service consumes [46].

To design a RESTful architecture we need to realize resources and representation of resources. Then we need to follow the design principles required for having a RESTful architecture.

2.2.2 RESTful Web Service

In this section we describe Resource Oriented Architecture (ROA) and the design principles for a RESTful architecture.

2.2.2.1 Resource Oriented Architecture (ROA)

As mentioned above, REST is not an architecture but a set of design principles. Therefore, [44] define a new architecture known as ROA to develop RESTful Web services. As understood from this term, the architecture is based on *resources*.

As the first phase of a Resource-Oriented Architecture design we need to grasp four concepts: Resources, their names (URI), their representations and the link between them.

A resource is anything that is important enough to be referenced [44]. A resource may be something that can be stored on a computer and represented as a document, a row in a database, the result of running an algorithm or a physical object like a robot [44].

Looking at this concept from the user point of view, If the user of the RESTful architecture wants to create a hypertext link to something, make an addition or deletion

about it, retrieve, delete , change or cache a representation of it then it should be considered a resource.

Each resource has a name, a representation and a link to be accessible [44] .

A representation of a resource is typically a document that captures the current or intended state of a resource [11]. In other words, representation is *any useful information* about the state of a resource either it is a data list or a physical object [44]. Data representation can be in any format or any media type such as XML, JavaScript Object Notation (JSON) [47], eXtensible Hyper Text Markup Language (XHTML) [48] or plain text representation.

Each resource must have a URI. The URI is the name and address of a resource. URIs should be descriptive: a resource and its URI must have a direct accordance. For instance: `http://www.example.com/EV/staff/123` which gives the information about an employee of EV building with the id number *123* [44].

In the relationship between the resources and URIs, no two resources can be accessible through the same URIs. A resource can be reached by one or more URIs. Resources are linked to each other via hyperlinks. Connection between the resources is an important concept in designing a good RESTful architecture. We can access the resource and manipulate it through a uniform interface which is described in the next section.

2.2.2.2 Design Principles

As the second phase of a ROA design we take four design principles into account: *Addressability, Statelessness*, *Connectedness* and a *Uniform interface*.

2.2.2.2.1 Addressability

We mentioned that every interesting piece of information the server can provide should be exposed as a resource and given its own URI. This is called Addressability. URIs are standardized and well-known. From a URI we know the object's protocol. In other words, we know how to communicate with the object: we know where in the network it is placed i.e. we know the host and the port number. We know the resource's path on its host, which is its identity on the server it resides on. From the end-user perspective, addressability is the most important aspect of any website or application [44].

2.2.2.2.2 Statelessness

The definition of the statelessness by the REST author [11] is: “each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.” From a RESTful perspective when the client sends an HTTP request, it includes all information necessary for the server to respond to that request. The server never relies on information from a previous request. If the information from a previous request was important, the client should resend it in the new request [44]. Statelessness means that the *possible states* of the server are also resources, and should be given their own URIs.

In a stateless application every time the client makes a request, it ends up back where it started. Each request is totally disconnected from the other [44].The client can make requests for resources any number of times, in any order. It can request page 2 before requesting page 1 (or not request page 1 at all), and the server will not care [44].

This constraint (statelessness) improves reliability and scalability: Reliability because it is easier to recover from partial failures; Scalability because the server does not have to store state between requests which allows the server component to quickly free resources. Statelessness also simplifies implementation because the server does not have to manage resource usage across requests. Detailed information can be found in [11] and [44].

2.2.2.2.3 Connectedness

RESTful service representations are hypermedia documents. These documents not only contain data but also carry links to other resources. The server guides the client's path by serving links and forms inside hypertext representations ("hypermedia"). The quality of having links is called "connectedness". Resources should link to each other in their representations. In a well-connected service, the client can make a path through the application by following links. Right now the Web is very well-connected, because most pages on a web site can be reached by following links from the main page [44].

2.2.2.2.4 Uniform Interface

Another fundamental feature of REST architectural style is its emphasis on a uniform interface between components [11]. The main goal of uniform interface is simplification. When clients are interacting with web resources they expect simplified interfaces. This can be achieved by using the uniform methods of HTTP protocol and combining the same with the resource operation.

By combining the standard HTTP methods and the resource names we can have uniform interfaces thus leading to simplified communication.

In the following we provide detailed information of each method in addition to a brief explanation of two other HTTP methods: HEAD and OPTIONS which can be used for retrieving meta-information.

- HTTP GET:

  The GET method retrieves information (in the form of an entity) identified by the Request-URI. A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given by the header. The conditional GET method is intended to reduce network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring unnecessary data. The GET method can also be used to submit forms. The form data is URL-encoded and appended to the request URI. As shown in Table 2-1 GET HTTP method is safe and idempotent which respectively means that it does not change the state of the server and multiple identical requests can be applied, having the same effect as a single request. The response code for a successful request is 200 (i.e. OK). When a resource does not exist the response is 400 (i.e. not found).

- HTTP PUT:

  To create or modify a resource, the client sends a PUT request that usually includes an entity-body. The entity-body contains the new representation of the resource that client intends. The data context and the format of it depend on the service. It is the point at which application state moves onto the server and becomes resource state. If a new resource is created, the response code is 201 (i.e. Created). If an existing resource is modified, a 200 (i.e. OK) response code should be sent to indicate

successful completion of the request. If the resource is not created or modified, an appropriate error response should be given; the error response reflects the nature of the problem.

PUT may be used to modify an existing resource identified by the URI that already exists. Furthermore, PUT method is idempotent like GET method.

- HTTP DELETE:

The HTTP DELETE method is used to delete the existing resources identified by the requesting URI. If the action is performed successfully, a successful response status code 200 (i.e. OK) is returned. Otherwise, an appropriate error response code will be returned indicating the nature of the problem. The HTTP DELETE method is also an idempotent like GET and PUT methods.

- HTTP POST:

POST is commonly used to create subordinate resources: resources that exist in relation to some other "parent" resource [11].The POST method is a way of creating a new resource without the client having to know its exact URI. The response to this request usually has an HTTP status code of 201 (i.e. Created). Its Location header contains the URI of the newly created subordinate resource. Now that the resource actually exists and the client knows its URI, future requests can use the PUT method to modify that resource, GET to fetch a representation of it, and DELETE to delete it [44].

Table 2-1 summarizes the list of the HTTP methods.

Table 2-1: HTTP methods description

Methods	Description
GET	Get the representation of a resource (safe and idempotent)
PUT	Create a resource to a new URI or Update an existing resource (idempotent)
DELETE	Deletes a resource (idempotent)
POST	Create a new resource to an existing URI

Besides the four main HTTP methods, there are two other HTTP methods; HTTP HEAD and HTTP OPTIONS are also considered uniform methods of HTTP protocol. HTTP HEAD method is used to fetch meta-data about a resource but not the resource itself. HTTP OPTIONS method is used to discover HTTP methods which are allowed for specific resources.

2.2.3 Procedure of Creating a RESTful Web Service

According to [44] the design procedure is as follows:

First the data set must be determined: What is our data? What information is important enough to be exchanged between different entities of the system? Then the information is classified into resources, thus making them accessible to the entities. We then name our resources with URIs, we will then have accessible uniform and linkable resources that are following three basic rules [44]:

1) Use path variables to encode hierarchy: /parent/child
2) Put punctuation characters in path variables to avoid implying hierarchy where none exist: /parent/child1;child2

3) Use query variables to imply inputs into an algorithm, for example: /search?q=student&start=m

The next step is to expose a subset of the uniform interfaces. As explained in the previous subsection, HTTP is widely used and is considered a simple yet efficient uniform interface.

Afterwards, we design the representation(s) data and format, a decision needs to be made as to what data must be sent when a client requests a resource and what data format must be used.

Besides conveying the state of a resource, the representation should provide links to other resources such as new *application states*. So the next step is to integrate a resource into existing resources using hypermedia links. The goal is *connectedness* which is the ability to get from one resource to another, following links.

After this, we need to consider the sequence of events that usually occur: The possible response codes or the HTTP headers that is sent. For example most read-only resources have a simple sequence of event: the client sends a GET request to a URI and the server responds with a response code like 200 (OK). From the header side, the main consideration is which HTTP headers the client should send in the request and which ones the server should send in response.

Finally, in the last step, the possible error conditions need to be considered. In some cases the response may be an error instead of conveying a representation.

The last two steps, although are conceptually simple take much of implementation time.

Chapter 3:
Companion Robots Applications Development and Deployment: Requirements and State of The Art Evaluation

Our goal is to provide an architecture that facilitates development and deployment of companion robot applications. This architecture must take into account the numerous applications, robots technologies, and the challenges of manoeuvring companion robots. We have derived requirements necessary for development and deployment of companion robots. In this chapter we discuss these requirements and categorize them into two groups: general and interface specific requirements.

We organize the related work for companion robots applications development into three groups: non-standard based, other standards such as SOAP and CORBA based, and REST based works. We then proceed to review some of the major works in each category and conduct an evaluation of the works with regards to our derived requirements. We conclude the chapter with a summary.

3.1 Requirements

In this section we discuss the two sets of requirements for the development and deployment of companion robots applications. The first set includes general requirements for the overall system; the second set is about specific requirements for the interface between the system components. The summary table of the requirements is presented at the end of this section.

3.1.1 General Requirements

As discussed in Chapter 2, robots are a part of daily life particularly for the elderly and the disabled. These two sectors of the population greatly benefit from companion robots, however, each group's needs are different and to further complicate matters each individual's needs are different. These differences require different applications. Therefore, our first requirement is that the architecture should be application independent, so it will support developing various application based on the target assistance.

As different applications are required so do different robots; various applications lead to robots with different technologies and capabilities. For example, a robot which is designed as a surveillance robot to provide monitoring services is different in hardware and capabilities from the one which is designed to clean up the house and water flowers. Therefore, our second requirement is that the architecture should be robot independent. So, developing various applications for different robots will be possible.

Robot technologies are changing rapidly. Every now and then, a new robot with new capabilities emerges. They either enhance applications that have formerly fulfilled some needs or they execute new applications that meet needs that were not previously considered. Communication approaches are evolving; this will eventually create new demands. This point leads us to our third and the last requirement, high level of flexibility and maintainability which can be achieved by the separation of concerns/decoupling among components, hence, companion robot applications developer will cope with the fast growth of technology without being concerned about developing applications from the scratch.

3.1.2 Interface Requirements

Based on the scenarios described in Chapter 1, the end-user may use his/her smart phone to control his/her robot remotely. Devices such as smart phones and PDAs have limited resources such as battery and processing power. Some robots are |IP based and can be connected directly to the internet. They have processing power to process a service on their own ,however, their power is limited in comparison to laptops and PCs, therefore, when designing interfaces resource-constrained devices need to be considered and an important requirement is that the interfaces need to be as light-weight as possible.

One significant concern for the developers is the design of the interfaces. If the interface is difficult to understand or requires a deep knowledge of robots' system, it will be difficult for developers to use it to develop different applications. If the interface is complex it will be difficult to modify if required. As the second requirement, Interfaces need to be easy to use, in order to facilitate the development of various applications, without burdening the developers.

Beside robots, there are other devices that need to work in tandem to execute an application, devices such as cameras, house sensors, body sensors, etc. Considering that there are numerous of these devices, developing interface for each device individually is a tiresome and complicated process, therefore, our third requirement is that the interface should be device independent so integrating new devices to the system will be easily fulfilled.

If an interface is designed based on one application it will not only be impossible to develop a new application using that interface, but also to modify it, therefore, our fourth

requirement is that the interface should be application independent so the developers are able to use the same interface for different applications.

Robotics application developers have different experiences and knowledge of programming. They might use Windows, Mac or LINUX operation systems. They might program their applications in C#, Java, python, etc. The system should not enforce using specific language and/or particular operating system to develop the application. Therefore, our fifth requirement is that the system should be OS and language independent so it will offer the developers the freedom of using any language and any OS they desire.

Earlier we mentioned that the evolving needs of the elderly and disabled create a demand for new robots and applications, therefore, we need to cater for changes, modifications and extensions. Our last requirement is that the interfaces should be extensible so it gives the ability to meet future requirement whom the developers might need to address.

Table 3-1 summarizes the general and specific requirements for development and deployment of companion robots applications.

Table 3-1: Derived requirements for development and deployment of robots applications

General requirements	1.Robotics platform (Robots) independent
	2.Application independent
	3.Separation of concerns among components to increase flexibility and maintainability
Interface specific requirements	1.Lightweight
	2.Easy to use
	3.Device Independent
	4.Application independent
	5.OS/language Independent
	6.Extensible

3.2 State of The Art

Companion robots applications are a growing research field that include different aspects of robotics in human life. Since companion robots are designed for the people with special needs, new challenges will arise. These challenges range from the sociological point of view to the robots design aspects. Developing and deploying applications for companion robots is an area that has many challenges that stem from the numerous applications and robots in elderly and disabled population. There has been some research conducted on this field, however, they are mostly proprietary designs. In spite of the importance of this field, a few works on standard based approaches have been suggested; therefore, a standard-based solution that addresses requirements of this field is a necessity.

In this section the related works are categorized in three sets: First we present non-standard based research works; in the second group, we discuss standard based solutions such as CORBA and SOAP based; in the third set, we discuss the REST-based research works. We evaluate all the related works based on our concluded requirements. To conclude this section, we present a summary of this evaluation in the Table 3-3. We summarize the chapter at the end.

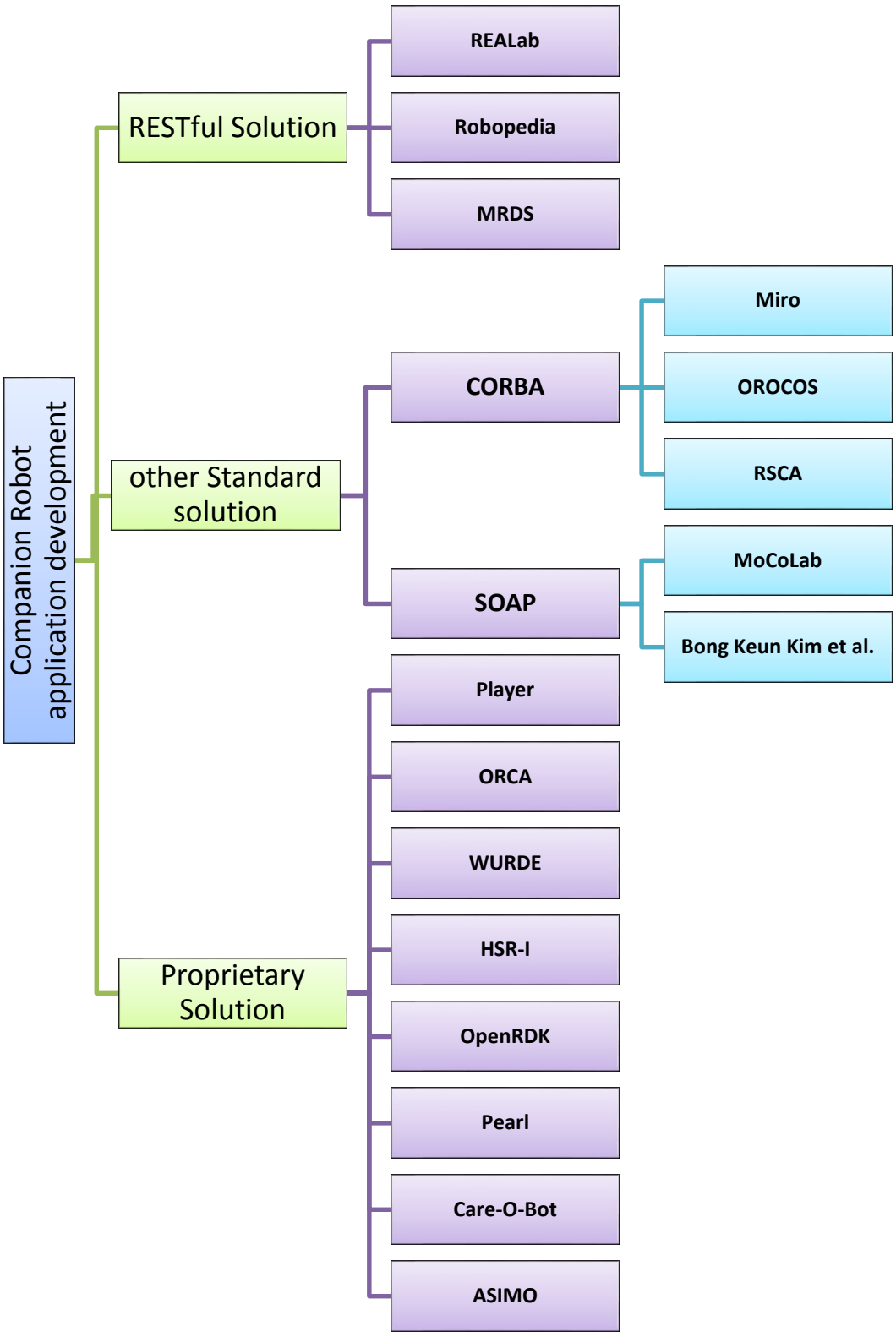


Figure 3-1: Companion Robot application development: state of the art

3.2.1 Non-Standard Based Solutions

This section provides detailed information on four non-standard based solutions and refers to four more works in this category.

3.2.1.1 A Framework for Robotics Application Development

The work presented in [49] is an open source Robotic Development Environment (RDE) developed at the University of Southern California. It is a robot programming framework that provides a set of tools for the robotics research community to simplify the development of robotics application. The Player server includes a collection of device drivers for many popular robot hardware devices. Client programs use proxy objects defined in a Player client library to write and read data to and from the desired device [49]. Using player programmers are able to support new hardware devices. Player is developed primarily under Linux; however, it also runs on other UNIX variants such as Solaris and FreeBSD that support TCP socket mechanisms and under Windows with Cygwin. In order to provide uniform abstraction Player follows the UNIX model of treating devices as files. In Player 2.0 the core system is a queue-based message passing system. Each driver has a single incoming message queue and can publish messages to the incoming queue of other drivers and to specific clients in response to requests. [49] uses an open standard called eXternal Data Representation, or XDR [50]. The XDR specifies a platform-independent encoding for commonly-used data types, including integers and floating point values. They developed a C library, libplayerxdr, which performs the XDR data marshaling. It provides a single function for each Player message type that packs and unpacks message payloads, converting between native and XDR formats.

Despite the flexibility and platform/language independency characteristics of [49] the developer is required to have a thorough knowledge of robot system and programming as well as an extensive understanding of the tools. Simplicity of the interface, however, is quite relevant and needs to be made a requirement. This work is not factoring in the light-weight characteristic, which imposes a limitation on using resource-constrained devices such as smart phone.

3.2.1.2 Tele-robotics System for Healthcare Management

Another work in this category is [51]. The objective of this work is to develop a tele-robotic system to support the elderly and disable in long-term health care management. The Internet based robot system is designed to function autonomously or semi-autonomously with tele-operation capability to help the elderly or the disabled in a hospital or at home setting [51]. The focus of the work is in three areas:

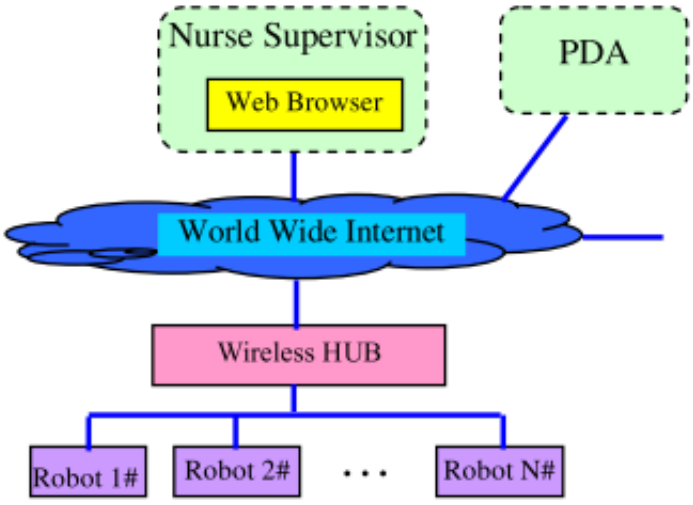


Figure 3-2: Network distributed tele-homecare robot system

Security: The interaction between robots and humans in the home or hospital makes security a primary consideration; *reusability:* In order to adapt to different scenarios the

existing modules should be easily reused or extended to new applications; *Friendly human-robot interface:* A healthcare robot should be practical for inexperienced users. People desire to communicate with the robot using human language and to receive friendly feedback. Thus, an optimal robotic tele-operation user interface must supply related information regarding the robot's state and environmental conditions (objects, persons, free space, etc.) along with an efficient command system to the operator. The block diagram of the Internet based Healthcare Robots (HRS-I) is presented in Figure 3-2. The HSR-I is connected to the Internet through a wireless adaptors. During normal operation (no emergency, no obstacles) the Supervisor will execute random patrol command to all platforms, display high-level graphical status and location information. Once the robot detects any exceptional action, the nurse can get the alarm information from the console and give a corresponding command to that robot. The HSR-I hardware platform is an autonomous indoor robot containing basic components for manipulating, motor control, sensing and navigation, including battery power, drive motors and wheels. The system software uses web based Client/Server architecture for the robot control and feedback information display. The client side (web browser) shows the robot information such as robot position, speed and security on the console so the nurse can observe the situation of the patients' rooms. Server side is responsible for motor control, sonar ranger, camera motion control and measurement of the patient's physiological parameters.

This work presents a mobile robot system in the healthcare domain and it has been developed based on the custom designed robot platform, however, the authors do not provide any information about the interfaces. Since the system proposed in [51] is based

on their own unique design we surmise that interfaces are not OS/language independent. The simplicity and light-weight of the interfaces are not features considered in this research.

3.2.1.3 Assistant Robot for Elderly People

[4] is another research effort in the elderly domain. Pearl is a mobile robot system designed to assist the elderly in performing their daily activities and manoeuvring their surroundings. The project was envisioned in 1998 by a multi-disciplinary team of investigators consisting of researchers in health care and computer science.

The research focus is on two tasks: reminding people about routine activities such as taking their medication and guiding them through their environment. They have developed two autonomous mobile robots along with software systems to enable these robots to assist the elderly.

The primary function of the system is to serve as a cognitive orthotic, providing the

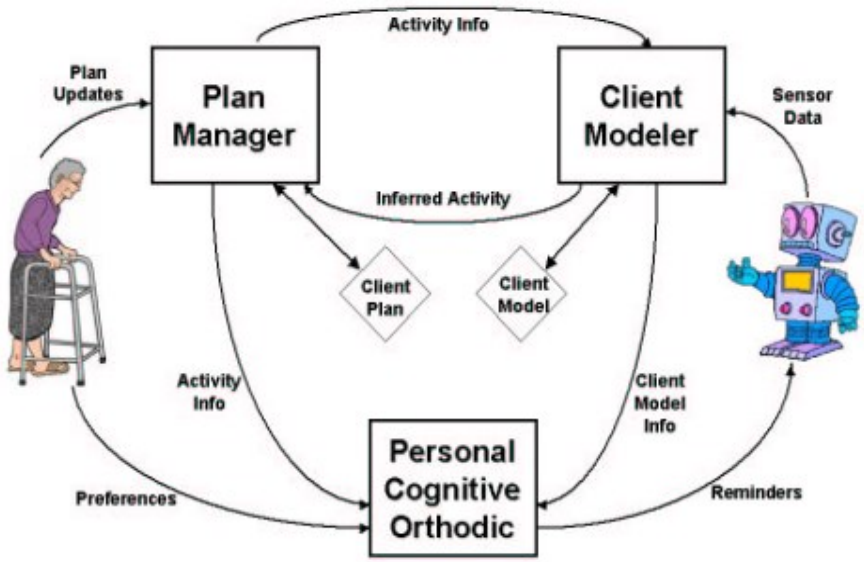


Figure 3-3: Autominder Architecture

elderly with reminders of their daily activities. The software component that has been developed for Pearl to provide the cognitive orthotic functions is called Autominder, and is depicted in Figure 3-3. As illustrated in this figure, Autominder has three main components: a Plan Manager (PM), which stores the user's daily tasks. It is also responsible to update the tasks and determining if there is any conflicts in the schedule; a Client Modeler (CM), which uses information about the visible user activities to track if the task is being performed according to the plan; and a Personal Cognitive Orthotic (PCO), which analyzes the difference between what the user is supposed to do and what he/she is doing. It also decides when to send out reminders.

Despite all the advantages that [4] has, it is limited to the proprietary robot platform. It is designed to develop specific applications in the elderly domain. Like [51], [4] does not provide any information on the interfaces. Pearl is not designed to be manoeuvred via the Web and important requirements not considered when designing Pearl are the simplicity and the light-weight characteristic of the interfaces.

3.2.1.4 Home-Care Robot for Elderly and Disabled People

The final reviewed related work in the non-standard based category is [52]. The work presented in [52] proposes a platform for a mobile home care system – called Care-O-Bot. The Care-O-Bot is a mobile service robot, which has the ability to operate different tasks in home environments. Main emphasis is on combining communicational and social features, like video telephone or automatic emergency calls. This work focuses on the mechanical design and realisation of the vehicle. It further provides the development of the control system architecture and the implementation and testing of navigation and motion algorithms.

The authors first list the necessary functions that an optimal home care robot should perform such as: Household tasks like delivery of food and drinks; Communication like automatic emergency calls; Technical House Management like control of home devices such air conditioning and lights; Mobility support like guidance assistance; Personal Management and Social activities like organization of daily tasks such as daily task organization medications, reminder to events, etc. ; and Personal Security like checking for personal safety.

The work further provides technological concepts that were developed by the research group. We summarize each concept with an example as following:

- Mechanical Concept: e.g. Mobile platform suited for home environments
- Electrical Concept: e.g. Independent battery based energy management and supply
- Control System Hardware: e.g. Modular and extendable control system hardware
- Control System Software: e.g. Map building and dynamic path planning
- Operator's Interface: e.g. Instruction of Care-O-Bot with natural speech

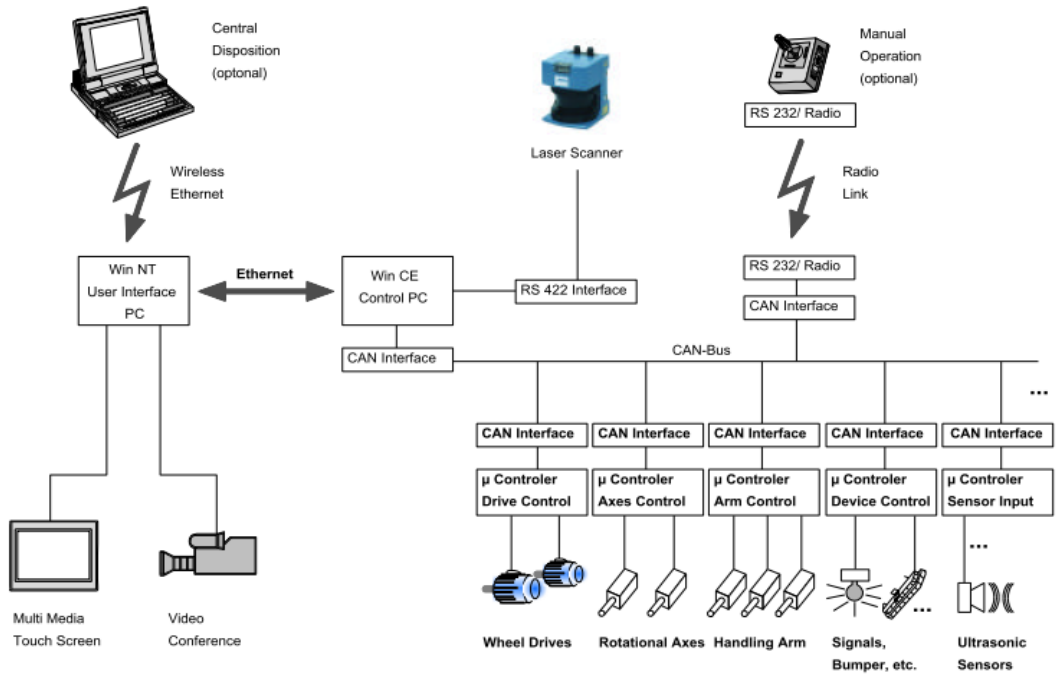


Figure 3-4: Care-O-Bot's decentralised control system architecture

To fulfil and perform all required functions of Care-O-bot a sophisticated control system is necessary. The designed control system architecture is shown in Figure 3-4. A master PC is employed to control the vehicle and command all devices and drive control modules. Different types of operation modes are used to drive the Care-O-bot according to the needs of the user (e.g. automatic, manual and reactive mode). There is another PC, connected via Ethernet running under Windows NT/95, which is responsible for the control of all communicational tasks like speech control, multimedia touch screen and linking via wireless Ethernet connection.

Explained above, [52] is a custom design for the specified robot platform. It suggests different applications to aid the elderly and disabled, however, the interfaces are proprietary i.e. they are not applicable to other robots. This work does not look at remote

manipulation of the robot while smart phones are an integral part of our lives and being able to manipulate a home care robot remotely is a necessity.

There are other works on the mobile robots that are shown in the related work hierarchy (Figure 3-1) such as [53] ,[54], [3] and [2]. For instance, [53] is robot/device dependent which has proprietary interfaces that do not meet any of our specific requirements. Another example, [3], proposes a hardware-independent architecture however it supports few applications. The interface proposed by this paper (RIDE) is a proprietary interface that does not meet the simplicity and light-weight requirements that were mentioned in the previous section. Flexibility and extensibility of the architecture has not been the consideration of the majority of non-standard research works.

3.2.2 Standard (CORBA and SOAP) Based Solutions

In this section, we begin by introducing a CORBA based related work. Then we criticize the work by discussing the drawbacks inherent to CORBA. Afterwards, we present SOAP based related works and criticize them by describing the disadvantages of SOAP compared to REST.

3.2.2.1 CORBA Based Solutions

A CORBA based solution has been proposed by [55]. It presents a middleware for mobile robot applications. It favours the use of object-oriented robot middleware to make the development of mobile robot applications. Miro also provides generic abstract services like localization, which can be applied on different robot platforms. Despite its advantages, it imposes some limitations inherent to CORBA. Below we explain some

major drawbacks of CORBA middleware that discourage using it for developing companion robot applications:

Lack of Distributed Transparency: The purpose of middleware is to hide implementation details and complexities from the applications. Using CORBA, each time a new component is added and the overall architecture is extended, several components that have previously been used will have to be reconfigured to adapt to the changes, and this causes overall configuration confusion. It also requires updating if the application needs to be migrated [56].

Implementation Complexity and Maintenance difficulty: reusing functionalities of existing applications is very complex when CORBA is employed. For example, CORBA to C++ mapping is extremely complicated and requires significant understanding and experience. Application maintenance in CORBA also requires a huge effort. If the properties of an object is changed or an object is later extended the IDL file has to be modified. As a result all the proxy components will have to be replaced by the newer ones [57].

Increase in dependency: Using object oriented propriety middle ware increases dependency on additional software components. Two distributed CORBA based applications can communicate with each other using IIOP; however these applications would not be able to apply security and transactions features supplied by CORBA. Such applications require thorough configuration and are highly dependent on other components [57].

Limited Interoperability: CORBA is language and platform independent, however, the interoperability level between different CORBA products is limited [12].

As a result, Web services have a great superiority over CORBA. If choosing between WS and CORBA, Web service will be the choice. The next section, however, will demonstrate that not all Web services are suitable for developing companion robots applications.

3.2.2.2 SOAP Based Solutions

[58] and [7] propose SOAP-based Web services approach for mobile robot applications. [58] presents a robot control platform for ubiquitous functions that is based on Web services. [7] proposes a service oriented architecture that provides a flexible distributed application model for the motion control of mobile robots.

Following description, however, clarifies that the above-mentioned related works, due to the drawbacks inherent to SOAP, are left aside.

Three major criteria that SOAP is assessed are as following:

- ✓ Light-weight and Simple
- ✓ Flexible in terms of data representation
- ✓ Easy to use for developing applications

Light-weight and Simple: SOAP based web services are sophisticated because of the verbose XML files and complex SOAP messages envelope. Using SOAP based web services, it is necessary that SOAP and RPC be supported on both client and server applications [59]. RESTful web services are very lightweight and simple compared to

SOAP -based web services. REST leverages W3C/IETF standards, i.e. HTTP, XML, URI, and MIME.

Flexible in terms of data representation: SOAP based web services force XML format for data representation. This limitation causes problem when devices with very limited resource are the target. RESTful web services, however, provide a greater flexibility when using different data representation formats such as XML, JSON and plain text.

Easy to use for developing applications: SOAP based services require greater effort to implement applications either on client or server side, due to the specific toolkits that are required on both client and server sides when developing SOAP based applications. Understanding these toolkits demands for a significant endeavour, whereas REST based web services require more effort just on the server side. Using familiar HTTP methods makes the developing applications much easier and simpler, thus there is no need for prior knowledge of a toolkit.

Table 3-2 clearly demonstrate here that REST has superiority over SOAP.

Table 3-2: REST vs. SOAP

Criteria / Solutions	Light-weight and Simple	Flexible in terms of data representation	Easy to use for developing applications
SOAP	NO	NO	NO
REST	YES	YES	YES

As a result of this section, none of the above-mentioned works will be taken into consideration since REST is superior to others.

3.2.3 REST-based Solutions

This section gives a detailed review on three REST-based research works on companion robots applications development.

3.2.3.1 A Platform for Network Robotics

[60] presents a distributed software platform that supports inter-domain interaction with mobile robots. Figure 3.5 shows the core packages of the platform presented by the authors in UML (Uniform Modeling Language) notation. The Embedded package includes the micro-server components that are able to run on embedded processors with limited processing power (e.g. mobile robot). The Protocol Handler package liberates the resources from performing tasks that require massive computation such as data encryption/decryption, protocol translation, and security checking. The Front-end package includes components that support interactions over the network between the application and the mobile robots. These components offer a high level interface for robot manipulation in different programming languages and platforms. The Management package performs management actions at the level of federations, domains, and

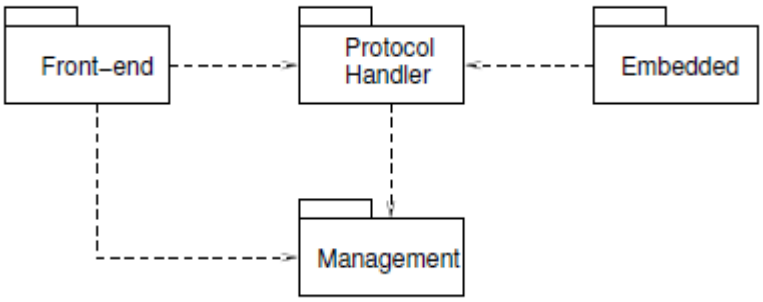


Figure 3-5: Main packages of the proposed platform

resources.

REALab is a proposed platform of the presented designed packages. It has two HTTP servers interacting with the ARIA and Player robotic frameworks. HTTP GET message is used for operations that do not change the state of the robot (e.g. sensor readings) and those that cause changes (e.g. movement) are performed through HTTP POST. All HTTP operations return a XML document. The Front-end package offers a set of functionalities such as rangefinder sensor, locomotion, and image acquisition. The REALab platform offers APIs in the following programming languages: C++, Java, Python, and Matlab. Figure 3-5 shows the components of the platform

The emphasis of the work is mostly on the components of the platform rather than interfaces. The authors claim that RESTful interfaces have been developed for this platform but it employs a RPC-like function call, where the procedure and parameters are

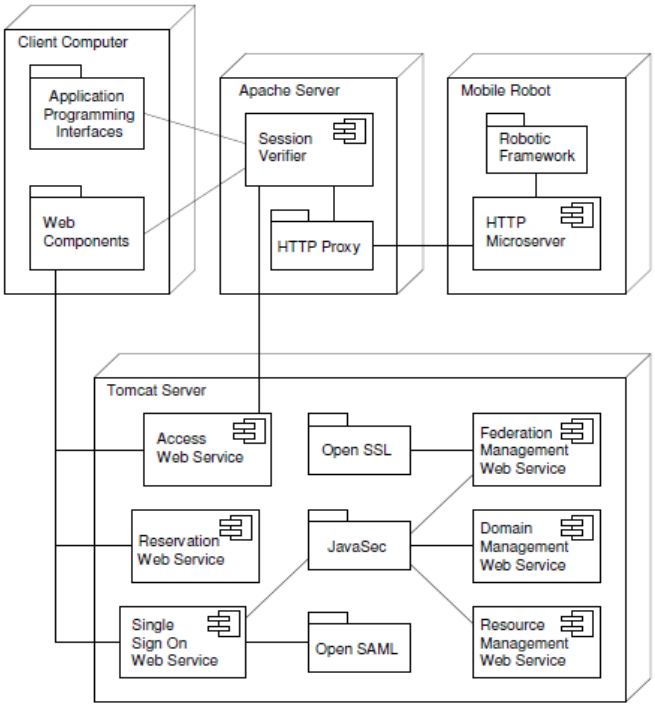


Figure 3-6: Components of the REALab platform

communicated in the HTTP Uniform Resource Locater (URL) , they also do not provide any resource modeling which is a key aspect of REST style architecture.

3.2.3.2 A Web-Enabled Framework for Robotics Application Development

[8] proposes a framework to integrate robotics application with Internet-scale sensor networks. It receives a sensor observation of a surveillance robot and commands the actuator of the robot over the Web. The framework allows the development of general web-based robotics application by hiding the most prerequisite of robotics knowledge behind the scene. According to authors quote "a general web-based robotics application includes status monitoring, specifying goals for the robot, and tele-operating actuators or sensors, such as a pan-tilt-zoom camera". Robopedia architecture is shown in Figure 3-7. The first part is the robot communication server which employs Player to receive robot sensor observations and to issue robot commands. The second part is a web-server which provides RESTful web-services to interface with the system. Robopedia's web-interface provides general reusable interfaces for controlling a robot via the web.

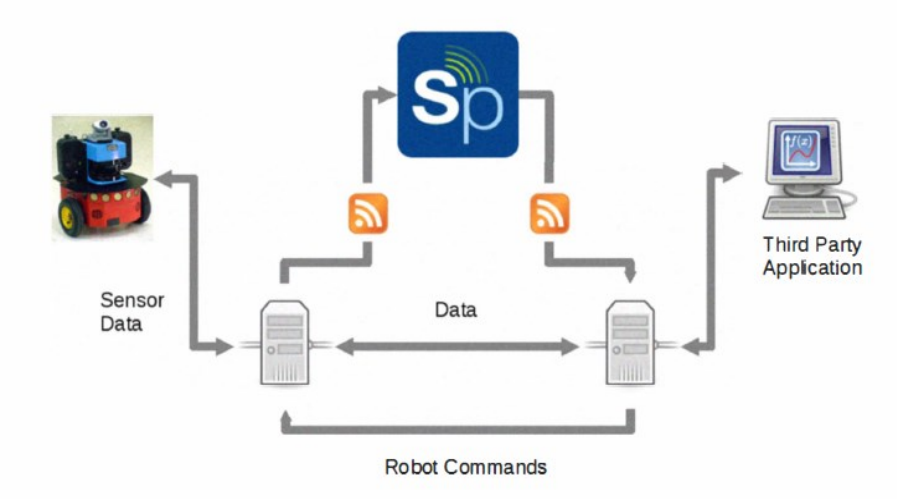


Figure 3-7: Robopedia Architecture

The work does not define a candidate data representation for every sensor type. It has limited the support to the Pioneer 3DX equipped with a camera, SICK laser range finder, odometer, encoders, and sonar.

The command interface issues commands in the proprietary Player server data format to the robot communication server.

One drawback of the system is that the command interface can only handle the Pioneer's Canon PTZ camera and motion commands such as velocity and specify goal points for the robot. Moreover, although the authors claim to use RESTful interfaces, they do not provide any information on REST resources. As mentioned in chapter 2, resource is the main concept that should be realized when providing REST style architecture.

3.2.3.3 Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio [61] is a Windows-based environment for robot control and simulation which allows development and execution of both RESTful and SOAP-based applications.

The architectural design of MRDS follows REST pattern that consists of two components: Concurrency and Coordination Runtime (CCR) and Decentralized Software Services (DSS). The CCR makes asynchronous programming simple. It handles asynchronous input from multiple sensors and output to actuators. The DSS Service Oriented Architecture (SOA) offers a simple access to a state of a robot using a Web browser or Windows-based application.

Third parties can extend the functionality of MRDS by providing additional libraries and services.

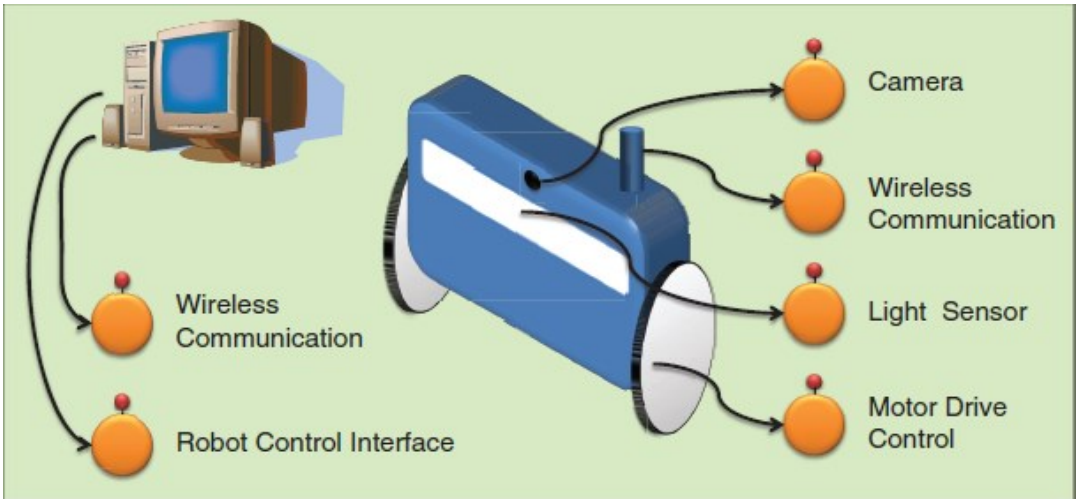


Figure 3-8: Typical service configuration for a mobile robot

Interaction with a robot is implemented through the use of multiple software services which are highly decoupled, providing the ability for modular reuse of the code. Services are not just limited to drive and sensor interaction but can also include implementations for industry floor camera observation, wireless communication, etc. One example of the service to hardware mapping is illustrated against the hypothetical robot in Figure 3-8[62]. The interaction of services in a certain control system is defined through the use of an XML configuration manifest file. MRDS provide developers with XML format as well as an interface for each service to interact with HTTP calls into the service.

MRDS is implemented using .NET; it is therefore required to program services in .NET language. Preferred implementation language of MRDS services is C#, though other languages such as Visual Basic are an alternative. MRDS is platform-dependent since

Windows and .NET are necessary to develop application. This is a limitation when resource-constrained devices are the targets.

3.3 Evaluation Summary

Table 3-3 summarizes our evaluation of the related works with respect to the requirements in Table 3-1. The discussed works clearly do not meet all of our requirements.

Table 3-3: Summary of the evaluation of the related work for robot application development

Related Works / Requirements	Player	HSR-I	Pearl	Care-O-Bot	Miro	SOAP-based robot control	REALab	Robopedia	MRDS
Robotics platform (Robots) independent	Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Application independent	satisfied	Not Satisfied	Satisfied	Satisfied	Not discussed	Not discussed	Not discussed	Not Satisfied	Not discussed
Separation of concerns among components to increase flexibility and maintainability	Not discussed	Not discussed	Not discussed	Not discussed	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Lightweight	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not discussed	Satisfied	Not Satisfied

Easy to use	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Not discussed	Satisfied	Not Satisfied
Device Independent	Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Application independent	Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied	Satisfied	Satisfied	Not Satisfied	Satisfied
OS/language Independent	Satisfied	Not discussed	Not discussed	Not discussed	Satisfied	Satisfied	Satisfied	Satisfied	Not Satisfied
Extensible	Satisfied	Not discussed	Not Satisfied	Not Satisfied	Satisfied	Satisfied	Not discussed	Satisfied	Satisfied

3.4 Chapter Summary

In this chapter, we stated the general and interface specific requirements for companion robots applications development and deployment. We introduced related works, categorized them into three categories: Proprietary solutions; standard based solutions, CORBA and SOAP; REST based solutions. We then evaluated these works with respect to our requirements, however, none of these works is meeting all our requirements, therefore, based on this evaluation and the appropriateness of REST we propose a

RESTful architecture for development and deployment of companion robots applications. This architecture is presented in the next chapter.

Chapter 4:
A RESTful Architecture for Development and Deployment of Companion Robots Applications

In the previous chapter, we stated the requirements for companion robots applications. With respect to these requirements, RESTful Web services are the most appropriate solution to be used for developing companion robots applications.

This chapter is organized into four sections. In the first section, we introduce our overall architecture for developing and deploying companion robots applications. This architecture is based on the separation of concerns into different layers. In the second section, we introduce our REST interfaces. We present our proposed resources modeling. We then provide an extensive table for the resources along with the HTTP methods that each resource support. In the third section, an operational scenario is presented to illustrate how the architecture works.

We conclude this chapter with an evaluation of our architecture with respect to the requirements.

4.1 The Overall Architecture

As described in Chapter 2, we have three general requirements: Separation of concerns, robot independency and application independency. Taking these requirements into account for our overall architecture, we have separated the concerns into two layers: an Application layer and a Service layer. The service layer is further organised into

high level and low level service layers. The abstraction of high-level services from low-level ones increases the flexibility and maintainability of the system.

The application layer includes the core logic, which interacts and coordinates different services to fulfil a desired task. This allows applications developers to reuse services and compose them to achieve their goals.

Figure 4.1 shows our overall architecture.

High-level services range from robotic utilities such as *MapBuilding* and *Navigation*, to generic database-like services such as *ObjectRepository* and *LineRepository*. The *ObjectRepository* maintains a list of the available objects to be handled by the robot. Some examples of such objects include a TV remote control and a phonebook that the end-user can ask the robot to bring. The *LineRepository* lists the existing lines that a robot can follow to reach specific locations, such as a blue line that leads from the kitchen to the table in the living room. A high-level service may reuse and compose other high-level and/or low-level services to provide a more complex functionality. In Figure 4-1, for instance, the *ObstacleAvoidance* and localization services are composed to build the navigation service, which is used by a more advanced service called *MapBuilding*.

Low-level services allow communication with the actual hardware devices and are categorized according to the type of these devices (e.g. robots, sensors, cameras and actuators).

The interactions between the different layers are carried out through REST interfaces. We chose REST because it is lightweight and platform/language independent. In addition, it provides developers with an easy and unified access to information. We propose two REST interfaces. The first one is used to access the high-level services (R1) and the second to communicate with the low-level services (R2). Separating the

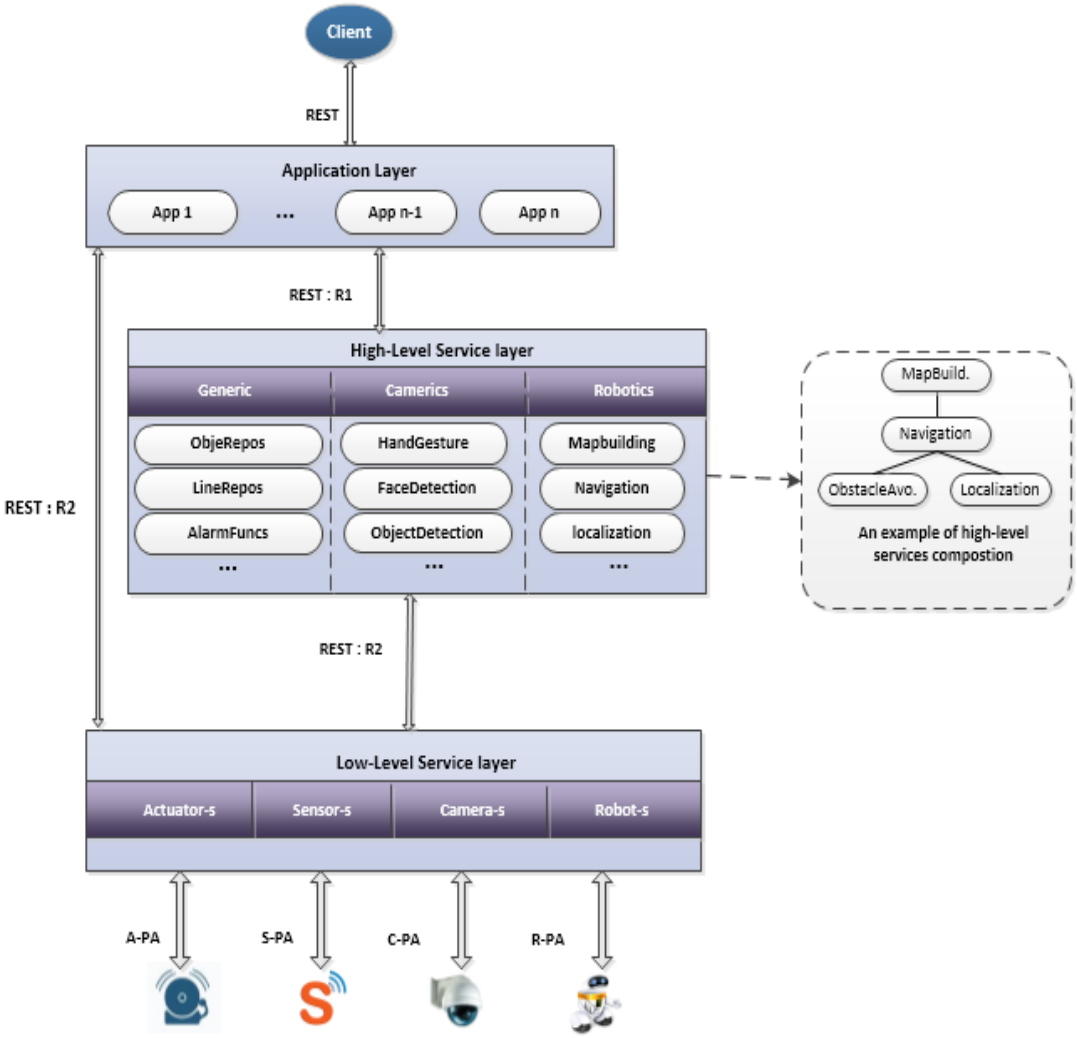


Figure 4-1: Overall Architecture

interfaces into two levels of granularity increases the simplicity and extensibility of the architecture, and broadens the number of application that can be developed. The interface between the client and the application is also REST-based. The low-level

services communicate with the hardware devices via the proprietary interfaces supported by these devices. Low-level services receive requests from an application or a high-level service through R2 and forward the appropriate commands to the hardware devices through their proprietary interfaces.

Our architecture is a distributed architecture. We have different entities with different roles, application provider, high level service provider, and low level service provider that may reside anywhere in the network. ‘

4.2 REST Interfaces

We provided detailed information on REST, its characteristics and principles. Before going to our resources and our proposed model for them we briefly review REST to have a better realization of the prospective sections.

As mentioned in Chapter 2, REST is an architectural style for distributed systems, whose main concept is the resource. Every piece of information that is important enough for a server to provide is exposed as a resource. Each resource has a unique identifier (i.e. URI). In order to manipulate these resources, the client and the server communicate through a standardized interface such as HTTP and exchange representations of these resources. HTTP is not the only protocol REST is based on, however, it is the one most widely used for RESTful Web services. A resource representation is typically a document that captures the current or intended state of a resource. Data representation can be in any format such as HTML, XML or plain text.

4.2.1 Resource modeling

For a better illustration we separate the resource models that we propose for both the high-level and low-level interfaces into two figures.

Figure 4-2 shows our high-level modeling. As shown in the figure, the first resource in the high-level hierarchy is *services*. It offers a subordinate resource for each service category, for instance, separate resources for camera-related (i.e. *camerics*) and robot-related (i.e. *robotics*) high level services. Generic services, such as *objeRepos*, *LineRepos* and *AlarmFuncs* are proposed under the *generics* resource. For example *ObjeRepos* service is a data-base like service that offers the list of available objects and the properties of each object such as location, the color, the last place the object has been, etc. The description of all services is available in Table 4-1.

Each specific service is given a unique identifier and is represented as a separate resource. The unique ID of the service is used to create the URI of its representing resource. Each of these resources may further offer one or more subordinate resources, depending on the specific functionalities they provide. A given camera service might expose a *ColorDetection* and a *faceDetection* sub-resource. This design approach not only simplifies access to the required services of each category, but also increases the system extensibility; i.e. new needs can be addressed by adding new service categories.

The resources for low-level services are shown in Figure 4-3. They are organized following the service categories and classified according to the type of devices that offer them. For example the services offer by a robot is under the *robot* resource. The services

offered by a sensor and an actuator are classified under the *Sensors* and *Actuators* resources, respectively. Finally the services offered by a camera device are placed under the *Cameras* resource.

Under each category, we either have a list of the specific devices available (e.g. the robot with ID 123), or a list of the available sub-categories. The sensors, for instance, may be of different types such as location and temperature sensors. The existing sensor nodes will then be presented under the related sub-categories (e.g. /sensors/location/sensor023). A robot may have a sensor or a camera that is attached to it. These are represented as subordinate resources of the specific robot they belong to, at the same level as the other components of the robot such as the wheels (e.g. robots/robot123/sensors/sensor093). While the robot enables the execution of basic functionalities such as turning the left wheel 25 degrees or moving 20m ahead, it should also allow for atomic execution of somewhat advanced operations. It should be possible, for instance, to ask a robot to move from location 'a' to location 'b' using a single request. This simplicity facilitates fulfilling the statelessness of RESTful services and avoiding conflicts among different services that may arise due to concurrent access to the same resource. We therefore modeled each of these operations as a separate resource. For example the robots that use wheels to move will be operated by *Drive* service to get from one location to another while moving function of the ones which are equipped with legs will be handled by the *Walk* service. Furthermore, robots that have the capability of grabbing objects (i.e. they are equipped with arm and gripper) will get their arms and grippers manipulated by the *Grab* service.

An interesting point in the services offered by a robot (i.e. *drive*, *walk*, *switch*, *grab*) is the actuators that operate those services. Each actuator resource (i.e. *wheels*, *legs*, *switch*, *and arms*) holds other resources under its branch. For example, *wheels* resource is divided to the *left* and *right* resources, each have their own sub-resource. Wheels of a robot can be manipulated separately for a specific purpose depending on the robot hardware design.

Another interesting low-level resource is *Sites*, the end-user homes can be of various sizes, with different plans. In addition, smart homes are more prevalent than ever. They are equipped with different sensors and actuators. To facilitate the management of the available devices including robots, cameras, sensors and actuators and their current locations, and to access to the set of devices currently available at a specific location, we introduced the *Sites* resource.

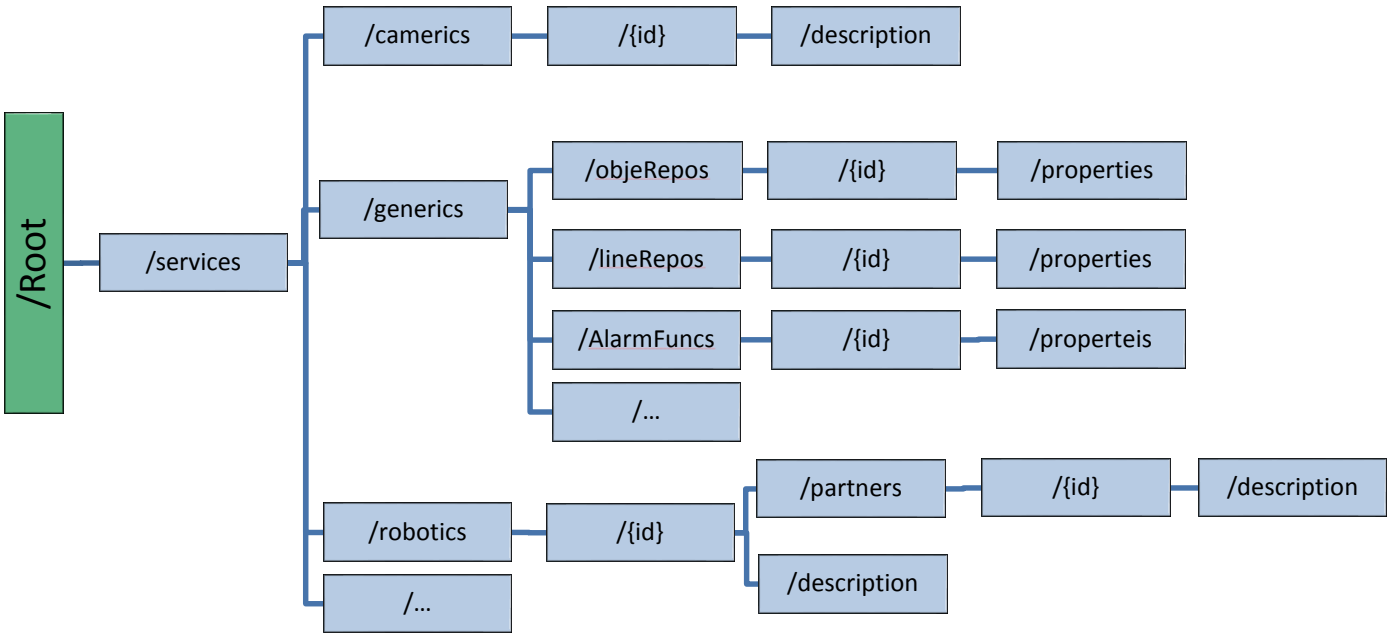


Figure 4-2: Resource model for high-level services: Interface R1

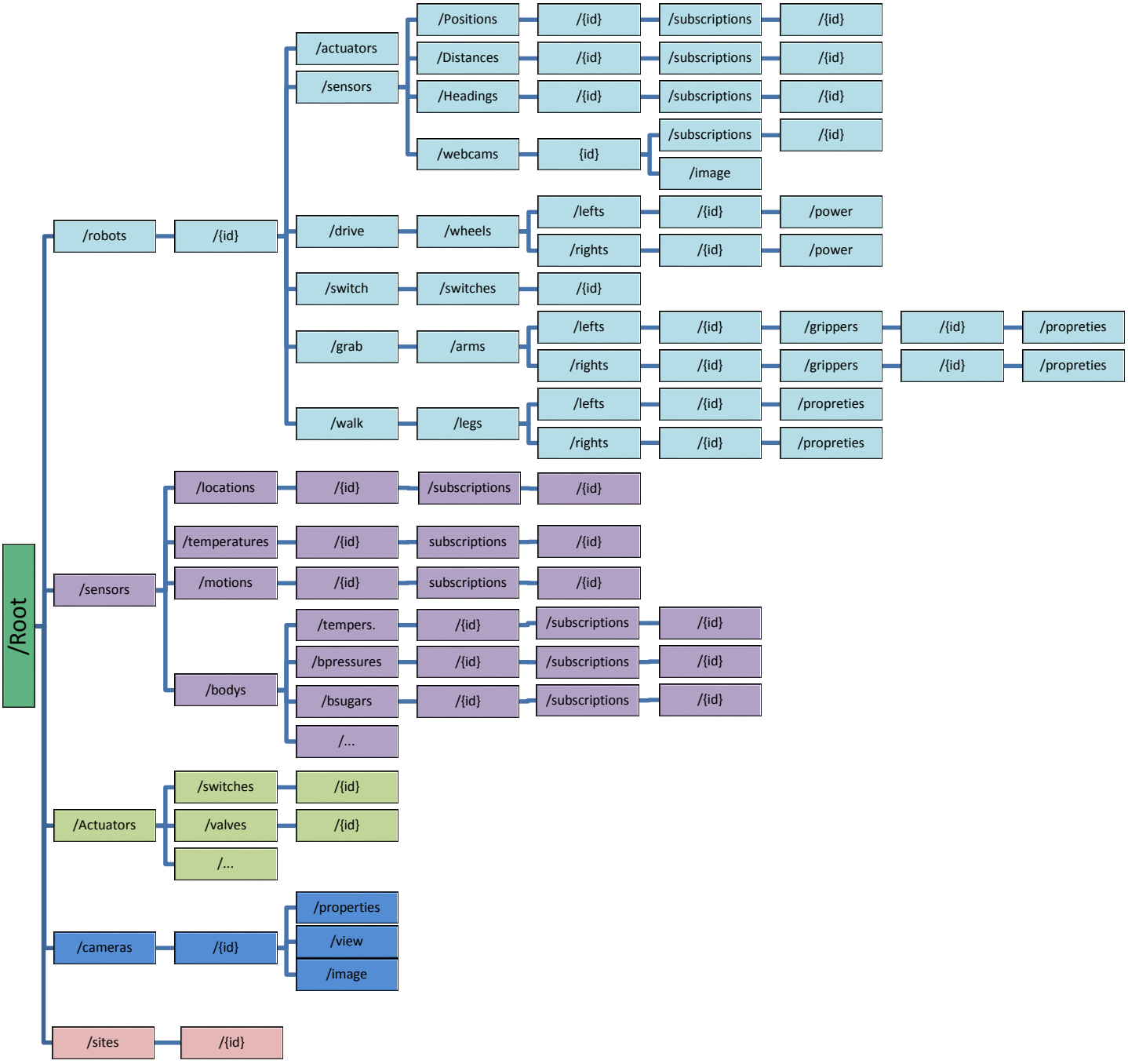


Figure 4-3: Resource model for low-level services: Interface R2

The proposed design for the resources along with the layering architecture described, simplify developing new applications or extending the existing one. This occurs by enabling new resources to simply be added to the system. These resources can be physical resources, high-level services or low level services.

4.2.2 Resources and associated HTTP methods

As described in Chapter 2, in order to use the resources, the client and the server communicate through standardized HTTP interfaces. The four HTTP methods, GET, POST, PUT, and DELETE are used to retrieve, create, update and delete a resource respectively.

Table 4-1 and Table 4-2 describe the defined resources, along with the HTTP methods they support. The *services* resource, for instance, only supports the GET operation; it returns the list of service categories that exist. The resource */sensors/temperature/id* refers to a particular temperature sensor and supports four operations. With GET, we retrieve the sensor readings (i.e. the temperature information). POST creates a new subscription to the sensor reading, to be notified under certain criteria (e.g. periodically). PUT updates the existing sensor subscription criteria. The DELETE operation removes the sensor from the list of available temperature devices. The resource *robots/id/actuators* provide a list of the actuators that are available on the specific robot such as wheels, arms, etc. The properties of each actuator will then be presented as the parameters.

Another example for the resources is */robots/id/drive/wheels/left/power*. This resource refers to the power of the left wheel of a specific robot. The speed and rotation degree of

the wheel depend on the value of this resource. With GET we retrieve the current value of the power of the left wheel and PUT updates the existing value.

Table 4-1: REST resources and the associated HTTP methods: High-level services

Resource	Interface	Description	HTTP action
/services	High level-R1	The list of all service categories provided by the service provider	**GET**: retrieve the list of categories
/services/camerics	High level-R1	The list of all existing high-level camera services	**GET**: retrieve the list of camera services
/services/camerics/id	High level-R1	A specific camera-related service	**GET**: retrieve the parameters the camera-related service **POST**: create new parameter for the camera-related service **PUT**: update/change the parameters of the camera-related service
/services/camerics/id/description	High level-R1	Necessary information about a specific camera-related services	**GET**: retrieve the description of the camera-related service
/services/generics/	High level-R1	The list of all existing high-level generic services	**GET**: retrieve the list of generic services
/services/generics/objeRepos	High level-R1	The list of all the available objects to be handled by the robot	**GET**: retrieve the list of all objects **POST**: add a new object to the list
/services/generics/objeRepos/id	High level-R1	A specific object with all necessary parameters	**GET**: retrieve the information about the object **POST**: add new parameters to the object **PUT**: update/change the parameters of the object

			DELETE: delete the object form the list
/services/generics/objeRepos/id/properties	High level-R1	The properties of a specific object such as last location it has placed	**GET**: retrieve the properties of the object **POST**: create a new property for the object **PUT**: update the properties of the object
/services/generics/lineRepos	High level-R1	The list of all the available lines to be followed by the robot to reach to a specific location	**GET**: retrieve the list of all lines **POST**: add a new line to the list
/services/generics/lineRepos/id	High level-R1	A specific lien with all necessary parameters	**GET**: retrieve the information about the line **POST**: add new parameters to the line **PUT**: update/change the parameters of the line **DELETE**: delete the object form the list
/services/generics/lineRepos/id/properties	High level-R1	The properties of a specific line such the start point and the end point	**GET**: retrieve the properties of the object **POST**: create a new property for the line **PUT**: update/change the properties of the line
/services/generics/alarmFuncs	High level-R1	The list of all the available alarm services such as emergency calls or turning of the lights	**GET**: retrieve the list of all the alarm services
/services/generics/alarmFuncs/id	High level-R1	A specific alarm service with all necessary information	**GET**: retrieve a specific alarm service **POST**: add a new parameter to the alarm service **PUT**: update parameters of the alarm service
/services/generics/alarmFuncs/id/properties	High level-R1	The properties of a specific alarm service such as emergency numbers for emergency call service	**GET**: retrieve the properties of the alarm service **PUT**: update the properties of the alarm service
/services/robotics	High level-R1	The list of all existing high-level robotics services	**GET**: retrieve the list of robotics services

/services/robotics/id	High level-R1	A specific robotics service	**GET**: retrieve the parameters the robotics service **POST**: create new parameter for the robotics service **PUT**: update/change the parameters of the robotics service
/services/robotics/id/description	High level-R1	Necessary information about a specific robotics service	**GET**: retrieve the description of the robotics service
/services/robotics/id/partners	High level-R1	The list of all existing partners of a specific robotics service	**GET**: retrieve the list of the partners of the robotics service **POST**: add a new partner to the list
/services/robotics/id/partners/id	High level-R1	a specific partner of a specific robotics service	**GET**: retrieve the partner of the robotics service **PUT**: update/change parameters of the partner of the robotics service **DELETE**: delete the partner from the list
/services/robotics/id/partners/id/description	High level-R1	Necessary information of a specific partner of a specific robotics service	**GET**: retrieve the description of the partner of the robotics service

Table 4-2: REST resources and the associated HTTP methods: Low-level services

Resource	Interface	Description	HTTP action
/robots	Low level-R2	The list of all available robots	**GET**: retrieve the list of available robots
/robots/id	Low level-R2	A specific robot	**GET**: retrieve the list of all the devices available on the robot **POST**: create a new parameter to the list **DELETE**: delete the robot from the list
/robots/id/actuators	Low level-R2	The list of all available actuators on a specific robot	**GET**: retrieve the list of all the actuators available on the robot **POST**: add a new actuator to the list
/robots/id/sensors	Low level-R2	The list of all available sensor on a specific robot	**GET**: retrieve the list of all the sensor types available on the robot **POST**: add a new sensor type to the list
/robots/id/sensors/positions	Low level-R2	The list of all existing position sensors on a specific robot	**GET**: retrieve the list of all the position sensor available on the robot **POST**: add a new position sensor to the list
/robots/id/sensors/positions/id	Low level-R2	A specific position sensor on a specific robot	**GET**: retrieve the readings of the position sensor **POST**: create a new subscription to the position sensor **PUT**: Update/change subscription criteria of the position sensor **DELETE**: delete the sensor from the list
/robots/id/sensors/positions/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific position sensor	**GET**: retrieve the list of all active subscription to the sensor
/robots/id/sensors/positions/id/subscriptions/id	Low level-R2	A specific subscription of a specific position sensor	**GET**: retrieve the subscription information of the sensor **DELETE**: delete the subscription to the sensor
/robots/id/sensors/distances	Low level-R2	The list of all existing	**GET**: retrieve the list of all the distance sensor

		distance sensors on a specific robot	available on the robot **POST**: add a new distance sensor to the list
/robots/id/sensors/distances/id	Low level-R2	A specific distance sensor on a specific robot	**GET:** retrieve the readings of the distance sensor **POST**: create a new subscription to the distance sensor **PUT:** Update/change subscription criteria of the distance sensor **DELETE:** delete the sensor from the list
/robots/id/sensors/distances/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific distance sensor	**GET:** retrieve the list of all active subscription to the sensor
/robots/id/sensors/distances/id/subscriptions/id	Low level-R2	A specific subscription of a specific distance sensor	**GET:** retrieve the subscription information of the sensor DELETE: delete the subscription to the sensor
/robots/id/sensors/headings	Low level-R2	The list of all existing heading sensors on a specific robot	**GET:** retrieve the list of all the heading sensors available on the robot **POST**: add a new heading sensor to the list
/robots/id/sensors/headings/id	Low level-R2	A specific heading sensor on a specific robot	**GET:** retrieve the readings of the heading sensor **POST**: create a new subscription to the heading sensor **PUT:** Update/change subscription criteria of the heading sensor **DELETE**: delete the sensor from the list
/robots/id/sensors/headings/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific heading sensor	**GET:** retrieve the list of all active subscription of the sensor
/robots/id/sensors/headings/id/subscriptions/id	Low level-R2	A specific subscription of a specific heading sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/robots/id/sensors/webcams	Low level-R2	The list of all existing	**GET:** retrieve the list of all the webcam sensor

		webcam sensor on a specific robot	available on the robot **POST:** add a new webcam sensor to the list
/robots/id/sensors/webcams/id	Low level-R2	A specific webcam sensor on a specific robot	**GET**: retrieve the readings of the webcam sensor **POST:** create a new subscription to the webcam sensor **PUT:** Update/change subscription criteria of the webcam sensor **DELETE:** delete the webcam sensor from the list
/robots/id/sensors/webcams/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific webcam sensor	**GET:** retrieve the list of all active subscription of the sensor
/robots/id/sensors/webcams/id/subscriptions/id	Low level-R2	A specific subscription of a specific webcam sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/robots/id/sensors/webcams/id/image	Low level-R2	Image properties of a specific webcam sensor	**GET:** retrieve the properties of the image captured by the webcam **PUT:** update/change the properties of the image
/robots/id/drive	Low level-R2	Necessary information about the drive status and parameters of a specific robot	**GET:** retrieve the drive parameters and status (stage)
/robots/id/drive/wheels	Low level-R2	The list of all existing wheels of a specific robot	**GET:** retrieve the list of all available wheels of the robot **POST:** add a new wheel to the list
/robots/id/drive/wheels/lefts	Low level-R2	The list of all available left wheels of a specific robot	**GET:** retrieve the list of all the left wheels **POST:** add a new left wheel to the list
/robots/id/drive/wheels/lefts/id	Low level-R2	A specific left wheel of a specific robot	**GET:** retrieve the properties of the left wheel **PUT:** update/change the properties of the left wheel **DELETE:** delete the left wheel from the list

/robots/id/drive/wheels/lefts/id/power	Low level-R2	Related to the actual command to a making a specific left wheel of a specific robot moves, stops or rotate	**GET:** retrieve the current power value of the left wheel **PUT:** Update/change the current value of the left wheel
/robots/id/drive/wheels/rights	Low level-R2	The list of all available right wheels of a specific robot	**GET:** retrieve the list of all the right wheels **POST:** add a new right wheel to the list
/robots/id/drive/wheels/rights/id	Low level-R2	A specific right wheel of a specific robot	**GET:** retrieve the properties of the right wheel **PUT:** update/change the properties of the right wheel **DELETE:** delete the right wheel from the list
/robots/id/drive/wheels/rights/id/power	Low level-R2	Related to the actual command to a making a specific right wheel of a specific robot moves, stops or rotate	**GET:** retrieve the current power value of the right wheel **PUT:** Update/change the current value of the right wheel
/robots/id/switches	Low level-R2	The list of all existing switches on a specific robot	**GET:** retrieve the list of all switches on the robot **POST:** add a new switch to the list
/robots/id/switches/id	Low level-R2	A specific switch on a specific robot	**GET:** retrieve the status and properties of the switch **PUT:** update/change the status and properties of the switch **DELETE:** delete the switch from the list
/robots/id/grab	Low level-R2	Necessary information about the grab status and parameters of a specific robot	**GET:** retrieve the grab parameters and status (stage)
/robots/id/grab/arms	Low level-R2	The list of all existing arms	**GET:** retrieve the list of all available arms of the

		of a specific robot	robot **POST:** add a new arm to the list
/robots/id/grab/arms/lefts	Low level-R2	The list of all available left arms of a specific robot	**GET:** retrieve the list of all the left arms **POST:** add a new left arm to the list
/robots/id/grab/arms/lefts/id	Low level-R2	A specific left arm of a specific robot	**GET:** retrieve the status and properties of the left arm **PUT:** update/change the status and properties of the left arm **DELETE**: delete the left arm from the list
/robots/id/grab/arms/lefts/id/grippers	Low level-R2	The list of all existing grippers of a specific left arm of a specific robot	**GET:** retrieve the list of all the grippers of the arm **POST**: add a new gripper to the list
/robots/id/grab/arms/lefts/id/grippers/id	Low level-R2	A specific gripper of a specific left arm of a specific robot	**GET:** retrieve the status of the gripper **PUT:** update/change the status of the gripper **DELETE:** delete the gripper from the list
/robots/id/grab/arms/lefts/id/grippers/properties	Low level-R2	The necessary properties to handle a specific gripper of a specific left arm of a specific robot	**GET:** retrieve the properties of the gripper **PUT:** update/change the properties of the gripper
/robots/id/grab/arms/rights	Low level-R2	The list of all available right arms of a specific robot	**GET**: retrieve the list of all the right arms **POST:** add a new right arm to the list
/robots/id/grab/arms/rights/id	Low level-R2	A specific right arm of a specific robot	**GET:** retrieve the status and properties of the right arm **PUT:** update/change the status and properties of the right arm **DELETE:** delete the right arm from the list
/robots/id/grab/arms/rights/id/grippers	Low level-R2	The list of all existing grippers of a specific right	**GET:** retrieve the list of all the grippers of the right arm **POST**: add a new gripper

		arm of a specific robot	to the list
/robots/id/grab/arms/rights/id/grippers/id	Low level-R2	A specific gripper of a specific right arm of a specific robot	**GET:** retrieve the status of the gripper **PUT:** update/change the status of the gripper **DELETE**: delete the gripper from the list
/robots/id/grab/arms/rights/id/grippers/id/properties	Low level-R2	The necessary properties to handle a specific gripper of a specific left arm of a specific robot	**GET:** retrieve the properties of the gripper **PUT:** update/change the properties of the gripper
/robots/id/walk	Low level-R2	Necessary information about the walk status and parameters of a specific robot	**GET:** retrieve the walk parameters and status (stage)
/robots/id/walk/legs	Low level-R2	The list of all existing legs of a specific robot	**GET:** retrieve the list of all available legs of the robot **POST:** add a new leg to the list
/robots/id/walk/legs/lefts	Low level-R2	The list of all available left legs of a specific robot	**GET:** retrieve the list of all the left legs **POST:** add a new left leg to the list
/robots/id/walk/legs/lefts/id	Low level-R2	A specific left leg of a specific robot	**GET:** retrieve the status of the left leg **PUT:** update/change the status of the left leg **DELETE:** delete the left leg from the list
/robots/id/walk/legs/lefts/id/properties	Low level-R2	The necessary properties to control a specific left leg of a specific robot	**GET:** retrieve the properties of the left leg **PUT:** update/change the properties of the left leg
/robots/id/walk/legs/rights	Low level-R2	The list of all available right legs of a specific robot	**GET:** retrieve the list of all the right legs **POST:** add a new right leg to the list
/robots/id/walk/legs/rights/id	Low level-R2	A specific right leg of a specific robot	**GET:** retrieve the status of the right leg **PUT:** update/change the status of the right leg

			DELETE: delete the right leg from the list
/robots/id/walk/legs/rights/properties	Low level-R2	The necessary properties to control a specific right leg of a specific robot	**GET:** retrieve the properties of the right leg **PUT:** update/change the properties of the right leg
/sensors	Low level-R2	The list of all existing sensors in the environment in which robot is used or on the end-user	**GET**: retrieve the list of all the sensor types in the environment and on the body of the end-user **POST:** add a new sensor type to the list
/sensors/locations	Low level-R2	The list of all existing location sensors for the end-user	**GET:** retrieve the list of all location sensor for the end-user **POST:** add a new location sensor to the list
/sensors/locations/id	Low level-R2	A specific location sensor on the end-user	**GET:** retrieve the readings of the location sensor **POST:** create a new subscription to the location sensor **PUT:** Update/change subscription criteria of the location sensor **DELETE:** delete the sensor from the list
/sensors/locations/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific location sensor on the end-user	**GET:** retrieve the list of all active subscription of the sensor
/sensors/locations/id/subscriptions/id	Low level-R2	A specific subscription of a specific location sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/sensors/temperatures	Low level-R2	The list of all existing temperature sensors of the environment in which the robot is used	**GET:** retrieve the list of all temperature sensors in the environment **POST:** add a new temperature sensor to the list
/sensors/temperatures/id	Low level-R2	A specific temperature	**GET:** retrieve the readings of the temperature sensor

		sensor of the environment in which the robot is used	**POST:** create a new subscription to the temperature sensor **PUT:** Update/change subscription criteria of the temperature sensor **DELETE:** delete the sensor from the list
/sensors/temperatures/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific temperature sensor of the environment in which the robot is used	**GET:** retrieve the list of all active subscription to the sensor
/sensors/temperatures/id/subscriptions/id	Low level-R2	A specific subscription of a specific temperature sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/sensors/motions	Low level-R2	The list of all existing motions sensors of the environment in which the robot is used	**GET:** retrieve the list of all motions sensors in the environment **POST:** add a new motions sensor to the list
/sensors/ motions /id	Low level-R2	A specific motions sensor of the environment in which the robot is used	**GET:** retrieve the readings of the motions sensor **POST:** create a new subscription to the motions sensor **PUT:** Update/change subscription criteria of the motions sensor **DELETE:** delete the sensor from the list
/sensors/ motions /id/subscriptions	Low level-R2	The list of all active subscriptions to a specific motions sensor of the environment in which the robot is used	**GET:** retrieve the list of all active subscription to the sensor
/sensors/ motions /id/subscriptions/id	Low level-R2	A specific subscription of	**GET:** retrieve the subscription information

		a specific motions sensor	of the sensor DELETE: delete the subscription to the sensor
/sensors/bodys	Low level-R2	The list of all existing sensor of the end-user body	**GET:** retrieve the list of all the en-user body sensor types **POST:** add a new sensor type to the list
/sensors/bodys/tempers	Low level-R2	The list of all available temperature sensor of the end-user body	**GET:** the list of all the body temperature sensors **POST:** add a new sensor to the list
/sensors/bodys/tempers/id	Low level-R2	A specific temperature sensor of the end-user body	**GET:** retrieve the readings of the body temperature sensor **POST:** create a new subscription to the body temperature sensor **PUT:** Update/change subscription criteria of the body temperature sensor **DELETE:** delete the sensor from the list
/sensors/bodys/tempers/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific temperature sensor of the end-user body	**GET:** retrieve the list of all active subscription to the sensor
/sensors/bodys/tempers/id/subscriptions/id	Low level-R2	A specific subscription of a specific temperature sensor	**GET:** retrieve the subscription information of the sensor **DELETE**: delete the subscription to the sensor
/sensors/bodys/ bpressures/	Low level-R2	The list of all available blood pressure sensor of the end-user body	**GET:** the list of all the blood pressure sensors **POST:** add a new sensor to the list
/sensors/bodys/bpressures/id	Low level-R2	A specific blood pressure sensor of the end-user body	**GET:** retrieve the readings of the blood pressure sensor **POST:** create a new subscription to the blood pressure sensor **PUT:** Update/change subscription criteria of the blood pressure sensor

			DELETE: delete the sensor from the list
/sensors/bodys/bpressures/id/subscriptions	Low level-R2	The list of all active subscriptions to a specific blood pressure sensor of the end-user body	**GET:** retrieve the list of all active subscription to the sensor
/sensors/bodys/bpressures/id/subscriptions/id	Low level-R2	A specific subscriptions to a specific blood pressure sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/sensors/bodys/bsugars	Low level-R2	The list of all available blood sugar sensor of the end-user body	**GET:** the list of all the blood sugar sensors POST: add a new sensor to the list
/sensors/bodys/ bsugars /id	Low level-R2	A specific blood sugar sensor of the end-user body	**GET:** retrieve the readings of the blood sugar sensor **POST:** create a new subscription to the blood sugar sensor **PUT:** update/change subscription criteria of the blood sugar sensor **DELETE:** delete the sensor from the list
/sensors/bodys/ bsugars /id/subscriptions	Low level-R2	The list of all active subscriptions to a specific blood sugar sensor of the end-user body	GET: retrieve the list of all active subscription to the sensor
/sensors/bodys/ bsugars /id/subscriptions/id	Low level-R2	A specific subscriptions to a specific blood sugar sensor	**GET:** retrieve the subscription information of the sensor **DELETE:** delete the subscription to the sensor
/actuators	Low level-R2	The list of all existing actuators in the environment in which robot is used	**GET:** retrieve the list of all the actuator types in the environment **POST:** add a new actuator type to the list
/actuators/switches	Low level-R2	The list of available switches in the	**GET:** retrieve the list of all the switches in the environment

		environment in which robot is used	**POST:** add a new switch to the list
/actuators/switches/id	Low level-R2	A specific switch in the environment in which robot is used	**GET:** retrieve the status and properties of the switch **PUT:** update/change the status and properties of the switch **DELETE:** delete the switch from the list
/actuators/valves	Low level-R2	The list of available valves in the environment in which robot is used	**GET:** retrieve the list of all the valves in the environment **POST:** add a new valve to the list
/actuators/valves/id	Low level-R2	A specific valve in the environment in which robot is used	**GET:** retrieve the status and properties of the valve **PUT:** update/change the status and properties of the valve **DELETE:** delete the valve from the list
/cameras	Low level-R2	The list of all existing cameras in the environment in which robot is used	**GET:** retrieve the list of all the cameras in the environment **POST:** add a new camera to the list
/cameras/id	Low level-R2	A specific camera in the environment in which robot is used	**GET:** retrieve the status and location of the camera **PUT:** update/change the status and location of the camera **DELETE**: delete the camera from the list
/cameras/id/properties	Low level-R2	The necessary properties of a specific camera in the environment in which robot is used	**GET:** retrieve the properties of the camera PUT: update/change the properties of the camera
/cameras/id/image	Low level-R2	The image properties of a specific camera in the environment in which robot is	**GET:** retrieve the image properties of the camera PUT: update/change the image properties of the camera

		used	
/cameras/id/view	Low level-R2	The view angle a and zoom of the camera in the environment in which robot is used from which images are captured	**GET:** retrieve the current value of the view angle PUT: update/change the current view angle
/sites	Low level-R2	The list of all defined sites (places) with the available devices of each site	**GET**: retrieve the list of all the sits and devices available in each site **POST:** add a new site to the list
/sites/id	Low level-R2	The list of all available devices of a specific site (place)	**GET:** retrieve the list of all the devices of the site **POST:** add a new device to the list

As described in the Table 4-2 we have defined different types of sensors for the robot, for the environment (e.g. home) and for the end-user body. Different types of robot's sensors include position sensors, distance sensors, heading sensors and webcam sensors. Position sensors supply information about the robot relative coordinates. Once the robot knows its relative coordinates within a given area and together with the knowledge of the surrounding environment, a path of movement from point A to point B can be planned. Distance sensors provide information about the distance between the robot and the objects around it. Heading sensors provide detailed information about the robot position in the term of the direction to which the robot is headed. Webcam sensors are the vision sensors of the robot which allows the robot see the environment around it.

Various types of sensors in the environment include temperature and motion sensors. Integrating the sensors and actuators installed at a home with the robot offer more complex applications that provide better welfare facilities for the targeted population.

Among people are those who need special cares such as continues health monitoring. Essential sensors that detect blood pressure, blood sugar and temperature may be implanted in the body. Integrating these sensors with other resources offers at-home-healthcare applications that increase safety and security.

4.2.3 Illustrative scenario

Figure 4-4 presents an illustrative scenario application called *FollowMe*. An end-user, Sam, asks his robot to help him carry his food tray and then follow him to the living room where he is going to have his food. The robot follows him while remaining at a safe distance.

In this scenario application, Sam is wearing a red shirt and gave the key word 'red' as input to the application. The robot uses its embedded camera to detect the red object and keeps following that object, always using the camera. We assume that the space the scenario is taking place in is obstacle free. It is because that in the case of having obstacles another high-level service, *ObstacleAvoidance*, would have been required which would add extra complexity to the scenario. We also assume that there is no other object with the same color in the location where Sam will be moving. It avoids confusing the robot as to which object to follow.

The scenario goes as following: First, Sam sends a POST request to the *FollowMe* application with the appropriate color (steps 1). The application acknowledges

the request receipt (step 2) and sends a GET request to the *ColorDetection* service to ask for detection of the color red (step 3). *ColorDetection* is one of the camera-related high-level services. In this example, we assume that its ID is 1. In step 4, the *ColorDetection* service requests images from the Camera low-level service, which communicates with the camera device. When the *ColorDetection* service receives the images (step 5), it analyzes them and gives the red object position back to the application (step 6). The application then sends a POST request to the *Drive* service, which instructs the robot to move toward Sam with a specific speed (steps 7-11). The speed information is given in the POST request. If Sam is not found (i.e. no red object is detected in step 6), the application asks the robot to turn around to find him (step 7').

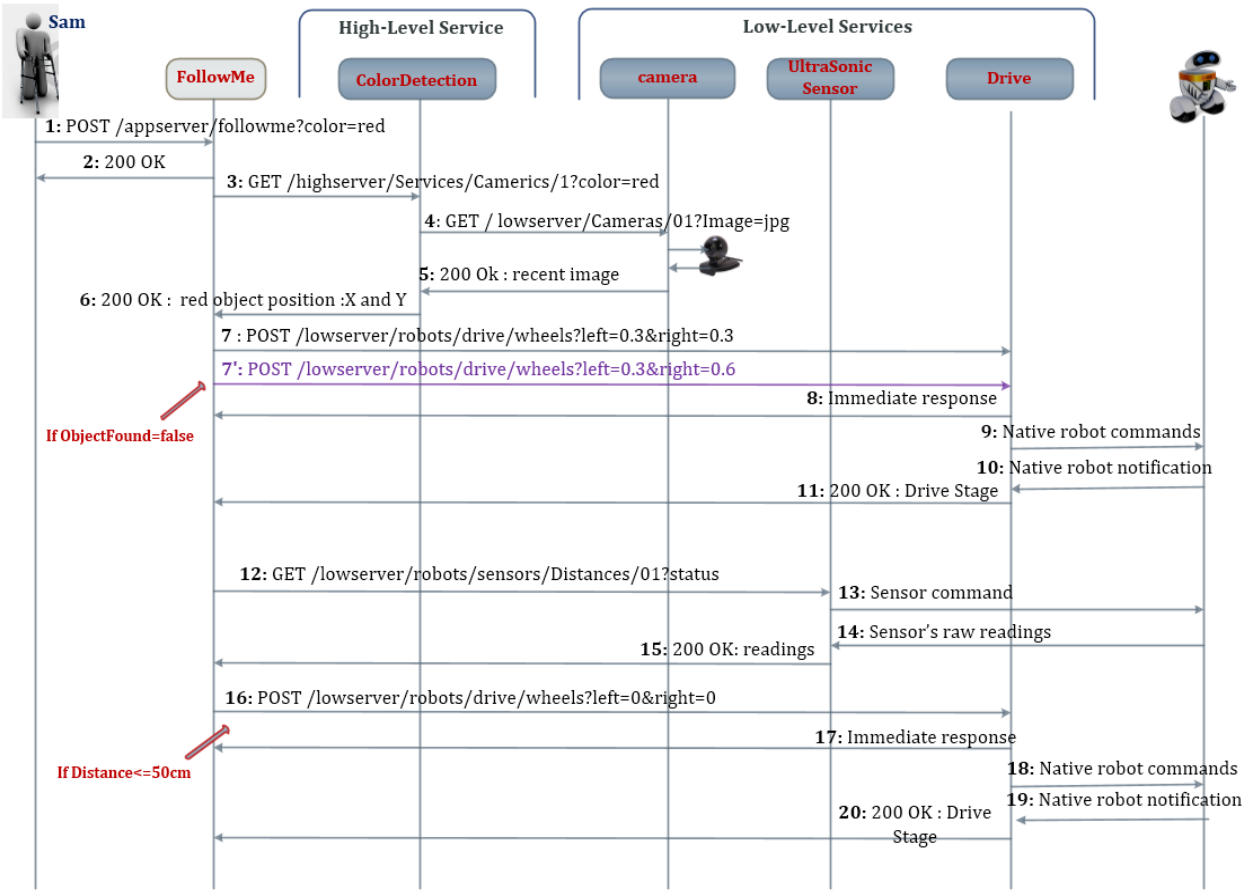


Figure 4-4: Illustrative Scenario Application

While the robot keeps moving, the application retrieves the distance -- between Sam and the robot -- from the *Ultrasonic Sensor* service, which gets the information via the robot's ultrasonic sensor readings (Step 12-15). If the distance falls below 50cm, the application asks the robot to stop moving (steps 16-20).

4.3 Chapter Summary

In this chapter, we proposed a RESTful architecture for the development and deployment of companion robots applications. The main three layers of the architecture were Application layer, High-level service layer and Low-level service layer. To facilitate interaction between these layers we employed two REST interfaces REST: R1 and REST: R2. Using REST as the communication approach requires to define resources.

We proposed resource modeling for each interface separately and provided two extensive tables to describe each resource along with HTTP actions that each supports. At the conclusion of this chapter, we described an illustrative scenario application that included all the steps needed to be completed.

In the previous chapter we designed certain requirements and our proposed architecture, RESTful architecture, satisfies all these requirements. Our proposed architecture is robot independent and application independent. Abstracting the concerns in different layers hides all the lower layer details from the end-users and application developers. Moreover, abstracting high-level services from low-level ones increase the flexibility and maintainability of the system which was our third general requirement. Our interface specific requirements were met by choosing RESTful web services as our communication interface due to the advantages that it offers and we explained in the Chapter 2.

Chapter 5:
Prototype Application and Performance Evaluation

In the previous chapter we presented our proposed architecture for a rapid development and deployment of companion robots applications. We introduced *application* layer and *service* layer. We then further abstracted high-level services from low-level ones; this facilitates rapid development of new applications with minimal cost of change in the system which led to a more flexible and maintainable system. As the communication approach, REST was employed. Combining RESTful Web services with our design approach provided a platform that hides the lower layer heterogeneity and details from developers allowing them to create new applications quickly and simply. In this chapter we describe the variety of applications that can be developed by the system. We later discuss a prototype application we implemented along with its performance evaluation.

5.1 Application variety

Following is four examples of application areas and the services used:

- Fetch and Carry Tasks
  - Task : Lift and carry stuff (e.g. Garbage bag)
  - Services used:
    - High-level: *ObjeRepos*, *ImageProcessing, Navigation, ObstacleAvoidance*

- Low-level: *Drive, Grab, Camera, Distance Sensor, Heading Sensor* (optional),*Position Sensor* (optional)

- Mobility and memory Support
  - Task : Walking aid and reminding of the task (e.g. Turn off the oven)
  - Services used:
    - High-level: *ImageProcessing, Navigation, ObstacleAvoidance, MapBuilding* (optional), *AlarmFuncs(*reminder),
    - Low-level: *Drive, Distance Sensor, Camera, Heading sensor*
- Household Tasks
  - Task : Cleaning the house (e.g. replace the dishes)
  - Services used:
    - High-level: *ImageProcessing, Navigation, ObstacleAvoidance*
    - Low-level: *Drive, Grab, Distance Sensor, Position sensor*
- Personal Security
  - Task : Health monitoring and reporting (e.g. Sending a message to the emergency center in case of falling)
  - Services used :

- High-level: *Image Processing, Navigation, AlarmFuncs* (emergency call)
- Low-level: *Health monitoring sensors* (e.g. Falling detection, Blood pressure, etc.) , *Drive, Switch* (optional)

5.2 Prototype implementation

As a prototype, we implemented the scenario application presented in Chapter 4, in Section 4.2.3. The prototype includes the *FollowMe* application along with the associated high-level and low-level services as shown in Figure 5-1. A LEGO MINDSTORMS NXT robot and a Logitech QuickCam Communicate STX webcam is used. The ultrasonic sensor is embedded in the LEGO robot. However, the robot does not come with an embedded camera. To avoid complex computations to infer the robot position in relation to the red object from the position of both objects with respect to the camera, we placed the camera on top of the robot. In this case, the position of the robot in relation to the red object is the same as the position of the camera in relation to the object.

The prototype was run in an obstacle-free area, where the robot and the red object were placed in area covered by a webcam.

Figure 5-1 shows a picture taken while the robot is following a red object:

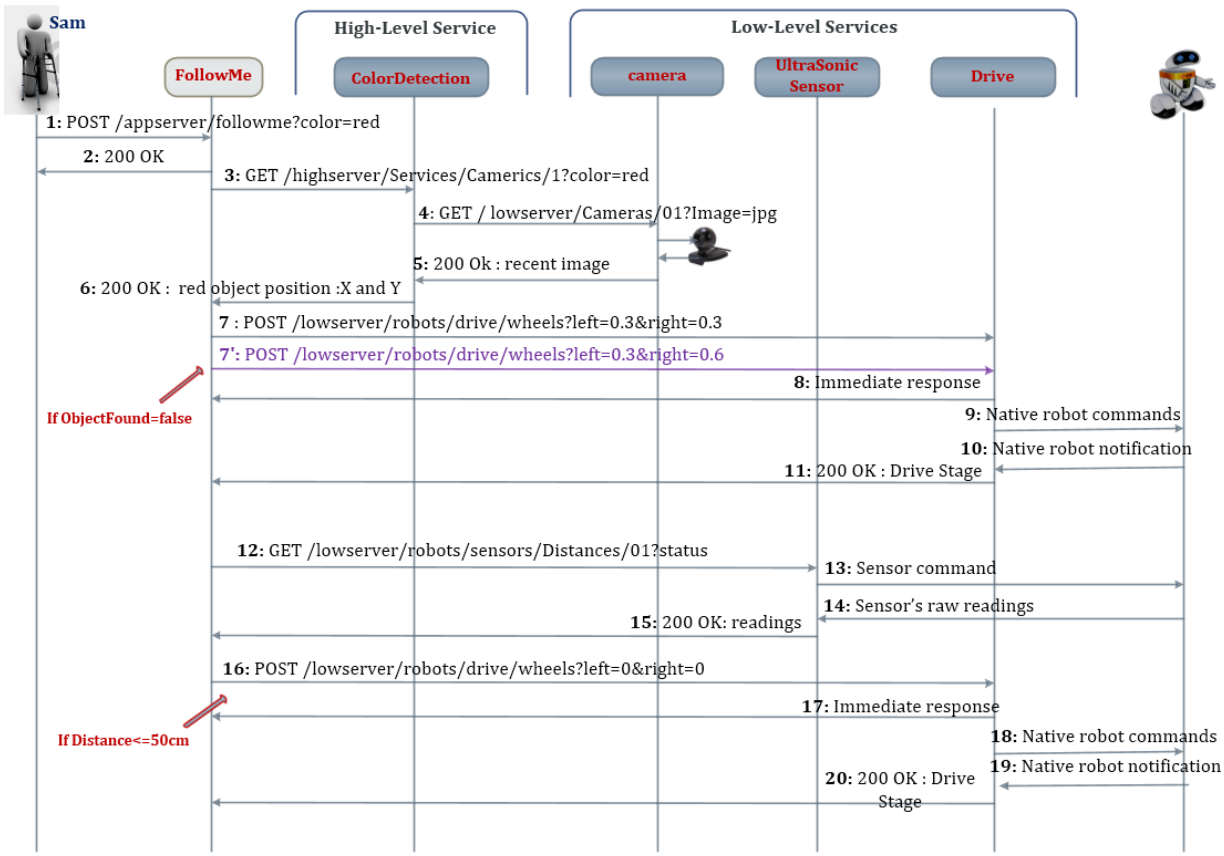


Figure 5-1: (above): Prototype scenario ;(below): LEGO following the red object

As we mentioned earlier, we have two assumptions for this scenario: First, that there is no obstacle in the way so we do not introduce the challenge of obstacle avoidance algorithm on the prototype; Second, the end-user and the robot are in the same area under camera surveillance so there is no need hand over algorithms to guide the robot from one site with camera 1 to the other site with camera 2.

5.2.1 Experiment Setup

We distributed the system using four computers. They are all connected to the same local area network. Figure 5-2 shows the prototype setup. We ran the client, the *FollowMe* application, and the high-level service on separate machines. The low-level services were all run on a single machine. The LEGO is connected to the low-level services` machine

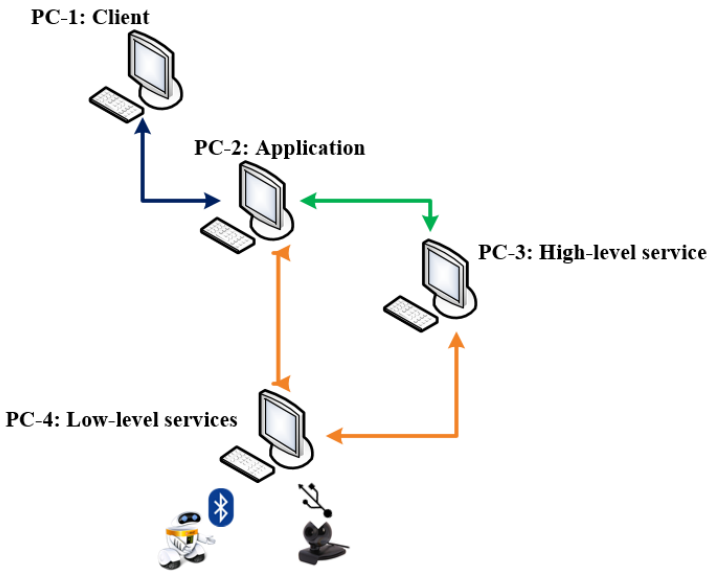


Figure 5-2: Prototype Setup

through Bluetooth interface. The Webcam is connected to the same machine through USB interface.

The operating systems installed on all machines were Windows 7. The hardware characteristics and the specification of our test environment are described in the Table 5-1 shown below.

Table 5-1: Specification of the test environment

Computers	Software Module	Hardware Configuration		
		CPU Model	CPU Speed	RAM
PC-1	• End-user (client) • Windows 7	Intel® Core ™ Duo E6550	2.33 GHz	2 GB
PC-2	• FollowMe (application) • Windows 7	Intel® Core ™ Duo E6550	2.33 GHz	2 GB
PC-3	• ImageProcessing (H.L. service) • Windows 7	Intel® Core ™ Duo E6550	2.33 GHz	2 GB
PC-4	• Drive , UltraSonicSensor (L.L. services) • Windows 7	2 Intel® Xeon® CPU X3450	2.67 GHz	4GB

5.2.2 Software tools: Microsoft Robotics Studio Developer

MRDS introduces a new way to program robots in the Windows environment. MRDS offers a set of tools and APIs that simplify the development and execution of robotics-based applications. These applications may be communicating with real or simulated robots. The MRDS`s Visual Simulation Environment is a 3D simulator with full physics simulation that can be used to prototype new algorithms or robots. MRDS provides support for a wide variety of robot hardware, including LEGO MINDSTORMS.

Key portions of the code for MRDS are available in source form. This offers many opportunities for programmers to write new services that integrate directly into the system.

The Microsoft Robotics Developer Studio Software Development Kit (SDK) consists of two main components. The CCR and DSS comprise the run - time environment. The tools come with programming examples and building blocks for user applications to help programmers understanding the concepts introduced in the tools [14].

5.2.2.1 Concurrency and Coordination Runtime (CCR)

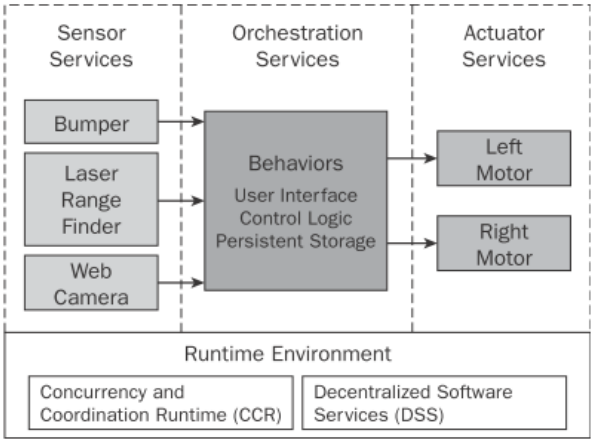


Figure 5-4: Example of service orchestration (taken from [14] **)**

The CCR is a lightweight library that is supplied as a .NET DLL. It is designed to handle asynchronous communication between loosely coupled services that are running concurrently.

It provides classes and methods to help with concurrency, coordination, and failure handling. CCR offers the ability of writing pieces of code that operate independently. When a message is received, it is placed in a queue, called a port, until it can be

processed by the receiver. The CCR library provides the necessary constructs when it is necessary to wait until two or more operations have completed [14].

Figure 5-3 shows an example of how the services might be orchestrated to control a robot. Every MRDS application will contain one or more services. Combining these services and transmitting messages between them, whether they are located on the same or different computers, is one of the tasks of DSS.

5.2.2.2 The Decentralized Software Services (DSS)

DSS is another library of Microsoft robotics tools. An application built with DSS consists of multiple independent services running in parallel. Each service has a state associated with it and particular types of messages that it receives called 'Operations' [14]. When a service receives a message, it may change its state and then send notifications to other services. The state of a service can be gained programmatically by sending a GET message to the service or it can be retrieved and displayed using a web browser. Services

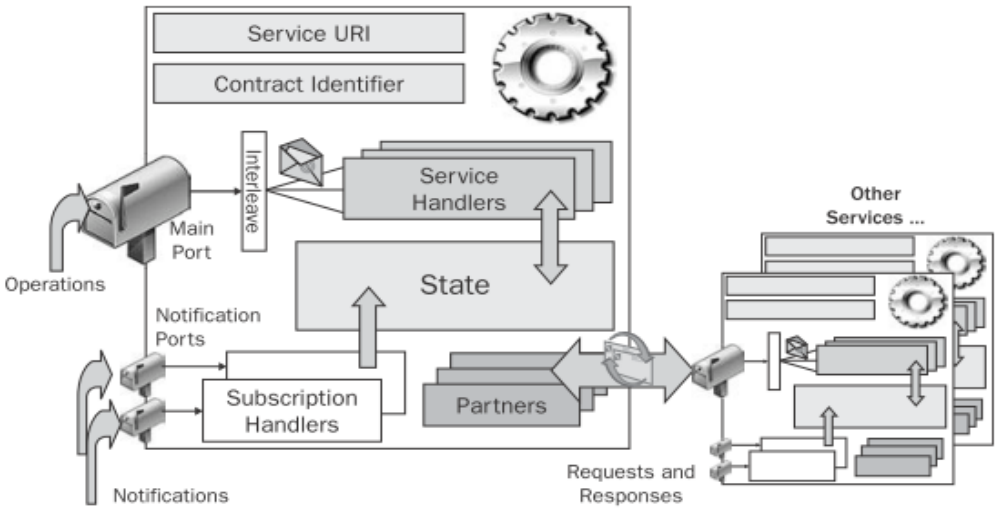


Figure 5-5: A DSS service components (taken from [14])

may subscribe to get notification when the state of a service changes or when other events occur.

Services may also partner with other services so that they can send messages to those services and receive responses [14]. Figure 5-4 shows a DSS service components and the interaction between those components.

MRDS has the role of service provider in our application. It provides us with the services that we need to develop the 'FollowMe' application. Instead of writing each service from scratch, we use them as provided by the tool to achieve our goal. The high-level service , *ImageProcessing*, and low-level services, *Drive, webcam and Ultrasonic Sensor*, in the prototype were implemented and run using MRDS. The tool also provides the proprietary interfaces of the LEGO robot and the Webcam. The low-level services communicate with the camera and with the robot using RoboticsCommon Dynamic Link Library (DLL), which contains most of the important robotics services in MRDS.

5.3 Performance evaluation

In this section the performance evaluation of our system will be discussed in the context of a set of performance metrics.

5.3.1 Performance metrics

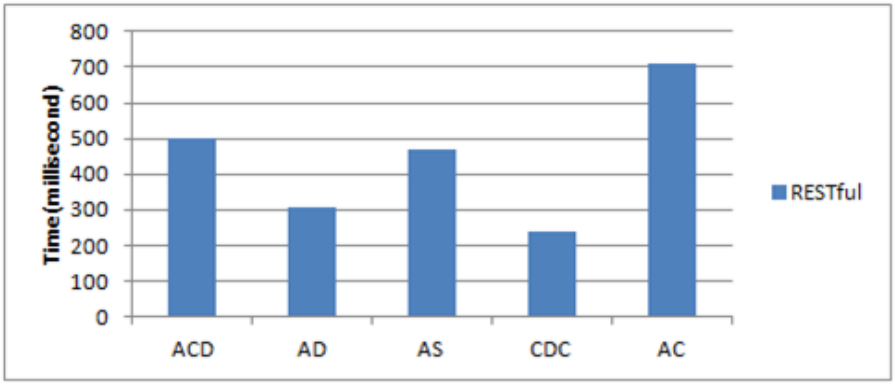
The performance of the prototype is assessed in terms of time delay and network load. We measured the end-to-end time delay from the end-user point of view, as well as the execution time delays for each of the services involved in the application provisioning.

The end-to- end time delay is the time difference between when an end- user sends a follow-me request to the application, and the time the robot starts moving, called *End user-Robot* (ER). The delays for the other parts of the scenario (e.g. the robots following the end-user) are not included in the end-to-end delay because they are human-reaction dependent (e.g. how fast the end-user is moving). A service execution time delay is the time difference between when a request is sent to the service and when a response is received. For instance, the execution time delay for the *ColorDetection* service is the time difference between when the application sends a request to the service (for an object position) and when it gets the object position, called *Application-Camera* (AC). The delays are measured in seconds.

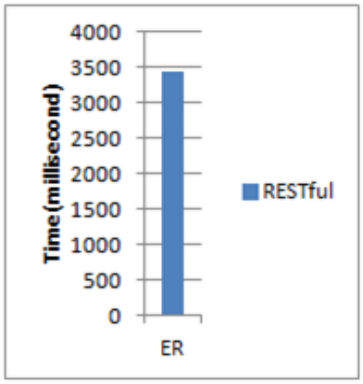
The network load indicates the total number of bytes sent and received for the execution of a given request. The network load to receive an object position, for instance, is the number of bytes sent and received by the application to execute the *ColorDetection* service. The end-to-end network load for the execution of a follow-me request includes the loads for getting an object position and for asking the robot to move. The message payloads related to image exchange (e.g. step 5 in Figure 5.1) are not counted because they depend on the image type (e.g. JPEG, GIF) and size.

5.3.2 Performance analysis

The measurements were taken using CommView, a network monitor and analyze tools for local area networks, and they are calculated as an average of 10 experiments. Figure 5-5 shows the performance result for the average time delay. As shown in this figure, the time delay for *ColorDetection-Camera* (CDC) has the lowest amount. The reason is that the interaction between the webcam service, which



(a)



(b)

ACD: Application GET object position from *ColorDetection*
AD: Application POST robot commands to the *Drive*
AS:Application GET distance from the *Ultrasonic*
CDC: *ColorDetection* GET image from *Camera*
AC: Application - webcam end to end
ER: End-user - Robot end to end

Figure 5-6: System performance in term of response delay

communicates with the actual webcam device, and the *ColorDetection* service does not need any processing since it is just the image captured by the webcam. *Application-ColorDetection* (ACD) spends more time for its execution. It is expected because image processing, carried out by *ColorDetection*, requires the biggest amount of time among the other processes. The second highest response time belongs to the *Application-Sensor* (AS). This interaction needs the *UltraSonicSensor* service processes the sensor reading received by the distance sensor (ultrasonic sensor) and checks for threshold (50cm for our scenario) to notify the application.

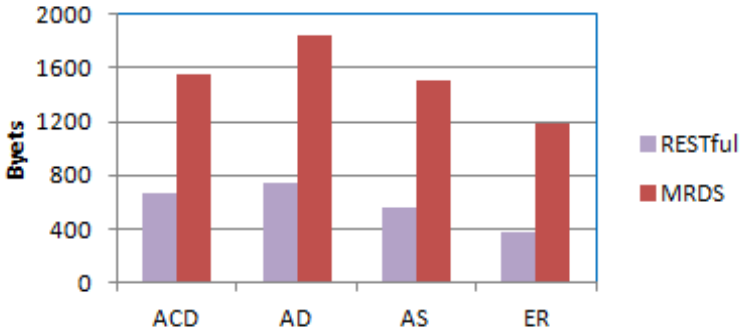


Figure 5-7: System performance in term of network load

The more complex the request, the more entities involved, the longer the delay, however, all delays are reasonable. This concept is applicable to ER as well. The total delay for a "follow me" request is in the range of 3.5 seconds, which remains acceptable from the end-user point of view.

Figure 5-6 shows the network load for different requests. For a better illustration of the advantage of REST compared to SOAP, we implemented the same application and services using MRDS basic building blocks where a SOAP-based protocol is used for the communication between the services over a distributed environment.

As this figure shows, RESTful interfaces have an average of 2 to 3 times less overhead compared to SOAP-based interfaces. This improvement is considerable when resource-constrained devices are the targets. The request which induced the most network load was the request to communicate with the Drive service (*Application-Drive* (AD)). This is because the request sent by the application needs to include different types of parameters to guide the service to correctly operate the robot, such as the speed information. The response payload also contributes to the total load by including the drive stage as well as robot's speed and the covered distance. From the other side, the least network load belongs to the end-user request. It is reasonable because the end-user request is a simple REST request issued through a web browser with the input `red` which determines the color that the robot should follow. The response to this request is simply a successful response code `200`. Then the actual response to the end-user is the robot moving.

5.4 Chapter Summary

In this chapter, we addressed the implementation of our prototype for our architecture proposed in the previous chapter. The implemented protocol and mechanisms are based on HTTP protocol that let us provide RESTful Web services as the communication interface. We ran the prototype in a local area network environment. We used four computers of the network, an actual LEGO robot and a webcam to develop and test the application. Microsoft Robotics Studio Developer (MRDS) was used as the development and deployment tools.

We evaluated the performance of our system with a small version of the corresponding architecture and obtained our results from 10 experiments. We have defined performance

metrics to evaluate the feasibility and efficiency or our architecture and collected the corresponding results from network monitoring tools.

Through the experiments we learnt that our proposed architecture is a promising approach for the rapid development and deployment of companion robots applications. The developers can develop their applications using our solution. It provides them with a platform that enables them to develop their applications by simply reusing and composing different services, and using the designed interfaces for the interaction between the required services, to achieve the goal. We also found that RESTful web services are the best existing solution for a simple and unified development while being suitable for resource-constrained devices.

Chapter 6:
Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis and discuss the remaining issues which can be considered as potential future work.

6.1 Summary of Contributions

Improving the quality of life for the elderly and disabled persons is essential for the society. Statistics reveal that the number of persons requiring home health care in the year 2040 will make up nearly 3.5 % of the population [52]. Companion Robot is targeted technology that enables this population to live independently with a good quality of life and longer in their own homes.

The numerous applications and robots technologies in the home-care domain, reusing and extending existing system effectively for individual projects are all important issues and challenges in this filed.

Future population of the elderly and disabled people will require the development of new and more sophisticated applications. It is important that the developers be able to reuse the existing applications and create new ones that realize more complex services.

In this thesis, we examined three main challenges in this domain: applications variety, robots heterogeneity and resource-constrained devices.

Application variety relates to the different demands of the individuals. Robot heterogeneity relates to the different needs, for which robots are designed, this suggest

that robots have different capabilities. The third challenge stems from the rapid progress in robot technology, which may lead to robots with higher processing capabilities, these robots are manoeuvred through devices such as Smartphone, PDAs, etc. Such devices are classified as resource-constrained devices with limited power and computation resource. This raises concerns when designing architecture that enables the development and deployment of companion robots applications.

Our thesis went further on to general and interface specific requirements for rapid development and deployment of companion robots applications. An extensive survey was conducted on existing solutions in the research domain. These solutions were then divided into non-standards based, standards based (CORBA and SOAP) and REST based categories. These categories were evaluated with respect to our requirements. We concluded that none of the solutions that currently exist meet all our derived requirements necessary for companion robots applications development and deployment.

As a core contribution of the thesis, we proposed an architecture that enables the rapid development and deployment of companion robots applications. Our architecture is REST-based and consists of different layers, ranging from low-level services to communicate with hardware devices to high-level application components for interacting with end users. The architecture enables different types of companion robots applications and can operate on heterogonous robots. Furthermore, REST resources were modeled and a detailed table providing the descriptions of the resources and the HTTP action that each resource support was presented.

To demonstrate our architecture, we implemented a prototype for an application called '*FollowMe*'. We evaluated the performance of the system based on two metrics: time delay and network load. We measured the time delay from the end-user point of view as well as the time of execution of each request. We concluded that the delay was reasonable for all requests. The network load results also showed that REST has superiority over SOAP due to its light-weight characteristics. Based on the overall results we concluded that our architecture is a promising approach for developing and deploying companion robots applications.

6.2 Future Work

The presented work is primarily focused on companion robots applications. In our implementation we assumed that the robot and the individual are in area surveyed by one camera. It is probable that more than one cameras are needed to guide the robot through different sites in the home, therefore a protocol is needed to hand over the robot from site A covered by one camera (e.g. camera 1) to the site B covered by another camera (e.g. Camera 2) without losing visual of the robot.

In this field security and safety is a significant consideration, a potential future extension may be to adjust the architecture to support Quality of Services (QoS). The need to adjust the architecture to include QoS is evident when the people in question require health monitoring or are prone to confusion due to memory loss. Prioritization is one of the QoS that may be added to the architecture, to aid in emergency situations that require indicating trouble, as well as sensing a changed condition in the home and with the

individual. The robot needs to perform different tasks in an order to insure the safety of the individual first and foremost.

Bibliography

[1] R. D. Schraft, C. Schaeffer, and T. May, "Care-O-bot/sup TM/: the concept of a system for assisting elderly or disabled persons in home environments," in *24th Annual Conference of the IEEE Industrial Electronics Society, IECON '98*, 1998, vol. 4, pp. 2476–2481.

[2] A. Makarenko, A. Brooks, and T. Kaupp, "Orca : Components for Robotics," in *International Conference on Intelligent Robots and Systems (IROS'06), Workshop on Robotic Standardization*, 2006.

[3] F. Heckel, T. Blakely, M. Dixon, C. Wilson, and W. D. Smart, "The WURDE Robotics Middleware and RIDE Multi-Robot Tele-Operation Interface," in *AAAI Mobile Robot Competition 2006: Papers from the AAAI Workshop*, 2007.

[4] M. E. Pollack, S. Engberg, J. T. Matthews, J. Dunbar-jacob, C. E. Mccarthy, and S. Thrun, "Pearl : A Mobile Robotic Assistant for the Elderly," in *Workshop on Automation as Caregiver: the Role of Intelligent Technology in Elder Care (AAAI)*, 2002.

[5] B. P. Gerkey and R. T. Vaughan, "The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems," in *International Conference on Advanced Robotics (ICAR)*, 2003, no. Icar, pp. 317–323.

[6] H. Bruyninckx*, "Open robot control software: the OROCOS project," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, pp. 2523–2528, 2001.

[7] C. Buiu and N. Moanta, "Using Web services for Designing a Remote Laboratory for Motion Control of Mobile Robots," in *World Conference on Educational Multimedia, Hypermedia and Telecommunications ((EDMEDIA))*, 2008, pp. 1706–1715.

[8] R. Edwards, L. E. Parker, and D. R. Resseguie, "Robopedia : Leveraging Sensorpedia for Web-Enabled Robot Control," in *International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010, no. March, pp. 193–188.

[9] "Common Object Request Broker Architecture (CORBA)," *Object Management Group (OMG®)*, 2011. [Online]. Available: http://www.omg.org/spec/index.htm. [Accessed: 11-Jun-2013].

[10] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," *W3C*

Recommendation. [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Accessed: 05-Jun-2013].

[11] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California ,Irvine, 2000.

[12] M. Henning, "The rise and fall of CORBA," *Commun. ACM*, vol. 51, no. 8, pp. 52–57, Jun. 2008.

[13] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," *W3C Recommendation*, 2008. [Online]. Available: http://www.w3.org/TR/REC-xml/. [Accessed: 11-Jun-2013].

[14] K. Johns and T. Taylor, *Professional Microsoft Robotics Developer Studio*, 1st ed. Indianapolis, Indiana: Wiley Publishing, Inc., 2008.

[15] "CommeView Network Analyzer," *TAMOSoft*. [Online]. Available: https://www.tamos.com/products/commview/. [Accessed: 11-Jun-2013].

[16] "Robotics," *Galileo Educational Network*, 2003. [Online]. Available: http://www.galileo.org/robotics/intro.html. [Accessed: 10-Jun-2013].

[17] Honda Motor Co. Ltd, "ASIMO Technical Information," Public Relations Division, 2007.

[18] N. A. Weir, D. P. Sierra, and J. F. Jones, "Printed October 2005 A Review of Research in the Field of Nanorobotics," Oak Ridge, TN, 2005.

[19] I. I. F. of Robotics, "History of Industrial Robots From the first installation until today Milestones of Technology and Commercialization," Frankfurt ,Germany, 2012.

[20] "International Organization of Standardization," 1947. [Online]. Available: http://www.iso.org/iso/home.html. [Accessed: 11-Jun-2013].

[21] P. McKerrow, *Introduction to Robotics*. Addison-Wesley, 1998, p. 260.

[22] P. Lima and M. I. Ribeiro, "MOBILE ROBOTICS," *Instituto Superior Técnico/Instituto de Sistemas e Robótica*, 2002. [Online]. Available: http://users.isr.ist.utl.pt/~mir/cadeiras/robmovel/Introduction.pdf.

[23] N. S. Board, *AUTONOMOUS VEHICLES IN SUPPORT OF NAVAL OPERATIONS*. Washington, D.C.: THE NATIONAL ACADEMIES PRESS, 2005, p. 256.

[24] S. Hirose, “Three Basic Types of Locomotion in Mobile Robots,” in *5th International Conference on Advanced Robotics, “Robots in Unstructured Environments” (ICAR)*, 1991, pp. 12 – 17 vol.1.

[25] “International Federation of Robotics,” 1987. [Online]. Available: http://www.ifr.org/home/. [Accessed: 11-Jun-2013].

[26] A. Haasch, S. Hohenner, M. Kleinehagenbrock, S. Lang, I. Toptsis, G. A. Fink, J. Fritsch, B. Wrede, and G. Sagerer, “BIRON – The Bielefeld Robot Companion,” in *International Workshop on Advances in Service Robots*, 2004, no. May.

[27] “A Multi-Purpose Robotic Platform,” *Mesa Robotics, Inc.* [Online]. Available: http://www.mesa-robotics.com/acer.html. [Accessed: 10-Jun-2013].

[28] J. E. Speich and J. Rosen, “Medical Robotics,” *Biomaterials and Biomedical Engineering*. Marcel Dekker, Inc., 2004.

[29] K. Tae-gyu, “Robots to Replace Native English Teachers,” *The Korea Times*, Seodaemun-gu, Seoul, 27-Jan-2010.

[30] J. Hsu, “South Korean Robot English Teachers Are Go,” *POPSCI*, 2010.

[31] J. Forlizzi and C. Disalvo, “Service Robots in the Domestic Environment : A Study of the Roomba Vacuum in the Home,” in *HRI’06*, 2006, pp. 258–265.

[32] Information and Robot Technology Research Initiative, “Cleaning and Tidying Technology for Home-Assistant Robots,” *The University of Tokyo*, Tokyo, 2008.

[33] S. Inc, “A Healthcare/Eldercare Robot based on Skilligent Technology,” Revision 7, 2008.

[34] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, “Towards robotic assistants in nursing homes: Challenges and results,” *Socially Interactive Robots, Robotics and Autonomous Systems*, vol. 42, no. 3–4, pp. 271–281, Mar. 2003.

[35] a. Saxena, J. Driemeyer, and a. Y. Ng, “Robotic Grasping of Novel Objects using Vision,” *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, Feb. 2008.

[36] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, “Mechatronic design of NAO humanoid,” in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 769–774.

[37] C. Fu, F. Belqasmi, and E. Canada, “RESTful Web Services for Bridging Presence Service across Technologies and Domains : An Early Feasibility Prototype,” *IEEE Communication Magazine*, no. December, pp. 92–100, Dec-2010.

[38] A. D. Birrell and B. J. Nelson, “Implementing remote procedure calls,” *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39–59, Feb. 1984.

[39] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,” *W3C Recommendation*, 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/. [Accessed: 09-Jun-2013].

[40] M. ELKSTEIN, “Learn REST: A Tutorial.” [Online]. Available: http://rest.elkstein.org/. [Accessed: 11-Jun-2013].

[41] J. Reynolds and J. Postel, *FILE TRANSFER PROTOCOL (FTP), RFC 959*. 1985.

[42] F. A. et Al., *The Internet Gopher Protocol (a distributed document search and retrieval protocol), RFC 1436*. 1993.

[43] F. Davis, B. Kahle, H. Morris, J. Salem, and T. Shen, “WAIS Interface Protocol Prototype Functional Specification.” Thinking Machines Corporation, 1990.

[44] L. Richardson and S. Ruby, *RESTful Web Services Web services for the real world*, 1st ed. O’Reilly and Associates, 2007, p. 454.

[45] M. Hadley, “Web Application Description Language,” *W3C Recommendation*, 2009. [Online]. Available: http://www.w3.org/Submission/wadl/. [Accessed: 11-Jun-2013].

[46] F. Belqasmi and R. Glitho, “RESTful Web Services for Service Provisioning in Next-Generation Networks : A Survey,” no. December, pp. 66–73, 2011.

[47] D. Crockford, *The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627*. 2006.

[48] Members of the W3C HTML Working Group, “XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition),” *W3C Recommendation*, 2000. [Online]. Available: http://www.w3.org/TR/xhtml1/#xhtml. [Accessed: 11-Jun-2013].

[49] T. H. J. Collett and B. A. Macdonald, “Player 2 . 0 : Toward a Practical Robot Programming Framework,” in *Australasian Conference on Robotics and Automation (ACRA)*, 2005.

[50] Sun Microsystems, *XDR: External Data Representation standard, RFC 1014*. 1987.

[51] Z. Dehuai, X. Gang, and W. Hai, "Study on Teleoperated Home Care Mobile Robot," in *Robotics and Biomimetics (ROBIO)*, 2007, pp. 43–46.

[52] D. C. Schaeffer, "Care-O-bot ™: A System for Assisting Elderly or Disabled Persons in Home Environments," in *AAATE 99, 5th European Conference for the Advancement of Assistive Technology*, 1999, pp. 1–6.

[53] Y. Sakagami, R. Watanabe, and C. Aoyama, "The intelligent ASIMO: System overview and integration," in *International Conference on Intelligent Robots and Systems, 2002*, 2002, no. October, pp. 2478 – 2483 vol.3.

[54] D. Calisi, A. Censi, L. Iocchi, and D. Nardi, "OpenRDK : a modular framework for robotic software development," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1872 – 1877.

[55] A. Hans Utz, Stefan Sablatnög, Stefan Enderle and G. Kraetzschmar, "Miro — Middleware for Mobile Robot Applications," *IEEE Transactions on Robotics*, vol. 18, no. 4, pp. 493–497, 2002.

[56] A. Gokhale, B. Kumar, and A. Sahuguet, "Reinventing the Wheel? CORBA vs. Web Services," in *International World Wide Web Conference*, 2002.

[57] H. R. Sheikh, "Comparing CORBA and Web-Services in view of a Service Oriented Architecture," *International Journal of Computer Applications*, vol. 39, no. 6, pp. 47–55, 2012.

[58] B. K. Kim, M. Miyazaki, K. Ohba, S. Hirai, and K. Tanie, "Web Services Based Robot Control Platform for Ubiquitous Functions," in *International Conference on Robotics and Automation, ICRA 2005*, 2005, no. April, pp. 691–696.

[59] C. Pautasso and F. Leymann, "RESTful Web Services vs . ' Big ' Web Services : Making the Right Architectural Decision," in *17th International World Wide Web Conference (WWW2008)*, 2008, pp. 805–814.

[60] F. P. and F. P. Eleri Cardozo, Eliane Guimar˜aes, Lucio Rocha, Ricardo Souza, "A Platform for Networked Robotics," in *International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 1000–1005.

[61] J. S. Cepeda, L. Chaimowicz, and R. Soto, "Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics," in *Latin American Robotics Symposium and Intelligent Robotics Meeting (LARS)*, 2010, pp. 7–12.

[62] B. Y. J. Jackson, “Microsoft Robotics Studio: A Technical Introduction,” *IEEE Robotics and Automation Magazine*, vol. 14, no. 4, pp. 82–87, 2007.

[63] B. T. Horowitz, “Cyber Care: Will Robots Help the Elderly Live at Home Longer?,” *Scientific American, Division of Nature America, Inc.*, 2013. [Online]. Available: http://www.scientificamerican.com/article.cfm?id=robot-elder-care. [Accessed: 11-Jun-2013].

[64] R. Andriany, “Robot Housekeeper!,” 2010. [Online]. Available: http://rajandriany.blogspot.ca/2010_11_01_archive.html. [Accessed: 11-Jun-2013].

[65] “RESCUE ROBOTS,” *Web Japan , Trends in Japan , Sci-tech*, 2010. [Online]. Available: http://web-japan.org/trends/09_sci-tech/sci100909.html. [Accessed: 11-Jun-2013].