Universidad de Sevilla

Departamento de Ingeniería de Sistemas y Automática

Doctoral Thesis

Integration of Service Robots in the Smart Home

Author:
Javier Ramírez de la Pinta

Supervisors:
Dr. Eduardo Fernández Camacho
Dr. José María Maestre Torreblanca

It is no measure of health to be well
adjusted to a profoundly sick society.

Jiddu Krishnamurti

No es saludable estar bien adaptado a
una sociedad profundamente enferma.

Jiddu Krishnamurti

Acknowledgements

In first place, I would like to express my sincere gratitude to professors Eduardo Fernández Camacho and José M. Maestre Torreblanca for the continuous support and guidance of my Ph.D study and research. During these years they have been very supportive and have taught me how to become a true researcher.

There have been many other researchers that have helped me with this work. I would like to thank all my colleagues in Seville. Working in the Systems and Automation Department and with people over there has been a great privilege.

In addition, I would like to thank my family, particularly my parents for their patience during my study life and I hope they will be proud about this thesis.

Last but not the least, I would like to thank my wife, Montse, for all the years she has been waiting me while I had been studying.

Javier Ramírez de la Pinta
Seville, January 2016

Contents

Chapter 1

Introduction

The home automation market is just beginning to heat up, and it is very much still in its infancy. This trend will change the life-style of the inhabitants of such homes. With the widespread of the smart home more and more people have “intelligent” devices like thermostats, lighting or TVs, that either run automatically or are controlled from a smartphone.

To date, new products have been developed from a wide range of companies and few of them are compatible with each other. This lack of normalization makes very difficult to establish a solid home automation in the humans’ life since both manufacturers and consumers can not settle a common framework to integrate new devices or equipment. This issue is known as interoperability problem. The lack of interoperability is one of the most important problems that hinders the take-off of the smart home market.

It is clearly necessary a more open set of products and services based on a standardized platform that is not restricted to certain manufacturers. The emergence of novel and integrative technologies, such the Zigbee wireless communication protocol or a more closed wireless home control platforms, including Z-Wave, is a good signal for home automation.

Interoperability deals with the integration of different devices from different vendors and with different technologies and architectures in the same network. There are systems that work properly with certain technologies, however, they are not compatible with other systems.

In the home automation market have emerged many technologies, nevertheless, they have not taken care about interoperability with the existing technologies. In addition, the inclusion of service robots in the smart home complicates, even more, the interoperability problem since they involve many different technologies.

This thesis focuses on the interoperability between service robots and the digital home. It aims to introduce the reader to the exciting world of the interoperability in the smart home. A large number of systems proposed to manage the interoperability problem are analyzed in this work. This integration problem is addressed with particular developments and proposals which are presented in this thesis.

This chapter presents the approach to the interoperability problem and the attempts to solve this issue. Section 1.1 surveys chronologically the most relevant systems which supposed a breakthrough for interoperability that can be found in the literature. In Section 1.2 the main goals of this thesis are presented. Finally, section 1.3 provides the outline of this thesis.

1.1 A Brief History of Interoperability

Interoperability is a key issue to face the digital home challenges. An ideal smart home should emerge from the integration of different technologies based on different fields: home automation, telecommunication, surveillance, energy management, health-care, digital entertainment... One of the biggest problem for the smart home is the great amount of digital home market vendors, which causes a lack of interoperability due that not all manufacturers create compatible devices with other systems.

Several proposals, whose objective is to interconnect all consumer electronic devices in a digital home, have been emerged in order to solve the interoperability problem. The first attempt was the Common Object Request Broker Architecture (CORBA) which appeared in the early 90s. CORBA is a distributed object oriented architecture that allows integrating distributed objects in an application that is implemented in workstations with different hardware architectures, operating systems, and with objects defined in different programming languages [1]. Unfortunately for CORBA, the lack of standardization on its early stages avoided a better deployment of this system.

At the end of the 90s, several important organizations proposed the Universal Plug & Play (UPnP) technology [2]. UPnP is an open and distributed architecture that provides mechanisms of interoperability among heterogeneous devices. The success of UPnP has been big enough to be considered a de facto standard in digital entertainment applications. For example, the Digital Living Network Alliance (DLNA) initiative, which is specifically integrated for the exchange of multimedia content, is based on UPnP [3].

The Open Services Gateway initiative (OSGi) is also a remarkable attempt to solve the interoperability problem within a Java context. It provides a service-oriented architecture, which enables applications to discover and collaborate among them dinamically.

Finally, it is also interesting to highlight the Web Services as a remarkable technology to solve the interoperability problem. It consists of a set of open protocols that provide interoperability between different software applications, running on a variety of platforms and frameworks.

Each interoperability technology has its own features and there are important differences between the other initiatives. For example, a UPnP or DLNA network is composed of physical or virtual devices which have an URL associated to present their features. CORBA services are provided by distributed objects, and Web Services has defined the figure of agents which contain the implementation of the service. Another difference is the language in which the services are described: UPnP uses XML to this end, whereas Web Services and CORBA use WSDL and IDL respectively. All the middleware technologies push in the same direction: to reach the transparent interoperation between devices or software applications. Most of them have been use in a smart home context [4][5][6][7], but the significant differences between these middleware technologies and the fact that none of them has been settled only demonstrate the complexity of the interoperability problem.

Chapter 2 provides deeper details of these and many more systems.

1.2 Objective of the Thesis

As will be seen throughout the thesis, the inclusion of new advanced services in the digital home makes more necessary than ever to enhance the interoperability between the different areas and systems available in the market. This is the main objective of the work carried out in this thesis. The first step in this work consists of knowing the most outstanding technologies in this sector, in order to know the advantages and the drawbacks of each system, in order to enhance the most interesting aspects of our proposals.

The lack of interoperability becomes more evident with the inclusion of service robots in homes around the world. In this sense, it would be interesting to control robots remotely and coordinate all robots to perform complex tasks and reduce the time to complete them. For this reason, the goal of achieving full interoperability between consumer electronics devices and service robots is becoming increasingly important.

In this context, all efforts are focused on reaching a common and accepted standard that allows interconnecting all devices in the home automation. Another interesting approach is to use an existing interoperability system (intermediate standard) to interoperate with the rest of the systems so that everything is interconnected through this intermediate system.

During this work, we have analyzed some of the most widespread middleware initiatives to provide interoperability. Existing protocols and systems for communicating heterogeneous platforms are described. The technologies studied and detailed in this work are the following: CORBA, Jini, RMI, OSGi, UPnP, DLNA, Web Services, Semantic Web Services, Military Standards, Salutation, Service Location Protocol, Ad hoc Developments, URBI, DH Compliant, ROS, OROCOS and OpenJAUS.

In general, previous interoperability solutions have been focused on improving available capacities. This implies that many systems are not compatible with the other systems available in the market. One of our most important objectives during this work has been to present a procedure to integrate different devices, and robots in particular, in the digital home. In this work, it has been demonstrated that any device is capable of being integrated into a home through the UPnP standard.

This thesis is based on a consolidated interoperability initiative, which has been one the most successful at a home level up to date: UPnP. According to this research, multimedia is going to be the core of the digital home, and no standard eases the integration of multimedia as UPnP. Likewise, UPnP is a mature technology and there are several devices already in the market that provide some kind of UPnP functionality. We considered also other reasons: UPnP is designed to work in small networks, such as domestic ones; it works with physical devices, such as robots, sensors or actuators; UPnP operation is based on events; UPnP has good development tools and a good learning curve; and finally, the drawbacks of UPnP are well known and they were considered less than the advantages of this technology.

The penetration of service robots in the digital home which are capable of interoperating with other systems in the digital home is a challenge that needs to be solved. This thesis attempts to provide mechanisms to integrate service robots into the digital home, reducing time and developments costs and maximizing the benefits originally proposed by UPnP. We have proposed solutions to integrate into the digital home different and heterogeneous systems and robots. As has been noted throughout the thesis, interoperability is a very important issue in the digital home as its lack is the biggest obstacle that prevents the penetration of digital home technologies in the market. It is true that the multimedia market has progressed a lot in recent years, however, the continued appearance of new protocols to reach a solution to share and transmit content from one device to another is still booming. Again, the problem of interoperability prevents multimedia devices from interoperating with each other in any environment and situation.

During this thesis, some robots have been integrated into UPnP networks and the capacities of such protocol have been enhanced as a result of the research carried out during the DH Compliant project. We have developed a protocol to transparently interoperate with each device in the home. This protocol, with the objective of maintaining backward compatibility, uses an existing and popular standard, and proposes the use of UPnP entities with double behavior (control point and device), allowing the exchange of information and cooperation between entities. On the contrary, the development of more complex logics is necessary.

In addition, as part of this thesis, research about providing existing computing services from cloud environments to robots available in households have been accomplished. This study aims to transparently provide services with high computing needs to the digital home environment.

In resume, the trend has been to use an existing protocol and to enhance its capabilities, as well as to propose gateways to integrate other devices totally independent to the protocol. In this way, service robots have been included, a new protocol has been developed based on the original and fully compatible with it, and also for processes that require high power computing has been proposed to be solved in the cloud.

1.3 Thesis outline

As it has been seen in the literature review, there is a large number of interoperability initiatives. This thesis tries to provide interesting mechanisms to integrate the robots in the smart home, reducing timing costs, developments and maximizing the benefits that UPnP originally proposed. Some robots are integrated in UPnP networks and the UPnP capabilities are improved as a result of the DH Compliant consortium researches. In addition, we have investigated about how to provide computation services from cloud environments to the robots available at homes.

The outline of the rest of the thesis is detailed below:

- **Chapter 2: Interoperability Systems**. The lack of interoperability is a problem that has always been attached to the digital home. This chapter presents a very large list of interoperability initiatives. Special emphasis has been placed on UPnP which is the base system of this thesis.

- **Chapter 3: Integration of Service Robots in the Smart Home by means of the Universal Plug and Play Protocol**. The integration of robots in the smart home brings a more comfortable life for its inhabitants. Nowadays, there are robots that works more or less independently in the home, a good example is the vacuum cleaner robots which are widely spread and accepted in homes around the world. In this chapter, the integration of two robots, the Roomba vaccum cleaner robot and the Rovio surveillance robot, in a UPnP network is presented. In addition, different scenarios developed using these robots are detailed.

- **Chapter 4: Collaborative Tasks between Robots based on the Digital Home Compliant Protocol over UPnP**. This chapter presents a novel interoperability protocol: Digital Home Compliant (DHC) which is focused on the solution of the interoperability problem between domotic and robotic devices. In particular, the DHC-Groups module has been implemented for Roomba, which allows a robot group (hive) to perform collaborative tasks. Different experiments are shown in order to test the DHC-Groups module and to show its possibilities.
- **Chapter 5: Cloud Robotics**. Basic concepts of cloud computing and cloud robotics are presented in this chapter. A cloud development is also detailed, as well as the scenarios carried out with it.
- **Chapter 6: Conclusions**. The thesis ends with a chapter that analyzes the most relevant contributions and, additionally, points out future research lines in the field of interoperability in the smart home.

Chapter 2

Interoperability Systems

Interoperability deals with the integration of heterogeneous devices in the same network regardless of their architecture, operating system, programming language or their location in the network.

The problem of the lack of interoperability arises when there are different devices that comply with a certain system but these devices are not compatible with other technologies in the smart home, so it is not possible to make all devices and technologies work together. This fact is not new. Actually it is one of the biggest handicaps of the smart home market. Real interoperability is necessary to make possible a future in which homes detect automatically user needs and satisfy them. The lack of interoperability causes that different systems and devices can not interoperate with each other and hinders the take-off of the current smart home. For example, in the home automation market many technologies have emerged but none of them worried about providing interoperability mechanisms with previous technologies. Many factors that concurred explain the situation: new technologies usually come with new possibilities that did not exist before, compatibility implies higher development costs and a risk of clients buying devices to other manufacturers, very strong lack of standardization in the sector... The list of causes is almost endless and helps to identify mistakes that should not be repeated in the future. Nevertheless, it does not help too much to solve the interoperability problem that we have to face nowadays. In particular, it is possible to identify the following areas that work more or less independently and that have to be integrated in the future smart home:

- **Home automation**: it arrived at homes 30 years ago with the arrival of X-10, which despite of its "antiquity" is still one of the most used systems. For this reason it has become a de facto standard. There are other systems that have enjoyed enough success to establish themselves as solid options in their market segments. This is the case, for example, of KNX and LonWorks, both of them supported by international standards. Apart from these systems there is a constellation of proprietary systems with good future perspectives such as Z-Wave or Ingenium.

- **Access and Surveillance systems**.
- **Digital entertainment**: this is the field in which interoperability is closer to become a reality. The success of UPnP for this kind of applications has greatly simplified the access to multimedia content shared among the computers of a network. In this sense, new protocols take advantage of this as DLNA.
- **Assistive Computing and Healthcare**.
- **Energy management**.
- **Advance telecommunication services**: probably it is easier to understand what we mean with advance service with an example: IP-Domo, which is a Spanish home automation system with many capabilities in this sense, allow to deviate the incoming calls from the entryphone to the mobile phone when the user is not at home. The future smart home must be able to support advanced services like this one, that demand a strong degree of interoperability.
- **Service robots**: they are a reality in many homes thanks to robots like the vacuum cleaning robots. Right now the integration of robots in the smart home is far away from becoming a standard. The most advanced robots from the interoperability point of view include some port to receive external commands or a server with a page to receive commands.
- **Electrical appliances and current consumer electronics**: it is not possible to walk to the future without using the bridge of the present. Nowadays homes are filled with many useful devices that are closed systems. In the case of the appliances, almost always it is possible to use a relay to control the plug that feeds the devices. This allows a small grade of integration. There are other devices such as air conditioning systems or DVD players, to name a few, that open a small door for a more advanced interoperability through a wise use of infrared.

However, the operation of all these devices, systems or robots does not really require them to "think" as they are simply programmed to perform a series of repetitive tasks. If anything interferes with the pre-programmed task, they would malfunction since none of them is able to sense the interference and "think out" a solution. As service robots are in greater proximity to humans, the technology involves more safety concerns over the human-machine interaction. Therefore, it remains a great challenge today for us to build smart homes and intelligent robots that can "think" like we do.

To achieve such a goal, scientists and engineers have been trying hard to capture the essence of human intelligence in our homes and robots to make them intelligent to function well in the real world. This is a challenging and ambitious task since the robot or home intelligence must cope with various noises, uncertainty and dynamic changes in the real world. Like human beings, smart homes and intelligent robots should be able to sense their environments, reason and make decisions and respond to tasks and unexpected events quickly.

In general, intelligent robots and smart homes are broad, interdisciplinary subjects that involve many different technologies such as sensor integration, data fusion, wireless sensor networks, map building, embedded computing, navigation, planning and artificial intelligence.

For each individual smart home or robot, the "thinking" process takes place at many different levels. At its lowest level,"thinking" needs to be fast to respond quickly to unexpected events. At higher levels, "thinking" enables the homes and robots to handle a dynamic and uncertain world. By contrast, "thinking" should exhibit an adaptation and learning capability at its highest level. Moreover, a close interaction among smart homes and robots should be realized for achieving the common goal cooperatively.

In all emerging markets the compatibility between systems plays a crucial role for the success of the market itself. The digital home market is like the dog that chases its own tail in this sense: as there is no ideal system between the existing ones, the market is guided by the offer and not by the demand, but how is there going to be a strong demand for devices and services when there is no simple way of combining them? It is necessary a solid commitment towards interoperability from all the actors of the smart home industry if they want to speed up the penetration of advanced services to the end users.

In this chapter we present some background for the research addressed in this thesis and review several systems that can be found in this field. Section 2.1 presents an introduction to the evolution of the interoperability systems and the approaches to solve the interoperability problems. Next, section 2.2 provides an introduction about interoperability systems that will be detailed in subsequent sections. Specific systems that try to solve the interoperability problem are deeply analyzed in sections 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11 and 2.12 which detail CORBA, Jini, RMI, OSGi, UPnP, DLNA, Web Services, Semantic Web Services, some military standards (JAUS, 4D/RCS and NANO STANAG 4586), and other interesting technologies (Salutation, Service Location Protocol, Ad hoc Developments, URBI, DH Compliant, ROS, OROCOS and OpenJAUS) which have not had enough success in the digital home market.

2.1 Interoperability evolution

Nowadays there are several initiatives and different standards available whose objective is to interconnect all consumer electronic devices in a digital home. However, none of them seems to be the definitive interoperability system. The lack of interoperability between the proposed systems and the lack of commitment from vendors to make devices that comply with such systems hinder the take-off of the smart home in new buildings.

Over the past few decades there has been an exponential growth in service robots and smart home technologies, which has led to the development of exciting new products in our daily lives. Service robots can be used to provide domestic aid for the elderly and disabled, serving various functions ranging from cleaning to entertainment. Service robots are divided by functions, such as personal robots, field robots, security robots, healthcare robots, medical robots, rehabilitation robots and entertainment robots. A smart home appears "intelligent" because its embedded computers can monitor so many aspects of the daily lives of householders. For example, the refrigerator may be able to monitor its contents, suggest healthy alternatives and order groceries. Also, the smart home system may be able to clean the house and water the plants. The lack of interoperability becomes more evident with the inclusion of such service robots in homes around the world. In this sense, it would be interesting to control the robots remotely and coordinate all the robots to perform complex tasks and reduce time to complete them. For this reason, the goal of achieving full interoperability between consumer electronic devices and service robots becomes more and more important.

All efforts are focused on achieving a common and accepted standard which allows to interconnect all devices in a digital home (Figure 2.1). Another interesting approach is to use an existing interoperability system (intermediate standard) to interoperate with the rest of the systems, in such a way that everything is interconnected through this intermediate system (Figure 2.2).

The continuous evolution of computers together with lowest prices in consumer electronics has allowed a wide deployment of the digital home. The

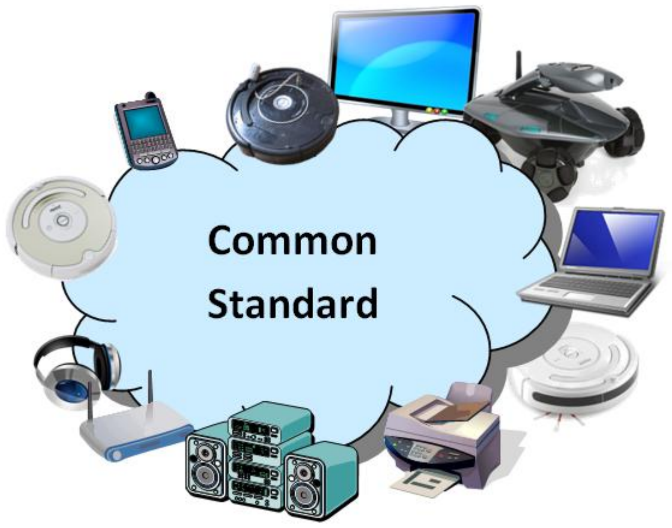


Figure 2.1: Interconnection through a common standard

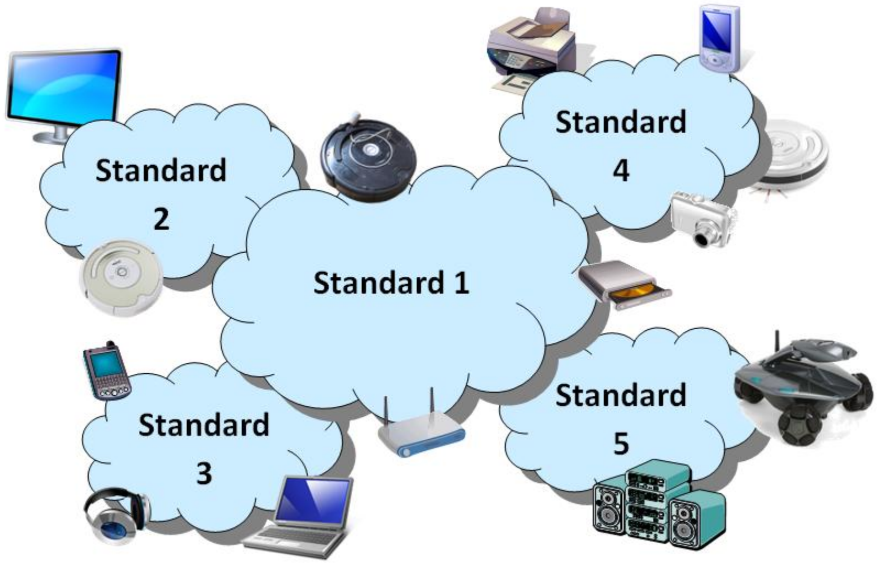


Figure 2.2: Interconnection through an intermediate standard

next revolution in the smart home is expected to come from the world of robotics. At present, the use of robotics is limited to industrial areas, although service robots that assist us in routine tasks such as cleaning the house, mowing the lawn or even preparing meals are becoming common. In recent years, the number of robotic standards has increased, and this progress has encouraged the integration of service robots in the smart home and the emergence of different communication protocols. Nevertheless, different problems have to be solved before service robots become as popular as computers. In particular, interoperability between the different systems that may exist in future homes is an ongoing issue.

2.2 Interoperability technologies

When looking through history from distributed systems to interoperability, it is important to recognize a need for cooperation and expansion of networks that are already in use. The beginning is the concept of *middleware*. Middleware is connectivity software that offers a group of services that make possible the running of distributed applications over heterogeneous platforms.

The idea of *middleware*, as an abstract layer of software, is to encapsulate all the available resources on a network, which can comprise all kinds of devices (from embedded processors to super processors, laptops, PDAs and mobile phones) and interconnect them in a transparent way. In other words, give an API (Application Programming Interface) to the programmers for the use of distributed applications.

There are some works related to the design and implementation of middleware generic distributed systems. For example [8], which represents an approximation to the design of a configurable system, based on the concept of *reflection*. The usefulness of this component's engineering is also important when giving a system the ability to configure.

These concepts are also commented on by [9]. These authors also talk about the link to the application layer by using this component's technology. They suggest the development of a model (OpenORB, based on the model CORBA) independent of the platform and the language of programming. They also define meta-structures and meta-data to give intelligence to the protocol so it can apply reason to its own interpretations, so the system's (re)configuration will be easier.

The idea that the reader must have in mind during this chapter is interoperability. Interoperability is the key component to solving the smart home jigsaw puzzle. Thus, in this chapter, we will place special emphasis on the interoperability aspects of the different standards. In addition, different research projects on the interoperability and control of robotic systems and unmanned vehicles will be surveyed. Behind all these standards and projects, there are stories of success and failure, and many valuable lessons about the complex world of interoperability. At this point, it is difficult to know if any of these alternatives will prevail and become a consolidated standard for the integration of robots in the digital home. However, what we know for sure is that any succeeding standard will have learnt from all that will be presented here.

In this chapter, different standards that could be used for the integration of mobile robots and unmanned vehicles in the digital home will be detailed. These standards have been proposed within different contexts. On the one hand, standards have been developed for military purposes such as JAUS or 4D/RCS, which is logical given that the control of autonomous vehicles has many potential applications in this field. On the other hand, standards have been developed in a computer science context, where interoperability between the different agents that may interact in a networked environment is a major problem. This chapter is mainly based on the tw chapters published in the Springer's book "Service Robotics within the Digital Home" [10][11].

Some of the most-used middleware initiatives to provide interoperability in the smart home are analysed. The protocols or existing systems used to communicate among heterogeneous platforms will be described.

2.3 CORBA

CORBA (Common Object Request Broker Architecture) is a standard that provides a platform for the development of distributed systems. CORBA is defined by the Object Management Group (OMG), which defines APIs, communication protocols and all necessary items to ensure interoperability between different applications running on different platforms. CORBA uses an IDL to specify the interfaces through their functionality. This is a way to indicate how CORBA data types must be used in implementations of client and server.

All this means that CORBA is a kind of middleware (platform of distributed services, independent of the operating system) that guarantees success in the transit of data across different platforms and applications. It is applied in RTS and is efficient enough for any kind of problem. The main features of this standard are:

- It is a distributed object standard.

- It specifies the architecture the system should have, is flexible and heterogeneous.
- Interoperability.
- Scalability.
- Transparency, facilitating client-object communication [12].
- Naming service.
- It sets a minimum object model.
- Each object implements an interface.

  - The definition of interfaces is made through the IDL, making it independent of the programming language.

  - The reuse in software is achieved through interface inheritance.

  - Multiple inheritance.

  - The details of an object's implementation cannot be accessed.

2.3.1 Components

- The Object Request Broker (ORB) is the CORBA object manager and is part of its core. It allows for the invocation of static and dynamic objects [12]. It can operate without the services and facilities provided by CORBA. It handles the invocation and search for remote objects using dynamic methods for the invocation. It is responsible for giving back the object attributes of the object accessed through the IDL of the object. Locally, it also collects information on the objects to pass to other ORBs and handles local computer security (Figure 2.3).
- IDL, Language for defining interfaces. Since it is a declarative language and not a programming language, it defines interfaces independent of the implementations of objects.
- Dynamic Invocation Interface (DII). Generic Stub. Client side.
- Dynamic Skeleton Interface (DSI). Generic skeleton. Server Side.
- Both DII and DSI are based on the interface repository, which is a CORBA object that contains information on the object's interfaces and their types. It allows applications to access this information in a static or a dynamic way. The main advantage is the support given to the dynamic calls.

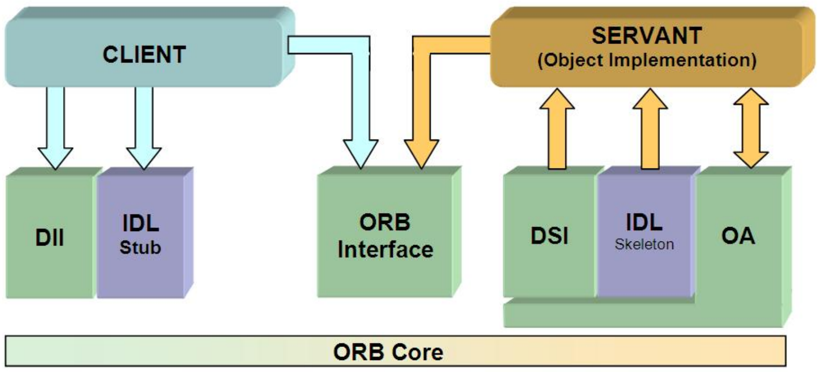


Figure 2.3: CORBA architecture

- The implementation repository is required when the objects are persistent. Most general purpose ORBs provide a repository of implementations that supports indirect connections for persistent references. This characteristic solves the problem of direct connections for persistent references. It has also a bad point; it slightly reduces the good working of the first invocation from client to server. It also offers various modes for the automatic activation of server objects [13].

- The object adapter is the bridge between the ORB and CORBA object implementations. This allows it to make requests to an object without knowing its interface, since the object adapter adapts the object's interface to that expected from the object making the request.

- Communication protocols between ORBs. CORBA is based on the protocols GIOP (General Inter-ORB Protocol) and the standard protocol IIOP (Internet Inter-ORB Protocol). GIOP specifies the types of messages and the format to transport requests between ORBs. IIOP specifies the way TCP/IP is implemented over GIOP. Thanks to these protocols, ORB can be integrated even if it comes from different developers.

2.3.2 Services

There is a large set of standard services offered by CORBA [14]. These services are added to the ORB interface to complete it; however, they are optional. The most important include:

- **Concurrency Service**. Mediates concurrent access to an object such that theconsistency of the object is not compromised when accessed by concurrently executing processes.

- **Event Service**. This defines two roles for objects: the supplier and the consumer. Consumers process information in the events that are produced.

- **Naming Service**. This is the main mechanism for objects that will be invoked by most customers from an ORB-based system.

- **Persistent State Service**. Replaces the persistent object service. These are interfaces that provide persistent information, namely data objects stored in databases.

- **Property Service**. Can attach dynamic properties to objects outside the static IDL-type system.

- **Security Service**. The security service of CORBA provides various security policies to cater for different needs that lead to a secure architecture. CORBA's security can be used in a wide range of systems. It also allows the reuse of its own security protocols. These include:

  - Authentication and identification of objects or users (i.e. verifying that they are who they seem).

  - Access control and authorization.

  - Security audits.

  - Secure communication between objects.

  - Non-repudiation policy

  The CORBA security service is included in the safety process of OMG. Among the OMG security specifications, we can find:

  At an API level:

  - ATLAS (Authorization Token Layer Acquisition Service)

  - RAD (Resource Access Decision Facility)

  In CORBA's infrastructure:

  - CSIv2 (Common Secure Interoperability, version 2)

  - CORBA Security Service

- **Time Service**. Allows an object to ascertain the time along with an estimated error associated to it.

- **Trading Object Service**. Facilitates the search for objects, services, features, functionalities and so on.

2.3.3 Application Examples

Some frameworks exploit the features of CORBA for telerobotic systems, whereas some applications may be based on the manipulation of complex systems remotely [15].

CORBA is commonly used in telecommunication robots in real time as well as to keep track on them. At the University of Auckland, researchers tested the LEGO Mindstorm and Khepera models to demonstrate the reliability of a design for the distributed control of robots using CORBA [16].

The Institute for Computer Design and Fault Tolerance at the University of Karlsruhe in Germany presented a distributed software architecture based on CORBA for the autonomous service robot Albert2. The development was focused on the modularity and integration of learning aspects [17].

The research group there proposed a system for controlling a humanoid robot based on CORBA. Using this architecture in a distributed environment such as a local network, it is possible that various humanoid robots all over the world can share their own modules via the Internet [18].

CORBA has been used to integrate a distributed system of multiple mobile robots in a simulated environment that offers the possibility of a collaborative control [19].

2.4 Jini

Jini is an architecture that provides an infrastructure for defining, publishing and search services on a network. It was developed with Java classes [20]. The main feature of Jini is the service discovery in multicast mode or search mode for specific services (similar to the idea of UPnP). It uses the multi-platform feature of the Java platform to provide universal services, registering each one as serialized objects with their own interfaces. A diagram of Jini's architecture is shown in Figure 2.4.

The main aims of Jini's platform are the immediate availability of services, the hardware abstraction on the Java environment, the service-based architecture and simplicity.

2.4.1 General Features

This is an easy protocol [22] (Figure 2.5). When a device connects, it registers in the lookup service of the Jini network. After that, the service sends a file with the bytecode that a customer needs to use its services.

- The lookup service stores this file in a special table and puts similar services together in groups. When a client asks the search service to use

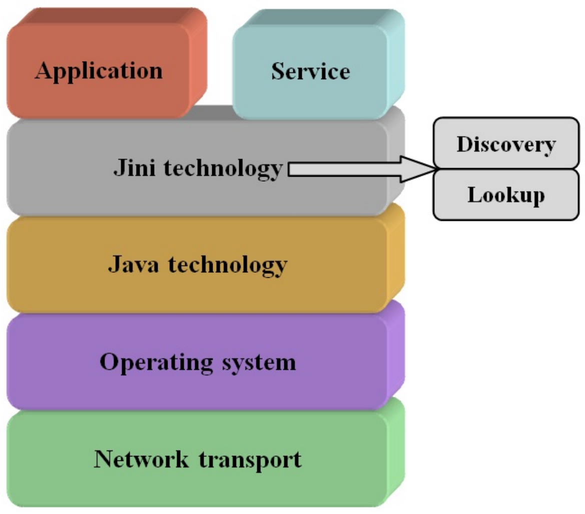


Figure 2.4: Layers model of a Jini system [21]

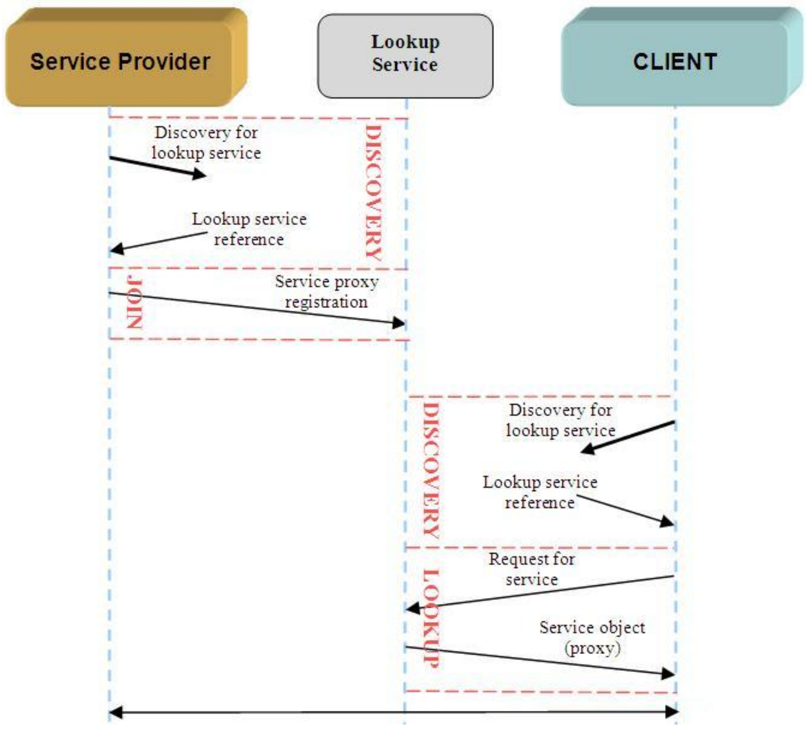


Figure 2.5: Diagram of the sequence events of Jini [21]

a device, it responds with a list of devices that provide these services.

- The client responds with the identifier of the specific device to be used and the search service responds with the bytecode mentioned above.
- The client will now be able to use the bytecode (during a specific time, in a shared way or in an exclusive one).

2.4.2 Specific Features

The purpose of the Jini architecture is to put devices and software into groups inside a distributed and dynamic system. This simplifies the access, management and maintenance services offered by each point separately, keeping the flexibility and control offered by a personal computer.

Services

The most important concept within the Jini architecture is the service. A service is an entity that can be used by one person, one program or another service. It may be a calculation, saved data, a communication channel with another user, a software filter, a hardware device or another user. As an example, we can mention the printing services of documents.

Members of a Jini system share access to services. A Jini system should not be considered a set of clients and servers, users and programs or even programs and files. Rather, it consists of a set of services used to perform a particular task.

The services may use other services, and the customer of a service can be a service itself for other customers. The dynamic nature of a Jini system enables services to be added or removed, at any time, from a set, according to demand, need or the changing demands of the working group. Jini systems provide mechanisms for service construction, lookup, transfer and use in a distributed system. The services may be:

- Devices, such as printers, screens and discs.
- Software, such as applications or utilities.
- Information, such as access to databases or/and files.
- Users of the system.

Services communicate with each other using a service protocol (set of interfaces written in Java). All these protocols are undefined. The groundings of the Jini system define a small number of these protocols, which in turn define the interactions among critical services.

Lookup Service

Services are found and resolved by a lookup service. This service is the central mechanism for the system to boot and the main point of contact between the system and users. In other words, it is a mapping service made up of lookup interfaces that indicates the functionality provided by a service to groups of objects that implement the lookup service. In addition, descriptive entries associated with a service allow finer lookup services, based on properties understandable by a human being.

The content of a lookup service may include other search services, providing hierarchical searches. In addition, this kind of service may contain objects that encapsulate other names or service directories, providing a system of pointers that connects Jini lookup services with other search services. Thus, references to a Jini lookup service can be mixed with these names and directory services, providing the customers of these services with a way to access a Jini system.

A service is added to a lookup service by a pair of protocols called discovery and join. First, the service locates an appropriate lookup service (using the discovery protocol) and then joins (using the protocol join).

Java Remote Method Invocation (Java RMI)

Communication between services can be performed using Java RMI. This infrastructure is not a service itself, but is rather part of the Jini technology infrastructure. RMI provides mechanisms to locate, activate and perform the garbage collection of Java objects. This method will be deeply detailed and presented as a different system for interoperability in the next section (Section 2.5).

RMI is a Java extension to traditional mechanisms for RPC. RMI not only allows data to pass from one object to another through the network, but also whole objects can be sent and received, including their codes. Much of the simplicity of the Jini system is because of this ability to move code through the network, encapsulated in an object.

Security

The design of the Jini security model is based on the concepts of a master list and an access control list. Jini services are accessed by an entity - the principal - which generally refers to a particular user in the system. The services themselves may request access to other services, providing the identifier of the object that implements the service. The access of an object to a service depends on the content of an access control list associated with the object.

Leasing

Access to many of the services in the Jini system environment is based on the concept of lease or loan. A loan is a grant of guaranteed access to a service for a certain period of time. Each loan contract is negotiated between the service user and provider, as part of the protocol service: a service is requested for a certain period of time and access is granted for the same time period (probably taking into account the time span taken to make the application). If a contract is not renewed before it is released - because the resource is no longer necessary, the client or the network fails or the contract cannot be renewed - both the user and the resource provider may agree that the resource can be released.

Leases are exclusive or non-exclusive. The first ensures that no one else can have a contract on the resource during the contracted period, whereas non-exclusive leases permit multiple users to share the same resource.

Transactions

A series of transactions, in a single service or spanning multiple services, may be involved in a transaction. The Jini transaction interfaces provide the service protocol needed to coordinate a two-phase commit. The responsibility for deciding how to implement transactions - and the semantics in a transaction - is left to the services themselves using these interfaces.

Events

Jini supports distributed events. An object may allow other objects to register in the events of an object and receive a notification with their histories. This allows event-distributed programs to be written with a great variety of liability and scalability guarantees.

General View of Components

The components of a Jini system can be divided into three categories: infrastructure, model of programming and services. The infrastructure is the set of components that builds a Jini system, whereas services are the entities inside it. The programming model is a set of interfaces that allows the construction of reliable services, including those that are a part of the infrastructure and those that are a part of the whole.

These three categories, although disjunct, are intertwined in a way that makes distinctions between them confusing. It is also possible to build systems with some of the features of the Jini system with variants on the cate-

Table 2.1: RMI components

	Infrastructure	Programming model	Services
Basic Java	JVM	API Java	JNDI
	RMI	JavaBeans	Enterprise Beans
	Java security	-	JTS
Java + Jini	Discovery/Join	Leasing	Printing
	Distributed security	Transaction	Transaction manager
	Lookup	Events	JavaSpaces services

gories or without any of them. By contrast, the main feature of Jini is that it is a system built with a particular infrastructure and described programming models, based on the concept of service.

The separation of the segments in the architecture means that only a slight change is needed in the inherited code to be used in a Jini system. However, the power of a Jini system is only available for services built using the integrated model from the beginning. A Jini system can be considered as an extension of the network's infrastructure, programming model and services that made Java technologies popular in the case of a single machine. These categories, along with the components for the Java application environment, are shown in the Table 2.1.

2.4.3 Organization of the Jini Architecture

Infrastructure

The infrastructure defines the basic core of this technology. It includes:

- ***A distributed system for security,*** integrated in the RMI, which extends the security model from Java to the world of the distributed systems.

- ***The discovery and join protocols,*** service protocols that allow other services (hardware or software) to discover and announce the services offered to the other members of the group.

- ***The lookup service,*** which is used as a backup for the services. The entries in the lookup service of objects are written in Java, and they can be downloaded as a part of a search operation and work as local proxies for the service that sets the code in the lookup service.

Programming Model

The infrastructure enables the programming model and makes use of it at the same time. The contracts made in the lookup service have a limited lifetime. This fact allows the lookup service to precisely check the set of available services at a specific moment. When the services binds to or separates from the lookup service, the events are notified about it, and the objects that have already shown an interest in receiving this information are updated about these new or defunct services.

The programming model is based on the ability to move the code, supported by the infrastructure. Both the infrastructure and the services that use it are calculation entities that live in the physical environment of the Jini system. However, services also constitute a set of interfaces that define the communication protocols used by services and the infrastructure to communicate between them.

These interfaces together form the distributed extension of the standard model in Java programming, which constitutes the Jini programming model. Among the interfaces that make up the Jini model are:

- *The leasing interface*, which defines a way to allocate and release resources through a model based on the renovation of their lifetime.
- *The event and notification interfaces*, which are extensions of the event model used by JavaBeans components for distributed environments. This feature allows event-based communication between services enabled by the Jini technology.
- *Operation interfaces*, which enable entities to cooperate so that all changes occur in the group or none take place.

Services

The technology infrastructure and the Jini programming model are designed to enable the services to offer themselves and to be found on the network. These services make use of the infrastructure to call and discover each other and announce their presence to other services and users.

The services appear as objects written in Java, perhaps made up of other objects. A service has an interface that defines the operations that may be requested of it. Some of these interfaces are intended to be used by programs, whereas others are intended to be administered by the client to enable the service to interact with a user.

The kind of service determines the interfaces of which it is composed and defines the set of methods used to access the service. A service can be implemented only by other services. Some of the Jini services are:

- *A printing service*, which can print from Java applications.
- *A service of JavaSpaces*, which can be used for simple communication and storing groups of objects written in Java.
- *A transaction administrator*, which allows groups of objects to participate in the Jini transaction protocol defined by the model of programming.

2.4.4 Application Examples

There have been various initiatives to implement Jini as a form of communication between devices. In this regard, we underline the Ronin Agent Framework [23], an environment based on Jini's distributed agents. This implementation attempted to improve the initial protocol, making it independent of the domain (so external devices could communicate with the local network), among other advances.

[24] and [25] described the implementation of a SOA (architecture oriented to services [26]), ICENI, using the Jini platform, among others. This environment is based on an independent specification of SOA.

The integration of different service platforms is not easy. [27] integrated Jini with UPnP, but it is important to note that these two protocols are incompatible by themselves. This new platform allows UPnP services to use Jini devices and vice versa by making just a few configuration changes. However, these authors do not answer several of the questions referring to the limitations encountered when working with these two architectures together.

In addition, for devices that do not have enough capacity to run a Java Virtual Machine (JVM), Jini offers the possibility of using a surrogate host. This is just another device capable of supporting a JVM, which works as a bridge between the original device and the Jini network architecture.

2.5 RMI

RMI (Remote Method Invocation) emerged from the need to communicate among different objects, and it is implemented on different machines as happens on distributed systems. Therefore, this technology is a remote invocation of Java objects. The initial version of Java RMI required a JVM in both the origin and destination machines [28].

After the RMI-IIOP was developed, it was added to the RMI, providing it with the best features of CORBA. RMI is pure Java and since it does not support other languages, CORBA emerged. The adaptation to a distributed system has not prevented the continued development of RMI as a secure system. The main characteristics of RMI are:

- Simple, easy to write and easy to maintain.
- Transparency, because the distribution of objects and parameters passing is transparent to the programmer.
- Pass an object by value (as parameters of methods).
- The definition of interfaces is done directly in Java.
- Implementation in Java.
- Independence of the communication protocol.
- Separation between interface-client and implementation-server.
- Naming service.

2.5.1 Architecture

RMI is a layer architecture made of a stub/skeleton layer, a remote reference layer and a transport layer. The programmer only interacts with the application layer. The RMI system manages the three previous layers (Figure 2.6), which could be replaced by others with the same function without altering the rest.

2.5.2 Components

RMI allows the programming of CORBA servers and applications via the RMI API. It is possible to work entirely in the Java programming language using the Java Remote Method Protocol as a transport or to work with any other CORBA implementation using IIOP Java RMI over IIOP.

RMI-IIOP is designed for developers who program in Java and want to use the RMI interfaces using IIOP as the transport layer. The RMI-IIOP interoperability with CORBA objects implemented in other languages is available only if all the remote interfaces have been previously defined as Java RMI interfaces [29].

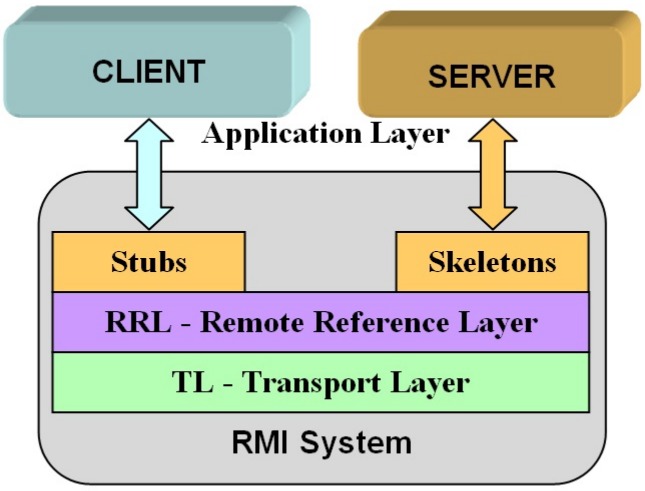


Figure 2.6: RMI architecture

2.5.3 Application Examples

At the University of Bielefeld, Germany, one research group has integrated memory-based software for the development of autonomous robots. This is an approach to an architecture of autonomous mobile robots operating in human environments. It replaced the use of data on a closed chain based on the long- and short-term memory. RMI was used for the exchange of critical information, such as the module that controls the hardware. RMI also allows the system to estimate when the configuration has been completed. The system can then send information on the result of the configuration [30].

[31] focuses on task-level programming and monitoring robots in their daily operations. It is not a framework limited to robots and it could be used in other distributed environments. During its development, the authors took advantage of technologies available in Java, such as Jini, RMI and Java Native Interface. [16] supported Java RMI over Bluetooth, GPRS and WLAN technologies. As a conclusion of this, the good work of Java RMI was tested in heterogeneous wireless environments, allowing parallel and distributed control.

In a study by researchers at the Information and Communications University in Korea, RMI is used to access external ontologies in the development of a selfexpandable software. This kind of software is useful for intelligent robots for two reasons. First, they study their environments and then they decide their appropriate behavior based on what they have learnt about their surroundings.

DEVS/RMI is a distributed, self-adaptive and reconfigurable simulation environment for engineering studies. It is based on the standard implementation of DEVS, in which Java RMI supports the synchronization of local and remote objects. It is designed for the intensive testing of programs, and this is the reason for it supporting dynamic models [32].

2.6 OSGi

OSGi (Open Services Gateway Initiative [33]) is an independent corporation that brings together about 40 companies in an alliance responsible for defining and promoting open specifications for the delivery of managed services in network environments. It is based on the modularity of the Java environment, trying to abstract the implementation of components (bundles) using services to communicate. One of its main aims was to resolve certain development and deployment conflicts, such as class conflict and the explicit dependencies [34].

2.6.1 General Features

OSGi is based on a layer model (Figure 2.7) that includes, among others, the bundles or packages (components developed as jar files), services (which provide communication between bundles through Java objects), modules and security layer.

The features of OSGi could be a good alternative for the development of complex systems because of its versatility and cross-platform feature (only a JVM resident on each node of the network would be required for the running). Nevertheless, OSGi has several problems (some delimited, some not) because of its poor basis of compatibility and poor management of dependencies.

2.6.2 Specific Features

Framework of OSGi

The OSGi framework is a module system for Java that implements a complete and dynamic model of components, which does not exist in independent environments of JVMs. The applications and components (which come in packets or bundles) can be installed, started, stopped, updated and uninstalled remotely without rebooting. The management of Java packages and classes is carefully specified. Lifecycle management is performed through APIs that make possible the remote download of management policies. The registry

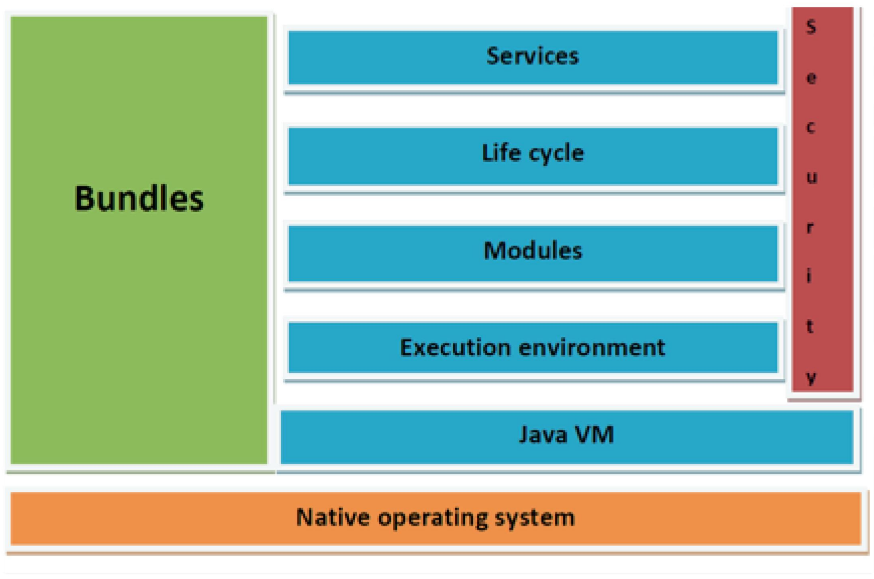


Figure 2.7: Layer model of an OSGi system

allows service bundles to detect if services have been added or deleted and acts accordingly.

Originally, it was focused on service gateways, but the scope has since widened. OSGi specifications are now used in applications ranging from cell phones to the Eclipse development environment (open source). Other application areas include automotive, automation in industry and buildings, PDAs, grid computing, entertainment (such as iPronto), fleet management and application servers. Figure 2.7 shows the hierarchical structure of an OSGi system.

Specification Process

OSGi specification has been developed by its members in an open process that is available for the public free of charge under the OSGi specification license. The OSGi Alliance has a performance program that is open to its members. In September 2008, the list of certified OSGi implementations contained five entries.

Architecture

Any framework that implements the OSGi standard provides an environment for the modularization of applications in small packages. Each package is a

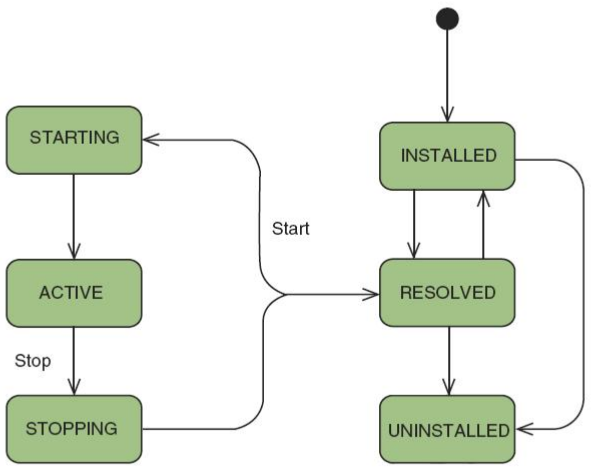


Figure 2.8: Lifetime of an OSGi bundle

collection of well-coupled and dynamically loadable classes, jar files and a configuration that explicitly state their external dependencies (if any). The framework is conceptually divided into the following areas (see Figure 2.8):

- *Bundles*: These are jar components with extra headers in a detailed manifest file.
- *Services*: The service layer connects bundles dynamically, offering a model of publication, search and link to plain old Java objects.
- *Register services*: The API of some management services (ServiceRegistration, ServiceTracker and ServiceReference).
- *Lifetime*: The API for the management of lifetime (install, start, stop, update and uninstall) bundles.
- *Module*: The layer that defines the encapsulation and declaration of dependencies (how a bundle can import and export code).
- *Security*: The layer that deals with security issues, limiting the functionality of the bundles to predefined capabilities.
- *Running environment*: This defines what methods and classes are available on a specific platform. Since they are susceptible to change, there is no fixed list of running environments. The Java community is creating new versions and editions of Java constantly.

1. Bundles A bundle is a set of Java classes and additional resources accompanied by a detailed manifest file (MANIFEST.MF) of all its contents as well as the additional services required to provide the included group of Java classes more complex behavior until the point of abstraction, where the whole is treated as one component. An example of a MANIFEST.MF file, typical of OSGi headers, is shown below:

   Bundle-Name: Hello World
   Bundle-Symbolic Name: org.wikipedia.helloworld
   Bundle-Description: A Hello World bundle
   Bundle-ManifestVersion: 2
   Bundle-Version: 1.0.0
   Bundle-Activator: org.wikipedia.Activator
   Export-Package: org.wikipedia.helloworld;version="1.0.0"
   Import-Package: org.osgi.framework;version="1.3.0"

2. Lifecycle A lifecycle layer adds bundles that can be installed, started, stopped, updated and uninstalled dynamically. The bundles trust in the module layer for class loading but they add an API to manage the runtime modules. The lifecycle layer provides mechanisms that are not usually part of an application. Some extensible dependency mechanisms are used to ensure the correct working of the environment. Lifecycle operations are fully protected by the security architecture (Table 2.2).

3. Services

   The OSGi Alliance has specified many services, all of them by a Java interface. The bundles can implement this interface and register it with the service registry. Service clients can find it in the service registry or detect it when it appears or disappears (Tables 2.3-2.5).

2.6.3 Organization

The OSGi Alliance was founded by Ericsson, IBM, Motorola, Sun Microsystems and others in March 1999 (before becoming a nonprofit corporation called Connected Alliance).

Among its members (as of May 2007) are more than 35 companies from different business fields, such as IONA Technologies, Ericsson, Deutsche Telekom, IBM, Makewave - before it was Gatespace Telematics - Motorola,

Table 2.2: Description of the lifetime of an OSGi bundle

State of the bundle	Description
Installed	The packet has been successfully installed
Resolved	Every Java class that needs the bundle is available. This state indicates that the packet is ready to be started or stopped
Starting	The package is being started; it will call the method BundleActivator.start, and this one has not finished yet. When the bundle has an activation policy, it will remain in the initial state until it is activated, according to this policy
Active	The package has been successfully activated and it is running. Its starting method, Bundle Activator, has been called and it has returned
Stopping	The packet is being stopped. The method Bundle-Activator.stop has been called but the stop method has not returned yet
Uninstalled	The packet has been uninstalled. It cannot be changed to a different state

Nokia, NTT, Oracle, ProSyst, Red Hat, Samsung Electronics, Siemens, SpringSource and Telefónica.

The alliance has a board that establishes the governance of the organization. OSGi officers have different roles and responsibilities to support the alliance. The technical work is carried out in the expert groups (EGs) organized by the board of directors, and the non-technical work is carried out in various working groups and committees. The technical work in EGs includes development specifications, reference implementations and compliance testing. These EGs have made four versions of OSGi specifications (as at 2007).

There are EGs dedicated to business areas, mobile phones, vehicles and central platforms. The Expert Group Company is the latest EG and handles applications regarding the company/server. In November 2007, the Residential Expert Group began working on specifications to remotely administer residential gateways orc homes.

Table 2.3: OSGi system services

System services	Description
Logging	The information register, warnings, debugging and errors are handled through this service. It receives log entries and dispatches others bundles that have already subscribed to this information
Configuration admin	This service allows an administrator to set and view information about the configuration of the bundles
Device access	This simplifies the detection and connection of existing devices. It is used in Plug and Play environments
User admin	This service uses a database containing user information (both public and private) to issue authentication and authorization
IO connector	This service is implemented in the packet CDC (`https://en.wikipedia.org/wiki/Connected_Device_Configuration`), CLDC (`http://en.wikipedia.org/wiki/CLDC`), javax.microedition.io (`https://docs.oracle.com/javame/config/cdc/ref-impl/cdc1.1.2/jsr218/javax/microedition/io/package-summary.html`) as a service. This one allows the bundles to provide new protocol diagrams
Preferences	It offers an alternative, friendlier mechanism with OSGi to use the default Java package java.util. Properties (`http://java.sun.com/javase/6/docs/api/java/util/Properties.html`) for storage preferences

Table 2.3: OSGi system services

System services	Description
Component runtime	The dynamic nature of the services - they can be opened and folded at any time - makes it difficult to write software. Runtime component specification can make it easier to manage these issues, providing a declaration of XML-based units
Deployment admin	This standardizes the access to some responsibilities of the administration agent
Event admin	This provides the bundle with a mechanism of internal communication, based on a publish and subscribe model
Application admin	This simplifies the management of an environment with different kinds of applications that are simultaneously available

Table 2.4: OSGi protocol services

Protocol services	Description
HTTP service	This allows the information to be sent and received by OSGi using HTTP
UPnP device service	This specifies how OSGi bundles can be developed to work with UPnP devices (`http://en.wikipedia.org/wiki/Universal_Plug_and_Play`)
DMT admin	This defines an API to deal with a device using concepts of the specifications for device administration from Open Mobile Alliance (`http://en.wikipedia.org./wiki/Open_Mobile_Alliance`) (OMA)

Table 2.5: OSGi miscellaneous services

Other services	Description
Wire Admin	This allows the connection between producers and consumers
XML parser	This service allows a bundle to locate a parser (XML syntax analyzer) with specified properties and compatibility with JAXP (`http://en.wikipedia.org/wiki/JAXP`)
Measurement and state	This allows and simplifies the correct use of measurements in an OSGi service platform

2.6.4 Application Examples

In October 2003, Nokia, Motorola, IBM, ProSyst and other members of OSGi formed the Mobile Expert Group, which specifies a service platform based on MIDP for the next generation of smartphones, dealing with some of the needs that CLDC cannot handle. MEG joined OSGi at the same time as R4.

Also in 2003, Eclipse selected OSGi as the runtime platform for the plug-in architecture to be used for the Eclipse Rich Client Platform and the IDE platform. Eclipse itself includes sophisticated tools to develop OSGi bundles, and there are some plug-ins for Eclipse to improve the development of OSGi (for example, ProSyst and Knopflerfish have Eclipse plug-ins available for OSGi developers).

There is a free software community with activity around OSGi. Some open source implementations are widely used such as Equinox OSGi, Apache Felix, the OSGi Knopflerfish project and the editing of embedded server Equinox (mBedded Server Equinox Edition, BSEE). Now talking about the support to the system's development and testing, projects Pax OPS4J provide a lot of components and useful knowledge.

Some examples of OSGi uses can be found in the literature. [35] discussed an intelligent system (SOCAM) based on ontologies integrated with OSGi to build a system that can deliver and manage context-aware services in a smart-home environment. Meanwhile, [36] fuse UPnP AV, which is used to provide media services, with OSGi, which manages each UPnP entity as a bundle.

2.7 UPnP

UPnP (Universal Plug and Play) is a group of protocols [37] or a much-extended architecture suggested by Microsoft [38] and promulgated by the

UPnP Forum, which ensures that some network devices can autoconfigure. The aims of UPnP are making sure that the devices can connect perfectly and simplifying the implementation of networks at home (exchange of data, communications and entertainment) and in corporate environments. It is an open and distributed architecture based on already existing protocols and specifications, such as UDP, SSDP, SOAP [39] or XML [40].

In addition, it is supported by the Internet protocol family TCP/IP, which (independent of the company, operating system and programming language) enables the APIs of the devices connected to a network control to negotiate and exchange information and data in an easy and transparent way for the user. This way, the user does not need to be an expert in networks, devices or operating system configuration. In addition, UPnP technology does not depend on the physical environment, so it can work on the telephonic line, the power supply, Ethernet, radio frequency and IEE 1394.

2.7.1 General Features

The main characteristic of the protocol is the transparency of installing a device that has just been connected to the power supply. All the services of the installed device are automatically available without the need to configure anything in the protocol [2]. UPnP notices when a new device is connected to the net, it gives it an IP address, a logic name and updates the rest of the devices about their functions and processing ability. As seems obvious, it also updates the new device about these same features of the others. This way, the user does not have to worry about the configuration of the net or losing any time installing new drivers or controllers for the devices. UPnP is dedicated to all these things each time a new device is connected (or disconnected) to (from) the net. It also optimizes the configuration of the devices.

Its application for development of a home automation system offers a new possibility to create distributed control architectures. In other words, robots have independent activation parts connected by an internal network. Because of this,UPnP gives more versatility and flexibility to the system. Moreover, any change in software or any device in the system is easily adaptable in the system.

A digital home based on UPnP is conceived to include all wire and wireless networks, entertainment devices, telephonic systems, home control and many more devices. It will also put some home networks together in a single logic made by programmable devices [41] (Figure 2.9).

One of the most common uses of this protocol is to enable devices or programs to open router ports, so they can properly communicate with the outside world.

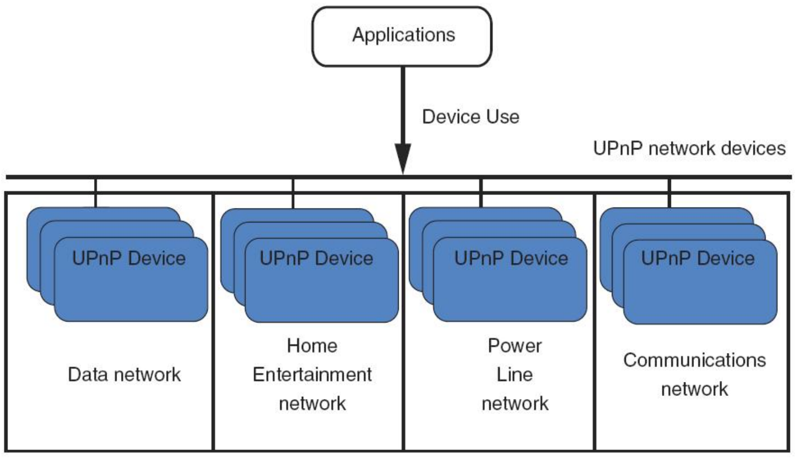


Figure 2.9: UPnP: network unification technology [41]

2.7.2 Specific Features

Since the (Universal Plug and Play) UPnP model is based on the existence of two different components, the control point and the device, this protocol makes identification of the roles of every element in a home automation network possible. The main idea is that every device (a robot, a router, etc.) can be accessible on a local area network (LAN). Some will announce the services they offer to the rest using a protocol such as the SOAP or something similar.

An XML file with the name of the device and a description of the services it offers are sent through the network each time a new device plugs into the network. The file may also include a URL pointing to the website of the developer. In addition, an external pointer to detailed information about the services could be included.

This fact gives a clearer idea of the ease of maintenance and transparency of use that this architecture provides to applications and interfaces. As shown in the figure above, Dynamic Host Configuration Protocol (DHCP) servers and/or DNS may be available on the network, so a new device may automatically be configured on the network upon connection. The next step will be discovering services.

2.7.3 Protocols

Some protocols and standards, which are part of the UPnP architecture, will be described in this subsection, special focus will be on protocols standard based on the concept of service. Some of the UPnP features related to the protocols TCP/IP, UDP/IP, HTTP, SOAP and XML are specified below.

TCP/IP

TCP/IP stands for Transfer Control Protocol/Internet Protocol. It is the grounding on which the development of other UPnP protocols takes place. TCP/IP is a set of protocols that covers different physical media and provides compatibility between different developers. It is based on the idea of an IP address or, in other words, the idea of providing an IP address to each computer connected to the network.

UDP/IP

The UDP (User Datagram Protocol) is the grounding that supports the HTTPU and HTTPMU sending of messages (see below). It makes the sending of datagrams possible before communication has been established.

HTTP, HTTPU and HTTPMU

These protocols are basic parts of UPnP. HTTP stands for Hypertext Transfer Protocol. HTTPU and HTTPMU are variants of HTTP, in particular, HTTP unicast and HTTP multicast. These variants are used for the delivery of messages over UDP/IP when multicast is used or it is not necessary to establish a connection ([42]).

SSDP

The Simple Service Discovery Protocol (SSDP) is a protocol that allows searching for UPnP devices on a network. It detects devices and network services that use the SSDP, such as UPnP devices. It also detects SSDP devices and services running on the local computer. Searches are made by sending a SSDP search request (on HTTPMU). In addition, it can refine its search to find only devices of a particular type, only certain services or even a particular device. A message is sent to all the devices on the network through the same channel, so each device must be listening through the multicast port. When it receives a search request, it checks the search criteria and, if there is a coincidence, answers by sending a unicast SSDP message, on HTTPU , with the code “200 OK”, which indicates that the request was successful. When a device is connected to the network, it sends several SSDP presence messages announcing the services it offers (delivery is not guaranteed over the UDP). The messages sent by the device have a link to the location of the document that contains its description, with its properties and the services it offers. In addition to the SSDP properties, it provides the device with methods for disconnection notification and updates the device's information using timeouts.

A SSDP packet is just an HTTP request with the statement "NOTIFY" (to announce) or with "M-SEARCH" (to find a service), leaving the HTTP body empty, and keeping UPnP-specific attributes in its head.

GENA

The General Event Notification Architecture (GENA) allows sending and receiving notifications using HTTP over TCP/IP and HTTPMU over UDP/IP. UDP multicast is useful because it allows a single report to be distributed to a potentially large group of receivers using a single request. GENA defines the terms of the subscriber and the publisher of the notifications, which enable the event's mechanism used by UPnP to warn of changes in the state of services. When a subscription to a service takes place, it sends event messages updating the changes in the status of the device. These event messages are in XML format. Apart from this, GENA is also used to create presence messages, which are sent using the SSDP protocol.

SOAP

The Simple Object Access Protocol (SOAP) provides a standard mechanism for packaging messages. It defines how two objects in different processes can communicate exchanging XML data. Thus, UPnP makes use of XML and HTTP to run remote procedure calls (RPCs), sending control messages to devices and getting the results or the errors in each case. Each control request is a SOAP message that contains the action invoked and all the necessary parameters. The response is another message of the same type with the state or the result of the action requested to the device.

Although many protocols are created to simplify the communication between applications (RPC from Sum, DCE from Microsoft, RMI from Java and ORPC from CORBA), the SOAP has received more attention because of the great support received from the industry. It has been accepted by almost all large companies. Consequently, it is becoming the standard for communication based on RPC over the Internet. Some of its advantages are:

- It is not associated with any language.
- It is not strongly associated with any transport.
- It is not tied to any distributed object infrastructure.

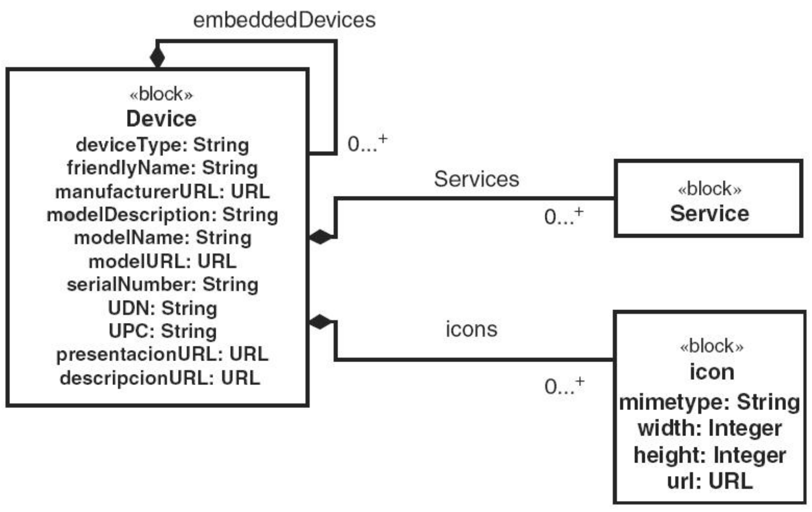


Figure 2.10: UML (Unified Modelling Language) class diagram of a UPnP device [41]

- It makes the most of the existing standards in the industry.
- It enables interoperability among multiple environments.

XML

Extensible Markup Language (XML) plays an important role in the exchange of data. It is similar to HTML, but its main function is to describe data and not to display them as is the case of HTML. XML is a format that allows reading data through different applications. Specifically, it can structure, store and exchange information ([43]). It is used in UPnP for device and service descriptions, control messages and events.

2.7.4 Components of a UPnP Network

A UPnP network defines various types of components, such as control points, devices and services. These are detailed below.

Devices

UPnP devices are logical containers for a service or set of services, and sometimes for other devices (embedded devices). Embedded devices can be discovered and used independent of the main container. Each UPnP device can offer any number of services. By itself, a device merely provides a self-description

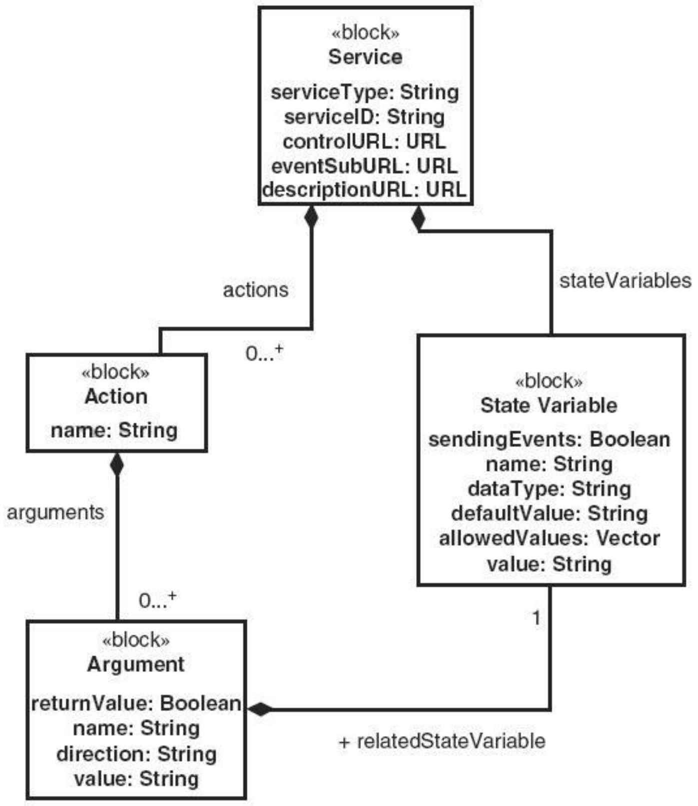


Figure 2.11: UML class diagram of a UPnP service [41]

of its information, such as developer, model name and serial number. Device services are those that provide real functionality (Figure 2.10).

There are different categories of UPnP devices, standardized according to the set of services provided by each device. This information (along with properties such as those mentioned above) is saved in an XML document that must be kept in the device until it needs to be sent.

Services

Each service in a UPnP device can contain any number of actions. An action has a name, a set of input parameters and a return value (optional). Each argument has a name, a value and an address. The address can be input or output depending on whether the argument is given to the action or returned by the action.

A service has a service identifier (URI) that identifies it from all the others; there cannot be two services with the same identifier. It can keep the

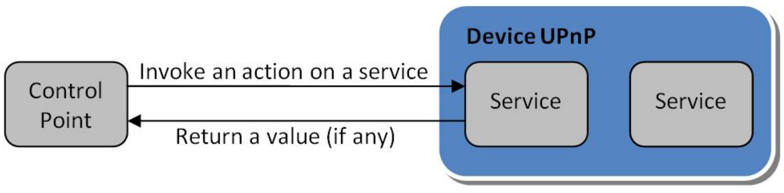


Figure 2.12: Control point invoking a service action [41]

variables that represent the current state of the service. These state variables have a name, type, default value, current value and a range of permissible values. If a variable sets an event to indicate a state, then it is an event notification variable.

A service is a state table, a control server and an event notification server. The state table contains the variables updated when there is any change in service status. The control server receives action requests and performs them, updates the state table and returns the result. The event notification server publishes updates of changes in the state of service (Figure 2.11).

Control Points

A control point is a network entity that invokes the functionality of a device. It is capable of discovering and controlling other devices. In terms of client/server in a UPnP network, the control point will be the client and the server role will be played by the device. Once the control point finds the device, it is capable of:

- Getting the description of the device and a list of related services.
- Getting the descriptions of the services in which it is interested.
- Invoking actions to control the service.
- Subscribing itself to the service's events (Figure 2.12).

When the status of the service changes, the event notification server sends an event to the control point. In short, a control point finds the devices, invokes the related actions to their services and signs up for event notifications. By contrast, a device responds to the actions invoked by the control point and sends the events when the variables change state.

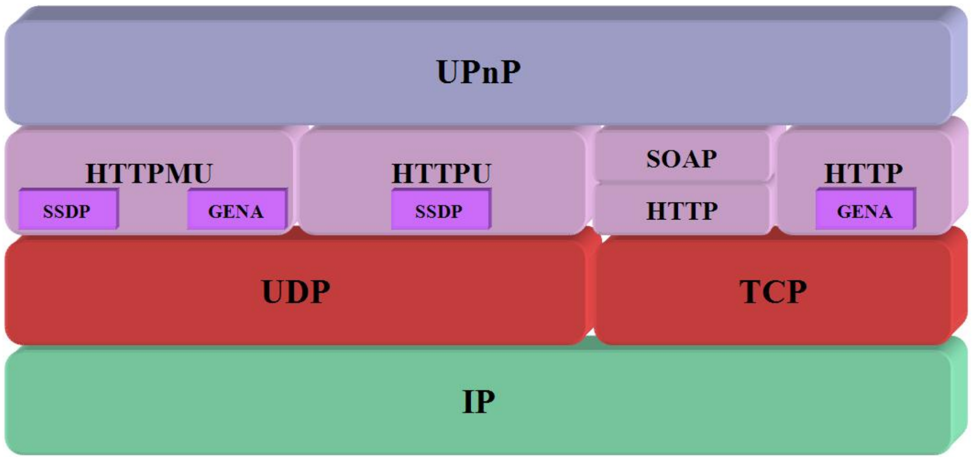


Figure 2.13: UPnP protocols stack [44]

2.7.5 UPnP Operation

To describe the UPnP way of working, we will show the development in six basic steps or stages: Addressing, Discovery, Description, Control, Event Notification and Presentation. The routing stage can be considered step zero. The representation of the protocol stacks used in each one of the following steps is shown below (Figure 2.13).

Addressing

Since all UPnP communications are based on the Internet Protocol (IP), a device must obtain an IP address before it can join to a network that supports UPnP.

The first step, also known as the ***zero phase***, is based on this; an address for the control points and devices connected to the network must be obtained. All the reasoning presented in this phase is valid for both devices and control points.

Addressing is the process by which a device automatically gets an IP address. It allows a device to join to the network and be prepared for communication with other devices and control points. The routing protocols implemented in the UPnP devices enables them to join dynamically to an IP network and to get an address without being configured by the user.

UPnP devices can use the DHCP, UDP-based, to retrieve an IP address from a DHCP server. To do this, both devices and control points must have a DHCP client. Being connected to the network, the first thing to do is to find a DHCP server that provides them an address. If this server already exists on the network, they must use the address they have been assigned.

If the network does not have a DHCP server, automatic IP addressing (Auto-IP) must be used to get the IP address. Through this mechanism, the device takes a random address within the 169.254/16 range to minimize potential collisions with other devices. This range was established by the ICANN (Internet Corporation for Assigned Names and Numbers) and the IANA (Internet Assigned Numbers Authority) for IP self-configuration in private networks. Once assigned an IP address using Auto-IP, it must be checked that this address is not used by any other device on the network using the ARP (Address Resolution Protocol). Each device must periodically verify the existence of a DHCP server on the network to manage the process of addressing. In this case, the automatically assigned IP is ruled out and so they start with the dynamic addresses assignment using the DHCP server.

First, a device or control point tries to contact a DHCP server to obtain an IP address. If it is unable to locate the server, the device uses Auto-IP, which allows devices to select addresses without having a server to assign it to them. It may be necessary to resolve the assignment to IP addresses because the devices can implement protocol layers higher than UPnP. To obtain this functionality, devices must incorporate a DNS client and support the DNS dynamic registration.

Discovery

The discovery phase defines how a device announces its presence and how control points discover it. A UPnP device is like a mini web server that can be detected and monitored by a control point. The discovery process allows control points to find devices and services of interest and obtain information about them. The devices use the SSDP to announce their services to control points. These last ones use the SSDP to search for devices. In the tower of protocols below, you can see certain color codes that match the parts of each message defined below. These color codes are useful until the operation's description of the UPnP technology is finished (Figure 2.14).

Once a device has acquired an IP address, the SSDP announces its services to all the control points of the network. Similarly, when you add a control point to the network, the SSDP searches for relevant devices on the network. They will answer if there is an agreement with the data of the search message. The message exchanged in both cases is a discovery message that contains essential details about the device or its services, such as the type of device, identifier and a pointer to more detailed information (Figure 2.15).

It must be kept in mind that when a control point or device initializes and connects to the network, it must wait a random time between 300 and

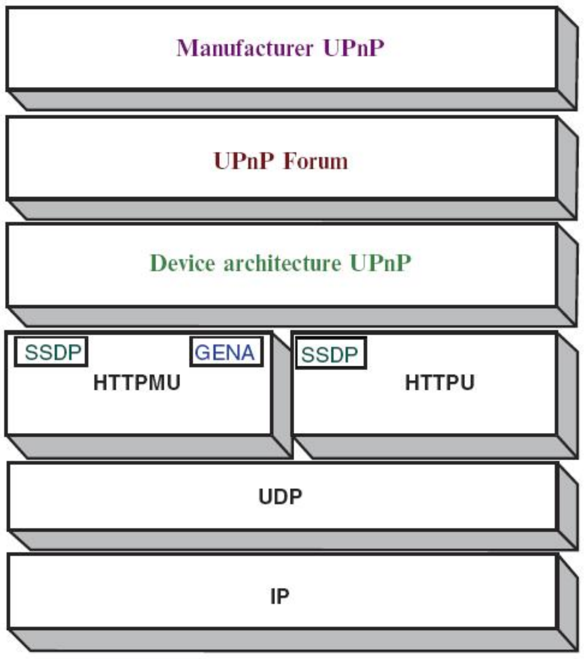


Figure 2.14: Protocol stack for discovery

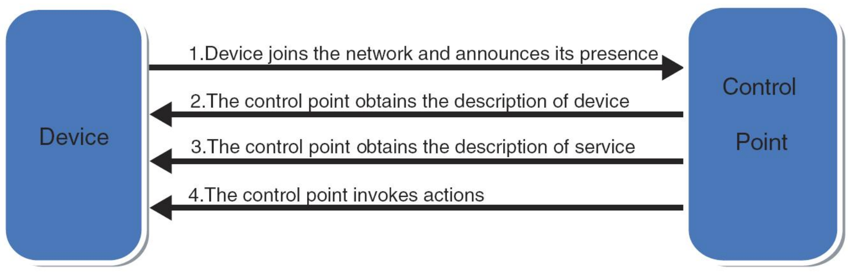

Device
Control Point
1.Device joins the network and announces its presence
2.The control point obtains the description of device
3.The control point obtains the description of service
4.The control point invokes actions

Figure 2.15: Recovery of the descriptions of a service and a device [41]

3,000 ms before sending any message of discovery. These ranges are set to avoid problems when many devices connect to the network at the same time (300 ms) and to minimize delays in the recovery of a network (3,000 ms) [45].

A URL of the XML document describing the device is included in the discovery and the advertisement responses of every device. This URL provides the necessary information to the control points to retrieve the descriptions of the devices and their services. All services contained in a device have three URLs that provide the necessary information to allow the control points to communicate with them:

- The URL of **control** is where the control point sends requests to control the service. UPnP device manufacturers specify one for each device.
- The URL of **subscriptions to events** is where control points send requests to subscribe to events. There is a URL for this kind of service in each device. If a service does not have event variables, and therefore no notification of events, the element URL of subscriptions to events must appear, but it will be empty.
- The URL of **description** indicates the location of the control points from which the service description document will be retrieved. The service description document is returned by an HTTP GET request.

A control point has two possibilities to search for devices. It can pick up a notification message sent by a device or it can request the response of the device using a discovery message sent by the control point itself (Figure 2.16).

The devices must refresh their advertisement messages periodically because they have a limited lifespan. For this reason, they are not obliged to cancel those sent previously (announcing their capabilities) when they disconnect from the network.

There are two ways of discovering devices:

1. **Advertisement**

   Once a device joins to the network, it announces its embedded devices and its services to control points through NOTIFY messages defined by GENA. These are multicast messages that use the SSDP. These messages are sent to the address and port (239.255.255.250:1900). This default value is indicated by ICANN/IANA to use it with the SSDP. The control points are supposed to listen to arriving messages in this port, knowing this way the capabilities that are available on the network. These messages do not require a response. One important fact

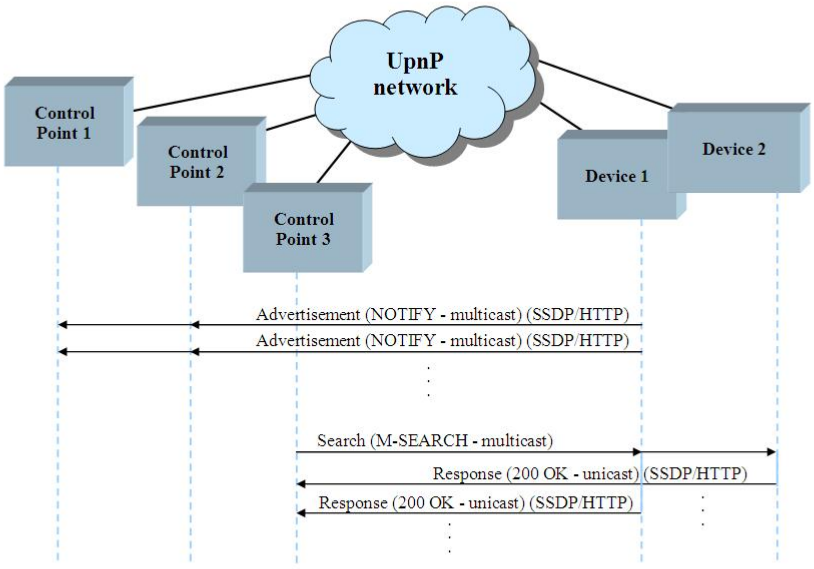

UpnP network
Control Point 1
Control Point 2
Control Point 3
Device 1
Device 2
Advertisement (NOTIFY - multicast) (SSDP/HTTP)
Advertisement (NOTIFY - multicast) (SSDP/HTTP)
Search (M-SEARCH - multicast)
Response (200 OK - unicast) (SSDP/HTTP)
Response (200 OK - unicast) (SSDP/HTTP)

Figure 2.16: Discovery step [46]

about advertisement messages is the time of validity, which indicates the period in which the device is available. After finishing this period without sending an advertisement message, the device will stop being available on the network.

During the advertisement process and considering that a root device has **d** embedded devices and provides **k** different types of services, a total of $\mathbf{3 + 2\ d + k}$ advertisement messages are sent to the network. This can be deduced, assuming they are different devices, by interpreting the number of messages that should be sent by a device:

- A message for each type of service with *NT=type of service*.
- A message for each type of device (root or embedded) with *NT=device type*.
- A message for each device (root or embedded) with *NT=UUID of the device*.
- A message regarding the root device with *NT=upnp:rootdevice*.

2. **Search**

   This procedure is activated when a control point requires a type of device or a specific service. This is when the control point sends a multicast message with the address and port specified above, i.e.,

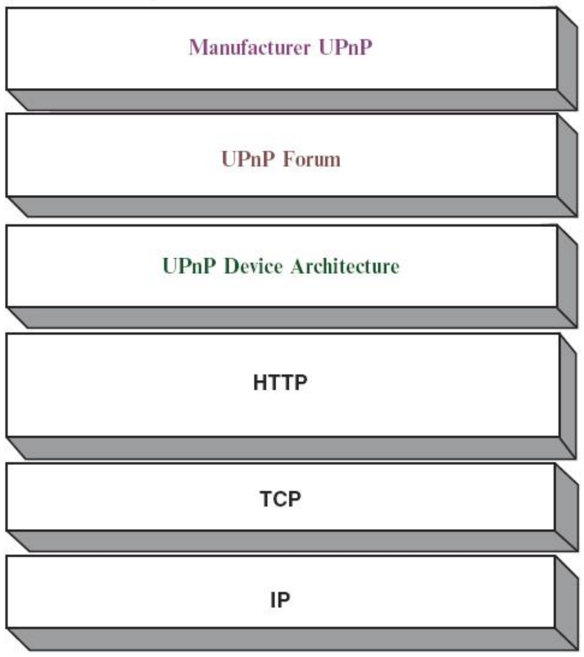


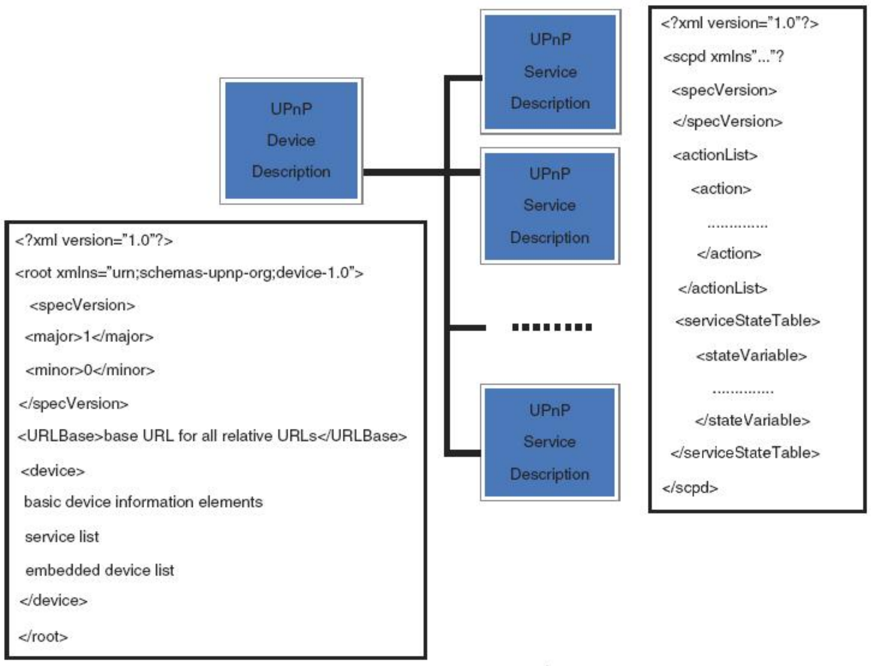
Figure 2.17: Protocol stack for description

239.255.255.250:1900. In this case, unlike in the method of advertisement, it will require answers from the devices that fit with the specifications defined by the control points. A control point must send multiple ***M-SEARCH*** messages since the messages are sent over the UDP (which does not guarantee delivery). A control point will receive multiple messages, but some will be duplicates. To filter these replies, the control point uses the USN header, which provides a unique identifier to look for answers.

Description

The description enables a device to list all the features it can provide. The descriptions of the devices and their services are stored in XML documents. The device summarizes its services and capabilities in a standard format. A device description document contains device information (such as developer, make, model and serial number), a list of the services provided by the device and a list of its embedded devices. A service description document contains detailed information about the device's service, the actions that the service provides, the parameters and values returned by the service (Figure 2.17).

The answers to the search messages received by a control point contain



```
<?xml version="1.0"?>
<root xmlns="urn;schemas-upnp-org;device-1.0">
  <specVersion>
 <major>1</major>
 <minor>0</minor>
</specVersion>
<URLBase>base URL for all relative URLs</URLBase>
 <device>
 basic device information elements
 service list
 embedded device list
</device>
</root>
```

```
<?xml version="1.0"?>
<scpd xmlns"..."?
  <specVersion>
  </specVersion>
  <actionList>
     <action>
        .............
      </action>
   </actionList>
   <serviceStateTable>
      <stateVariable>
        .............
      </stateVariable>
  </serviceStateTable>
</scpd>
```

Figure 2.18: Hierarchy in the device and service description [41]

URLs that provide descriptions of the capabilities of the device. Control points use these description documents to get more information from the devices, trying this way to get their features and interact with them.

The description of a UPnP device consists of two parts: the device description, which refers to the physic and logic container, and the service description, which refers to the capabilities offered by the device. Both descriptions are provided by the developer and are written in XML.

Devices may contain other logic devices apart from services. The UPnP description document includes a list of integrated devices and a description of the available services. For each service, its description includes a list of actions to which the service replies and the arguments for each action. The service description also includes a list of variables that reflects the state of the device. These variables are described in terms of their types of data, ranges and characteristic events (Figure 2.18).

To receive the description of a device, the control point sends an HTTP request using the GET method to the URL contained in the discovery message that had previously been received by the device. When it receives the request, it replies with an HTTP message that contains the device's description in the message's body.

The URLs of the device's description of its services are included in this description. The information contained in the device description consists of:

- An XML document containing various data from the device.
- The meanings of all nested devices.
- A list of all services supported by the device, including state variables and actions.

The control point can send another HTTP request containing the URLs of the service descriptions to reacquire the service descriptions. The format of the control point's request is shown below (it is important not to forget about the blank line at the end of the header):

GET: description route HTTP/1.1
HOST: host:port
ACCEPT-LANGUAGE: control point's favorite language.

The syntax of the device's response message is shown below, and the device's or service's description will appear in the body.

HTTP/1.1 200 OK
CONTENT-LANGUAGE: language used in the description
CONTENT-LENGTH: length of the body, in bytes
CONTENT-TYPE: text/xml
DATE: time to answer

For each service that contains a device, the description contains (in addition to what was stated above) the name and type of service, service description URL, URL for control and URL for event notification. Finally, the device description also provides a description of all nested devices and a URL for presentation (Figure 2.19).

Control

Control is the UPnP phase in which the control points invoke actions to the services of the devices. Once a control point has all the information about a device and one of its services through its description, it will be able to control the service by invoking actions. The protocol stack that supports the control phase in the running of UPnP is shown in Figure 2.20.

To control the device, UPnP is based on the SOAP, which uses XML and HTTP to provide web messaging and RPC. XML makes public the content of the message and HTTP sends the message to its destination. The SOAP comprises four parts:

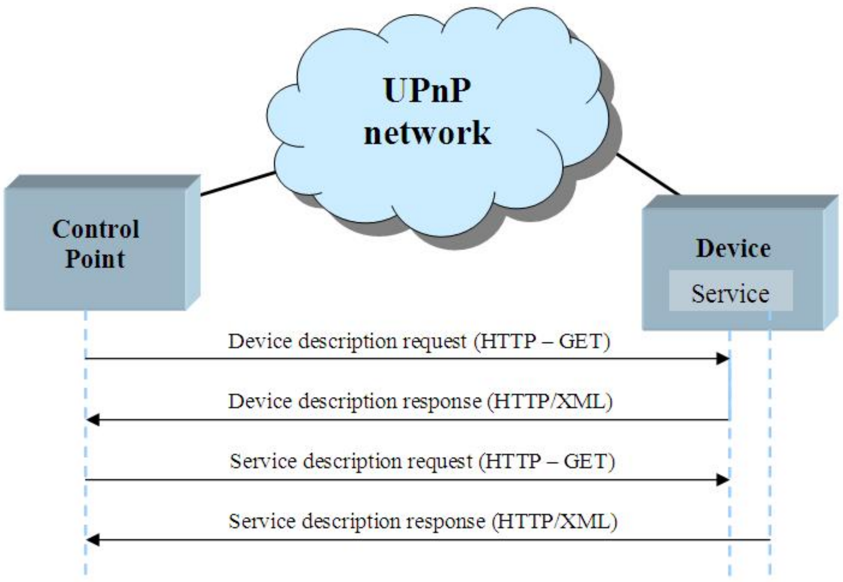

UPnP network
Control Point
Device
Service
Device description request (HTTP – GET)
Device description response (HTTP/XML)
Service description request (HTTP – GET)
Service description response (HTTP/XML)

Figure 2.19: Description step [46]

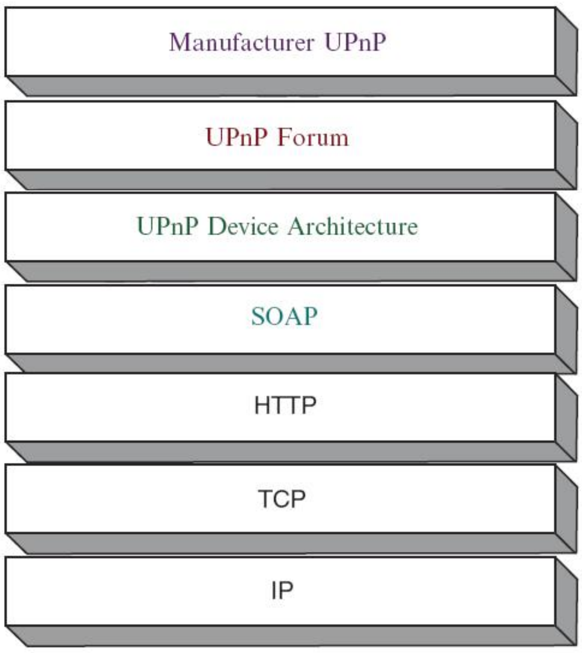


Figure 2.20: Protocol stack for control

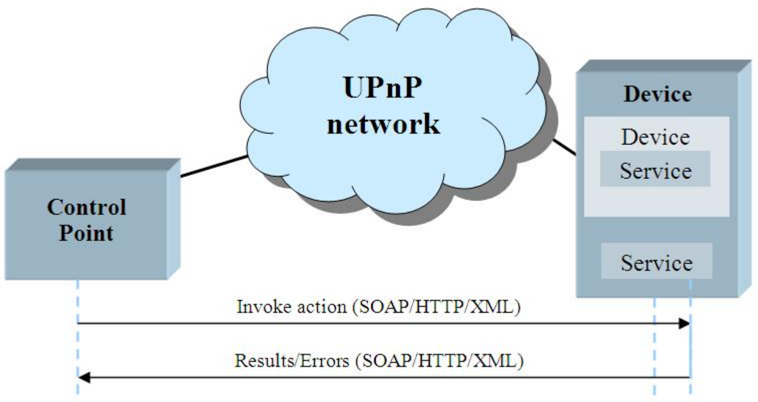


Figure 2.21: Control step [46]

- *Extensible and required envelope to encapsulate the data.* The SOAP envelope defines a SOAP message, and this is the basic unit of exchange between SOAP message processors. This is the only obligatory part of the specification.

- *Optional rules for encoding data represent data types defined by the application.*

- *Link between SOAP and HTTP.* This part is also optional since the SOAP can be used in combination with any transport protocol or mechanism that can transport the SOAP envelope.

- *RPC Model.* Its purpose is message exchange (request/response). It is a convention to represent RPC and its responses.

To invoke an action, the control point sends a message to the control URL that it already knows from the description phase explained above. The device will respond with the result or the errors obtained after running the service action. Moreover, this action may change the state of the service and, therefore, change some of its variables (Figure 2.21).

To invoke a specificaction, the control point must send a SOAP request using the POST method to the service device. This control message contains information specific to the manufacturer, name of the action, names of the arguments and variables that are defined by the UPnP Forum.

Requests for the state variables were considered in UPnP, but this way of working has been discarded by the UPnP Forum and must not be used for control points. Instead, the working committees and the manufacturers

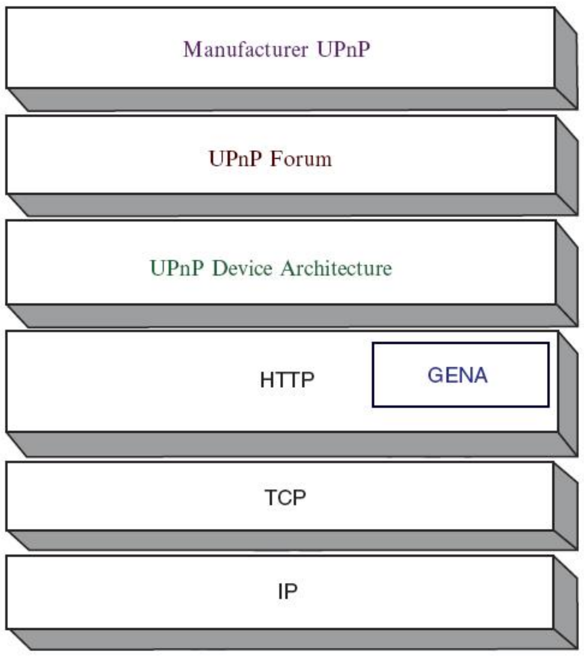


Figure 2.22: Protocol stack for event notification

define actions that return the variable's value and that can be invoked by the control points.

Event Notification

Event notification offers the possibility of notifying a control point when the state of a device changes. As explained above, a service description contains a list of variables that model the state of the service. If any of these variables is likely to be reported as an event, the service publishes updates when any of these variables are modified. The protocol stack used in this case is shown in Figure 2.22.

The event notification system uses a publisher/subscriber model in which the control points can subscribe to events sent by a service. The services publish event notifications to subscribers. An event is a message sent from a service to the subscribed control points. The events inform the subscribed control points about the state changes in the service (Figure 2.23).

A control point that wants to be notified about changes in the state of the variables subscribes to an event source by sending a subscription request to the URL of the events, which is contained in the corresponding device description. The subscription application must include the service to subscribe,

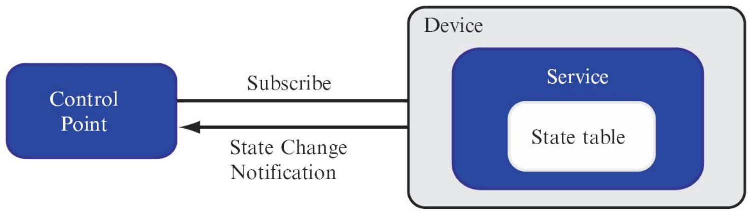


Figure 2.23: Subscribing and notifying [41]

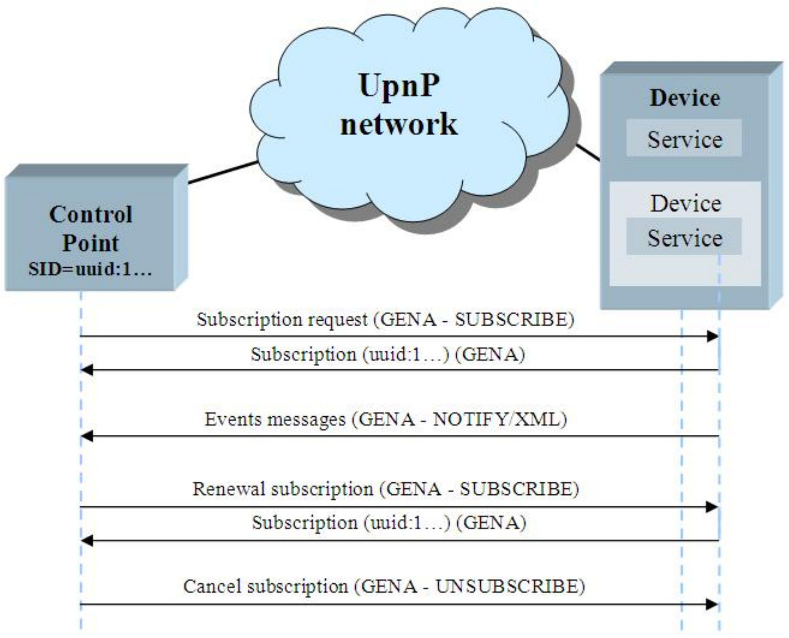

UpnP network
Control Point
SID=uuid:1...
Device
Service
Device
Service
Subscription request (GENA - SUBSCRIBE)
Subscription (uuid:1...) (GENA)
Events messages (GENA - NOTIFY/XML)
Renewal subscription (GENA - SUBSCRIBE)
Subscription (uuid:1...) (GENA)
Cancel subscription (GENA - UNSUBSCRIBE)

Figure 2.24: Event notification step [46]

a URL to send events and a subscription time.

If a service accepts the subscription request, it responds with a unique identifier of subscription (SID) and the life of the subscription, which indicates its validity period. This unique identifier allows the control point to refer to the subscription service for future applications to the service, such as renewing or canceling the subscription (Figure 2.24).

Event messages are sent to all subscribers regardless of the reason for the change in state variables. These messages contain information expressed in XML with the names and values of those state variables configured in the service as event variables. The event notification protocol is GENA and, as seen in the previous protocol stack, it is used in the TCP transport, which guarantees message delivery to the subscriber. When the subscription expires, the subscription identifier becomes invalid and the service stops sending events to the corresponding control point. If this control point attempts to send any message (renewal or cancellation, but not the subscription), the service is rejected because the ID is no longer valid.

The control point will send a subscription message to the URL of the service to receive its events. This message uses the SUBSCRIBE method defined by GENA and its syntax is:

SUBSCRIBE event route of the service HTTP/1.1

HOST: host:port

CALLBACK: <delivery URL> NT:upnp:event

TIMEOUT: request for the lifetime, in seconds

A blank line must be added at the end of the last header. When this message is received, the service establishes a list of subscribers with the following information for each of them: SID, URL for the event messages delivery, event counter and length of subscription.

If the subscription is accepted, the service sends a message with the identifier of the subscription and validity period. This message has the following syntax. It is important not to forget about the final blank line:

HTTP/1.1 200 OK DATE: request time

SERVER: OS/version UPnP/1.0 product/version

SID: uuid:subscription UUID

TIMEOUT: lifetime of the subscription, in seconds

The first event notification message must be sent after the message above. It contains the names of the variables and their current values in XML. In addition, each time that one of these variables, which are set as event variables, changes the service must send an event message to all subscribed control points.

These event notification messages are labeled with a different key for each subscriber to detect errors. In every control point, in the initial event message, this key is set at zero and increases with each subsequent notification message. This way, if the subscriber receives a notification with an incorrect key, it will reply to the service with an error message.

All subscriptions must be renewed periodically for the control points to go on receiving notifications. To keep a subscription active, the control point must send a renewal message before the subscription expires. The renewal message is sent to the same URL as the original subscription message, but this time it does not include the URL for event message delivery. Instead, the renewal message includes the subscription identifier received in the initial message, which confirmed the subscription. We can see this message format

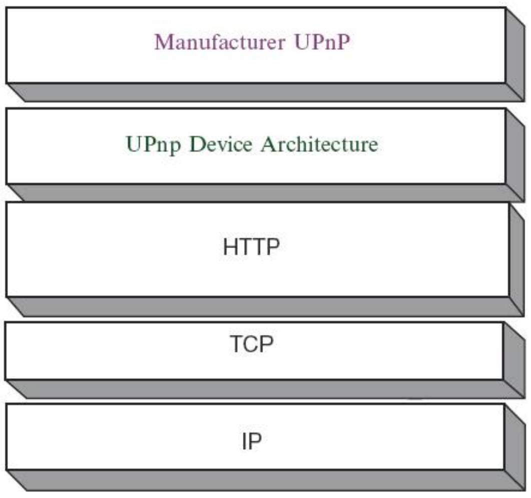


Figure 2.25: Protocol stack for presentation

below and, as already mentioned, it must include the blank line:

SUBSCRIBE: service route HTTP/1.1
HOST: host:service port
SID: uuid:susbcription UUID
TIMEOUT: request for the time of subscription, in seconds

The answer to this message is exactly the same as in the subscription message case. When a control point does not want to get any more events from a service, it can call off its subscription by sending a cancellation message:

UNSUBSCRIBE: service route HTTP/1.1
HOST: host:service port
SID: uuid:subscription UUID

The answer to this message is, as in the case above, an HTTP confirmation. If the control point abruptly disconnects from the network without sending the message to cancel the subscription, the service will keep on sending it notifications until the subscription time expires.

Presentation

In a UPnP network, a control point can monitor a device or check its status through the presentation of an HTML page. A home page can be loaded by the control point in a browser and this allows users to view and control the device. The protocol stack required for this is shown in Figure 2.25.

Home pages are not necessary; if a device has no home page, it can still be controlled through standard control messages. If the device allows a home

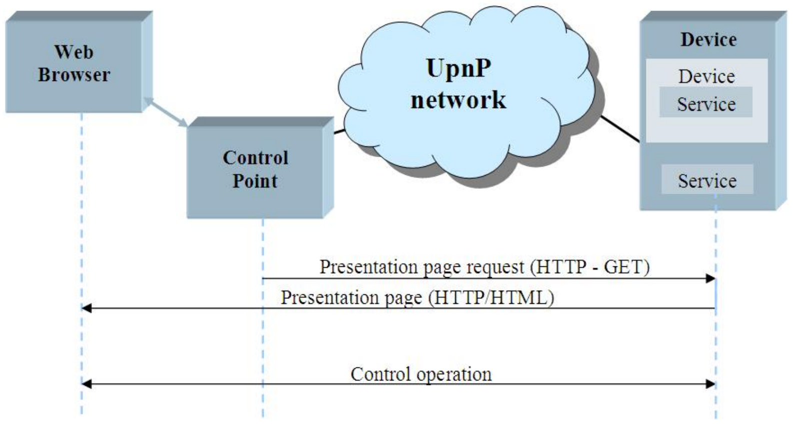


Figure 2.26: Presentation step [46]

page, its description document contains the URL for the presentation page on the label <*presentationURL*>. This label must always be present. If the device has no home page, the label will be empty.

In the presentation phase, the control point sends an HTTP request using the GET method to the presentation URL (available in the device description) and the device then returns to the home page. After loading the page in the browser, the control point can monitor the device or check its variables. The diagram in Figure 2.26 shows this.

The presentation message for requests includes the field *ACCEPT-LANGUAGE*, and the language of the presentation page will be defined by the *Content-Language* field, which is defined in the device.

An additional component of a UPnP network is the *application layer*. The capabilities of a device are defined by itself and the service models that provide the framework for the network components (description, control and events). A device manufacturer can develop these models by itself or work with other manufacturers inside the UPnP Forum to prepare the standard for the device and the service models. Currently, the working committee for UPnP has developed definitions for standard models.

2.7.6 Application Examples

Some modern projects work with UPnP, such as [47], which compares it with other distributed systems such as CORBA. [48] used UPnP to build a middleware layer for a home network. It is important to notice that, for this project, we have used device emulators (TV, fridge, etc.). This will be important for future implementations of the final solution.

[49] demonstrated UPnP protocol integration in a system consisting of a robot to manipulate objects. This paper described the design of the system systematically, the data used to compile the XML document of services and the definition of actions and control variables. In [5], there were conclusions to develop a flexible and low cost home automation, which has been implemented using UPnP.

This architecture is also used for sensor networks. [50] studied the design of a network with an interface between wireless sensor networks and UPnP via TCP/IP. This application makes possible the communication between control points and sensors and provides the use of web technologies for the control interface. By contrast, [51] discussed the few resources that present the sensing devices using a UPnP agent (BOSS, bridge of the sensors) that acts as an interface between the PC and the elements not supported by UPnP.

Currently, you can find different solutions in the market for developing UPnP systems, highlighting initiatives such as CyberLink for Java [52], a Java implementation that automatically controls all the internal aspects of the protocol and allows the programmer to focus on the business layers and the tool's interface.

2.8 DLNA

The DLNA (Digital Living Network Alliance) is an international and collaborative organization of companies involved in consumer electronics, industrial computers and mobile devices.

DLNA is a standard that allows different devices from the same network connected together to share information easily and without complicated configurations [53]. This system works with both wireless and Ethernet networks and even with the power supply. The DLNA has established a set of standards for the platforms and infrastructure software to be completely compatible. It focuses on the interoperability among mobile devices associated with multimedia images, digital audio and digital video.

Thus, assuming that all available devices on the network support this technology, a copy of the content and the network can be accessed from any device. In other words, we can listen to music from the files stored on our computer, watch movies stored on the digital video recorder on our computer or see photos of our camera on the TV. Figure 2.27 shows a possible scenario using this technology.

The objectives proposed by this technology are listed below:

- Digital music should be easily captured, stored and accessed from anywhere in the house. Digital photos should be managed, viewed and

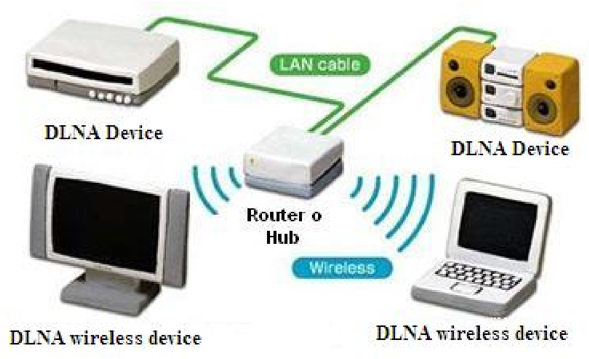


Figure 2.27: Interoperability between devices using DLNA

printed very easily.

- It must be possible to read content anywhere and enjoy it while traveling by car or walking down the street (there are already projects to synchronize information).
- It must be possible to save the distributed content to be able to see it as many times as we want.

2.8.1 General Features

The digital home is an electronic network made up of PC and mobile devices that cooperate transparently. The aim of DLNA is to become a home network for all its global customers. This objective integrates the interoperability of the three digital islands within the home: the Internet, broadband electronic network and island of mobile devices (Figure 2.28).

The DLNA network must have at least a server and a client to work. The main objective of DMS (Digital Media Servers) is to provide multimedia content to DMP (Digital Media Players), which act as clients. These devices include camcorders, digital cameras, game consoles and mobile phones, but they need to be certified, that is, they must have integrated the electronics and configure to the DLNA standard.

DLNA makes use of a part of the technology developed for UPnP that allows the discovery of other devices on the local network. DLNA is based on UPnP and IETF (Internet Engineering Task Force) technologies [54]. The DLNA standard is based on standards established in the industry and developed by groups such as IETF, World Wide Web Consortium (W3C), Motion Picture Experts Group (MPEG) and the UPnP Forum. Interoperability between devices is transparently performed by providing a particular service to

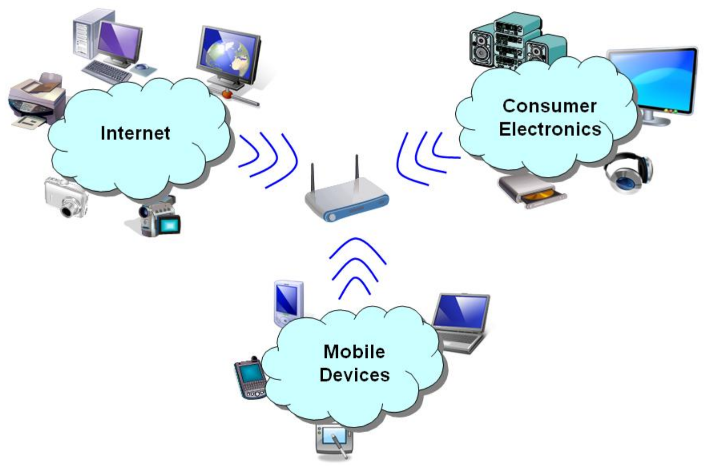


Figure 2.28: Objective of DLNA: digital islands at home

the user. This includes the ability of the devices to communicate with each other and exchange useful information.

The interoperability guidelines require that all devices must support connectivity via Ethernet, Wi-Fi or Bluetooth. It uses TCP/IP for all network connections and works with HTML and the SOAP for transport and media management. The required formats to support images, audio and video are also defined. They are JPEG, LPCM (Linear Pulse Code Modulation) and MPEG2, respectively.

DLNA is based on a specification created by the working groups of the UPnP Forum. This specification is the UPnP AV (Audio and Video UPnP), and it has been the greatest success for these working groups, at least in terms of digital content (Figure 2.29).

DLNA Model for Devices

The model for devices used by DLNA comes from the UPnP Forum and consists of devices, services and control points. The devices are network entities that provide services. These services are the basic control units and they perform actions to keep a state through its variables. The control points are network entities used to discover and control other devices on the network. A group of multiple devices can be controlled by a control point.

In the UPnP standard, interoperability was first between the control point and a single device. However, with the evolution of the specification of UPnP AV (and DLNA as well) the basic model of devices was improved. For this reason, although interoperability between the control point and device still

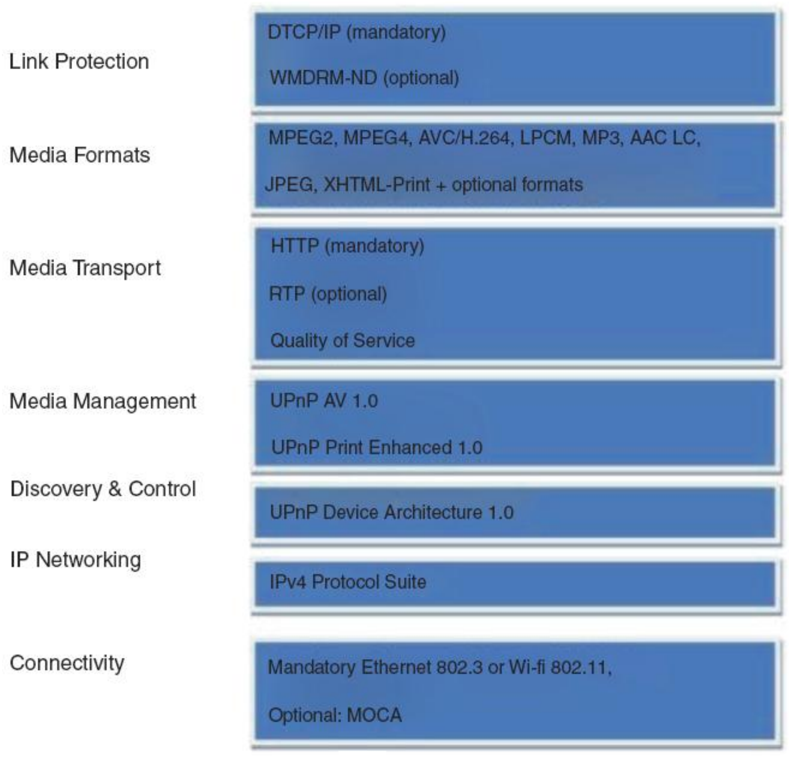


Figure 2.29: DLNA interoperability model

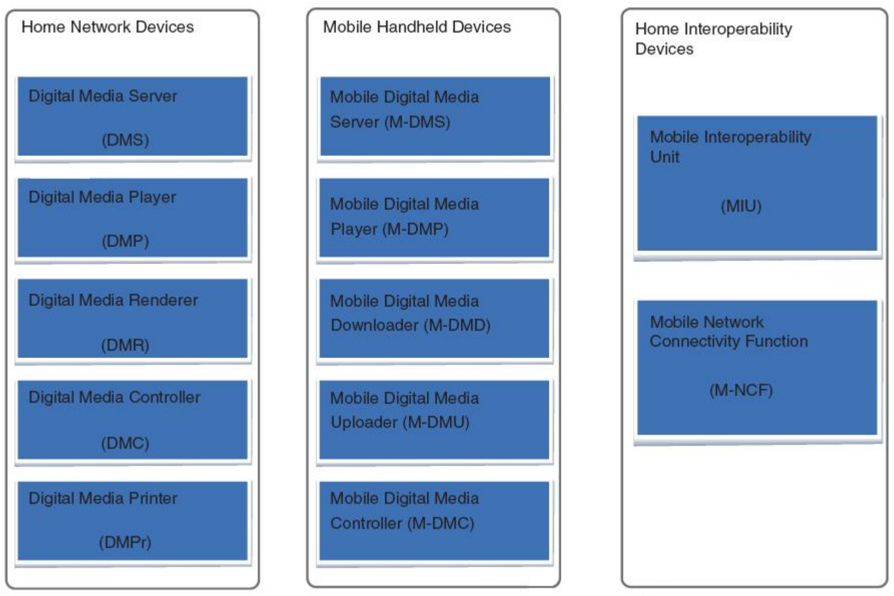

Home Network Devices
Digital Media Server (DMS)
Digital Media Player (DMP)
Digital Media Renderer (DMR)
Digital Media Controller (DMC)
Digital Media Printer (DMPr)
Mobile Handheld Devices
Mobile Digital Media Server (M-DMS)
Mobile Digital Media Player (M-DMP)
Mobile Digital Media Downloader (M-DMD)
Mobile Digital Media Uploader (M-DMU)
Mobile Digital Media Controller (M-DMC)
Home Interoperability Devices
Mobile Interoperability Unit (MIU)
Mobile Network Connectivity Function (M-NCF)

Figure 2.30: Categories and kinds of DLNA devices

works, it has been extended to other devices so that they can interact with each other by exchanging digital content using different communication protocols (Figure 2.30).

There are 12 kinds of DLNA devices in three different categories. The category Home Network Device (HND) consists of five classes of devices that share the same use on the network system, with the same media formats and connectivity requirements.

- *DMS*. These are devices that can originate, acquire, record and store media on the model of interoperability in the digital home. There are DMS that help to protect the content saved. These devices, in case a customer is not able to handle a particular format, must be able to convert the file into another format before sending. Some examples of these devices include digital video recorders, computers, home cinema with hard disk drives (such as music servers), devices to capture video and images and multimedia mobile phones. We can see the protocols and services of DMS in Figure 2.31.

- *DMP*. These devices select and play the digital media stored on the network and include TV monitors, home cinema, PDAs, multimedia mobile phones, consoles and digital media adapters.

- *Digital Media Renderer (DMR)*. Devices that reproduce the content

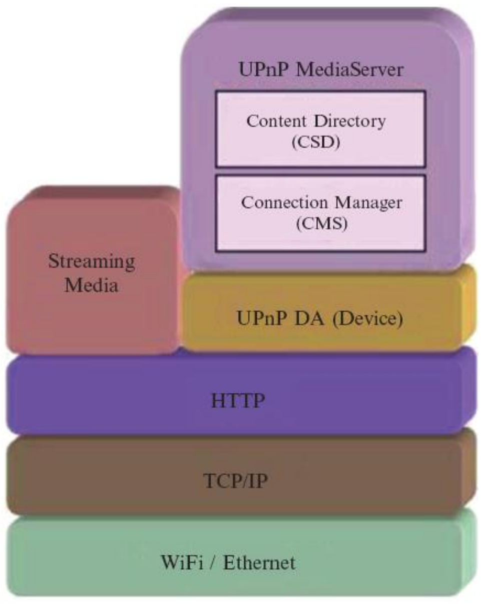


Figure 2.31: Protocols and services of DMS

received from DMS or their mobile counterparts after being configured by another device on the HND, such as a Digital Media Controller (DMC, see below). DMC and mobile DMC devices will be explained in subsequent studies. Examples of such devices include televisions, audio/video receivers and remote speakers for music. The services and protocols of a DMR are show in Figure 2.32.

- *DMC*. This device has the ability to find content exposed by DMS and adapt it to the rendering capabilities of a DMR, establishing the connections between them. It can also send instructions to another device, such as telling a server to play a particular video on a TV or sending a photo to a printer. A possible example of a DMC could be a learning remote control or a multifunctional device such as a multimedia mobile phone.
- *Digital Media Printer (DMPr)*. These devices provide printing services to the home network. Some examples are a network printer or an application running on a PC with a USB-connected printer.

The category Mobile Handheld Device (MHD) consists of five classes of devices that use the same model as in the HND category, but have different requirements for media formats and network connectivity. This category includes the following kinds of devices and features:

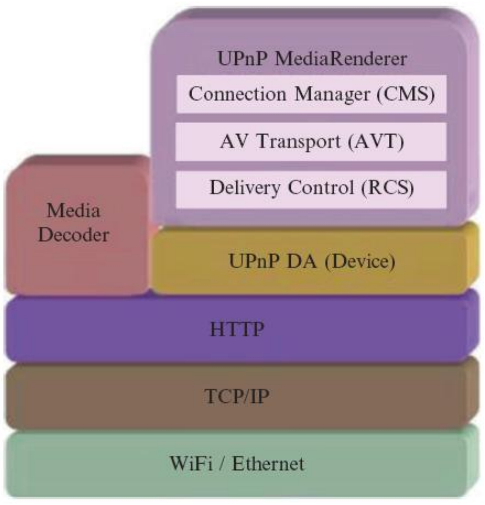


Figure 2.32: Protocols and services of DMR

- ***Mobile DMS***. Wireless devices that provide and distribute content to a mobile DMP, DMR or DMPr. Examples of these devices are mobile phones and music players.

- ***Mobile DMP***. These devices are able to find and play the content offered by DMS or mobile DMS and play it in a local environment. An example of this kind of device may be a media tablet, which is a portable player with Wi-Fi connectivity that can be used as an Internet browser.

- ***Mobile Digital Media Controller***. A device that finds content offered by a mobile DMS and adapts it to the capabilities of a DMR, establishing connections between the server (DMS) and renderer (DMR). A PDA and an intelligent remote control are examples of such devices.

- ***Mobile Digital Media Uploader***. These wireless devices send (load) a mobile DMS or DMS with an upload functionality. A digital camera and a phone with an integrated camera are examples of such devices.

- ***Mobile Digital Media Downloader***. This finds and downloads the content exposed by DMS or mobile DMS and reproduces it after downloading. An example is a portable music player.

MHDs interact with stationary devices in the DLNA digital home and allow a wide variety of uses. Some examples include:

- Play images and videos taken from a MHD on a TV.
- Remote control function.
- Uploading images, music and video clips to a media server.
- Download images to a server using its controls.

The category Home Infrastructure Device (HID) integrates two kinds of devices. These devices are designed to enable MHDs and HNDs to interact.

- ***Mobile Network Connectivity Function***. These devices provide a bridge function between the network connectivity of MHDs and HNDs.
- ***Media Interoperability Unit***. Devices that make possible the change of format in multimedia content between HNDs and MHDs.

These 12 kinds of devices enable the sharing of digital content over a network. The three basic classes that must be in a DLNA network are DMS, DMP and DMC, and a particular device can do the functions of one or more of these basic devices.

The ways of working of these devices on the DLNA network, or the phases that it has to carry out, are similar to those described for UPnP. In the next picture, we see a representation of how to proceed in DLNA (Figure 2.33).

2.8.2 Specific Features

Nowadays, the IPv4 protocol family is used, but the IETF is standardizing IPv6 as an enhancement of this version. The use of IP in the digital home brings us many benefits:

- It allows us to run applications over different means that can communicate in a transparent way. IP provides the framework that allows applications to be independent of the transport technology.
- It allows connecting all the devices in the home to the Internet. Using IP, every digital home device can connect to any other connected to the Internet.
- IP connectivity is cheap. Its implementation makes sure that IP is available at a lower cost than that of other technologies. Therefore, IP support in the current digital home is essential for interoperability among devices. The Figures 2.34 shows the protocol stack used by DLNA 1.5.

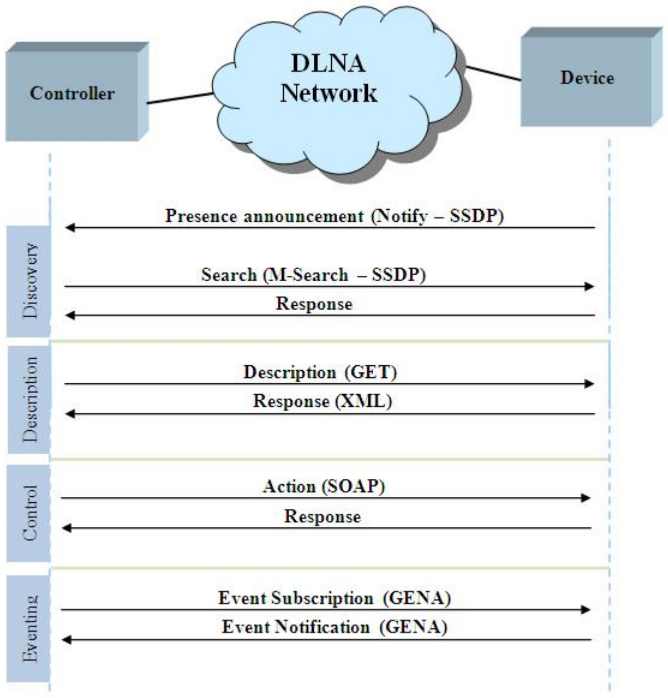


Figure 2.33: DLNA operation

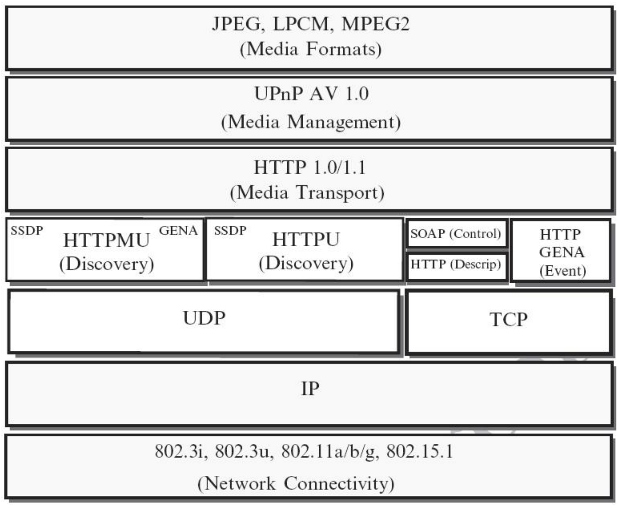


Figure 2.34: DLNA protocols stack [55]

The base for DLNA is the TCP/IPv4 protocol. Each device must implement a DHCP client and look for a DHCP server the first time it connects to the network. The device must use the IP address assigned by this server and, in case it does not find any server, the device will use Auto-IP, which means that it will generate an IP address within the 169.254/16 address range. The first and last 256 addresses in this range are reserved and cannot be used. Once it has an address, it must determine whether that IP is available using ARP. If the device receives a response, it is assumed that the chosen IP is currently in use on the network and must generate a new one. In addition, the device must periodically check the existence of a DHCP server.

The technologies for the network connectivity that can be used in DLNA are Ethernet 10Base-T and 100Base-T (802.3i/802.3u) for wire connections, Wi-Fi (802.11a/802.11b/802.11g) for wireless connections and Bluetooth for wireless connections in handheld devices. In future, the idea is to start working with Ethernet 1000Base-T (802.3ab) and faster Wi-Fi connections (802.11n). It is also important to know that technologies such as LonWorks, CeBus, X-10 and Universal Powerline Bus are supported through UPnP bridges.

To protect digital media devices, DLNA technology makes use of digital rights management, which restricts the use of the media and devices. To protect the links (encryption/decryption) it is necessary to include a layer above all others in the protocol stack. This layer is based on Digital Transmission Content Protection (DTCP)/IP, which is needed to establish secure interoperability, and WMDRM-ND, which is optional and provides access to additional content.

DTCP/IP is a technology to protect links and is particularly adapted to work over IP [56]. It is used to provide security to commercial content. It allows for the establishment of a secure authenticated channel that supports data flow (streaming) with limited copying rights: copy once, never copy and copy-restricted rights.

Media Format

The media format describes the way to encode and the format for each one of the three kinds of media: audio, video and video with audio (AV). The term format is equivalent to codec or codec family.

The media format model is intended to achieve interoperability on the network, while the innovation in the media codec technology goes on. It defines a set of media formats and a set of optional media formats for audio, video and AV. DLNA also provides rules for the use of optional formats between compatible devices and converts optional formats into mandatory

Table 2.6: Mobile household appliances

Media Format	Mandatory formats for household devices	Optional formats for household devices	Mandatory formats for mobile devices	Optional formats for mobile devices
Image	JPEG	GIF, TIFF, PNG	JPEG	GIF, TIFF, PNG
Audio	LPCM	MP3, WMA9,	MP3 y MPEG4	MPEG4, AMR,
	(2 channels)	AC-3, AAC, ATRAC3plus	AAC LC	ATRAC3plus, G.726, WMA, LPCM
Video	MPEG2	MPEG1, MPEG4,	MPEG4 AVC	VC1, H.263,
			(AAC LC	MPEG4 part 2,
			Asoc. Audio)	MPEG2, MPEG4 AVC

ones and vice versa. In the following table, we can see both the mandatory and optional formats for fixed and mobile household appliances (Table 2.6).

Media Transport

Media transport defines how the data move through the network. The grounding of the DLNA transport for any device that deals with media content through the network is HTTP 1.1. It is necessary to use this protocol, but there is also an optional protocol of transport in DLNA, namely the real-time transport protocol (RTP).

Management of Media, Distribution and Control

Media managing allows devices and applications to identify, manage and distribute digital content across devices on the network. UPnP technology AV is the solution for the management and control of devices developed according to the guidelines for the interoperability of devices on the network. UPnP AV architecture allows devices to support the entertainment content in any format and in any transport protocol. The services provided by this technology are:

- ***Content Directory Service***. This service provides a mechanism for each content server on the network as well as a standard directory and all its available content to any interested device. It enumerates the content and presents a logic structure for the multimedia library available on the server, such as videos, music and images.

- ***Connection Manager Service***. Determines the way content can be transferred from the media server to a media player device. This service is used to carry out one of the following actions:

  - Match the capacity between the server and player devices.

  - Set up and remove connections between devices.

  - Find out information about current transfers on the network. When connections are made, the connection manager service is the interface between the devices and the TCP/IP stack.

- ***AV Transport***. This controls the flow of audio and video including the functions of play, stop, pause and search.

- ***Delivering Control Service***. Many devices contain attributes that can be configured dynamically. They make differences in content delivery, such as brightness and contrast in video devices or volume, balance and the equalizer in audio devices. This allows the control point to discover the attributes that support a device and retrieve, change and restore the configuration of any of these attributes.

Figure 2.35 shows the typical sequence when reproducing multimedia content.

2.8.3 Application Examples

DLNA is currently implemented in the home in the usual way, especially with the appearance of lots of important manufacturers of devices that incorporate this technology. Among the most common devices using DLNA and incorporating it into our home are TV sets (with 400 certifications in the second quarter of 2009), games consoles, mobile phones (such as Nokia N95, which incorporated this standard), players and even cameras. There were 2,000 certified devices during the first half of 2009. Thus, it seems that this standard is becoming more and more relevant for the exchange of information and interactivity between terminals.

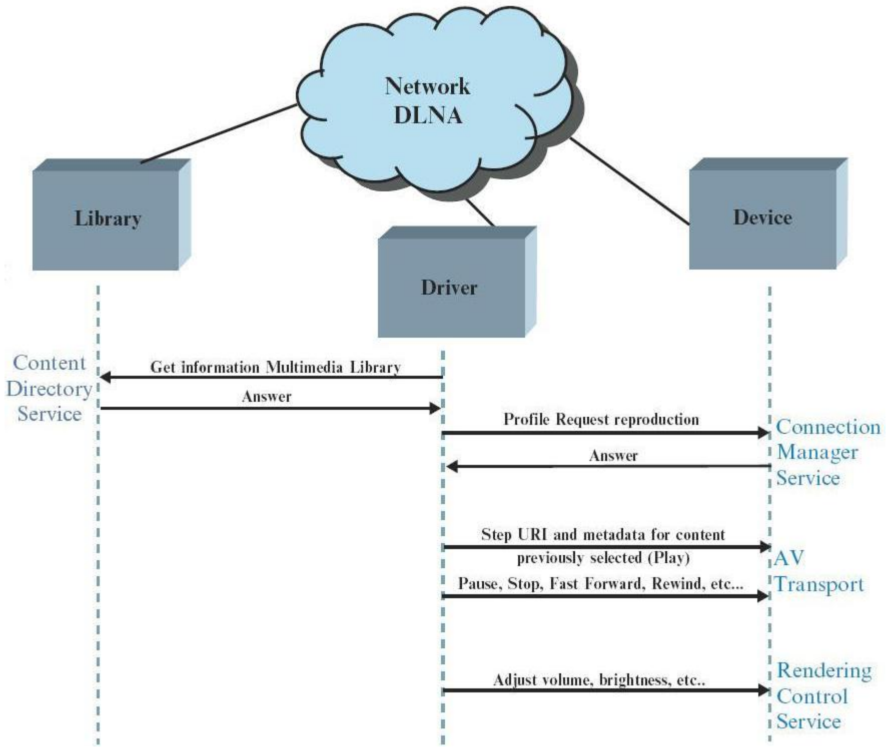

Network
DLNA
Library
Driver
Device
Content
Directory
Service
Get information Multimedia Library
Answer
Profile Request reproduction
Answer
Connection
Manager
Service
Step URI and metadata for content
previously selected (Play)
Pause, Stop, Fast Forward, Rewind, etc...
AV
Transport
Adjust volume, brightness, etc..
Rendering
Control
Service

Figure 2.35: Sequence of actions

Attempts have been made to expand the DLNA domain further so a device will be able to connect to any network. For example, [57] implemented the DLNA proxy server to service any virtual network [58].

2.9 Web Services

WS (Web Services) is a technology that allows websites to use distributed applications and offers features such as access to the information and functionalities of any platform. At first, they were created to meet the need to standardize communication between different platforms and programming languages because earlier attempts such as CORBA had little success. In the case of CORBA, this was because there are certain limitations for more complex applications that require a security control or transaction management.

WS provide a standard means of interoperating between different software applications, running on a variety of platforms and frameworks. WS are functions or procedures that can be accessed via the web. Regardless of the programming language of the service and its platform, they enable the exchange of data and provide services between different applications.

Such a degree of interoperability is only possible using open protocols. WS are mainly used with HTTP because this is widely used and is rarely blocked by firewalls. WS are a set of protocols and standards used to exchange data between applications, and they are used on important websites for tasks such as e-commerce, web browsers and computer services by companies such as Google, eBay or Amazon. The W3C is responsible for managing the specifications. The main features of WS technology and its advantages and disadvantages are listed below:

- It is supported by any platform and any programming language.
- It is a W3C standard.
- It provides functionality to websites.
- It uses HTTP to transport data.
- It uses standard elements for each of its components (SOAP, UDDI, Web Services Definition Language (WSDL) and XML).

  - One of the main advantages of WS is that they allow applications to communicate efficiently, regardless of the platforms used, offering greater interoperability. WS use standards and text-based protocols, which allows a better understanding and easier access to the data exchanged. They also use HTTP to allow the information to pass through

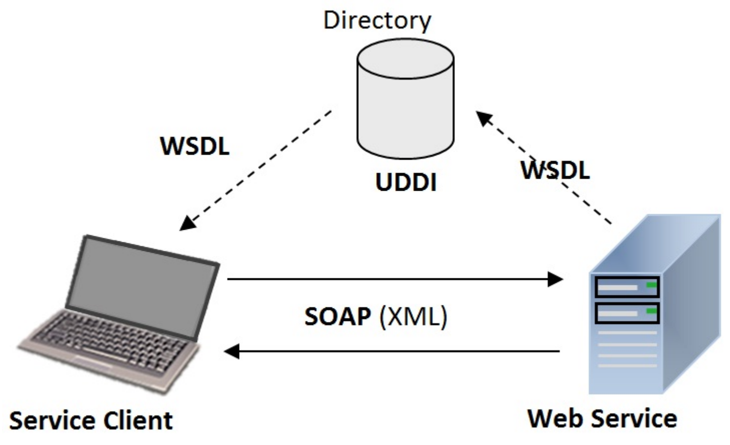


Figure 2.36: WS communication architecture

firewalls without major complications. This fact together with the use of XML promotes interoperability.

- However, WS are much less efficient than are CORBA or RMI because they make use of formats based on text, such as XML, which are not the best options to process tasks. Nevertheless, new WS standards may define more optimized protocols. Also they are not as developed as standards such as CORBA. Both HTTP and XML have a high run-time cost compared with other distributed applications approaches. Skipping the firewall security can also be seen as a drawback.

2.9.1 Components

WS use text-based standards and protocols, and this involves the components listed below. Figure 2.36 shows the diagram of the interactions between the entities and flows of the incoming and outgoing data of each component.

1. WSDL

   It is desirable that WS have information on the operations and data types involved. For this reason, WSDL is used. This is a standard adopted by the W3C that defines the public interface of WS. It is structured as follows:

   - Ports (<portType>): these describe the operations provided by WS. Its function is similar to an object-oriented class.

- Messages (<message>): these define the data involved in an operation, where each message can have one or more parts. It is considered one of the parameters used in object-oriented programming.
- Types (<types>): these define the data types involved in WS, using XML Schema, an XML language that accurately describes the structures and constraints of the XML file. It has been in the W3C since 2001.
- Links (<binding>): these describe the message formats and the protocols for each one of the ports.
- Operations (<operations>): these can be one-way, request-response (makes a request and waits for a response), request-response (receives a request and makes a response) or notice.
- Services: these define a set of web service ports.

2. UDDI

   In order to register and publish WSDL we use Universal Description, Discovery and Integration (UDDI). This is a standard developed for the publication and registration of WS. Its way of working is similar to a database and has two different parts:

   - Registration of business:
     - White Pages (Overview)
     - Yellow Pages (categories of services)
     - Green Pages (business rules)
   - Registration of services

3. SOAP

   In addition, there was a need to define the way of exchanging data between different processes on different machines. For this task, we use the SOAP, which defines the format of the messages to send. It is independent of the transport protocol. The elements of a SOAP message are [59]:

   - Encapsulation of the message.
   - Description of the data coding.
   - Body, which contains the specific message of the application.

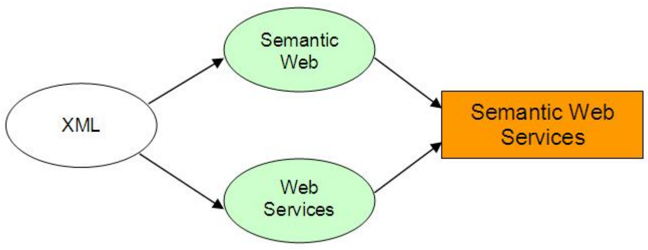


Figure 2.37: The emergence of SWS

2.9.2 Application Examples

Websites ask WS for a series of functions. They are currently used in almost all websites and they provide most logic to the website. Another possible application of WS is for the control of robots. WS are used to control robots from anywhere in the world via the Internet through a user interface, which will provide the services offered by the robot as well as its status [60].

2.10 Semantic Web Services

SWS (Semantic Web Services) were derived from the combination of WS with the emergence of the semantic web (Figure 2.37). Tim Berners-Lee created the semantic web states that the *"Semantic Web is not a separate web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation."* WS meet the requirement of a specified syntax; however, they have a lack of semantics so they cannot resolve ambiguities. This is solved by using SWS, optimizing this way the reuse of WS and creating smarter websites, resulting in the concept of Web 3.0. This simplifies the sharing and integration of web resources.

To represent knowledge, ontologies that structure information, resources or services based on the meaning of words emerge. This allows computers to interpret and process this information to work automatically.

The languages of high-level ontologies are backed by a formal logic, which makes sure that the ontology can be interpreted by the machines. This means that the computer and its software can interpret the semantics of the model without direct human intervention. The ontological software rises to the level of human conceptual knowledge; humans do not have to descend to the machine's levels [59].

SWS are an important line of the semantic web, which aim to describe not only information but also WS's functionality ontologies and procedures: its inputs, outputs, conditions for implementation, effects produced or steps followed. These machine-processable descriptions will automate the discovery, composition and implementation of services, as well as the communication among them. The semantic web has emerged to provide the syntactic web with semantic intelligence and has the following main features:

- Automatic data interpretation.
- Ontologies as data models.
- Discovery, selection and automatic service composition.
- Service implementation through the web.

2.10.1 Required Functionalities

- Publication of service descriptions.
- Services discovery.
- Service selection.
- Composition of services.
- Resolution of problems caused.
- Implementation of automated services.
- Monitoring of implementation.
- Compensation.
- Substitution of services for similar ones.
- Verification of implementation.

2.10.2 Main Technologies

- Web Ontology Language (OWL-S). This is an ontology based on OWL, which is a markup language for publishing and sharing data using ontologies. It was created by [61], which is part of the US Department of Defense, where they automate tasks such as the discovery, invocation and composition of WS.

- Web Service Modeling Ontology (WSMO). This is a conceptual model for the relevant aspects of SWS and it belongs to the European Semantic Systems Initiative. The WSMO working group includes the technology of Web Service. Modeling Language, which formalizes the WS that model the ontology [62]. Its main components are:

  - Goals. These are the customer's aims when they access the web service.

  - Ontologies. A formal description of the semantics used by all components.

  - Mediator. These are connectors that provide interoperability among different ontologies.

  - WS. These can include the functional and usage descriptions of WS.

  - OWL-S has a weak point in the architecture because it is undefined. It also has little development in comparison with WSMO. Its difficulty is also higher and less intuitive than WSMO is. However, its groundings of use are well developed. However, WSMO is not mature in key areas of use. It has a robust and flexible architecture for the consumer in contrast to OWL-S. It has defined important aspects such as languages and mediation. There are also plans to automate the creation of WS based on WSMO to semi-automate this process, thereby saving money, time and resources; the same as in the IRS III project.

2.11 Military Standards

2.11.1 JAUS

The JAUS (Joint Architecture for Unmanned Systems) standard was developed for the US Defense Department [63] by the JAUS Work Group, which is composed of research groups from the government (US Army ARMDEC), industry (SSC San Diego, WINTEC Inc., iRobot) and academia (Virginia Tech, University of Florida). JAUS was defined as an open and scalable standard that would meet the needs related to the communication of unmanned systems regardless of the platform used. The development of JAUS has tried to meet the following six goals [64]:

1. Independence of the vehicle's platform.

2. Isolation of the mission.

3. Hardware independence.

4. Independence from the technology.

5. Independence from the operation.

6. Independence from the connection used.

The JAUS architecture is composed of three levels:

- Level 1 - Inter subsystem: The purpose of this level is to support interoperability between subsystems. It is responsible for specifying requirements between the subsystems (Robot to Robot, Robot to Controller, Controller to Controller).

- Level 2 - Inter nodal: The purpose of this level is to support the interoperability between nodes. To this end, it specifies requirements between the subsystems (interoperability between data loads or between the on-board control and data loads).

- Level 3 - Inter components: The purpose of this level is to provide a reusable software source. It specifies requirements for each component (component by component, such as sensors and motors).

In 2004, a process of transition from the JAUS Work Group to the Society of Automotive Engineers [65] started. This developed the standard through the AS-4 (Technical Committee on Unmanned Systems) [66]. The following norms migrated from JAUS to a framework based on the following services:

- JAUS Transport Standard, AS5669 [67]. This is in charge of defining the creation of packets with the destination and source addresses and TCP and IP headers and links.

- JAUS Core Service Set, AS5710 [68]. This is responsible for providing the means for the software entities in an unmanned system to communicate and coordinate among their activities.

- JAUS Mobility Service Set, AS6009 [69]. This is in charge of making the migration from the first drivers to the new development platform of the AS-4.

Today, the main application of JAUS is focused on the use of unmanned civilian and military vehicles.

Application of Military Unmanned Vehicles

A major center for development of military unmanned vehicles exists at the SPAWAR Systems Center (SSC) in San Diego (California). There, a JAUS work team focuses on the development of surveillance systems, such as MDARS (Mobile Detection Assessment Response System), which are used in autonomous vehicles for military bases with restricted access.

The US Defense Department uses MDARS to meet security and surveillance needs in hostile environments for humans. In this way, it provides an integrated solution, where unit patrol vehicles are controlled just by a single control operator. Moreover, SSC has developed a distributed processing system called Multiple Resource Host Architecture [70] which, along with MDARS, was tested by the JAUS work team in December 2003 to demonstrate the level of interoperability between control operator units (COUs) and unmanned systems [71]. In this experiment, COUs were equipped with a screen capable of displaying the statuses of each patrol vehicle, and thereby they controlled each one of the unmanned systems [72].

These experiments show how the JAUS architecture provides interoperability for the remote control of unmanned systems while fulfilling the objectives mentioned in the general characteristics section.

Application of Civil Unmanned Vehicles

In 2004, Virginia Tech launched a project to implement simultaneously the JAUS standard in the following seven unmanned vehicles:

1. MATILDA

   This was the first interoperable vehicle designed by Virginia Tech in 2002. It was designed as an evaluation, development and demonstration platform of the JAUS standard. It had to fulfill some functional requirements:

   - It had to be teleoperable through a COU.
   - It had to be capable of driving autonomously via GPS commanded by a COU.
   - It had to interact with other subsystems of JAUS (either vehicle or COU).
   - It had to accept JAUS workloads from other devices.
   - It had to allow an easy modification and/or addition of intelligent software.

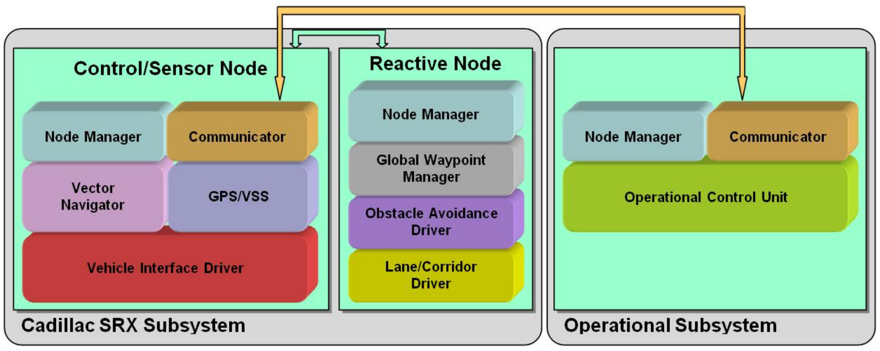


Figure 2.38: JAUS topology

- It had to ease the demonstration, evaluation and testing of the JAUS standard.

2. JOHNNY-5

   This was developed in 2004 to participate in the AUVSI Intelligent Ground Vehicle Competition in 2005. Owing to its robustness and its capability to navigate via GPS, it quickly replaced MATILDA. The main problems of this model were the failures in the camera interface and the starting force on the wheels.

3. CADILLAC SRX

   Grant Gothing and Jesse Hurdus, researchers from Virginia Tech, managed to implement the JAUS standard on the Cadillac SRX, creating the first luxury unmanned vehicle in the world [73]. The challenge of this model depended on development of a JAUS-based vehicle able to use potential field methods [74] for navigation. The result was the creation of a software topology, based on operational subsystems, nodes and components (Figure 2.38).

   However, when they launched this vehicle in the Blind Driver competition [75] they detected some issues that could be improved [76]. For example, every driver had to know the turn angle of the vehicle and, according to the control messages of the JAUS specification, only one controller per component was allowed.

4. GEMINI

   Gemini was developed as an extension of Johnny-5. The idea was to create an articulated robot with four wheels. It won the JAUS Award at the AUVSI Intelligent Ground Vehicle Competition in 2006 because of its refined design, the long life of its batteries (5 h), its innovative mobility and the ability to deal with bigger workloads under the JAUS architecture.

5. HELIUM RED (Unmanned Ground Vehicle; UGV) and THE RMAX (UAV)

   HeLiUm RED (HElicopter LIfted UnManned Reconnaissance and Exploration Drone) redefines the traditional notion of collaboration between UAVs and UGVs (RMAX-HELIUM THE RED). This small unmanned vehicle is light enough to be carried by the VT Yamaha RMAX UAV. Initially, the JAUS standard was implemented to simplify communication with the vehicle; however, vehicles are usually treated as subsystems of the JAUS architecture, but in the project HELIUM RED, the UGV operates as a single node.

6. ROCKY

   This is another example of the vehicles used by Virginia Tech in the DARPA Grand Challenge. The JAUS implementation in Rocky has taken place in two stages:

   - Teleoperability: Through the primitive driver, they could make sure that the vehicle was teleoperated making use of the COU, but nowadays with the use of Global Position/Speed Sensors, the COU, speed and position can be kept on track and transmitted through a connection service.
   - Portability of the basic code from Cadillac SRX directly to Rocky. This feature can be seen as a demonstration of the reusability existing when developing autonomous vehicles under the JAUS architecture.

Owing to these achievements, Virginia Tech established, as functional requirements, that their prototypes had to be interoperable with other JAUS subsystems (applied to both COUs and vehicles). Throughout this research, they realized the need to integrate some specifications in the JAUS Service Specification standard that would make use of messages in charge of waiting for a response that will allow the COU and the vehicles to make behavior decisions for a better interaction between them.

With respect to the development of unmanned vehicles, the company TORC started the ByWire XGV Project [77]. This project is being developed over a Ford Escape Hybrid using the JAUS standard as a platform to interact with the different parts of the car (steering, throttle, brakes and gear system). The vehicle has an Ethernet interface installed in a central console that allows for remote control of the vehicle by a COU, making use of the SAE AS-4 (JAUS) architecture. The use of the JAUS standard makes sure that ByWire XGV is compatible with any other platform developed on JAUS. It is important to note that the ByWire XGV has maintained speeds of 160 km/h. The DARPA Urban Challenge [61] checks the utility of unmanned vehicles in traffic environments and assesses how they stick to conventional rules of the road. This is a challenge for participants to ensure that unmanned vehicles can perform complex movements such as parking or taking navigational decisions at intersections. In 2005, the DARPA Grand Challenge competition, the University of Florida and Virginia Tech competed with their unmanned vehicle projects based on JAUS.

Applied Research Inc., Virginia Tech, University of Florida, iRobot and the US Air Force Research Lab showed the importance of interoperability in robotics in an experiment [78][79]. To this end, each consortium member made their COU able to interact with all robots and control all loads. The benefits of the JAUS standard were successfully proven after showing the independence of the technology used in unmanned vehicles and robots.

Sean Baity [80], talking about the future of JAUS, mentions the need to focus on development of software. This author says that it is a primary point to take into account to minimize problems in the progress of UGVs.

2.11.2 Other Military Standards

4D/RCS (Real-Time Control Systems)

The 4D/RCS architecture provides a reference model for military unmanned vehicles. 4D/RCS is a method of designing, integrating and testing intelligent systems software for vehicles that have a certain degree of autonomy [81]. It is an autonomous intelligent control system architecture for vehicles that can be either teleoperated or fully autonomous.

4D/RCS [48] specifies the way in which software components are distributed and interconnected, and that is the reason why it became a model for military unmanned vehicles. The importance of this standard lies in the way in which unmanned vehicles must manage situations in hostile environments to complete their missions. As a result of the above features, the

4D/RCS fulfills perfectly the specific needs of the Department of Defense and US Army standards [82].

4D/RCS architecture was based on the assumption that different knowledge representation techniques may offer greater advantages. The aim was to cover all of them to create a real-time control system for objects that move in the real world [83].

The Demo III UGB Program [84] developed and demonstrated advances in control of unmanned systems, especially small UGVs under supervised control. That is where the 4D/RCS architecture and its characteristics arose. This protocol allows intelligent vehicles to adapt to a changing world, to extract deeper information from a dynamic world and to merge such information with previously available information to improve a vehicle's performance.

The intelligent control of a 4D/RCS system is based on three layers of abstraction:

- A conceptual framework. This is the highest layer of abstraction and covers the full range of operations that involve intelligent vehicles, from a simple actuator for some milliseconds to lots of vehicles during long periods of time.
- A reference model architecture. This defines a hierarchical control structure and at each level functional processes are included.
- Engineering guidelines. These are the lowest layer of abstraction in intelligent control. They define how to design intelligent vehicles to work in groups with other intelligent vehicles.

NATO STANAG 4586

In 1998, a NATO expert team, composed of members of government and industry [85], started working on the development of the standard STANAG 4586 (Compliant Ground Control System for UAV) [86], which was ratified by NATO in 2002 for the communication and interoperability of its UAV.

The search for interoperability between unmanned systems is essential when meeting objectives in military terms. The line of development should be focused on interoperability between land systems, aerial systems and elements of control, command, communication, computer and intelligence (C4I) [87].

STANAG 4586 was developed as an interface control definition capable of defining a common number of data packets for two new interfaces [85]:

- A data link interface among ground control stations and aerial vehicles.

- A command and control interface among ground control stations and C4I systems.

According to [88], STANAG 4586 is the only standard that promotes interoperability in control networks of UAVs. There are five interoperability levels defined in this standard [86]:

- Level 1: Reception/transmission of data packets related to UAV.
- Level 2: Received live data about intelligence, surveillance and reconnaissance.
- Level 3: Control and monitoring of data packets of UAVs in addition to the reception of intelligence, surveillance and reconnaissance and other data.
- Level 4: Control and monitoring of UAV, except from launch and recovery.
- Level 5: Control and monitoring of UAV including launch and recovery.

STANAG 4586 supports Electro-Optical/Infrared, Synthetic Aperture Radar, communication transmission and data link interface resources.

2.12 Other Technologies

2.12.1 Salutation

This platform is independent of the architecture, language and operating system on which it is installed. It is based on the operation of the translation manager, specific for the Runtime Environment, and the salutation manager, which provides an API for publishing and search services [89]. For example, [90] demonstrated the integration of this protocol with Bluetooth Service Discovery.

2.12.2 Service Location Protocol

This was created for client/server applications and it defines three kinds of agents: user, service and directory [91]. For more information, many of the protocols discussed in this chapter are compared and classified in [92] and [93].

2.12.3 Ad hoc Developments

Before the appearances of concepts related to the automatic installation of devices in distributed networks, algorithms were developed for specific types of robots. This is the case for the Multi-Robot System of UNIX, which uses TCP/IP connections in a client/server architecture.

Standards have also been created for a particular type of technology such as the service discovery protocol [94]. This protocol can discover information on existing services in other Bluetooth devices.

2.12.4 URBI

URBI (Universal Robot Body Interface) [95] is an open source software platform written in C++ for robotics, complex and parallel systems. It is based on the UObject component architecture and on a parallel and event-driven script (interpreted) language. It is an orchestration script language, known as urbiscript, to glue the components together [96]. Urbiscript is a programming language primarily designed for robotics. It is a dynamic, prototype-based, object-oriented scripting language that supports and emphasizes parallel and event-based programming by providing core primitives and language constructs. The urbiscript language syntax is very close to C++ syntax and is fully integrated with C++. It allows both a low level operation with motors and sensors and a high level with complex commands. URBI is not only available for robotics system, but also for non-robotic purposes. The objective of Urbi is to help make robots compatible and simplify the process of developing software and behaviors for those robots.

URBI supports Windows, Linux and Mac OS X [97]. This development platform for robotics takes into account interesting features to provide parallelism, task and event handling, time handling, code tagging, UObject architecture (plugins) and Client/Server architecture.

An example [98] of the parallelism and the time handling may be ilustrated tak-ing into account the following line executed on an Aibo robot [99]:

```
headPan.val = -90 time:2s & tailPan.val = 45 time:2s;
```

The above line allows to rotate Aibo's head till the ninety degrees (headPan.val = -90) for two seconds (time:2s), and at the same time (&) it moves its tail until it ar-rives to forty-five degrees (tailPan.val = 45) during two seconds (time:2s).

2.12.5 DH Compliant

DH Compliant (Digital Home Compliant) [100] is a system providing interoperability between all devices existing in a home network. It is based on the UPnP architecture and is currently under development by the University of Oviedo, the University of Seville and a consortium of companies composed of Ingenium [101], Domotica Davinci [102], MoviRobotics [103], Applied Research Associates [104] and the Cartif Foundation [105]. The main goal of DH Compliant architecture (Figure 2.39) is to integrate consumer electronics devices, robots, sensors and other interesting components that may be useful in a home automation framework. The aim of the DH Compliant system is development and implementation that allows the integration of service robots within the digital home. This architecture will provide interactions between robots and digital homes to make life easier, more secure and more comfortable. This protocol integrates the intelligence of a UPnP control point and the functionality of a UPnP device in a single DHC device. This entity network is managed by other entities that provide new services such as the localization service, energy-saving service and the service for collaborative tasks between robots. This interoperability system will be deeper detailed in chapter 4.

2.12.6 ROS

ROS (Robot Operating System) [107] is an open source robotic operating system for controlling robotic components from a PC. ROS was originally released in 2007 by the Stanford Artificial Intelligence Laboratory, but from 2008 until 2013, development was performed primarily at Willow Garage. It provides libraries and tools to help software developers create robot applications. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model.

The main module in ROS system is the Master node. It allows all other software nodes to discover and communicate with each other by publishing and subscribing to events as it is illustrated in 2.40.

A simple example is a sensor implemented as a node. This node publishes the data from the sensor by sending messages to the subscribed nodes. This basic coordination may be extended to a complete and more complex environment. ROS allows to group several specific capabilities from different nodes in order to reach a common goal. For example, a group of robots (ROS nodes) create maps of the environment, which are used to navigate by another group of robots (ROS nodes) that capture images of the environ-

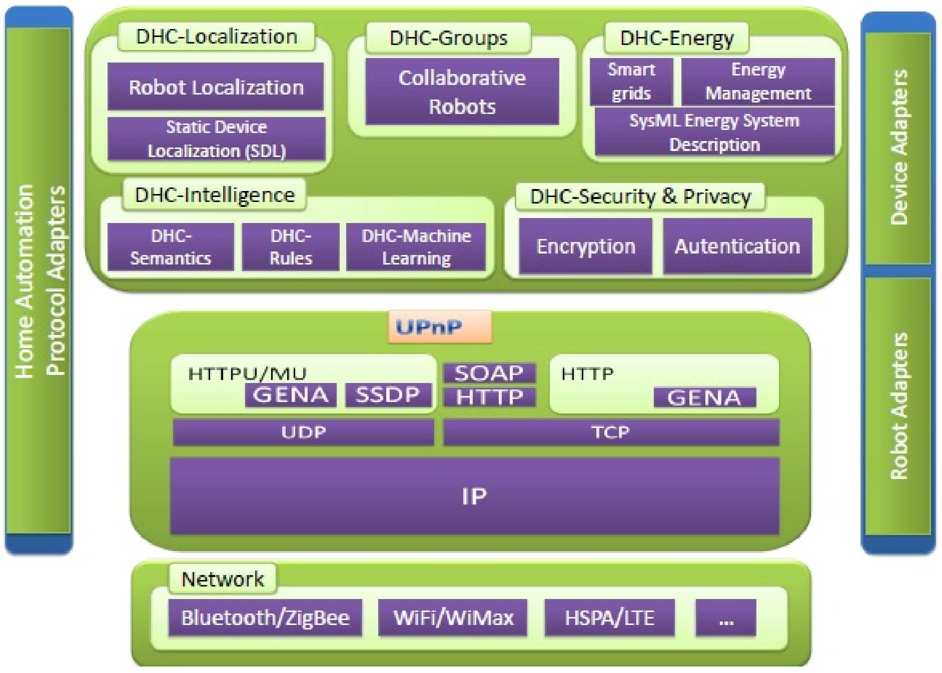


Figure 2.39: DH Compliant architecture [106]

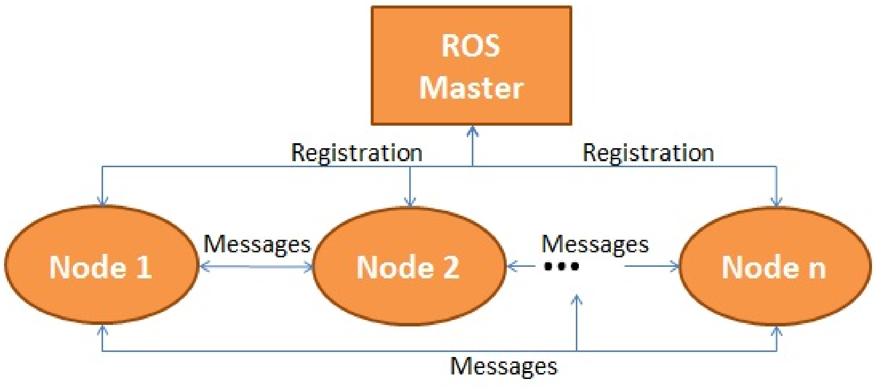


Figure 2.40: ROS coordination

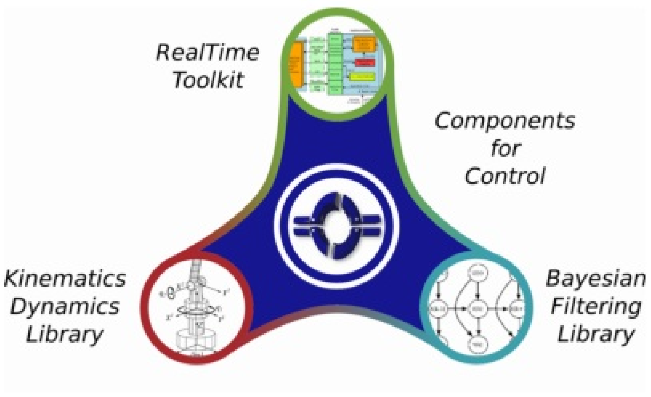


Figure 2.41: OROCOS libraries [109]

ment; finally, these images are processed by a computer vision node to find particular objects in the environment.

2.12.7 OROCOS

Orocos (Open Robot Control Software) is an Open Source C++ software framework for building real-time component-based applications in automation and robotics [108]. It is a general-purpose and modular framework for robot and machine control. The Orocos project supports 4 C++ libraries (Figure 2.41):

- **Orocos Real-Time Toolkit (RTT)** provides the infrastructure and the functionalities to build robotics applications in C++.
- **Orocos Components Library (OCL)** provides some ready to use control components (Component management and Components for control and hardware access).
- **Orocos Kinematics and Dynamics Library (KDL)** is a C++ library which allows to calculate kinematic chains in real-time.
- **Orocos Bayesian Filtering Library (BFL)** provides an application independent framework for algorithms implementation such as Kalman Filters or Particle Filters.

2.12.8 OpenJAUS

OpenJAUS [110] is a free Open Source middleware implementation for Unmanned Systems based on the Joint Architecture for Unmanned Systems

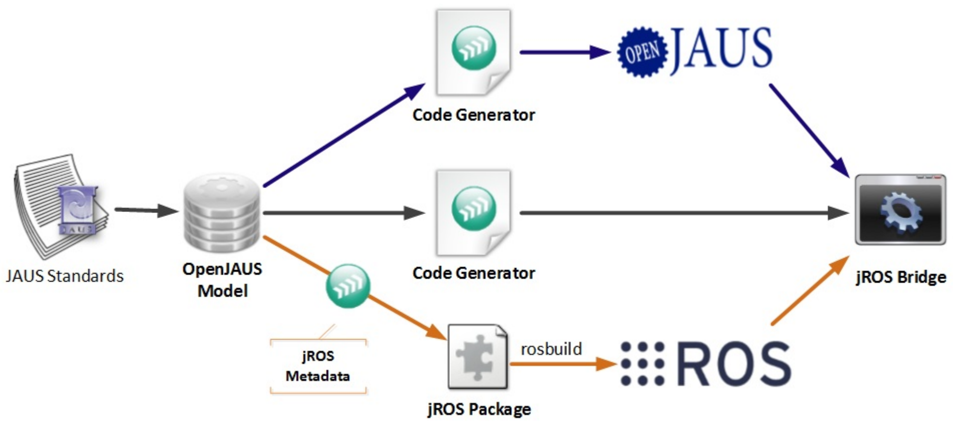


Figure 2.42: jROS design [110]

(JAUS). OpenJAUS emerged as the libraries and applications developed at the University of Florida in the 2005 DARPA Grand Challenge.

OpenJAUS has created jROS, a software library to integrate JAUS and ROS solutions, which is required to use a JAUS interface for a ROS-based system or vice versa. jROS is distributed as a set of ROS Packages which consist of custom ROS Messages for each of the JAUS Service Sets (i.e. Core, Mobility, Manipulators, etc). Figure 2.42 shows how OpenJAUS uses code generation to automatically create the jROS bridge.

Chapter 3

Integration of Service Robots in the Smart Home by means of the Universal Plug and Play Protocol

The digital home concept has recently emerged. It is characterized by a network of sensors and actuators which provides services in terms of comfort, security and energy management. Moreover, the future points to the paradigm of ambient intelligence whereby home should automatically adapt to the needs of its inhabitants.

The emergence of home automation systems was launched in the late 1970s with the development of a system based on X-10 Power Line Carriers. From that moment up to the present, the number of home automation systems has increased significantly: KNX, Lonworks, Zwave, Delta Dore, Control4, Cardio, ... In this sense, it is remarkable the situation in Spain, where the variety of home automation systems is even larger because of its real estate bubble in the beginning of this century. Because of the large investments made in this sector new systems emerged, such as IPDomo [111] or Ingenium [101], which were able to survive despite the subsequent real estate market drop.

Not only the large number of home automation systems should be considered, but also the wide variety of consumer electronic devices that can be integrated within the digital home network. Different interoperability standards have been proposed in [6], [112] and [113] in order to solve the problems resulting from the integration of such heterogeneous systems. In this sense the Universal Plug and Play (UPnP) [114] is the standard that has achieved the greatest success, although with some nuances.

Nowadays, multiple heterogeneous electronic devices coexist at homes. However, most of these systems are independent and only ad hoc integrations between them are possible. Given that smart home applications are based on the integration in a same network of a set of sensors and actuators, it is obvious that this lack of interoperability hinders the development of more advanced services for the home inhabitants and it is a major issue in the road towards the ambient intelligence paradigm [115]. This problem is not new, though there have been several attempts to standardize the way in which heterogeneous systems can be integrated [116]. The first important attempt to solve this problem in a systematic way was CORBA (Common Object Request Broker Architecture) in 1991. CORBA is a distributed object oriented architecture. It allows to integrate distributed objects regardless of the hardware architecture, operating system and programming language [1]. In 1998 Sun proposed a different standard, Jini, with the goal of setting the basis for the definition, publication and searching of services in a network [20]. A similar alternative was given by UPnP, which was proposed by Microsoft [41] in 1998. UPnP has been very successful in the field of multimedia interoperability. For this reason, there are other interoperability standards based on UPnP. This is the case for example of DLNA (Digital Living Networks Alliance), whose main focus is multimedia interoperability [57]. Another remarkable standard is OSGi (Open Services Gateway Initiative), which is focused on domestic applications. OSGi defines its own architecture, but it is designed to work together with other protocols such as UPnP or Jini [117]. Finally, some authors [118] bet on web services as the solution for the interoperability problem.

Despite the absence of interoperability, the relevance of technology in our lives has grown significantly during the last decades. In particular, the evolution of the role played by robotics is remarkable. Despite the fact that the use of robots has traditionally been associated to military and industrial applications, it is common to find them now even in domestic environments.

The purpose of UPnP is the integration of new devices within the home network transparently to the user as many devices that operate on a computer. To that end, the devices in a network exchange information about the services and their capabilities that may provide to the network. In addition, these devices must expose their services to be invoked remotely. The most successful application field of UPnP is the interoperability between multimedia devices. This fact is confirmed in different studies such as [119] or [120]. Nowadays, the UPnP features related to audio and video streaming are widely used by many devices. However, UPnP is not so employed in other different fields. Nevertheless, the success of UPnP is greater than other interoperability standards that have aslo been proposed, such as OSGi and Jini [116].

This chapter presents a methodology for creating UPnP virtual devices. This methodology can be used to ensure real systems interoperability by means of UPnP gateways working on a computer. In other words, it allows the creation of hybrid UPnP devices that are capable of translating UPnP orders into actions performed by real systems. In this chapter we explain how to design UPnP devices via software to integrate robots in the smart home. The problem of integrating service robots in the smart home is also addressed in this chapter, and the integration of two service robots by creating UPnP gateways is detailed.

The stages for the creation of UPnP devices have been detailed in the Paraninfo's book "Home Automation for Engineers" (Domotica para ingenieros) [121] written in spanish language. This chapter contains information extracted from the paper "Robots in the smart home: a project towards interoperability" [44] published in the international journal IJAHUC in which the Roomba robot and its integration in the smart home is deeply studied. In this chapter, the Rovio robot and its inclusion in the digital home through UPnP is also analyzed. This research has been published in the international journal RAS with the title "Integration of service robots in the smart home by means of UPnP: A surveillance robot case study" [122]. We have developed UPnP applications for each robot and different experiments have been performed to illustrate the possibilities derived from the integration of service robots with home automation technologies.

The outline of the chapter is as follows: In section 3.1 Roomba vacuum cleaner and Rovio surveillance robots are presented. Section 3.2 details how to create virtual devices based on the UPnP architecture. The next section explains the UPnP application developed in order to integrate the robots in the digital home. Some experiments related with the robots were carried out in the next section. Finally, the conclusions are presented in section 3.5.

3.1 Smart Homes and Robots

Nowadays, the use of service and industrial robots is spreading worldwide. In general, robots reduce costs and improve the precision in the tasks they are assigned to do. Moreover, they substitute human beings in many complicated tasks. The take-off of robotics is motivated mainly by the reduction of production costs.

Service robots are still at a very early stage. Until now, the definition for service robot was not well defined. The International Federation of Robotics (IFR) [123] had defined the term service robot as

> "a robot which operates semi or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations."

However, it seems that the IFR has specified the definition:

> "A service robot is a robot that performs useful tasks for humans or equipment excluding industrial automation application."

Other entities as Hisparob [124] defined service robotics as:

> "...nowadays industrial robotics extension in which new robots are being developed to answer the industry needs not directly linked with the productive tasks..."

The most basic way to interpret these definitions is that any robot that is not used in manufacturing or production operations is a service robot. For the purposes of this chapter, this simple definition is enough.

Service robots can be classified into different types. We present a classification inspired on the one provided by the IFR:

- **Professional service robots**
  This group includes robots (Figure 3.1) capable of collaborating in tasks performed by humans. These robots are designed to enhance the capacities of people in their jobs. Examples of this type of service robots are: Communication Robot, developed by AEON MALL [125] in collaboration with Tmsuk Co. [126]; An-9RR created by the Japanese company ALSOK [127]; da Vinci Surgical System designed by the company INTUITIVE SURGICAL [128], among others.

Figure 3.1: Professional service robots

- **Personal service robots**
  Personal service robots (Figure 3.2)have become a reality in many houses. They can solve daily tasks at home. They improve the quality of life of their users. The Roomba is included in this category. Besides the Roomba, the following can be included: Asimo, presented by Honda [129]; Aibo, developed by Sony [99]; Scitos G5 created by MetraLabs Robotics [130]; Home Assistant Robot is a joint research between Toyota [131] and the University of Tokyo, among many others.

- **Defence robots**
  These robots (Figure 3.3) are prepared for dangerous situations such as deactivation of explosives or intervention in contaminated areas. Within this group, we can cite the following robots: Pointman, designed by Applied Research Associates (ARA) [104] or mSecurit, developed by MoviRobotics [103].

Service robots will bring a robotic revolution to homes, streets, schools, hospitals, etc. One company, which has taken some of the steps towards penetrating service robots into the home, is iRobot Corporation [132], the creators of the service robot Roomba. Roomba is an autonomous robotic vacuum cleaner that has sold more than 10 millions of units worldwide.

Figure 3.2: Personal service robots

Figure 3.3: Defense robots

In this chapter, we are interested in how service robots can be integrated in the digital home through middleware. The problem is complex because of the large number of systems available in the home automation market. The absence of real interoperability between the systems makes the development of new services based on the combination of heterogeneous systems more cumbersome. Given that at the present time there is no ideal choice between all the technologies that can be installed in the home, the best choice to develop new advanced services comes from mixing different systems and technologies in the same installation [133][134].

Given that one of the most important objectives of smart homes is the development of new services that make the life of its inhabitants better, service robots should be easily integrable in a smart home environment. The variety of robotic and smart home information systems' companies represents the main problem for the interoperability approach. The absence of minimum requirements makes it difficult to assume the existence of systems or services in the home. The consequence is that difficulties arise for service developers. At this point, it is needed to emphasise that the interoperability among systems deviates far from the simple exchange of information. The intercommunication among the information systems that support the new services must satisfy a large number of needs. The use of mark-up standards, consult services and some web services provide the functionality and interoperability required but there are also some other important issues, such as the ability of interaction and parallel implementation tasks, to name a few.

UPnP has been successfully used for home automation, see for example [5], and it has been considered as a middleware platform to integrate robots [135]. In addition, UPnP is an important element of other home networking standards such as the Digital Living Network Alliance [136] (formerly the Digital Home Working Group) and Intel's Networked Media Products Requirements (NMPR) specifications. Despite its relative success, UPnP still has some drawbacks that should be improved to be the ideal middleware platform, specifically in the case of robot integration. Here, we show a list with some of the features that have to be improved. The list is based on works [137][116], and it has been complemented with other deficiencies that we have found in the standard during our research.

- Lack of priority mechanism for messages.
- Lack of a deterministic time of response for commands.
- Lack of a mechanism for synchronous request and responses.
- Unclear definition for elements in the network that assume functions of

both control point and device.

- Lack of complex data types such as arrays. This problem can be avoided indirectly by defining an extension of the standard in the XML schemas, something that is mentioned in the specification. However, it is desirable to count also with a standardised version of complex data types.
- Possibility to select a network when there are several available.
- Lack of security mechanisms.
- Absence of directory and proxy services in the network.
- The query mechanism is very basic.

During our experiments, we especially found the first two deficiencies to be critical. They are particularly important for tasks that involve the coordination of two or more robots and they could be a potential source of problems such as collisions. The lack of a directory or proxy services is also troublesome in case that the UPnP devices are connected in a wireless fashion. Without a directory, it may happen that a control point cannot detect one of the devices. In addition, a proxy could save petitions for the device until connectivity is restored.

In particular, in this chapter we focus specifically on UPnP, which seems to have had a larger impact than the others in the market of consumer electronics, especially in multimedia applications. Other works have explored the use of UPnP for the smart home and the integration of robots. For example, different home automation systems and consumer electronic devices are integrated via UPnP in [5]. In [135] and [137], requisites for the use of UPnP as middleware to control robots are studied. In this chapter, we do not only address these issues but also develop software bridges to integrate the Roomba and Rovio robots in a UPnP environment, and test them in real homes equipped with the system IPDomo [111], which is natively UPnP. In addition, we have extended the services offered by the robots by developing new capabilities such as a basic garbage detection routine for Rovio or an application which enables Rovio to track the autonomous vacuum cleaner Roomba. In this manner, it is possible to illustrate the benefits of interoperability in the smart home.

3.1.1 Roomba Robot

The first ancestor of Roomba was the Rug Warrior, a robot developed in 1989 in the MIT Artificial Intelligence Laboratory. Ten years later, its creator, J.L. Jones, together with Paul Sandin, proposed to iRobot to investigate the development of a floor-cleaning robot focused on the home market. As a result of their investigation, the first prototype of the Roomba was born with the name of Scamp [138]. Three years later, the first commercial version of Roomba was released. Nowadays, more than 2 million Roomba units have been sold and its impact on daily life has been strong; even emotional relationships have emerged between Roomba and its owners, see [139], [140] or [141].

Roomba's commercial spawned many applications that can be found on the internet and in the literature. For example, in [142], the Roomba is evaluated and presented as a valuable tool for robotics teaching and research. There are even books [143] that deeply analyse the Roomba and explain how to take advantage of all its capabilities. Another good example that shows a very interesting application of this vacuum cleaner is [144] in which a set of Roomba are used to implement a real version of the popular PacMan game.

For the above-mentioned reasons, Roomba became the perfect candidate to be the first service robot that we integrated in the smart home of research. In particular, the Roomba 560 model has been chosen for the application. Similar robotic vacuum cleaning technologies, like the ones provided by Infinuvo [145] or P3 international [146], might benefit as well from the integration method used in this chapter. In the rest of the section, we describe Roomba's hardware and software.

Roomba is a vacuuming disc-shaped robot as the one presented in Figure 3.4. According to [138], the microprocessor runs at 16 MHz, has 256 B of RAM and has 30 IO. It is directed by a behaviour-based programming scheme composed by many strategies to avoid stuck situations and its cleaning strategy is mainly based on bounce and wall following algorithms. During its task, Roomba switches between the following cleaning patterns: Spiralling, Wall Following, Room Crossing and Dirt Detection.

Roomba has several sensors and actuators available to it, which enable navigation and allow it to realise its work.

- *Drive motors*: Roomba has two electric drive motors, which allow differential steering and speed control.
- *Cleaning motors*: Roomba uses three motors to control the vacuuming functions of the Roomba, these motors move the main brush, side brush and vacuum. Motors speed cannot be controlled.
- *Speaker and LEDs* for user feedback.
- *Wheel encoders*: These measure the number of rotations of each wheel.

Figure 3.4: Roomba robot

- *Optical interrupters*: They provide a mechanism to detect a bump of the Roomba. When bumper moves, it triggers one or both of these sensors.
- *IR wall sensor*: This infrared distance sensor enables Roomba to follow walls. It allows Roomba to decrease speed automatically and to get close to the walls.
- *IR cliff sensors*: The cliff sensors are four infrared sensors. They face down and detect when Roomba has started to go over a cliff.
- *IR receiver*: This is a 360 infrared receiver for signals emitted from a virtual wall, used to confine Roomba in a region, or the home base.
- *Microswitches*: The three wheels on Roomba are equipped microswitches that detect when the wheel has dropped (wheel drop sensor).
- *Capacitive sensor*: It allows Roomba to detect dirt.
- *Battery level and motor current sensors.*

To ease the external control of the robot, iRobot published the Roomba Serial Command Interface (SCI) [147], which is nowadays known as the Roomba Open Interface (ROI). Through this interface, it is possible to command Roomba and also to extract data from its sensors. The ROI allows full control of the Roomba with simple commands. The control itself is performed through a Mini DIN 7-pin jack located on top of Roomba and covered by a circular piece of plastic. Through this port, it is possible to establish serial communication at 115200 bps, with 8 bits per character and no parity and one stop bit. Roomba uses a command-response model and it never sends data unless requested. This last feature simplifies the structure of the communication but it is inconvenient because periodic state information of the Roomba is needed, specifically to know the value of those sensors more important for the navigation. Finally, the ROI protocol allows users to set Roomba in one of the five internal modes:

- *Sleep* (off)
- *On*: Roomba can be operated only through its buttons.
- *Passive*: The sensors can be read, but no control can be accomplished.
- *Safe*: This mode allows the external control of the Roomba. In this mode, the Roomba will automatically stop for security reasons if it detects a cliff, a wheel drop alarm or a plug-in power supply.
- *Full*: Users take total control through the ROI without any security exceptions.

3.1.2 Rovio Robot

Rovio is a mobile robot equipped with a webcam and manufactured by WowWee [148]. It has an embedded web server where its control interface is stored. Through this interface Rovio can be controlled via WiFi by means of any web-enabled device. Its main applications are surveillance and telepresence. Due to its characteristics, it has been used in different projects in the literature. For example, in [149] a Rovio was equipped with a laser pointer and maps a region of space by using this pointer and its camera. In [150] Rovio was used as a platform to install a recognition agent, Cassie, which allows a verbal and interactive communication with the robot. In [151] Rovio is used to map a region of space using its camera. The goal of this work is to implement SLAM (*Simultaneous Localization And Mapping*) although it focuses finally on the identification of some concrete position marks. In [152] the Rovio API is implemented in the Python programming language. The result is *PyRovio*, which is used with the aim of providing robots with cognitive features using the MGLAIR (*Modal Grounded Layered Architecture for Integrated Reasoning*) architecture. In [153] Rovio is used to localize and identify a concrete object: a beer barrel. In this work movement and image analysis are implemented, using the OpenCV libraries [154]. The robot is intended to identify a small barrel in an autonomous manner in an unknown environment but with a recognizable movement direction (a corridor, typically).

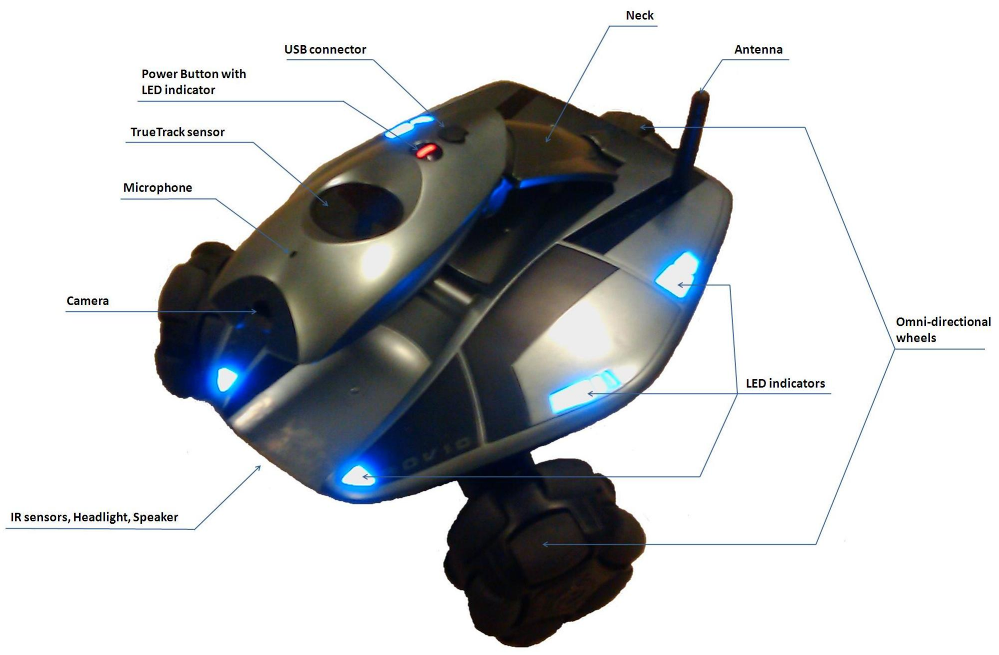


Figure 3.5: Rovio general scheme

A general scheme of the Rovio and its charging dock are shown in Figure 3.5 and figure 3.6.

Rovio is equipped with the following set of sensors:

- Head-mounted moveable VGA camera
- Microphone for 2-way audio
- Infrared sensor for positioning purposes
- Infrared sensor for collision detection
- Battery charge monitor

From all these sensors, the most remarkable one is the infrared sensor for positioning purposes. Rovio uses infrarred beacons projected on the ceiling as guiding lights. These lights are projected from its base (Figure 3.6), and help Rovio to find its dock back and simplify the navigation of the robot by the definition of waypoints by the user.

In addition, Rovio has the following set of actuators:

- Position controller for the robot neck

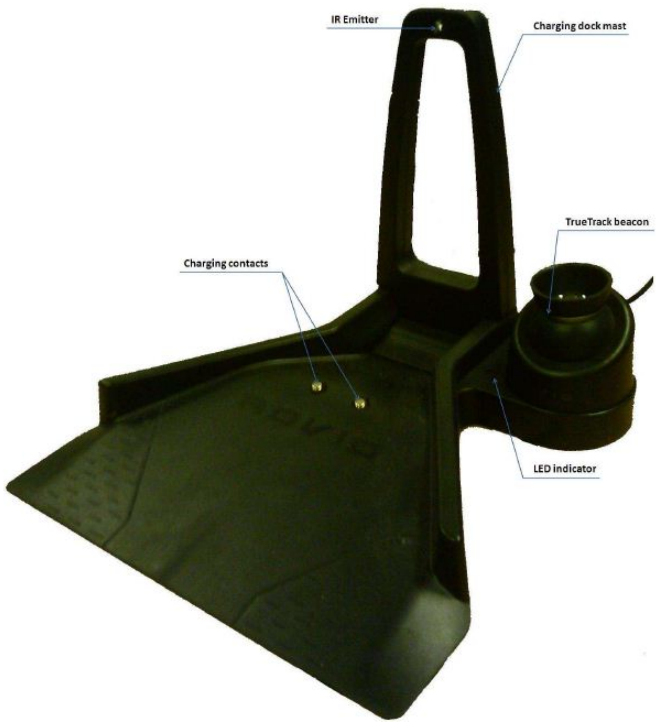


Figure 3.6: Charging dock

- Speaker for 2-way audio
- 3 omni-directional wheels
- Headlight (LED illumination)

The Rovio API [155] is based on the HTTP protocol. In particular, Rovio works by means of CGI commands within HTTP GET requests. CGI stands for *Common Gateway Interface* and is a *World Wide Web* technology which allows a web-based client to send and require some data from an application which is executed in a web server. In this manner, Rovio's web server passes user's request to the CGI application program which controls the robot. Note that this implies that the only requirement to create a Rovio control application is to create a TCP/IP binding with its web server in order to send the HTTP requests and receive the corresponding responses. Therefore the communication with Rovio does not depend on any particular platform, programming language or operating system.

There are nine categories of CGI commands which can be sent to the Rovio robot: movement control, camera control, user management, time, network, MAC address, HTTP server, mail and other commands. For example, the command *HTTP://192.168.1.2/rev.cgi?Cmd=nav&action=1* corresponds to a HTTP request to the IP address 192.168.1.2 whose argument is a CGI command to order Rovio to go to its home location in front of its charging dock. More details about other commands of the Rovio API can be found in [155].

3.2 UPnP Virtual Devices Development

In chapter 2 we explained the operation of UPnP. The steps to follow in order to develop UPnP virtual devices are detailed below. Note the emphasis on the virtual nature of the device, i.e. they are not physical devices. Therefore, it is possible to define a virtual UPnP device as that which is only implemented at a software level and is executed on a general purpose computer as a PC.

We have classified UPnP devices in three groups: physical, pure virtual and mixed virtual. These types of devices differ in how they are implemented.

- **Physical**: These devices are a physical implementation, i.e., the device is a printed circuit board (PCB) or a real system that provides a UPnP interface and must be fed properly. Specifically, a physical UPnP device is the hardware and software that provide the UPnP functionality. Within this group we can include the IPDomo cards for home automation.

- **Virtual**: This group includes devices implemented at a software level. An example of a virtual device may be a weather station.
- **Mixed**: They are characteristic virtual devices that translate orders from the UPnP network to another physical system, so they are at an intermediate point between the two previous groups. They are gateways that provide a UPnP interface and another interface to the corresponding system. On the one hand, it collects data from the UPnP network, interprets and translates the information into commands known by the system at the other side which actually perform the action; on the other hand, it collects data from the agent and provides them to the UPnP network. In this group we can include the UPnP gateways for the Roomba or Rovio robots.

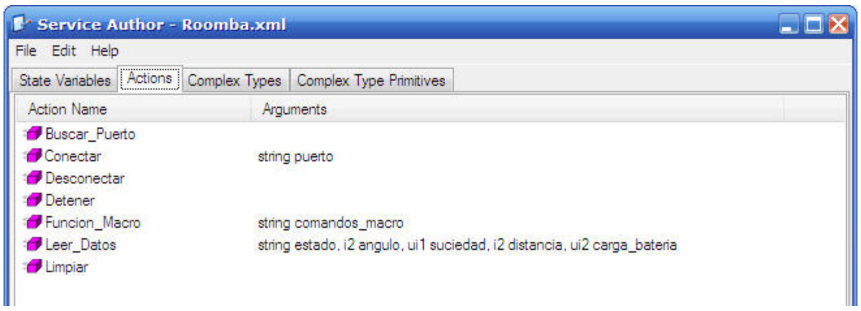
As it can be deduced from these devices, both the virtual and the mixed devices require a computer program to provide support for the connection in a UPnP network. The development of virtual devices is also useful as the basis for adapting non-UPnP devices within a UPnP network. To that end, it is necessary to program a virtual device that works as a gateway between the equipment you want to provide UPnP functionality and the UPnP network.

In general, the development of virtual devices would be very complex if we had to write all the code associated with the UPnP protocol. This difficulty may be avoided using any of the existing development kits for UPnP such as Allegro [156], Cybergarage [157], GUPnP [158] or the Atinav [159]. In particular, this chapter explains how to use the Intel SDK for UPnP [160]. This development kit is a suite of programming tools for UPnP which can be downloaded for free on the Net. The kit generates a code template that implements the basic features of UPnP, so developers may focus on developing the code associated with the functionality of the device. The template may be generated in different programming languages (C, C++, C#). From all the programming languages, we have chosen C# for the development of the Roomba and Rovio UPnP gateways.

The design of a UPnP device can be summarized in four main steps or stages which are described below.

3.2.1 Step 1: Creation of the Service Description

The first step is to create a description file for each of the services the device will provide. This file contains information related to the actions and the state variables that characterize the service (Figure 3.7). In this step we will use the Service Author program which is one of the applications included in the Intel tools. This program simplifies the definition of variables, actions



Action Name	Arguments
Buscar_Puerto	
Conectar	string puerto
Desconectar	
Detener	
Funcion_Macro	string comandos_macro
Leer_Datos	string estado, i2 angulo, ui1 suciedad, i2 distancia, ui2 carga_bateria
Limpiar	

Figure 3.7: Creation of the device service description

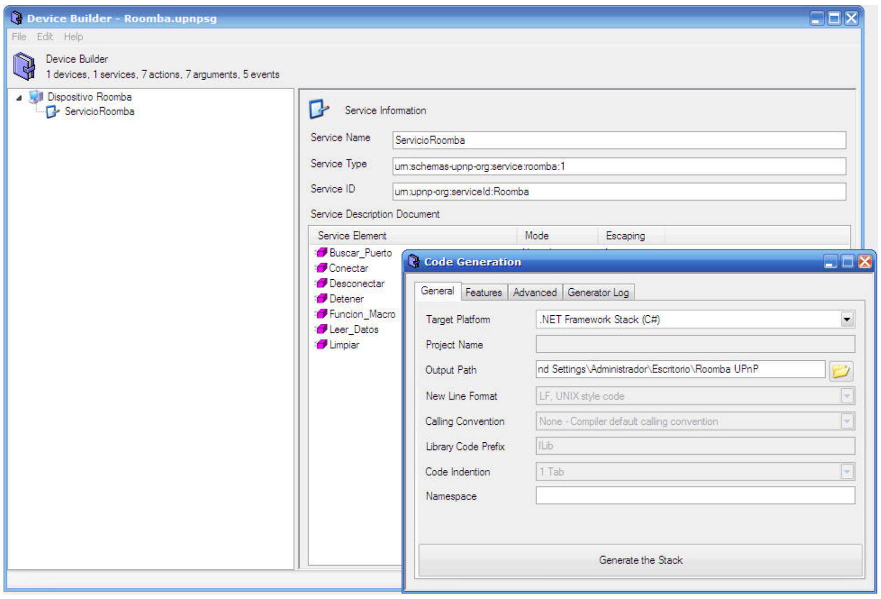
and arguments that form a given service. Firstly, it is necessary to define the state variables required. In particular, it is necessary to specify the format of each variable and if it is evented type or not. When the value of an evented type variable changes, it generates event notifications to the control points subscribed to this variable. This is the basis of subscription and event notification in UPnP.

In order to create the service description file it is also necessary to declare the actions. Each action may optionally receive input arguments or generate output results. To that end, some state variables previously defined must be employed. In particular, the type of each variable must be specified: input, output or return. Thus, if the state variables are declared as input variables, they will be arguments or values sent to the device when the user invokes the actions. On the contrary, if they are declared as output or return variables, they will be values returned by the device and stored in such variables during the execution of the action; or as the result of the action.

Once the actions and the state variables are declared, all this information is saved in an XML file. Therefore, a device will have as many service description XML files as services offered. Therefore, a device has so many XML service description files as services it provides. Note that the actions and the state variables must be created separately for each service.

3.2.2 Step 2: Creation of the UPnP Stack

The second step employs the service description files created in the previous step to generate the device UPnP stack (Figure 3.8). The stack contains all the UPnP protocols as it is illustrated in Figure 2.13, i.e., it contains all the code responsible for the registration and announcement of the device and its services, and manages the subscriptions and requests whith the devices in a UPnP network. The creation of the stack requires the use of the *Device*


Device Builder - Roomba.upnpsg

Device Builder
1 devices, 1 services, 7 actions, 7 arguments, 5 events

Dispositivo Roomba
ServicioRoomba

Service Information

Service Name: ServicioRoomba
Service Type: urn:schemas-upnp-org:service:roomba:1
Service ID: urn:upnp-org:serviceId:Roomba

Service Description Document

Service Element	Mode	Escaping
Buscar_Puerto		
Conectar		
Desconectar		
Detener		
Funcion_Macro		
Leer_Datos		
Limpiar		

Code Generation

General | Features | Advanced | Generator Log

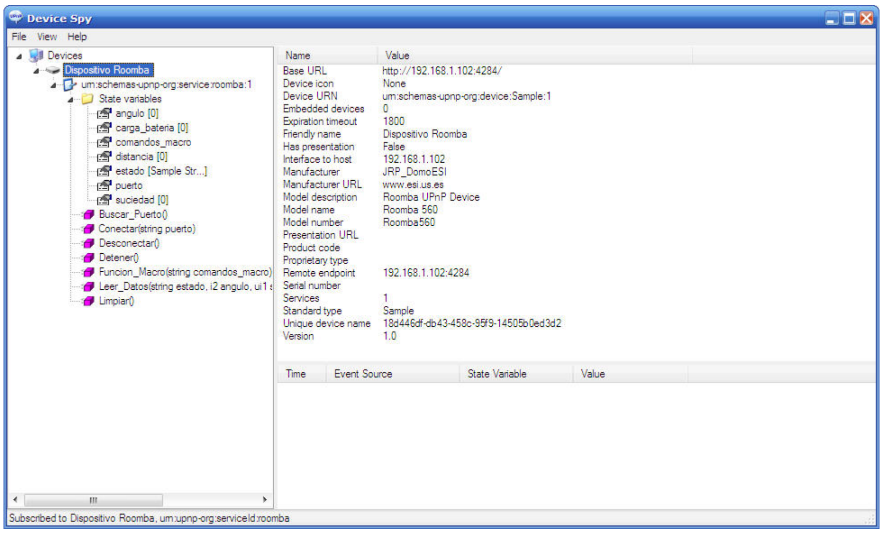
Target Platform	.NET Framework Stack (C#)
Project Name	
Output Path	nd Settings\Administrador\Escritorio\Roomba UPnP
New Line Format	LF, UNIX style code
Calling Convention	None - Compiler default calling convention
Library Code Prefix	ILib
Code Indention	1 Tab
Namespace	

Figure 3.8: Creation of the device stack

Builder program which is another tool from Intel. This program supports the creation of UPnP stacks for both control points and devices. In case of developing UPnP devices is necessary to import XML service description files from the previous step. It is possible to customize certain parameters that characterize the device or control point in the network. For example, it is possible to edit the *Friendly Name* of the device that is the name by which the device will be identified in the network; or the *Service Name* and the *Service ID* attributes of each service, i.e., the name and identifier of each service. After importing the service description files and editing the attributes required, the developer must generate the code in the chosen programming language for the development of the application. This will create all the code files required for the compilation of the device or the control point. Note that at this point it is available the source code required to compile and build a fully functional device at a network level.

3.2.3 Step 3: Implementation of the Actions

Despite the code required to create a UPnP device is available after the previous step, it is essential to type the code that implements the functionality of each action defined in the first step. Otherwise, the device does not respond to any requests made by other entities in the network since it has no code to do it. Therefore, it is necessary to use an editor and a compiler for the



```
Devices
  Dispositivo Roomba
    um:schemas-upnp-org:service:roomba:1
      State variables
        angulo [0]
        carga_bateria [0]
        comandos_macro
        distancia [0]
        estado [Sample Str...]
        puerto
        suciedad [0]
      Buscar_Puerto()
      Conectar(string puerto)
      Desconectar()
      Detener()
      Funcion_Macro(string comandos_macro)
      Leer_Datos(string estado, i2 angulo, ui1 s
      Limpiar()
```

Name	Value
Base URL	http://192.168.1.102:4284/
Device icon	None
Device URN	um:schemas-upnp-org:device:Sample:1
Embedded devices	0
Expiration timeout	1800
Friendly name	Dispositivo Roomba
Has presentation	False
Interface to host	192.168.1.102
Manufacturer	JRP_DomoESI
Manufacturer URL	www.esi.us.es
Model description	Roomba UPnP Device
Model name	Roomba 560
Model number	Roomba560
Presentation URL	
Product code	
Proprietary type	
Remote endpoint	192.168.1.102:4284
Serial number	
Services	1
Standard type	Sample
Unique device name	18d446df-db43-458c-95f9-14505b0ed3d2
Version	1.0

Time	Event Source	State Variable	Value

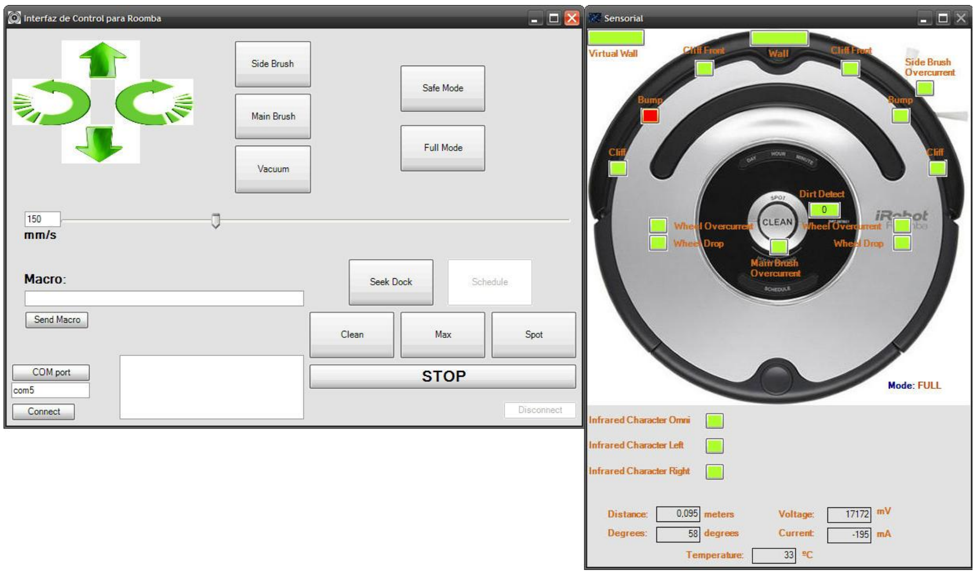
```
Subscribed to Dispositivo Roomba, um:upnp-org:serviceId:roomba
```

Figure 3.9: Operational testing

full development of the device. The UPnP gateways detailed in this chapter have been developed using the Visual C# 2010 Express [161] program. Obviously, the code in this step is custom designed for each application. Also, note that it is not necessary to implement only the functionality of the UPnP actions. In other words, the program can be as complex as desired and, even, a very complex program may provide only a simple UPnP action. Finally, the resulting code is compiled and an executable file is obtained. From this moment on, any UPnP control point can control the virtual device.

3.2.4 Step 4: Operational Testing and Validation of the Device

Finally, you should check the correct operation of the device developed. This procedure could be considered as another step in the design process of UPnP devices since the device may provide a non-expected functionality or cause errors during execution. To that end, the application is run and is tested with any control point. The Device Spy program is an Intel tool that allows to invoke actions and to subscribe to events. This verifies the proper operation of our UPnP device (Figure 3.9). If you detect any deficiency in the operation of the device you should check the code implemented during the third step. However, if there is a deficit in the services provided by the device, you should change some details from the first and second steps, and add the appropriate code associated with the third step.



150 mm/s

com5

Mode: FULL

Distance: 0.095 meters
Degrees: 58 degrees
Voltage: 17172 mV
Current: -195 mA
Temperature: 33 ºC

Figure 3.10: UPnP application for Roomba robot

3.3 UPnP Gateways for Robots

3.3.1 Roomba Gateway

The first gateway developed in this thesis was the UPnP Roomba bridge. It is an application (Figure 3.10) that offers Roomba's services and state in a UPnP network. Specifically, this application is a UPnP virtual device that works as a bridge intercommunicating the home automation network and the Roomba (Figure 3.11). Roomba's service provides the actions programmed and the necessaries variables to know Roomba's state. Users may invoke actions and Roomba will execute them transparently.

Implementation

The first problem to be solved is the physical communication. The virtual device establishes the communication with Roomba using a Bluetooth modem, called RooTooth (Figure 3.12). RooTooth is connected into Roomba's external serial port. Thus, what the virtual device does is to collect the actions from the UPnP control points and translate them into Roomba ROI commands that are transmitted via Bluetooth (Figure 3.13).

Once that communication between the Roomba and the PC is solved, we can focus on the creation of the UPnP device. This procedure was exposed previously and during the step of the implementation of the actions, most

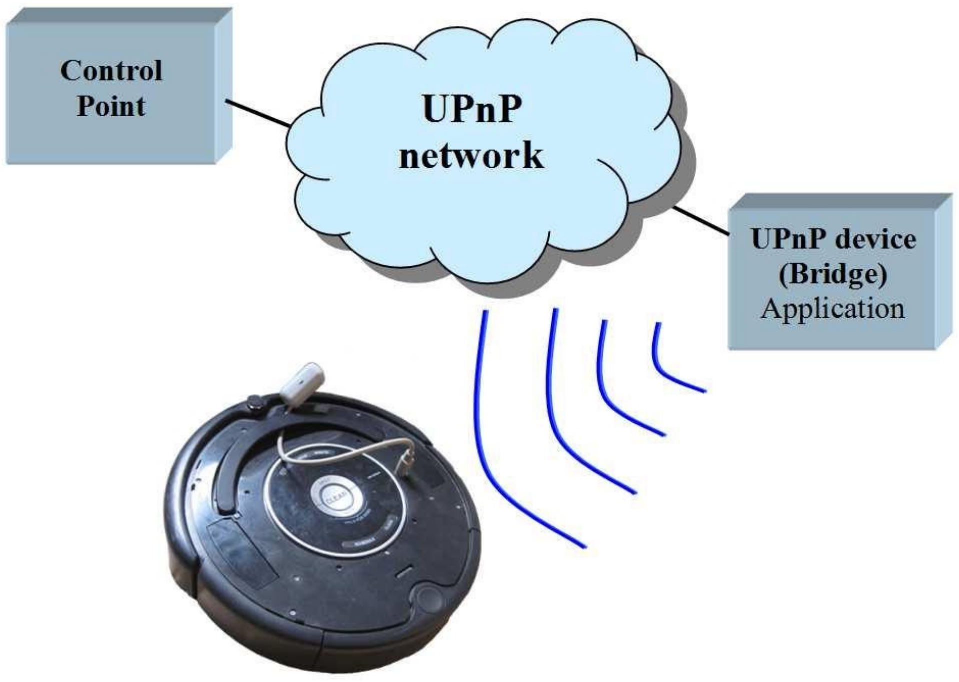


Figure 3.11: Roomba UPnP gateway

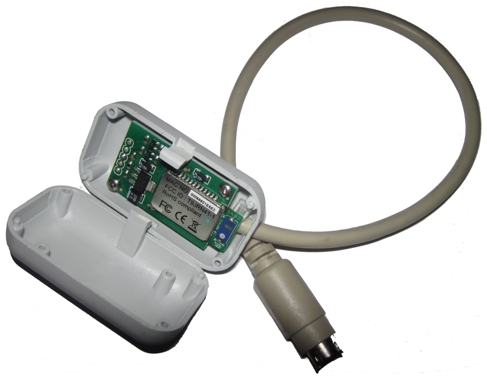


Figure 3.12: RooTooth

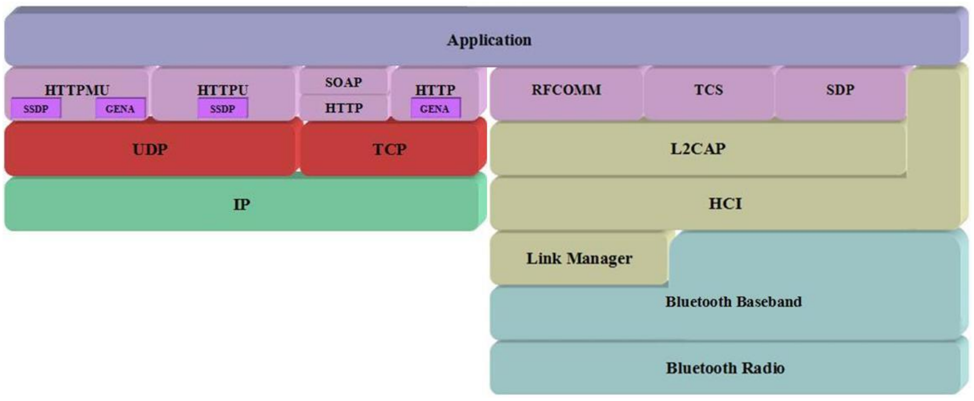


Figure 3.13: Roomba UPnP-Bluetooth bridge stack

hand-written code is dedicated to managing the communication with the RooTooth and sending via the serial link the command opcodes that will be obeyed by Roomba. A description of all the opcodes can be found on the iRobot SCI document [147]. After this step, an executable file is obtained. From this moment on, any UPnP control point can control the virtual device. The virtual device works like any standard UPnP device, i.e., it follows the operational steps shown in Section 2.7:

- It gets an IP address from a DHCP server or the AUTO-IP mechanism (addressing step).

- It announces its services and control points discover them (discovery step). Next, the device sends information about itself and its services to the control point (description step).

- Upon discovery, the control points are able to control Roomba transparently; they have a list with the actions users may invoke and a list with the variables they may check (control step).

- It is even possible to include evented variables (eventing step). However, it was preferred not to implement them to reduce the UPnP traffic in the network. Therefore, to have an updated list of the variables, our virtual device polls the Roomba periodically and saves the data into internal variables. Then, when users invoke the proper action, the bridge will update and return the value of the UPnP device variables.

Actions and State Variables

Once the application is running, it can be detected from a UPnP control point. The variables and services offered to the network were chosen carefully as the ROI allows the control of an overwhelming number of options. To keep the application as simple as possible, the device offers the same basic actions that can be controlled with the Roomba buttons complemented with some extra features for the digital home. In Figure 3.14, it can be seen a schematic representation of the virtual device. As can be seen in the figure, the following variables were implemented:

- *Port (string)*: Stores the virtual serial port used to connect with the Roomba.
- *Macro_commands (string)*: Stores a path to be followed by the Roomba. The format is composed of instructions to go forward (U), backward (B), turn to right (R) or turn to left (L) followed by the time in milliseconds to go in the specified direction. For example, if the string is "U700L200", the Roomba will go forward for 700 ms and then it will spin leftwards for 200 ms.
- *Dirt (unsigned 8-bit integer)*: Stores the dirt level detected by the capacitive sensor.
- *State (string)*: Stores the Roomba internal mode.
- *Battery_charge (unsigned 16-bit integer)*: Stores the charge of Roomba's battery in milliamperes-hours (mAh).
- *Distance (signed 16-bit integer)*: Accumulates the relative distance travelled by Roomba in millimetres (mm) since the device was executed.
- *Angle (signed 16-bit integer)*: Provides the relative angle that Roomba has rotated.

To reduce the network traffic, these variables are not evented. It is needed to invoke an action so that the application updates the published value of the variables. Otherwise, the Roomba virtual device would be constantly sending event messages because the value of the sensors is continuously changing. An option to lower the network traffic with evented variables would be to implement a threshold in the change of value of the variables before proceeding to their update.

The actions we have implemented are:

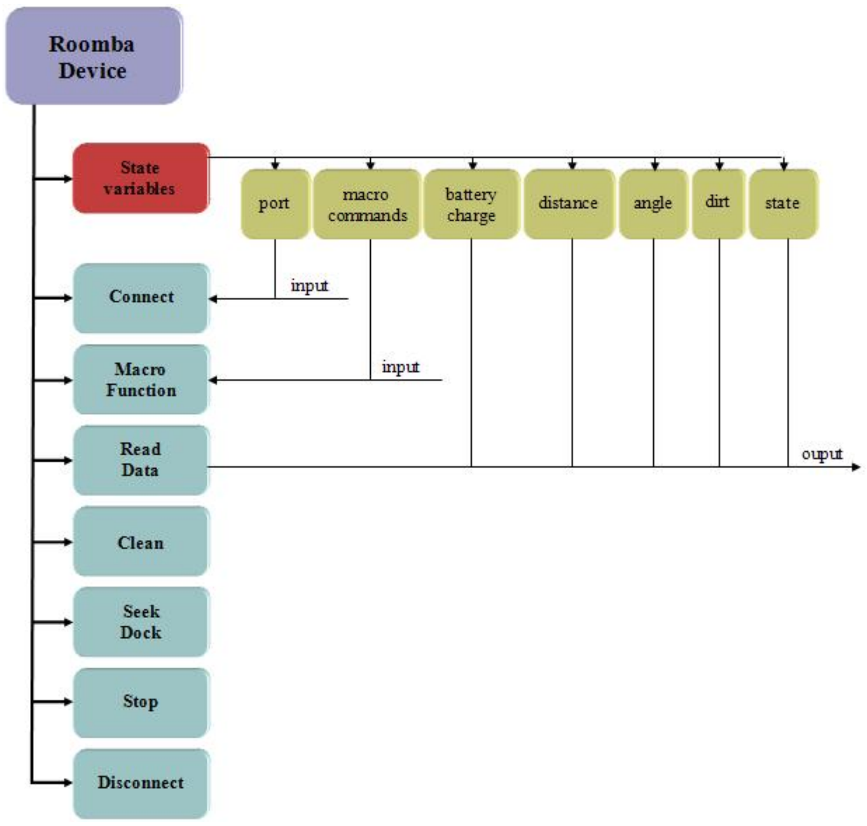


Figure 3.14: Roomba UPnP virtual bridge scheme

- *Connect (input: port)*: Establishes the link between the RooTooth and the UPnP virtual device through the virtual serial port used by RooTooth. A passkey is required to establish the communication. Then, the function "wakes up" Roomba using several special commands (see [132] for details); Roomba will not respond to any commands when it is "asleep". Finally, it starts the ROI port and finally it puts Roomba into the Safe Mode.

- *Macro_Function (input: macro_commands)*: Allows to send macros with the proper format for Roomba movements using the variable "macro_commands".

- *Clean*: Starts the Roomba's autonomous cleaning mode.

- *Seek_Dock*: Makes Roomba start its seek dock routine.

- *Read_Data*: Retrieves sensor values from the robot through the virtual device. These data are: mode, dirt level, distance and angle since the last measurement and the current charge level of Roomba's battery. They are received in the variables "state", "dirt", "battery_charge", "distance" and "angle", respectively.

- *Stop*: Stops any of the routines started by Roomba.

- *Disconnect*: The virtual device removes the link with the robot. Also, this function must be called when the bridge application ends.

3.3.2 Rovio Gateway

The second gateway that enables the interoperability between a robot, in this case the Rovio robot, and a smart home is presented in this subsection. This application provides a Rovio interface with a UPnP network. It is a UPnP virtual device that works as a bridge or gateway between the home automation network and the Rovio. The UPnP bridge provides the Rovio API functionality to the UPnP network. In this manner, users may invoke the existing actions in Rovio's service and then, the robot will execute them.

We used for this research the aforementioned Tools for UPnP Technology provided by Intel which helps to accelerate the developing, testing and deploying of UPnP compliant devices, including full support to audio and video distribution.

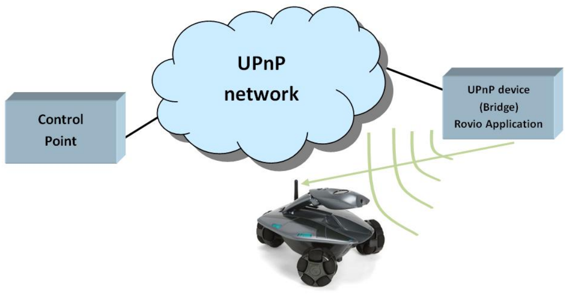


Figure 3.15: Rovio UPnP bridge (Router-Bridge and Bridge-Rovio connections).

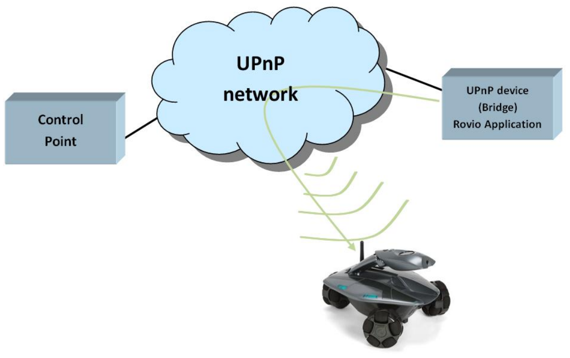


Figure 3.16: Rovio UPnP bridge (Router-Bridge and Router-Rovio connections).

Implementation

First of all, a wireless network connection has to be established in order to control the Rovio from the PC where the UPnP application is programmed. There are two possibilities at this point: the first one is to connect the robot directly to the PC by means of an ad hoc network and at the same time to connect the PC to the UPnP network (Figure 3.15); otherwise the Rovio robot and the PC have to be in the same network (Figure 3.16). Note that in the last case, the same network can be used for the UPnP messages and the Rovio HTTP requests (both protocols use the TCP/IP stack).

Once Rovio is able to receive messages from the PC where the bridge is implemented, the proper implementation of the UPnP functionality can take place. The steps accomplished to create the bridge are the previously detailed for the creation of any UPnP virtual device. During the programming step of the UPnP device behaviour we have coded the actions to be executed by the robot. Most handwritten code deals with the translation of the UPnP commands into HTTP requests for the Rovio. Likewise, the value of the evented variables is updated taking into account the information submitted by the robot. Once the application is running, any control point can invoke the actions offered by the Rovio UPnP bridge.

From this point, the device will work as any common UPnP device,i.e., it follows the UPnP operation mechanism described in Section 2.7 in a similar way that the Roomba virtual bridge does.

Actions and State Variables

The Rovio UPnP bridge offers several actions from the Rovio API to the UPnP network. Likewise, it has several input and output variables which determine the way in which the communication with the UPnP network is performed. In this subsection we offer an overview of the actions and variables which characterize the device. A representation of how actions and variables are related in this device is illustrated in Figure 3.17. The complete list of variables is the following:

- **User (string):** is the variable sent to Rovio which identifies the username.
- **Password (string):** allows to authenticate the user provided previously.
- **IP_address (string):** it is the IP assigned to the robot.
- **Movements (string):** stores a sequence of movements. The variable must contain the instructions to go in a specific direction followed by the movement duration in milliseconds. The instructions are: “U” to go forward, “B” to go backward, “R” to turn right and “L” to turn left. For example, if the variable is “B1000R500” the Rovio goes backward for 1 second and then it will turn to right for 500 milliseconds.
- **Path (string):** identifies a path stored by Rovio.
- **Speed (unsigned 8-bit integer):** specifies the velocity of the robot.
- **Garbage_position (string):** provides the x and y coordinates where Rovio have detected garbage.
- **State (string):** stores the robot state.
- **Path_list (string):** provides the paths saved by Rovio.

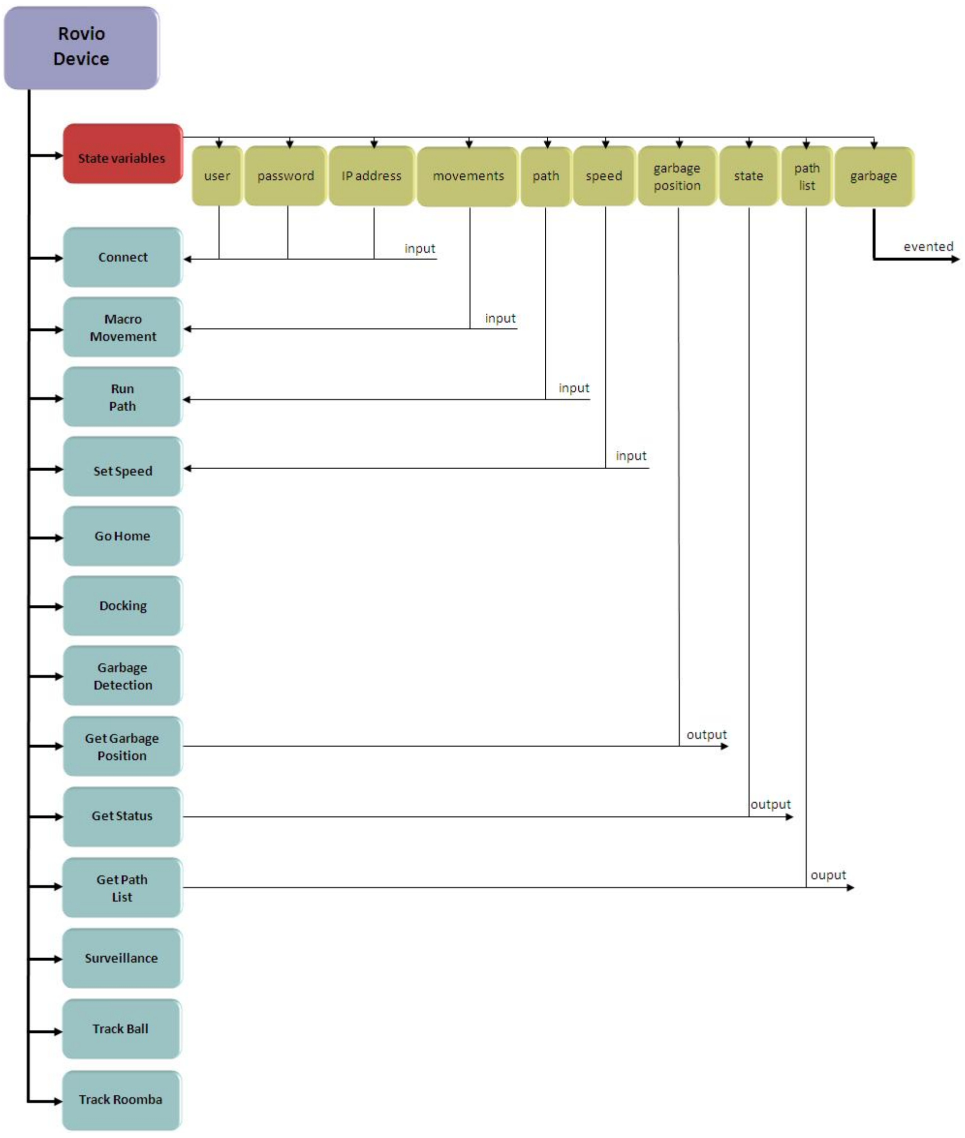
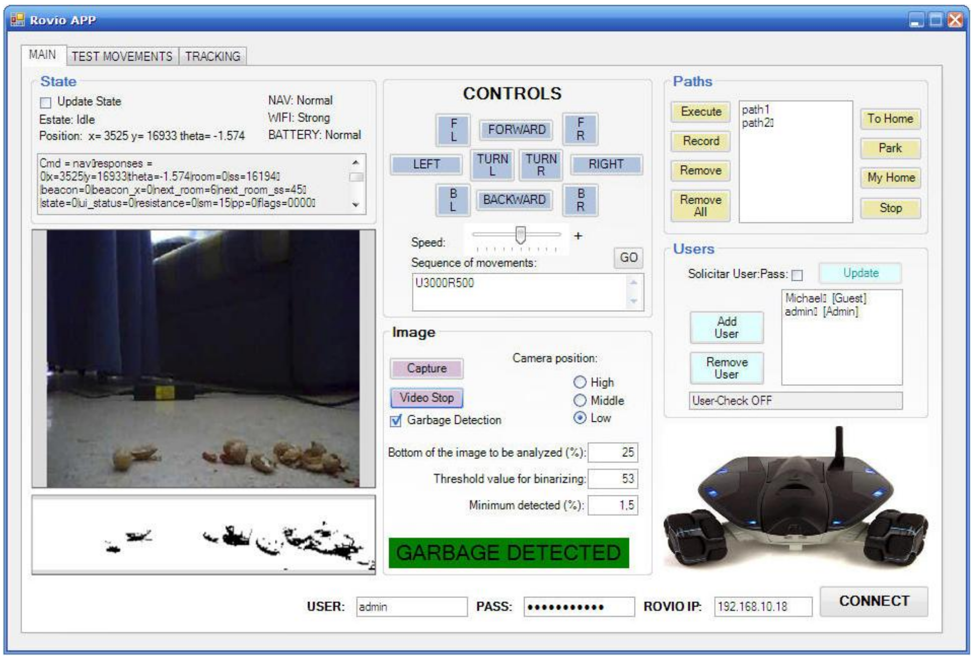


Figure 3.17: Rovio UPnP virtual bridge scheme

- **Garbage (boolean):** is an evented variable that announces when the Rovio detects garbage.

Next, we list the actions that the device offers:

- **Connect** (input: user, password, IP address): establishes a communication between the Rovio and the virtual bridge via WiFi. It is not possible to send any command to the robot until the connection is established. To initialize the connectivity, it is necessary to provide a user and its related password, and the IP address.
- **Macro_Movement** (input: movements): orders to the Rovio a sequence of movements indicated in the "movements" variable. When the Rovio receives this invocation, it analyzes the instructions and it performs the proper sequence of movements.
- **Run_Path** (input: path): executes a sequence of movements to follow a path previously saved by Rovio.
- **Set_Speed** (input: speed): adjusts the speed with which the Rovio will execute the movements.
- **Go_Home**: sends the robot to a position near the Home Base.
- **Docking**: sends the robot to dock in the Home Base.
- **Garbage_Detection**: activates the detection garbage system. Rovio takes pictures periodically with its camera, and it will process the images in order to detect garbage.
- **Get_Garbage_Position** (output: garbage position): gives back the coordinates of the garbage position detected by the robot. The operation of these last two actions will be deeply analyzed in the experiments section.
- **Get_Status** (output: state): provides a report with the state of the robot.
- **Get_Path_List** (output: path list): provides a list with the paths previously saved by Rovio.
- **Surveillance**: with the cam in the high position the robot will take pictures continuously. By subtracting two consecutives images, if the resulting image has too much information, the robot will consider the presence of an intruder and from now on it will upload the images files to a server. This action is justified in next section.



Update State: ☐
Estate: Idle
Position: x= 3525 y= 16933 theta= -1.574
NAV: Normal
WIFI: Strong
BATTERY: Normal

```
Cmd = nav|responses =
0|x=3525|y=16933|theta=-1.574|room=0|ss=16194|
|beacon=0|beacon_x=0|next_room=6|next_room_ss=45|
|state=0|ui_status=0|resistance=0|sm=15|pp=0|flags=0000|
```

Speed:
Sequence of movements: U3000R500

Camera position: High Middle [✓ Low]
Garbage Detection: ✓
Bottom of the image to be analyzed (%): 25
Threshold value for binarizing: 53
Minimum detected (%): 1,5
GARBAGE DETECTED

path1
path2

Solicitar User:Pass: ☐
Michael [Guest]
admin [Admin]
User-Check OFF

USER: admin PASS: ••••••••••• ROVIO IP: 192.168.10.18

Figure 3.18: Rovio GUI

- **Track_Ball**: the Rovio will find a ball by means of some image processing algorithms and once it has found the ball, the robot will follow it. This action is explained on detail in 3.4.4.

- **Track_Roomba**: this action is similar to the Track_Ball one. The robot can find and follow the Roomba vacuum cleaner robot. This case is detailed in 3.4.5.

All the actions and the variables presented before allow the user to control the Rovio transparently through a UPnP network using a control point. In addition, the application provides a graphical user interface (GUI) shown in Figure 3.18 to control the robot regardless of the UPnP standard.

3.4 Experiments

We have carried out different experiments in order to test the UPnP bridges and to explore some of the new possibilities the Rovio and Roomba robots may offer into an interoperable digital home.

3.4.1 Robots in a UPnP Network

The first experiment to test the UPnP gateways was to control the robots through the control point included in the Intel UPnP Tools. In [162] a case of use with the Rovio robot is shown. It consists of invoking the *Connect* action, then the robot state is obtained by invoking the *Get_Status* action. Afterwards, a movement sequence is performed by means of the *Macro_Movement* action, then the robot speed is changed by invoking the *Set_Speed* action, and finally the same movement sequence is executed. It worked as it was expected, the robot offered all the services to the UPnP network and it performed all the actions invoked by the control point. The Roomba robot was also tested with this tool and every action was executed to check the proper operation.

In addition, the UPnP devices were tested with a different manufacturer equipment, in particular, in an installation of IPDomo [111], a native home automation system. Once the application was proved to work in lab conditions, it was tested in a real smart home. The IPDomo installation was composed of a PC in which it was installed the IPDomo control point and several multifunction and security cards. Thanks to this equipment, a total amount of 4 lights, 4 blinds and different sensors (presence, gas and fire detectors) were controlled. The main difference between the control points of Intel and IPDomo is that the second allows the programming of macros with all the UPnP devices present in the network. The Roomba application and the necessary Bluetooth stick were installed in the same PC that ran the control point. Once the Roomba software bridge was executed, the UPnP device was recognised immediately and some macros were programmed. For example, cleaning tasks were scheduled in some days if and always if the condition that there was nobody at home in the moment the cleaning was fulfilled. In addition, the Rovio UPnP virtual bridge was executed in the same PC as the IPDomo control point and macros were programmed to test the operation of the device in the installation. The system was configured to invoke the Surveillance action in the case that presence sensors detected something. This is an useful application in order to reinforce the security at home. As we checked from these tests, the UPnP devices work well in a common UPnP network and they perform all the actions perfectly.

3.4.2 Roomba tests

Different test were carried out with the Roomba robot:

- *Trajectory tests*: Although it has not been developed the necessary software to calculate the position of the Roomba using the measurements

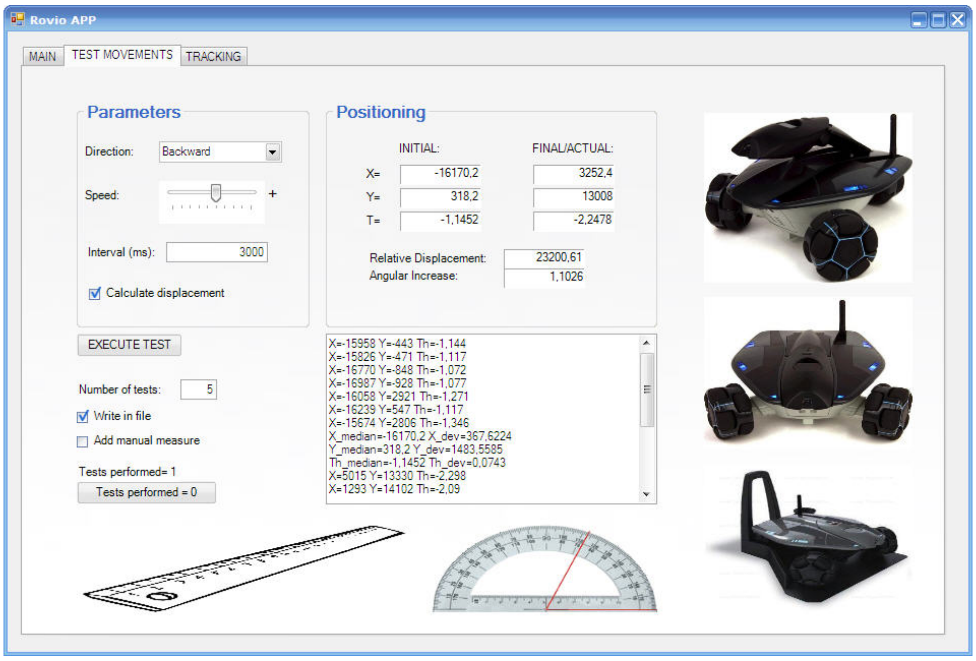
Table 3.1: Dirt sensor tests

Dirt	*Measurement*
Chickpeas	2
Beans	51
Star-shaped noodles	128
Lentils	251

provided by the wheel encoders, it is still possible to control the movement of the vacuum cleaner in open loop. The Roomba has shown a very lineal time-distance relationship when the action Macro_Function is invoked. The repeatability in the behaviour of the robot is good enough to guarantee that Macro_Function can be used to command Roomba to go to a room.

- *Dirt sensor test*: The dirt sensor provides a value between 0 and 255 as a function of the dirt density. This information is useful to determine the type and degree of dirtiness in the house. The tests consisted of spreading over a small area typical cooking ingredients on the floor and the results are shown in Table 3.1. The conclusion that can be extracted from the experiments with this sensor is that it registers higher values as the flow of individual elements grows. It is also remarkable that if the dirt size is equal or greater to chickpeas then jams may occur in the main brush of the cleaner.

- *Distance and angle sensors*: We also made some experiments related to the use of the distance and angle sensors. Our results showed that there exist linear relations between the distance and angle displacement of the robot and the measurements obtained. The results show a correlation between the speed of the robot and the quality of the measurements. The faster the Roomba goes the worse performance of the measurements was obtained. This application also depends dramatically on the reliability of the communication link between the control point and the Roomba. In case, there are lost packages, the estimated location of the Roomba can be very wrong. We recommend the interested reader to see [142] to get more information about location of the Roomba using only information coming from its own sensors.



Direction: Backward

Speed:

Interval (ms): 3000

[✓] Calculate displacement

	INITIAL:	FINAL/ACTUAL:
X=	-16170,2	3252,4
Y=	318,2	13008
T=	-1,1452	-2,2478

Relative Displacement: 23200,61
Angular Increase: 1,1026

Number of tests: 5

[✓] Write in file

[ ] Add manual measure

Tests performed= 1

```
X=-15958 Y=-443 Th=-1,144
X=-15826 Y=-471 Th=-1,117
X=-16770 Y=-848 Th=-1,072
X=-16987 Y=-928 Th=-1,077
X=-16058 Y=2921 Th=-1,271
X=-16239 Y=547 Th=-1,117
X=-15674 Y=2806 Th=-1,346
X_median=-16170,2 X_dev=367,6224
Y_median=318,2 Y_dev=1483,5585
Th_median=-1,1452 Th_dev=0,0743
X=5015 Y=13330 Th=-2,298
X=1293 Y=14102 Th=-2,09
```

Figure 3.19: Tool for statistical measures

3.4.3 Rovio tests

The tool shown in Figure 3.19 facilitates the measuring task and it allows to study the movements of the Rovio robots.

It is possible to adjust the direction of the movement, the speed and the movement duration in order to get the displacement and rotation measured by Rovio's positioning system.

The statistical operation of this tool is determined by the number of iterations of each test. In each iteration Rovio gets some samples of its position and the mean and standard deviation is calculated. The robot accomplishes the movement and the mean and deviation of its position is calculated again. The tool will ask for manual measures if the corresponding tick box is marked. Then, a new test iteration will start. After the last iteration the tool provides the mean and standard deviation of the robot displacement/rotation, and a relation between the manual measures and that provided by the positioning system.

In order to get the model of each robot movement, several tests have been developed. All tests have been performed for three different time intervals for each direction, and linear interpolation has been used between samples. The results obtained are illustrated in Figures 3.20, 3.21 and 3.22 and the

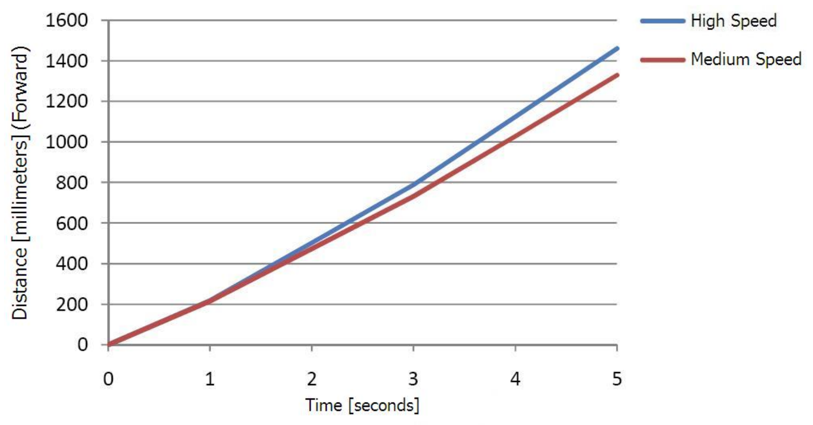


Figure 3.20: Forward

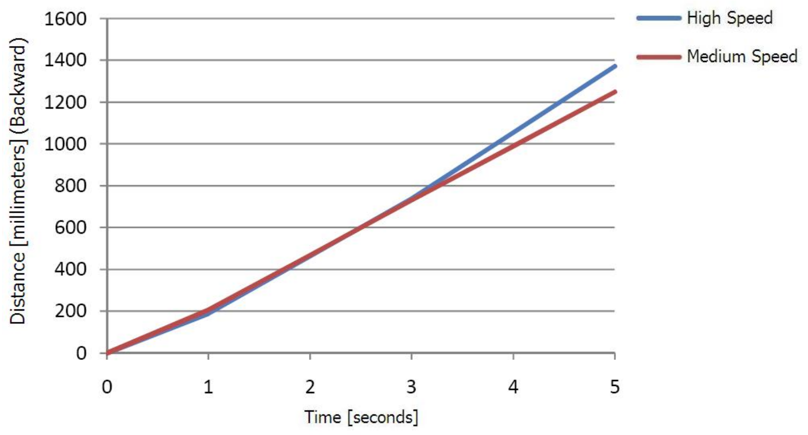


Figure 3.21: Backward

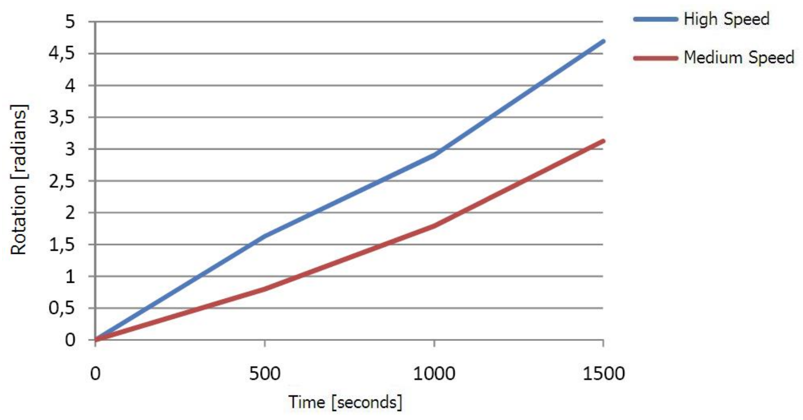


Figure 3.22: Left rotation

Table 3.2: Rovio movements models

Direction	Speed	Equations of motion
Forward	High	y [mm] = 294,46x - 54,3
	Medium	y [mm] = 266,9x - 37,67
Backward	High	y [mm] = 277,63x - 58,41
	Medium	y [mm] = 253,06x - 25,97
Rotation	High	y [radians] = 3,068x + 0,004
	Medium	y [radians] = 2,07x - 0,125

models are represented in table 3.2.

3.4.4 Rovio Robot Tracking a Ball

In this experiment we have used the Rovio's webcam to track a ball. We have implemented this tracking application as the base for the tracking of more complex objects as it will be shown in the next subsection. In order to detect the ball, the robot takes periodic pictures using its webcam and processes these images in real time looking for the ball. When the robot finds it, Rovio moves in order to maintain a certain alignment and position with respect to the ball.

The ball-tracking application is based on the x-coordinate of the centroid and the size of the ball. Each image taken by Rovio is processed, in order to get the centroid and the radius of the ball, by applying several image filters available in the AForge [163] and OpenCV [154] libraries:

- HSL: allows to extract objects from the image with similar features in hue, saturation and lightness. It has been used to get the ball by adjusting the saturation parameter.

- Median: is used to reduce noise in the image.
- Grayscale: allows to convert a color image into a gray scale image.
- Gaussian: is similar to the median filter. It is used to blur and smooth the images and it removes details and noise.
- Circular Hough Transform: detects the presence of circular shapes on an image. For this application it has been configured to detect one circle by adjusting the filter parameters. It is the last image processing and it provides the radius and the centroid coordinates if the ball is on the image.

Once the application detects the ball, it calculates its centroid and it tries to align the robot with the ball by performing rotary and lateral movements. In addition, when Rovio has centered the ball, it tries to keep at a certain distance from the ball using its radius to measure this distance. If the robot loses the ball, Rovio tries to find it on the direction that the ball was moving. On the other hand, if the Rovio is moving and it detects the ball, and it has disappeared in the following image, it will spin the opposite of its last move. Finally, after looking for the ball, if the robot does not find it, the Rovio will move randomly.

As we can see in [164], it has been developed an application for Rovio that is able to look for the ball, to align with it and to follow it if you move the ball.

3.4.5 Rovio Robot Tracking a Roomba Robot

In this case we have developed an application that allows the Rovio to follow the autonomous vacuum cleaner Roomba.

In order to simplify the tracking problem, we added red rectangles around the Roomba. Thus, the image processing is based on that performed to detect the ball. First an HSL filter is applied to get red objects by adjusting the hue and saturation parameters, then, a grayscale filter is applied. Next, the image processing is a bit different and follows these steps:

- Segmentation: this process allows to locate objects and boundaries. The contours extracted from the image are filtered with a size filter in order to remove unwanted red small objects.

- Sobel edge detector: performs a spatial gradient on an image and emphasizes regions of high spatial frequency. It detects and remarks the edge of the objects previously filtered.

- Rectangles validation: this step consists on look for objects that fulfill some geometrical conditions. The object should be composed of four vertices, and thus it should have four sides. In addition, two consecutives sides should form an angle close to ninety degrees. Once the rectangles have detected, the average of the centroids and the average height of the rectangles' sides are calculated. The mean centroid is used to determine if Rovio has focused Roomba or not, and the direction that Roomba is following. The mean height is useful in order to know if Roomba is too far or too close to Rovio.

The Rovio spins on its axis looking for Roomba. When it identifies some of the red rectangles included on Roomba, Rovio moves towards or away from the robot depending on the average height of the sides. If Roomba goes forward the Rovio will follow it. If the Rovio does not find Roomba during the pursuit, it will turn in the direction where it was shifting the mean of the centroid and it will go forward to find it again. The goal of this application is to follow the Roomba in order to know its position through the Rovio. Samples of this experiment are available in [165] and [166].

3.4.6 Garbage Detection

Another interesting application in a domotic environment is the detection of garbage by means of image processing. In this scenario, Rovio is programmed to patrol the house while looking for garbage. In case of garbage detection, an UPnP event is generated and the digital home acts consequently. For example, an autonomous vacuum cleaner robot such as the Roomba [44] could be sent to clean the area. An example of this is available in [167]. This scenario is a good example of the digital home operation based on the interoperability between domotic and robotic devices. Rovio acts as a mobile sensor since it moves and can detect garbage, and the vacuum cleaner robot has the role of an actuator performing a cleaning task. Both devices are coordinated by a control point which manages this operation.

It is important to remark that precise detection of garbage in an arbitrary floor is in general a very difficult problem which is beyond the scope of this work. For this reason, we have developed a simplified garbage detection application which allows the Rovio to detect garbage in some particular scenarios.

Within a testing framework, we have developed an application that looks for garbage when a control point invokes the *Garbage_Detection* UPnP action or when the user enables it directly through the GUI. In addition, users may adjust three parameters in order to detect garbage:

1. The percentage of the bottom of the image to be analyzed.

2. The threshold to binarize the image which processes and transforms it into a black and white image.

3. The percentage of black pixels after the threshold step that are necessary to consider them as garbage.

If Rovio detects garbage it announces it to the UPnP network by changing the value of the boolean evented variable (Garbage). Even, it is possible to invoke the *Get_Garbage_Position* UPnP action that returns the position of garbage in the *Garbage_position* variable.

3.5 Conclusion

We have presented applications to integrate the service robots, Roomba and Rovio robots, into the smart home. The integration is based in UPnP software bridges that are executed in a computer. One of the program controls the Roomba via Bluetooth, and the other controls the robots using Wifi, and both of them offer UPnP interfaces so that UPnP control points can command the Roomba and Rovio robots. Both robots are deeply analyzed in this chapter as service robots and UPnP is analyzed as a middleware platform for the integration of service robots into the smart home.

In addition, we have performed several experiments which have shown that the developed applications can be successfully integrated in a commercial UPnP home automation system. Moreover, we have shown that there are interesting services which arise from the full integration of robots in smart homes. For example, we have presented an experiment in which Rovio acts as a surveillance camera after a presence detector sensor warns the home automation system. Another interesting example is the application in which the Rovio's bridge uses its garbage detection routine and communicates the control point the coordinates where garbage was detected. After that, the control point sends Roomba robot to clean the dirty area. These are good examples which highlight the benefits of interoperability in smart homes. Despite that total interoperability seems to be far away due to the large number of existing systems and the lack of a common standard, the potential benefits of the integration make the necessary efforts worthwhile. The application presented in this chapter is a contribution to go towards this objective.

The goal of this research is to provide a framework for the integration of service robots in the smart home. Although the applications has proven to be useful in a real home automation environment, there are some aspects that have to be improved and new lines of research that have to be explored.

The new services that emerge from the integration of robots and smart homes are promising. For example, in this application, we have made possible that the house control system can send the Roomba to any room. Consequently, it is possible to schedule the cleaning of the rooms or, imagining a more technological and integrated scenario, if a camera at home would detect dirty spot the control system could send the Roomba to clean it. Nevertheless, we have to remark that the home automation market is not mature enough to incorporate these ideas now, at least not at a mass market level, but we must not forget that in the long run ambient intelligence is the paradigm to follow, and it is clear that some of the steps needed to go towards this objective depend on the development of applications like the one that we have presented in this chapter.

Chapter 4

Collaborative Tasks between Robots based on the Digital Home Compliant Protocol over UPnP

The lack of interoperability is one of most important problems in the smart home. As a consequence, the integration of heterogeneous devices into the same system is a very difficult task which usually requires ad hoc solutions. For this reason, different interoperability standards have been proposed during the last two decades.

The Digital Home Compliant (DHC) protocol [100] is an attempt to provide interoperability between domotic and robotic devices. This protocol is developed and supported by a consortium comprised of different companies and public institutions. DHC is an UPnP-based protocol aimed at making consumer electronic devices and robots work in a collaborative and effective way, and also to improve their energetic efficiency. Given the novelty of the protocol, in this chapter we provide a brief description of its architecture. Nevertheless, we will focus on one particular component, the DHC-Groups module, which is the DHC core since it allows a group of robots to work together and provides the necessary mechanisms to perform collaborative tasks.

Given that this protocol is based on the UPnP architecture, robots working in accordance with UPnP technology are likely to meet the DHC requirements. UPnP has important features which make it a good starting point for developing a new home automation system for device interoperability. In addition, several studies have been developed for the implementation of UPnP in smart homes and the integration of robots. For example, in [5], consumer electronic devices and home automation systems are developed according to UPnP, which can also be employed as a middleware for the integration of robots [137]. Likewise, different gateways between UPnP and other technologies have been developed previously. A good example is the gateway proposed in [168], which solves the interoperability problem between X-10 and UPnP. In addition, a ZigBee UPnP gateway is detailed in [169].

To present the DHC protocol we have chosen as a practical example a common service robot: the Roomba vacuum cleaner [132], which has been succesfully marketed around the world. for research applications, the Create robot [170].

Since DHC is intended to be used with several different types of robots, software adapters specific for each robot must be designed to meet the requirements of the DHC protocol. In the case of Roomba, a DHC adapter has been developed from the UPnP Roomba adapter presented in [44]. The specific DHC adapter for Roomba will serve as a basis for explaining how DHC-Groups works. In addition, other adapters are currently being developed for robots such as Rovio Mobile WebCam [171], Pioneer Research Robot [172] or Sacarino Robot [105].

The outline of the rest of the chapter is as follows: Section 4.1 provides a brief overview of the DHC protocol and its goals . In section 4.2 the DHC-Groups module is presented. Section 4.3 deals with the Roomba vacuum cleaner, presenting its main features. The DHC Roomba adapter, which is based on the Roomba UPnP interface, is also explained in section 4.3. In section 4.4 some experimental results based on the operation of DHC-Groups are presented. Finally, section 4.5 provides the conclusions obtained from this research.

4.1 The DH Compliant protocol

The DHC protocol is designed to make robotic and domotic devices work together [173]. The DHC architecture (Figure 4.1) consists of a set of modules which are based on UPnP technology. Every module provides a key feature of the standard. These modules are:

- **DHC-Groups**: this is the key module for the performance of collaborative tasks by a robot hive. This module provides the mechanisms required to choose the group of robots that can perform a task and to choose a leader from among them. The foundations of collaborative work between robots are given in [174].

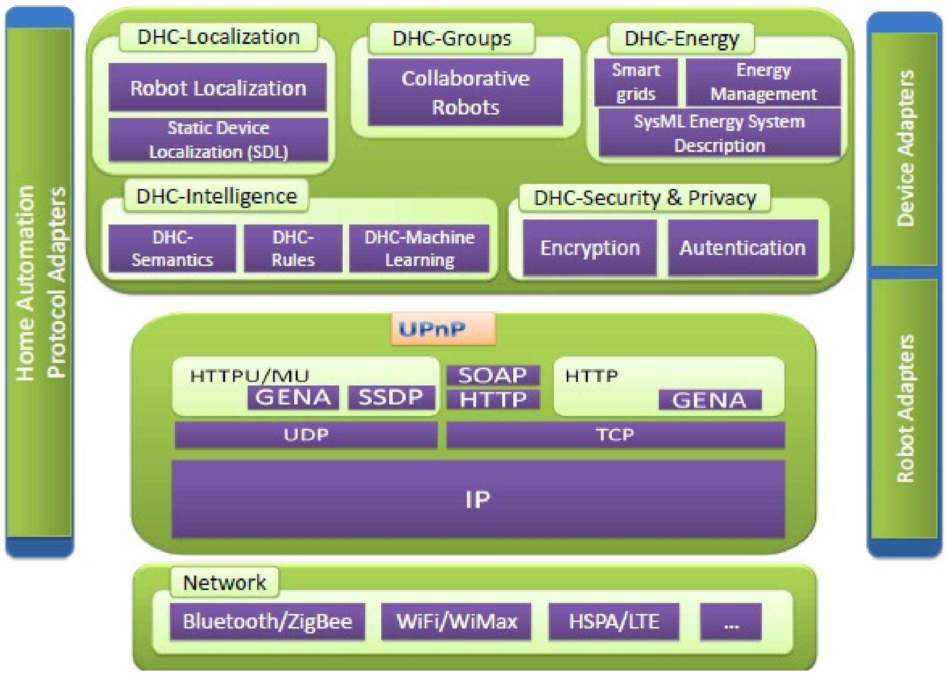


Figure 4.1: DHC Architecture [106]

- **DHC-Localization**: this module is necessary to require a robot to perform a task at given coordinates. It is possible to specify the coordinates where the task must be performed. An external location system must be used so that the robot can have its own coordinates and calculate the path to the task point [175].

- **DHC-Energy**: this module provides features related to the energetic efficiency in tasks performed by several devices [176].

- **DHC-Intelligence**: this module is still under development and has three submodules: DHC-Rules [177], which is useful for specifying certain rules that devices must follow when performing their tasks; DHC-Semantics, which deals with the semantic language used on the environment of DHC; and the DH-Machine Learning submodule, which is based on WEKA Machine Learning Services [178].

- **DHC-Security & Privacy**: this module focuses on the security which is required for communications using the DHC standard for the digital home. It deals with issues such as authentication, confidentiality, or data integrity [179]. The identification mechanism used by DHC-Security & Privacy is based on the decentralized digital identity standard called OpenID, which allows devices to identify themselves through an URL.

4.2 DHC-Groups operation

The steps that compose the operation of the UPnP standard are an essential part of the mechanisms on which DHC is based. DHC is an architecture composed of peer entities that have both control point and device features. Depending on the situation, DHC entities may behave as either a control point (for example, invoking actions or processing an event) or as a device (receiving commands from control points and performing tasks).

This section shows how the DHC-Groups module works. A DHC application is structured in three layers (Figure 4.2). The top layer is the human-machine interaction layer and consists of a graphical user interface (GUI) that is used to send orders to the robot adapter, which is the bottom layer. Between the top and bottom layers there are several DHC modules, one of which is the DHC-Groups module, which receives commands from the user interface and sends them to the robot adapter. DHC-Groups implements all the mechanisms required to make a robot group work in a collaborative manner. These three layers base their communication on the changes of UPnP state variables, which trigger several events. Next, we examine one by one each of these layers:

- Human-machine interaction: in this layer, a GUI is used to collect and store important information for the development of the tasks that will be launched. This information is provided by the user and includes fields as a task identifier (*TaskID*), the destination floor and room, important timing information about the task, the number of robots that can be used, or the *CollaborationKeyword*, which clearly indicates the kind of task that is going to be performed. This last field is chosen from a list of predefined keywords and determines the type of task to do and, consequently, the robots that can perform the task. Finally, all this information is parsed into a set of UPnP variables that are offered to the network so that the DHC-Groups modules can retrieve them.

- DHC-Groups module: when it is launched, it subscribes to the GUI's evented state variable *TaskID*. When the value of this variable changes, it means that the user has commanded to perform a new task after introducing the task parameters in the GUI. Likewise, the DHC-Groups module obtains the task parameters using several getters to this end. After that, the DHC-Groups module publishes the information regarding the *TaskID* and *CollaborationKeyword* so that all the robot adapters that are subscribed to this service are notified. Those robots that can perform the task will propose themselves to take part in the task and also to lead it. The DHC-Groups module will collect all the proposals received from the available robots and it will decide which robots will participate in the task and also which robot will lead it. When the leader has been chosen, the DHC-Groups state variable *Leader* changes its value to the ID of the robot that will be the leader. Every robot adapter is sent an event and the robot whose ID is the same as the leader ID must start the task. Finally the DHC-Groups module, by means of notifying messages, provides a list with the IDs of the robots that will perform the task, and it will subscribe to the *endTask* leader's state variable. *Remark:* The problem of deciding the robots in the hive and the leader is open and admits different possible solutions. During the development of the DHC architecture several options were considered by the consortium. For example, the simplest ones consisted almost in random choices based on the proximity of the robots to the coordinates of the task destination. Other more sophisticated options in the line of auction methods [180] were considered as well. For example, the quality of service provided by a robot in previous executions of the task, the expected cost of the new task, and the energy level and the load factor of the robot were weighted and used to determine the robots in the hive and its leader. Given that an assessment of the validity of the choice depends on the tasks to be performed, the particular algorithm implemented is kept outside by now of the specification of DHC-Groups, which simply defines the communication protocol between the robot adapters and DHC-Groups module. Hence, the specific algorithm used depends on the particular implementation of the DHC-Groups module. *Remark:* The DHC-Groups specification also defines the case in which a robot rejects to perform a task with a TaskNACK message. This can happen under different exceptional circumstances. Examples of this kind of situations are: the destination cannot be reached by the robot, the energy level of the robot is critical, or the robot is busy.

- Robot adapter: when the robot adapter is launched, it subscribes to two DHC-Groups state variables: *TaskID* and *CollaborationKeyword*. This second variable is the key that makes possible to decide whether a robot can perform a certain task. Every robot meeting the DHC requirements has a set of *CollaborationKeywords* that indicates its capabilities. When a robot adapter is sent an event, it reads the *CollaborationKeyword* value, and compares it with every value of its set of keywords. If the current keyword belongs to the robot set of keywords, it means that the robot can perform the required task, so that the robot proposes itself as a candidate to collaborate in the task and also to lead it. The DHC-Groups module will decide whether the robot takes part in the task. The way that robot leader coordinates the team depends on its capabilities and how the adapter has been developed. Nevertheless, it must be based on the UPnP invocation messages. In this case, we have developed a simple algorithm on each adapter that divides the place to clean and assigns a region to each robot. It is possible to implement more complex algorithms, but this is beyond the scope of this work. Once the leader has the coordinates for each robot, it invokes the action *CleanAtPoint* of every robot in the hive and subscribes to the corresponding *EndTask* state variable. Once the robots accomplish their subtasks, they change the value of their *EndTask* variable, which generates a notification event for the leader. Then, if the leader has finished its own subtask, it modifies its own *EndTask* variable in order to notify it to the DHC-Groups module, which is subscribed to this state variable.

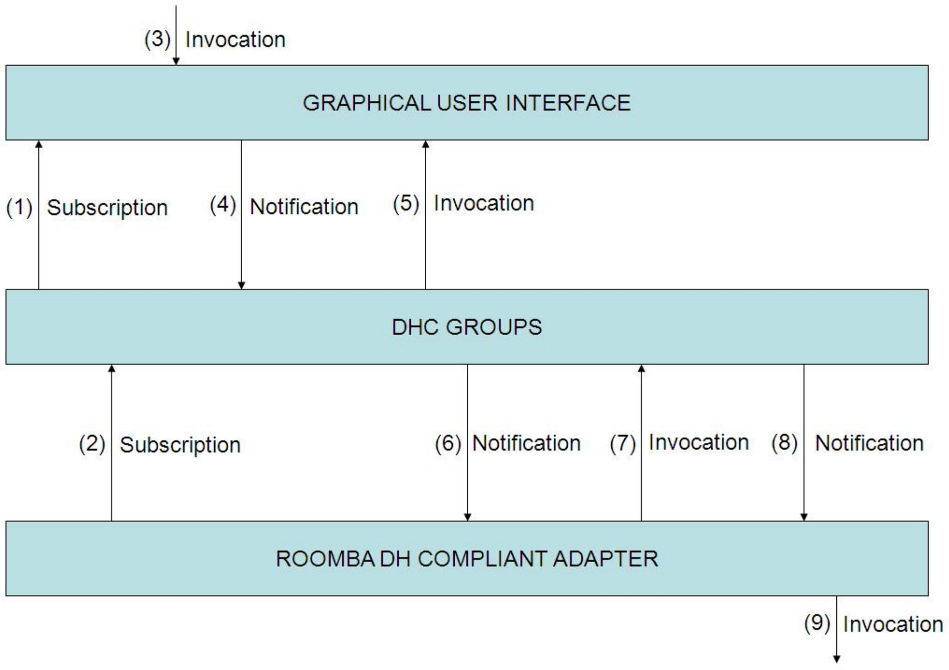


Figure 4.2: DHC-Groups Operation

In Figure 4.2 DHC-Groups operation is summarized:

1. When the DHC-Groups module is launched, it subscribes to the state variable *TaskID*. Obviously, the graphical user interface must be launched as well.

2. When the DHC Roomba adapter is launched it subscribes to the state variables *TaskID*, *CollaborationKeyword* and *Leader*.

3. The user introduces the task parameters using the GUI.

4. When the *TaskID* value changes, the GUI generates an event, so that DHC-Groups is informed.

5. The DHC-Groups module obtains the task parameters and stores them in state variables. The most important variables are *TaskID* and *CollaborationKeyword*, which are evented and hence generate the corresponding events.

6. The aforementioned events inform the robot adapters. In our case, the DHC Roomba adapter compares the *CollaborationKeyword* and the set

of *CollaborationKeyword* associated with the robot. There are as many *CollaborationKeyword* as tasks that can be performed. A robot will have a *CollaborationKeyword* associated with each task it is capable of executing. For example, *spot* and *clean* are *CollaborationKeyword* associated with Roomba or any robot with similar features.

7. If the robot has the corresponding *CollaborationKeyword*, it will propose itself to perform the task and to lead a working hive.

8. The DHC-Groups module receives all the proposals and calculates the working hive and the leader. Then the state variable *Leader* has a new value, and an event is generated. In addition, the robots that will collaborate on the task are notified by several event messages. Finally, the DHC-Groups module subscribes to the leader's *EndTask* event state variable.

9. When the adapter detects the new value of the variable *Leader*, it determines whether the leader ID is equal to the robot ID. When the IDs match, the adapter must send the order to the robot to start the task.

4.3 The Roomba vacuum cleaner: from Roomba UPnP to Roomba DHC

Roomba is a floor-cleaning, disc-shaped robot developed by iRobot. As it was designed with the retail home market in mind, Roomba's hardware is composed of standard hardware components.

In this work, we have developed an application that offers Roomba's services and state according to the DHC protocol. This application has been built from the adapter previously developed in [44], the Roomba UPnP virtual device. This adapter acts as a virtual bridge between the UPnP network and Roomba. The software bridge provides services to the network that will be performed by Roomba. To this end, the virtual adapter receives invocations from control points and it translates them into orders recognized by Roomba via Bluetooth. Once the robot interprets the orders, it executes the corresponding actions.

DHC Roomba adapters are subscribed to all DHC-Groups variables at startup. When the user sends one task through the GUI, the working hive (size of the robot team) and the kind of task to be performed must be specified. Then, this task is received by the DHC-Groups module and it collects all information about the task using invocations and UPnP event notifications messages. The values of the DHC-Groups variables such as *TaskID*, *CollaborationKeyword* or *WorkingHive* change and the DHC Roomba adapters are in turn informed of these changes through event notifications. In this way, the DHC adapters have all the necessary information to perform the task. The adapters receive a notification message with the ID of the selected devices by DHC-Groups in order to perform the task, and they check if their own ID appears in the ID list or not. If it does, the adapter configures the corresponding profiles to perform the task. In addition, they receive another notification in which the task to be performed is indicated (the *CollaborationKeyword* value) and the adapter invokes an action to be executed by the robot, working in similar fashion to the UPnP adapter.

As it can be seen, the DHC adapter operation is based on the exchange of UPnP messages. When DHC-Groups updates its variables, the adapter saves the information, invokes actions to get the information required to perform the task, configures its profiles, takes any necessary action based on information collected and performs the task properly.

4.4 Experiments

In this section we describe briefly how the implemented DHC-Groups module was developed. In addition, experimental results are presented.

4.4.1 Implementation

We used Visual C# 2010 Express [161] integrated development environment to develop the adapter. Each of the three layers of Figure 4.2 has been implemented as a DHC entity relying on UPnP technology. Thus, we have a GUI device, a DHC device (which includes the DHC-Groups service) and a Roomba adapter device. To create and develop these devices, we used the Developer Tools for UPnP Technology [160]. This is a set of tools which are used in the design, creation and management of devices with UPnP features. Two of them, the Service Author and the Device Builder are used in the DHC project.

Since DHC works over UPnP, DHC entities are essentially UPnP devices with some advanced capabilities. These capabilities have to do with the mechanisms that allow several entities to work in a collaborative manner. In addition, in DHC devices a dual behaviour is introduced, which means they can act as UPnP device or they can have control point features according to the situation. The development of DHC entities is based on the steps given to implement UPnP devices:

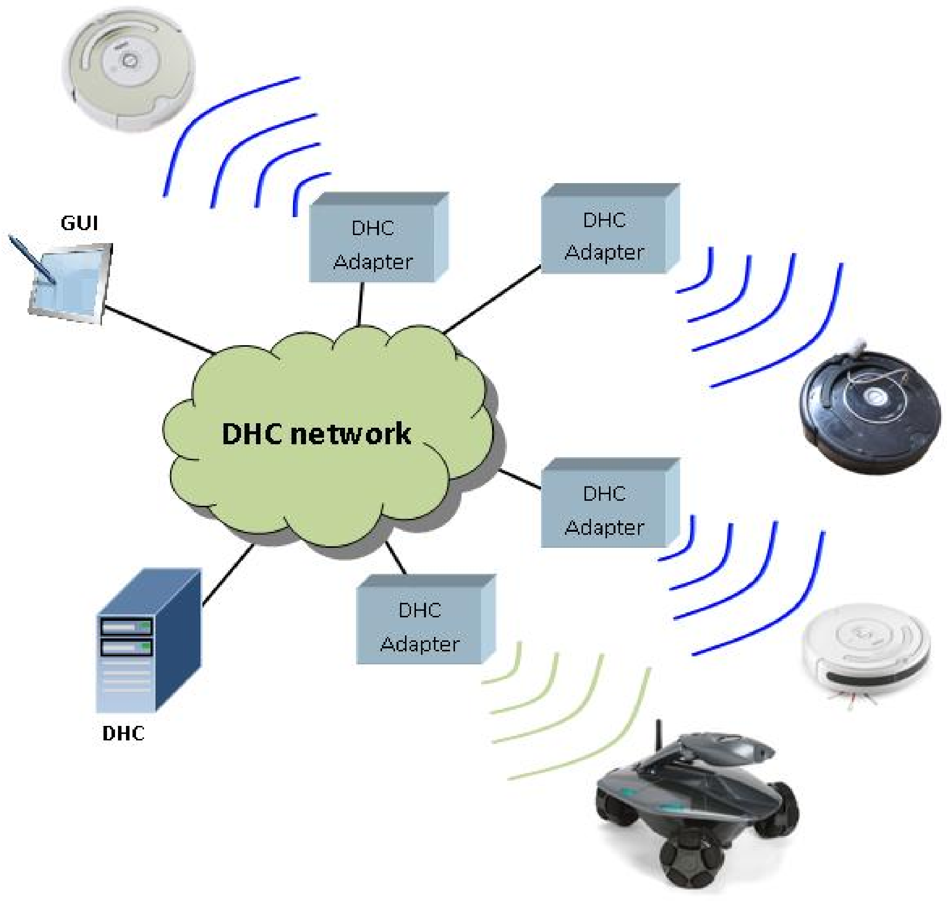


Figure 4.3: DH Compliant Network

1. In the first step the XML service description file is created. This file contains the actions included in each service provided by the device. The description file is generated and implemented with the Service Author tool. It allows the developer to define the variables and the actions of the service. Every device must have at least one service description file in order to provide some information about the actions that the device offers to the network.

2. Once all the services included in the device are defined by means of their description files, we used the Device Builder to define the device. This tool allows for specification of the device and properties of services and generates the associated UPnP stack in the .NET framework.

3. In the third step the code related to the service actions must be developed in C# in order to implement the real functionality of the device. The developer must decide the technical complexity of the algorithm to implement each action. A dual behaviour must be implemented in the DHC entity, meaning that it can work either as a UPnP control point invoking actions and subscribing to events or as a UPnP device offering services and executing actions. It can notify of events as well.

4. Finally, it is necessary to check that the designed DHC entity works properly. In case that it does not, it may be necessary to repeat and debug one of the previous steps.

4.4.2 Experimental results

In this subsection we present results from a real experiment in order show the DHC-Groups module in action (see [100] for videos of the DHC). In this experiment, the user is provided with a touchscreen that presents the graphical user interface (GUI) with UPnP features. This is the top layer of the scheme illustrated in Figure 4.2. The user must specify the task that the robot team will accomplish. To do this, the user must complete a form where the task parameters are indicated. The most important parameters are the following:

- A task identifier.
- Identifiers for the floor and the room where the task will be performed.
- A collaboration keyword which represents the task type to be accomplished.
- The number of robots in the working hive.

- The time when the task should begin.

In this case of use we design a task for a team composed of two Roombas. They are required to perform a cleaning task which is specified in the collaboration keyword value. Once the form is filled, the user can launch the task. The information about the task is sent through the DHC network and collected by the DHC-Groups module, which is subscribed to the GUI variables. This module processes the information and establishes a communication by means of events and subscriptions with the DHC Roomba adapters. When these adapters receive the task, they send a request for the coordinates of the room where they are, and how to reach the point where they should go to accomplish the task. An external localization system is used to exchange this information with the robot team using the DHC protocol and to guide the hive to the desired coordinates. In particular, the localization system consists on a zenithal high definition camera and a PC with the OpenCV library that provides the coordinates and orientation of the robots (see [173, 181] for more information). Although it is beyond the scope of this work, the DHC Localization module works as any other DHC entity: it provides an UPnP service that can be invoked whenever a robot needs to know its current coordinates and orientation. This service is connected to the localization system and sends the position information back to the adapter as the return value of the invocation received. Finally, when the robot hive reaches its destination, they start the cleaning task. This process is shown in Figure 4.4 in which the following steps are distinguished:

(1) The GUI allows the user to create a new task and send the task parameters to the DHC module.

(2) The robot adapters are subscribed to the DHC-Groups module state variables. As the task received from the GUI changes the values of the evented DHC state variables, the DHC adapters are notified.

(3) The adapters analyze the evented variables and check the kind of task to be performed. If the robot has enough capabilities to accomplish the task, the adapter will propose the robot's candidancy to perform the task by invoking a DHC Groups module action.

(4) Once the DHC-Groups module has all the candidancies, it will choose the robots that will accomplish the task and the leader of the alliance by updating its state variables. At the end of this step the DHC-Groups module subscribes to the leader's *EndTask* state variable.

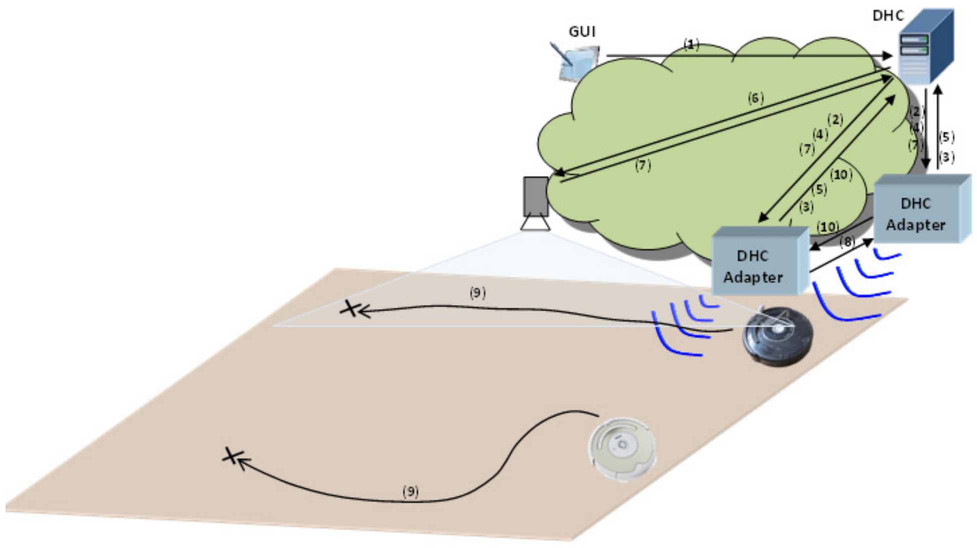


Figure 4.4: Roombas scenario.

(5) Once the leader adapter has divided the cleaning place and has calculated the coordinates for each subtask, it invokes the *CleanAtPoint* action of every robot in the hive. This action performs a cleaning task at a certain coordinates. Then, the leader subscribes to the *EndTask* state variable of every robot.

(6) The adapters involved in the task invoke the DHC-Groups module to obtain their robot's current location.

(7) The DHC-Groups module requests the robots' position from a DHC localization system. In this particular case, a zenital camera has been used for locating the robots.

(8) The DHC localization system responds with the current position of each robot. This information is forwarded to each adapter.

(9) Each adapter guides its robot to the destination point provided with the *CleanAtPoint* action in order to start the subtask. This navigation is performed by the adapters, which can update the robot's position by requesting it from the DHC-Localization module.

(10) When the robots accomplish their subtasks, they change the value of their *EndTask* variables to send the corresponding event to the leader.

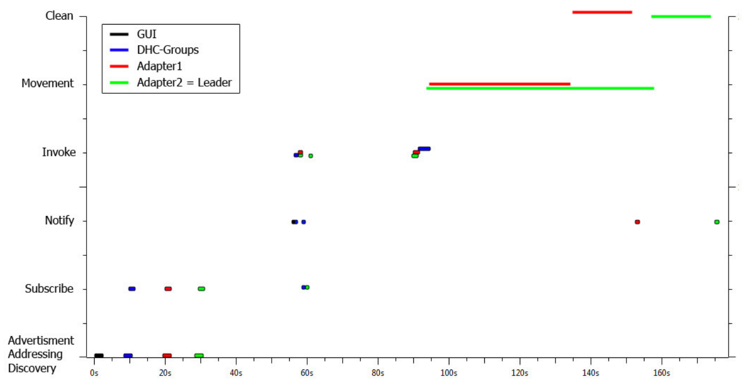


Figure 4.5: A cooperative task over the DHC Compliant protocol.

Once the leader accomplishes its own subtask and has received the *EndTask* event from every robot, it sends a similar event to the DHC-Groups module so that it knows that the task has concluded.

In Figure 4.5 the behaviour of the DHC Groups entities over time at our scenario can be seen. The sequence of states is: (1) The GUI is launched; (2) the DHC-Groups module starts and it subscribes to the GUI state variables; (3) the DHC-Adapter1 starts and it subscribes to the DHC-Groups state variables; (4) the DHC-Adapter2 starts and it subscribes to the DHC-Groups state variables; (5) the task is launched from the GUI and it notifies by changing its state variable; (6) DHC-Groups notifies the task to the adapters by means of its state variable and it invokes a GUI action to ask for the task information; (7) the adapters present their candidacy by invoking the Groups action; (8) DHC-Groups notifies the leader (DHC-Adapter2 for our particular case) and the task participants (DHC-Adapter1) and it subscribes to the leader *EndTask* state variable; (9) the leader invokes the *CleanAtPoint* action of the robots in the hive and it subscribes to the corresponding *EndTask* state variable; (10) the DHC-Adapter1 and the leader invoke actions to get their coordinates from the DHC-Localization through the DHC-Groups; (11) the DHC-Adapter1 finishes it subtask and it notifies to the leader by changing its *EndTask* state variable; (12) the leader finishes its subtask and it notifies to the DHC-Groups by changing its *EndTask* state variable.

Finally, we examine the communicational burden of DHC-Groups module, which is shown in Figure 4.6 for the aforementioned experiment. During the three minutes of the experiment, 169 messages were sent, with an average size of 322 bytes. The average bandwidth used by DHC was 305 bytes/s and the total amount of bytes sent in the scenario was 54395. Notice that the largest part of this amount was sent during the launch of each entity. It is

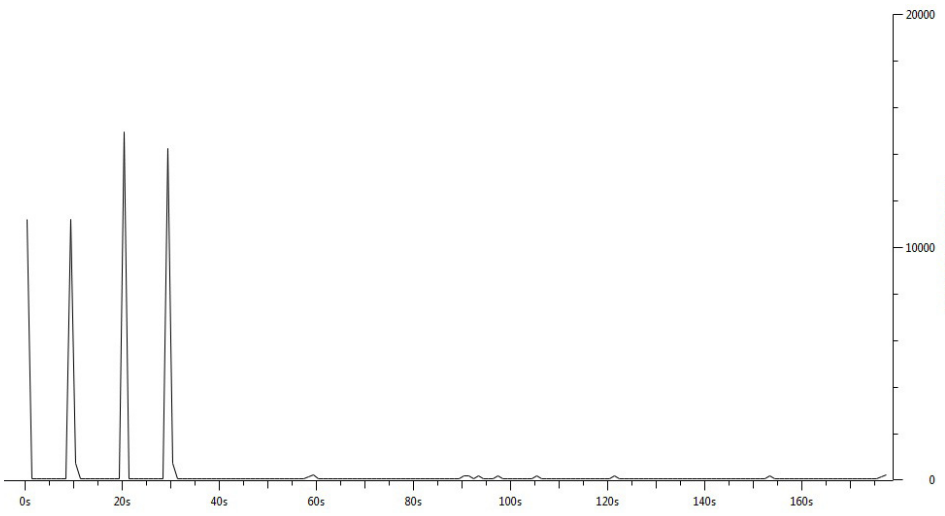


Figure 4.6: A cooperative task over the DHC Compliant protocol.

important to remark that these amounts are low when compared with the bandwidth available on modern home networks, which shows that the DHC architecture is suitable for current smart homes.

According to the results that have been obtained from our tests we can conclude that DHC-Groups module works properly and allows for the scheduling and execution of several tasks performed by a robot team. Additional performance measurements for the Rovio adapter can be found in [182].

4.5 Conclusion

In this chapter an implementation of the DHC protocol in the service robot Roomba is presented. One of the DHC modules (DHC-Groups) is deeply detailed and is implemented on this robot. DHC is based on UPnP technology and has a three-tier stack. Devices meeting DHC requirements are implemented in specific adapters that act as a bridge between DHC and the robot proprietary technology. In the case of Roomba, its adapter translates orders received via UPnP into orders that are recognized by the robot and are transmitted using Bluetooth technology.

As it has been seen, DHC relies on UPnP in order to provide additional M2M mechanisms to make different types of devices work together (as seen in Figure 4.3). The standard is composed of different modules which implement key features, such as collaborative tasks, energy management, localization, security and privacy. The chapter deals with the operation of the DHC-Groups module, which manages the collaboration between different types of robots to perform tasks. The paticular adapter presented in this chapter, DHC Roomba, allows users to clean a house with a set of cooperating Roombas.

As a consequence of this first project a new alliance emerged. DH Compliant 2 [183] continued the goal proposed by the previous consortium to develop a universal protocol for communicating service robots, smart appliances and digital homes using Web Services.

Chapter 5

Cloud Robotics

This chapter is an adapted copy of a paper sent to Sensors journal, which is an open access journal published by MDPI, with the title “Off the shelf cloud robotics for the smart home: empowering a wireless robot through cloud computing” [184] for the special issue Sensors for Home Automation and Security.

Nowadays service robots are gradually being introduced at homes. For example, autonomous vacuum cleaners such as Roomba or Navibot have become common devices in many homes. Other popular robots are Rovio, which provides users with telepresence services, and the quadrotor ARDrone, which has become a successful domestic recreational device. All of them have contributed to provide a more realistic view of the current capabilities of service robots, still very far away from the expectations generated by science-fiction and futuristic movies.

Besides the proliferation of service robots at homes, there are other remarkable household technologies that are being adopted to improve people’s lives and to optimize energy consumption. This is the case of smart home technologies, which provide users with services regarding automation, security, or multimedia management. Unfortunately, the lack of standardization of the smart home market is a major issue that hinders its development [185, 186]: there are many different technologies that offer smart home services (e.g., the old X-10, Lonworks, KNX, Z-wave,. . .), but none of them has become a true reference for marketeers and developers. For this reason, different middleware platforms have been proposed to integrate heterogeneous technologies in the same network, e.g.: OSGi [33], UPnP [45, 37, 41, 187], or Jini [20], to name a few.

In this chapter, we deal with the integration of service robots in the smart home. In the literature, different works that deal with this interesting problem have been reported. In [44, 122], the service robots Roomba and Rovio were integrated in a smart home network by means of UPnP adapters. A more general approach is provided by DH-Compliant [188, 173], which is an architecture tailored for the integration of home automation systems and service robots. Other alternatives are: Miro [189], which is an object-oriented robot middleware that makes the development of mobile robot applications easier and faster, and promotes the portability and maintainability of robot software; ROS (Robot Operating System) [107], which is an open source framework to support modular, tools-based software development; OROCOS (Open Robot Control Software) [108, 190, 191], which is an open distributed framework for robot control, consisting of object libraries and components, and a standard middleware; and RoboMaidHome [192], which consists of a wireless sensor networks for service robots to provide reliable services.

The integration of service robots and smart home systems allows providing new services. However, the development of such services may be limited by the computational power available at home. In particular, the hardware used by robots and smart home systems typically has little computation capabilities. For this reason, we rely here on cloud computing whenever heavy computations must be performed. In order to illustrate the approach, we present a case study based on the following off-the-shelf components: a Rovio robot and a Vera controller, which is a controller for Z-Wave networks with low computing capabilities. In particular, we have integrated Rovio in a Z-Wave home automation network governed by Vera. The integration of this robot allows us to offer new services based for example on image processing and voice recognition. Given that these services are computationally expensive, and taking into account that there may not be devices capable of performing the corresponding computations in the local network, the solution we propose is to use the cloud. This idea and the whole scenario can be seen in Figure 5.1.

The rest of the chapter is organized as follows. In Section 5.1, basic concepts of cloud computing and cloud robotics are introduced. Section 5.2 deals with the integration of the surveillance robot Rovio in a Z-Wave network. Section 5.3 details the cloud computing implementation developed. Finally, Section 5.4 shows the results and the last section provides conclusions and future work.

5.1 From Cloud Computing to Cloud Robotics

Cloud computing has become a very popular topic in the last few years, but the concept is not new; many *old* services for e-mailing or file sharing are

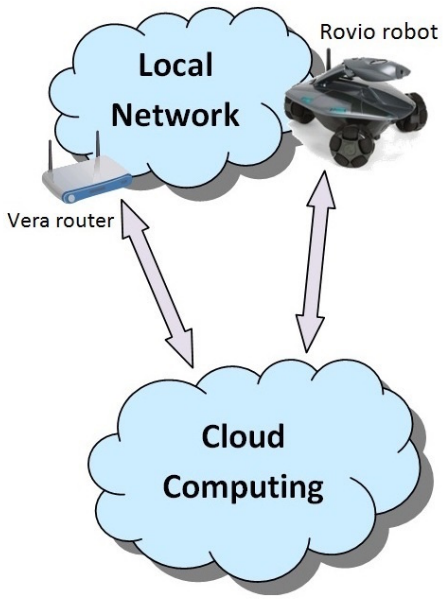


Figure 5.1: Cloud robotics scenario.

based on the same principle. The main idea is to provide remote computational resources as services invoked through a network. In this way, it is possible to provide low-resource devices with the access to a massive amount of data and computation power, including distributed and parallel processing. In general, the advantages of cloud computing are [193, 194]:

- Ease of data exchange with external servers.
- Flexible configuration.
- Middleware: cloud computing servers can provide a virtualized platform for each component in the network to collaborate with each other.
- Heavy processes can be handled by a server, so vendors can manufacture smaller and less expensive and less power consumption components.
- High scalability.
- Maintaining and updating the software and drivers is simpler.
- Services and resources are located in the cloud, not on a particular address.

In [195], the extensive resources of Cloud infrastructure are studied. It provides a survey of research on cloud robotics and automation based on four of its features: Big Data, Cloud Computing, Collective Robot Learning and Human Computation.

The robotics community has also been attracted by these advantages, and nowadays robots are being prepared to connect to cloud computing infrastructures. This approach is called cloud robotics, which is a particular case of cloud computing that deals only with robots [196]. Cloud robotics allows to allocate resources in the cloud dynamically in order to support task offloading and information sharing in robotic applications [197]. Moreover, cloud robotics are expected to enhance robots capabilities such as speech recognition or mapping. It can provide benefits in different applications, e.g.: SLAM, navigation, voice recognition, weather monitoring, intrusion detection, surveillance, language translation or communication. In this way, cloud robotics enables robots to use remote servers to perform the heavier processing, so robots could be designed smaller, smarter and cheaper [198]. However, there are some important drawbacks that should be considered [199]:

- Robots need a permanent connection to keep full capabilities.
- It is necessary to protect the network to preserve the data privacy and to avoid robots hacking.
- It is recommended to use on-board processing for strict real-time tasks such as those related with motion control or path planning.

5.2 Integration of Rovio in a home network through Vera

In this section we introduce briefly the main elements that compose our case study – surveillance robot Rovio and home automation controller for Z-Wave Vera – and how they are integrated in the same network.

5.2.1 Rovio

Rovio is a surveillance robot that has WiFi connectivity and provides an HTTP interface to control its capabilities. Rovio, that was released in 2008, has a set of sensors that includes a head-mounted moveable VGA camera, a microphone for 2-way audio, infrared sensors for positioning and collision detection and a battery charge monitor. On the other hand, Rovio has the following actuators: a position controller for the robot neck, a speaker for 2-way audio, 3 omni-directional wheels, and LED lights for illumination. There exists an API based on the HTTP protocol over TCP/IP that allows users to send commands to Rovio and to receive data from it. In particular, Rovio works by means of common gateway interface (CGI) commands within HTTP GET requests. For example, in order to retrieve the status report from Rovio it is necessary to send the following command:

```
http://ip_address/rev.cgi?Cmd=nav&action=1
```

where ip_address is the IP address of Rovio in the network.

Finally, we must remark one of the main features of Rovio: it can navigate through the home thanks to the use of infrared lights projected on the ceiling from its base. In order to extend the number of rooms that a Rovio can navigate autonomously, additional beacons known as Rovio Truetrack Room Beacon can be used. Each beacon must be set to a unique ID (1-9) using a selector that identifies each room. It must be remarked that the production of Rovio has been discontinued, although there are other robots with similar and higher capabilities available in the market.

5.2.2 Vera

Vera is a controller for Z-wave, which is a wireless home automation technology based on IEEE 802.15.4. The role played by Vera is to act as a centralized controller in the network that offers the following services:

- It provides a platform for the creation of macros and scenarios in the home automation network.
- It works as a bridge between the Z-wave and Wifi, allowing the user to command his home automation devices via Wifi.
- It works as any common router: it provides access to the local home network and to the Internet. In this way a user can control the smart home network from the Internet. In addition, this also provides a means to connect to an external cloud computing service.
- It provides a native UPnP interface, which allows integrating UPnP compliant devices in the smart home network.

5.2.3 Creation of a basic Rovio interface for Vera

As it has been stated previously, one of the contributions of the chapter is the integration of a service robot in a Z-wave network[1], which has been updated

[1]A basic, and earlier, version of our Vera's Rovio adapter for UI5 can be downloaded freely from [200].

to the UI7 version for Vera. More specifically, our first goal is to be able to invoke the basic robot capabilities from Vera so that it can be used as any other Z-wave device. In this way the robot can be integrated in scenes or events that can be triggered from Vera depending on user actions, scheduling, or sensor information. For example, Rovio could be programmed to go to a certain location to get a picture as a response to a change in the state of a sensor of the smart home network.

Vera uses an UPnP internal model for each device integrated in the smart home network, even if the device itself is not UPnP compliant. Hence, if the device does not have its own UPnP interface, it is necessary to create one. To this end, three different types of files have to be created[2]: 1) a device description file, 2) service description files, and 3) implementation files. The first two types of files are UPnP compliant description files while the third type contains the implementation of the services in LUUP (Lua UPnP) or in JavaScript, which are the programming languages used by Vera.

For the Rovio plugin the files uploaded are:

- The device description file that contains the top level information about Rovio's plugin and it is linked to the service files.
- The services description files which list the services (actions and variables) that the device will provide.
- The implementation files that execute the actions to provide the services.
- An interface file in JSON format that allows the user to control the device using a graphical interface.

Once these files have been created and uploaded to Vera, Rovio is integrated in the Z-wave network.

In Figure 5.2, it can be seen the graphical user interface offered by Vera. On the top left of the interface there are some buttons available to invoke the following Rovio movements: forward, backward, left, right (and their respective diagonals), the clockwise and counterclockwise rotation; and a button to send Rovio to its dock station. There is also a speed field available that users can modify to change Rovio's velocity when it is controlled by using the aforementioned buttons. On the top right of the interface, the image captured by

[2]There is an optional fourth type of file to provide a graphical user interface for the device in Vera. In this case, JavaScript Object Notation (JSON) is used to describe the appearance of the interface and how it interacts with the UPnP actions and variables defined in the other files. The details of this fourth file are omitted here since they are irrelevant for the scope of the chapter.

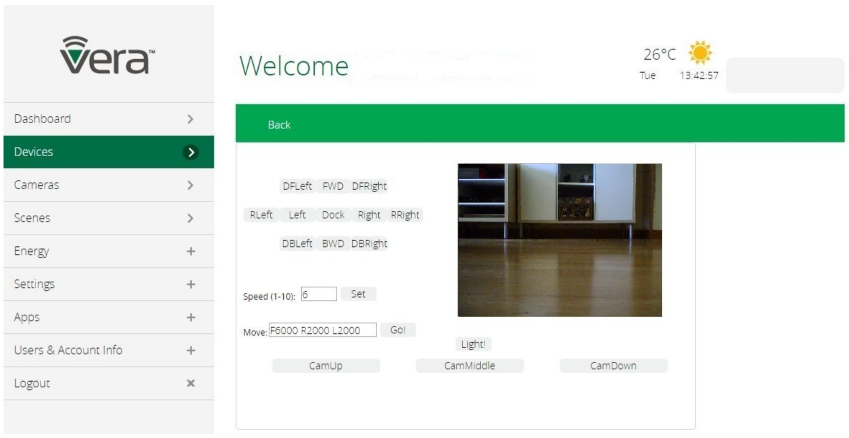


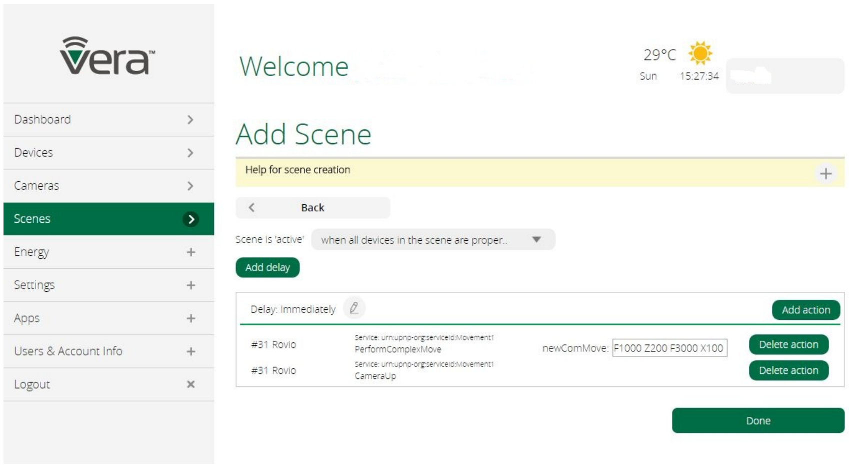
Figure 5.2: Vera Control Graphical Interface for Rovio.

Rovio's webcam is shown. In addition, there is a button that turns on and off the LEDs installed on Rovio for low light environments. Another interesting functionality is the possibility to adjust Rovio's cam position by means of three buttons (one for each position) on the interface. Finally, there is a field where a string that represents a function of complex movements, can be inserted. This string contains several elements, where each element consists on a letter that indicates the movement direction and a number that represents the movement duration (in milliseconds). The movement directions implemented are: F (forward), B (backward), R (right), L (left), Z (counterclockwise rotation), X (clockwise rotation). For example, if the string is F5000 R2000 B1600 L2000, Rovio goes forward for 5 seconds, it turns right for 2 seconds, then it goes backward for 1.6 seconds, and, finally, it turns left for 2 seconds.

The actions provided by Rovio are also available for macros and scenes, as shown in Figure 5.3. In this figure, it is possible to see how to program a movement string using the *PerformComplexMove* function implemented in the plugin. Likewise, these actions can be seen and invoked from any UPnP control point in the smart home network due to Vera's native UPnP functionality.

The interaction with the device can also be accomplished by simple HTTP GET requests to the Vera IP address router with the following format:

```
    http://ip_address:3480/data_request?id=device&action=forward&
device=31
```



Scene is 'active' when all devices in the scene are proper..

Delay: Immediately

#31 Rovio Service: urn:upnp-org:serviceId:Movement1 PerformComplexMove newComMove: F1000 Z200 F3000 X100

#31 Rovio Service: urn:upnp-org:serviceId:Movement1 CameraUp

Figure 5.3: Scene section of Vera for Rovio device.

5.3 Cloud capabilities

The entities that take part in the cloud platform are listed below:

- **User:** It is the actor that launches the Rovio service and sets the credentials through a GUI browser. It also interacts with Rovio robot by commands voice.
- **Vera router:** It receives HTTP requests to execute actions and sends HTTP requests to Rovio Web Service using Rovio's plugin installed.
- **GUI Browser:** It enables the user to interact with Vera router by sending HTTP requests.
- **Rovio Web Service:** It receives requests from Vera router and assigns a Rovio application in the Cloud to the home to resent the requests.
- **Rovio Application:** This is the main entity in the Cloud. It receives the requests originated in Vera, performs advanced and the heaviest tasks and sends the corresponding orders to Rovio or Vera according to the individual case.
- **Google Cloud Speech API [201]:** It is an external Cloud service that enables the Rovio application to convert audio into text.
- **Rovio robot:** This robot receives the orders from Rovio application and it performs tasks, i.e. it moves to an specific location or streams audio.

Once a basic integration of Rovio and Vera is attained, we can focus on the development of advanced capabilities based on voice recognition. Given that Vera's computation power is too low for these tasks, cloud computing is used to overcome this issue.

5.3.1 Rovio, home and cloud interaction

In order to inter-operate with the cloud, an extension of the original Rovio's plugin [200] has been developed, so the new version of the plugin requests services through Vera by sending a label that identifies the operation and Rovio's IP address. In this way, the remote cloud computing server has a means to exchange information with both Rovio and Vera.

The scenario consists on a Rovio robot and a router Vera with Rovio's plugin, which are available locally in the home. On the other side, a cloud server has been deployed with a Web Service that accepts HTTP requests for the following services:

- Login Service: it allows the user to get access to the rest of the services available. It requires the cloud server IP address and the Web Service URL to send the request, which are saved in a variable of the plugin for subsequent requests; and the service username and password, which are sent to the service.

- Rovio Connect Service: it allows the connection with Rovio. It requires Rovio's IP address, username and password, which should be previously set in Rovio's plugin as variables for Vera. Once this service is invoked, a Rovio application is launched in the cloud server with the information provided and the connection is established. The Web Service keeps a table with pairs of IP addresses from clients and Rovio application ports in order to manage different clients and future requests.

- Launch Voice Recognition Service: it activates the voice recognition feature.

- Stop Voice Recognition Service: it deactivates the voice recognition feature in Rovio application.

- Garbage Detection Service: it allows to enable/disable a garbage detection algorithm. This service has been created to show the capabilities of a cloud application to create a very complex algorithm to detect garbage and use this information in Vera, e.g., to send another robot to clean the area.

Once all the elements are available, the user should send a login request to the Rovio Web Service in the cloud through the Vera interface. When the Rovio Web Service receives a login request, it launches a Rovio application to communicate with each service client using an specific port. Then, a connection request should be sent to connect the cloud application with the robot. The Rovio Web Service identifies the request client and sends a request to the Rovio application using the corresponding port to connect to the robot. Finally, in order to process command voices, a request to activate the voice recognition service should be sent to the cloud server. Vera sends a request to Rovio's Web Service and it requests Rovio application to collect audio streaming from the robot. In this way, the remote Rovio cloud application collects voice data from home and processes it by using the Google Cloud Speech API, which requires sending requests to an external cloud service. Once the command is analyzed and recognized, the cloud application sends the coresponding command to Rovio or Vera to execute the action according to the command voice.

The operation sequence is detailed below:

- The user logs in Vera and accesses Rovio.
- The user sends the login HTTP request to the Web Service in the cloud through Rovio's plugin in Vera. It authenticates the user in the cloud to get access to the rest of the services.
- The user sends the HTTP request connect to the Web Service with Rovio's IP address, the user's name and the password. The service launches the cloud application locally with an associated port and stores the pair port - Vera IP address to identify a home with its corresponding application. Then, the service sends a request to the Rovio application with the provided parameters to establish a connection between Rovio's application in the cloud and the robot at home.
- The user can send the voice activation HTTP request and the service acts as a bridge and resends the request to the Rovio application in the cloud, which starts to collect audio data streams from the Rovio's microphone. The application records the audio periodically during two seconds and processes it by using the Google Cloud Speech API, which converts audio into text by applying powerful neural networks. The text retrieved from audio is analyzed to check if it matches the commands loaded (there are four basic commands that identify four operations and that are loaded at the beginning). When one of these operations is recognized, a new grammar is loaded to specify the information for the

specific operation. The voice commands are configurable and included in an xml file whose fields and meanings are described below:

- Move: when the command voice identified by this label is recognized, the Rovio application loads a new grammar which is composed by the paths saved in the robot and by the two default paths stored on it, dock base and home, which are identified, in the same order, by the *Place* labels inside the GoTo section. The user says the voice command defining the *Position* label and then the command that sets the path. Rovio, using its tracking beacons, will move to the position defined by the path.
- Detect: once this voice command is recognized, the Rovio application loads the new grammar with the name of the objects identified by the *Object* labels inside the *ObjectDetection* section. The name of the object should be previously defined in the application and a jpg image should be uploaded in order to look for the object. This application uses a SURF algorithm.
- Action: when the Rovio application recognizes such voice command, it loads a grammar with the scenes defined in Vera. To this end, the Rovio application sends a request to Vera to retrieve the scenes, and then, the application awaits for the voice command that identifies the scene. When the application detects this command voice, it sends a request to Vera to execute the scene.
- Cancel: when Rovio the application recognizes any voice command, it also adds the *Cancel* option to the grammar loaded. Then, when this voice command is recognized, the initial grammar is reloaded.

The working sequence described above is illustrated in Figure 5.4. In this figure, the messages exchanged by the equipments at home, the Rovio application in the cloud, and the Google cloud are represented as a sequence diagram.

5.3.2 User interaction

Regarding the user, he interacts with Rovio transparently using the local plugin installed in Vera.

We have implemented a voice recognition service that allows the use of voice commands with Rovio. The speech recognition service relies on a list of possible voice commands that are stored in the cloud server in a XML file. The orders we have considered are:

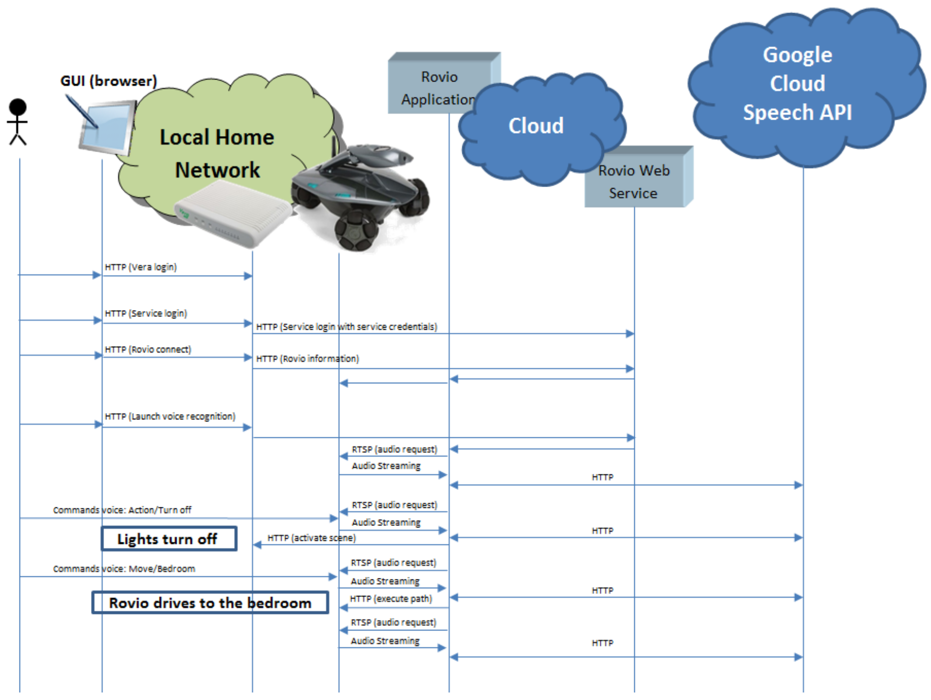


Figure 5.4: Messages exchange for Rovio and Vera router integration.

- Follow a default path or one of the recorded paths available in Rovio.
- Begin the search of an object using the image recognition capability developed.
- Execute one of the scenes previously defined in Vera to trigger events in the house.
- Cancel a certain action.

The voice recognition service can be activated through the plugin. Once the user activates it, the plugin on Vera sends the corresponding message to the cloud server so the remote application can retrieve the information provided by Rovio's microphone. There are important applications behind this service:

- It is possible to use voice commands to trigger events in the house as long as the user has Rovio available in a room, identified with the corresponding Truetrack beacon.
- The cloud server application for Rovio retrieves the scenes defined in Vera by sending a request, therefore the user can simply say a scene's name to execute it.
- The cloud server application commands Vera to perform the corresponding action.
- Another example is the activation of the object search algorithm by means of voice recognition: the user says the corresponding voice command and the name of the object to be found, and then Rovio starts to look for the object.

5.4 Experimental Results

We have carried out some experiments to test the cloud robotics implementation.

The reader should take into account that the experiments have been performed in Spanish but English subtitles have been inserted to clarify the scenarios in the videos, where experiments can be seen `https://www.youtube.com/watch?v=Z-jwaY_pUjA&feature=youtu.be`, `https://www.youtube.com/watch?v=ZEcCFR_bsOs&feature=youtu.be` and `https://www.youtube.com/watch?v=rUDAxxSrGDc&feature=youtu.be`.

For simplicity, we have used a local computer to act as the cloud server. The deployment would be similar to a remote one. Moreover, the deployment of the Web Service and the Rovio Cloud Applications could be distributed among a remote server farm. In addition, there is another cloud element that takes part in the experiments, the Google Cloud Speech API, which receives request to identify the commands voice and sends the results to the Rovio Cloud Application.

5.4.1 Teddy bear search experiment

The first experiment consists on requesting the robot to look for a teddy bear, [202]. Using the Vera interface, the user sends a login request to Rovio Web Service, which launches a Rovio application. Then, a connection request is sent, and finally the voice recognition algorithm is launched. When the user says the *detect* command (Spanish command: *detectar*), the Rovio application loads the objects grammar available and then, once the user says the *teddy* command (Spanish command: *peluche*), it guides the robot and launches the SURF algorithm to look for the indicated object.

5.4.2 New object for search service experiment

The video in [203] shows how easy is to insert a new object in the application. First, the user includes the object *candle* (*vela* in Spanish) in the xml file. Once the user says the command *candle*, the Rovio application ignore it and prints the message "The pattern image of the object to look for does not exist" (in Spanish language: "La imagen patron del objeto a buscar no existe"), which indicates that the object pattern to compare with does not exist. In order to identify the object, the user takes a snapshot of the candle and saves it. Finally, the user tries again with the candle command voice, and this time, as it is shown in figure 5.5, the Rovio application launches the SURF algorithm and finds the object successfully.

5.4.3 Z-wave + Rovio integration experiment

Finally, in [204] an integration scenario between Rovio and the smart home has been carried out. In this scenario, Rovio receives voice commands and performs the corresponding tasks by executing the scenes previously defined in Vera. When the Rovio application receives the *action* command (Spanish command: *accion*), it sends a request to Vera indicating the room where Rovio is located to perform the action. Firstly, the user says the *action* command (Spanish command: *accion*), and then the *turn off* command (Spanish command: *apagar*), and the Rovio application sends a request to

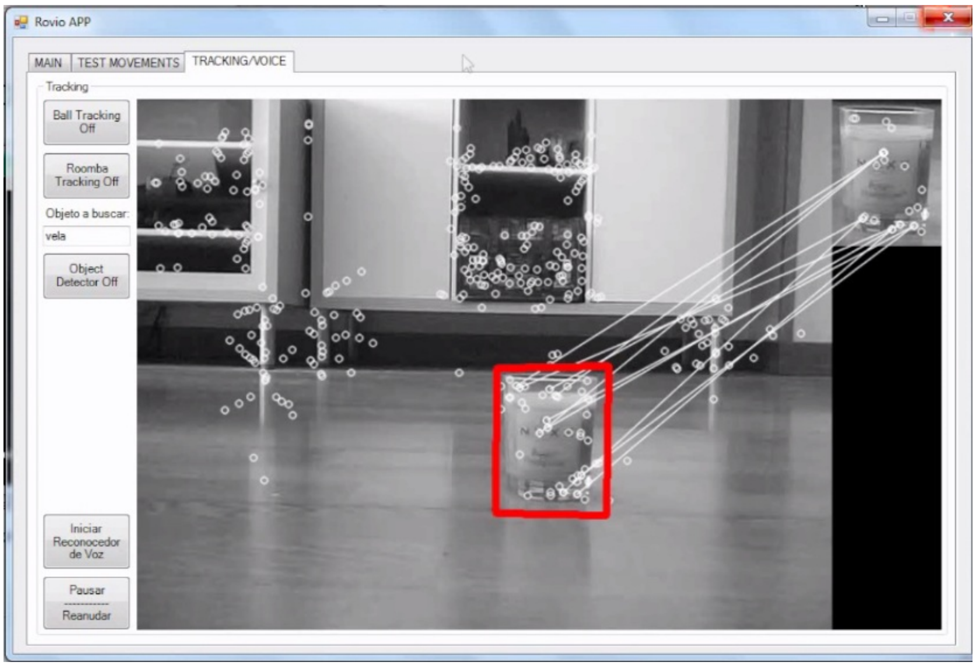


Figure 5.5: Rovio finds candle object.

Vera to turn off the lights in the room where Rovio is located (the bedroom). Next, the user orders Rovio to go to the living room by saying the commands voice *move* and *living room* (Spanish commands: *mover* and *salón*). Once the robot arrives to the living room, the user says the commands *action* and *turn on* (Spanish commands: *accion* and *encender*), and the Rovio application sends a request to Vera to turn on the lights in the room where Rovio is located (the living room). Finally, the user turn off the lights in the living room with the voice commands *action* and *turn off* (Spanish commands: *accion* and *apagar*), which send a request to Vera to perform the task. Note that the *turn off* command executes on Vera two different scenes according to the room where the Rovio is located. When Rovio was in the bedroom, the *turn off* command triggered the scene for the bedroom, and on the other hand, when Rovio was in the living room, the *turn off* command triggered the scene for the living room.

5.5 Conclusion

This chapter shows a practical case study of how new advanced services can emerge in the smart home by using cloud computing to overcome the limitations of local computational resources. In particular, a Rovio robot has allowed us to offer services such as object detection and voice recognition. This is a remarkable achievement because it allows using a low cost off-the-shelf robot to provide services that are only possible with much more sophisticated and expensive robots, e.g., Zenbo. Taking into account that we have developed an interface to integrate the service robot Rovio in a Z-wave network, which is also a very economic home automation technology, it can be concluded that users can enjoy the benefits of advanced cloud robotics at home inexpensively.

The integration of cloud robotics into smart homes may reduce the costs in robots manufacturing. This fact together with the proliferation of cloud services and its advantages could promote a new vision for the home automation systems.

Therefore, the rationale for cloud robotics consists of making the operating logic as independent from the hardware as possible, or at least the heaviest tasks.

The following step could be to integrate a smart home with robots completely in the cloud with a central orchestrator entity which manages the communications among the cloud and the systems at home. The smart home will have a single entry point to and from the cloud to orchestrate all the communications. The orchestrator could send messages to the cloud requesting services for different elements in the smart home, and the responses will be resend to the corresponding smart home units in order to enhance their capabilities.

Chapter 6

Conclusions

In this thesis we have proposed solutions to integrate different and heterogeneous systems and robots in the smart home. As it has been seen, interoperability is a very important issue in smart homes since the lack of it is the major obstacle that hinders the penetration of smart home technologies in the mass market. It is true that the media market has progressed a lot in the last few years, however, the continuous appearance of new protocols to come up with a solution to share and stream content from one device to another is still booming. Once again, the interoperability problem prevent media devices to interoperate between them in any place, and in any situation.

The problems derived from the lack of interoperability have been studied during several decades from different perspectives and with different applications in mind, but the basic problem is always the same: a machine-to-machine (M2M) communication mechanism has to be defined to ease the implementation of services that cannot rely solely on ad-hoc implementation.

It is worthwhile to stand out that in this thesis a great effort has been made to integrate different and heterogeneous systems, such as robots and smart home controllers. Likewise, we have developed a protocol to interoperate transparently with every device at home. Such protocol uses an existing and popular standard in order to keep backward compatibility. In general, previous interoperability solutions have been more focused on improving the capabilities instead. This implies that many systems were not compatible with the rest of the systems that there were available in the market.

One of our most important objectives during this work has been to reach a general procedure to integrate different devices, and robots in particular, in the smart home. Every work presented in this thesis has been developed around the UPnP protocol which is a consolidated smart home technology.

In this work it has been demonstrated that any device is susceptible to be integrated in a smart home through the UPnP standard. Additionally, a novel protocol which enhance the UPnP capabilities has been developed. This protocol is backward compatible and works properly with any native UPnP device. However, it proposes UPnP entities to work with both behaviours, control point and device (master and slave), enabling the possibility to exchange information between entities and to work cooperatively. On the contrary, it is necessary to develop a more complex algorithmic.

Finally, it is important to stand out that all the systems and techniques developed throughout this work have been tested, at least, in controlled scenarios.

6.1 Conclusions

We present next the main contributions of each of the chapters of this thesis:

- **Interoperability Systems**. In chapter 2 we have addressed the problem of the lack of interoperability in general, and in the smart home in particular. In addition, a literature review of previous systems has been done from the interoperability point of view, that is, we have detailed a very large list of interoperability proposals. Special attention has been paid to the UPnP system which is the base of this thesis.

- **Integration of Service Robots in the Smart Home by means of the Universal Plug and Play Protocol**. In chapter 3 we focused on the integration of service robots in the smart home through the UPnP protocol. In particular, we have proposed a methodolgy to create UPnP devices to control the robots. The robots studied in this chapter have been Roomba and Rovio. Both robots have been described and the implementation of the UPnP devices has also been deeply detailed. It is also important to remark that both robots have been integrated in a real UPnP network and different scenarios have been presented.

- **Collaborative Tasks between Robots based on the Digital Home Compliant Protocol over UPnP**. In chapter 4 a novel interoperabiliy protocol is presented: the Digital Home Compliant (DHC) protocol. This protocol is focused on solving the interoperability problem between robotic and smart home devices keeping in mind the smart home market state. In particular, we have presented the DHC-Groups module which allows a group of robots (hive) to perform collaborative tasks. An adapter to provide DHC-Groups capabilities has been implemented for Roomba robot. In order to test the DHC-Groups module and to show its capabilities the Roomba was included in a controlled scenario and different experiments were carried out.

- **Cloud Robotics**. In chapter 5 the basic concepts of cloud computing and cloud robotics are explained. Rovio robot is integrated in a real smart home through a comercial smart home controller. Basic commands for Rovio robot are locally and directly controlled using this controller, which also provides a UPnP interface to control the robot. In order to provide more advanced capabilities to the robot we have developed a cloud service and a cloud application which receive requests from the controller, send commands to Rovio robot and, as result of Rovio responses, send orders back to the smart home controller to perform actions at home. In addition, different experiments have been carried out in order to test these developments. These scenarios have been tested in a real smart home network, with a simulated cloud environment for our developments which uses another real cloud environment, the Google Cloud Speech API.

6.2 Future research

Although the research of interoperability systems have become a hot issue in the last years, there are still many interesting topics that will have to be studied in the future. In this section we enumerate some research lines that are interesting from our point of view:

- There are many interoperability systems proposed in the smart home market but none of them has became a de facto standard. The *Digital entertainment* area has taken advantage in the smart home market among other areas and therefore it is necessary to reach a good interoperability approach with the most popular technologies in this field. Thus, an important research line would be to integrate UPnP protocol, as large as possible, with different sharing and streaming media protocols in order to get a relevant position in the market.

- During this thesis there have been developed complex algorithm in order to provide advanced capabilities to devices. Despite the fact that the creation of UPnP devices is relatively simple, when it is necessary to provide advanced services or to control complex devices, developments become a really heavy task. In this line, another interesting research, or perhaps a good practice, is to define simple interfaces to control smart home devices, which will simplify the developments to integrate devices in a smart home network. In particular, each interface should provide a very specific task. For example, robots should have specific orders to reach specific location, not just commands to guide the robots. In the same line, cloud robotics applications should be simple and very specific, and they should also manage one task for application.

- Cloud Robotics is an important issue to improve the smart home market since it could provide valuable services at a very low cost. We believe that cloud robotics have a great potential to enhance the services provided by robots in the smart home. We also believe that there are strong economical incentives for the development of these services. On the one hand, it allows simplifying hardware requirements so that the production costs and retail prices of both robots and smart home systems can be reduced. On the other hand, cloud services are easier to update and maintain. It also has to be taken into account that the novel services that emerge from the integration of heterogeneous devices also increment the value that users perceive from their smart homes and service robots, which is one of the drivers of the technology industry.

- Cyber-Physical Systems (CPS) is also a very interesting research approach that will become a key issue in the smart home. CPS are integrations of computation and networking elements monitoring and controlling physical entities. CPS is typically designed as a distributed network which has a tight coupling and coordination between its computational, communication and physical sensors and actuators element. The continuous evolution of embedded and ubiquitous computing technologies will increase the adaptability, autonomy, efficiency, functionality, reliability, safety, and usability of these Cyber-Physical Systems. The advances in computation and communication in embedded systems as well as the emergence of CPS networks will allow them to cooperate, share information and be active elements of a more complex system.

6.3 Publications derived from this thesis

As it has been seen, this thesis promotes the integration of service robots in the smart home and the improvement of UPnP capabilities. Basically its purpose is to increase the interoperability in the digital home. In this section we will enumerate the major contributions made in this work and the results from the thesis that have been published in conferences and journals:

1. "Localización de Robot Móvil Mediante Zigbee" (Robot Mobile Location using Zigbee) was a publication for the XXXI Automation Workshop in Jaen in 2010.

2. "Robots in the smart home: a project towards interoperability" is a paper published in 2011 in the International Journal of Ad Hoc and Ubiquitous Computing

3. "Integration of Service Robots in the Smart Home" was published in 2011 as a chapter of the Springer book "Service Robotics within the Digital Home: Appications and Future Prospects".

4. "Interoperabiliy Systems" was published in 2011 as another chapter of the Springer book "Service Robotics within the Digital Home: Appications and Future Prospects".

5. "Smoke Detectors: Development of an Alarm Management System for UPnP" is a paper published in the Intera Workshop in 2011.

6. "Integration of service robots in the smart home by means of UPnP: A surveillance robot case study" was published in the international journal Robotics and Autonomous Systems in 2013.

7. "Collaborative tasks between robots based on the Digital Home Compliant Protocol over UPnP" is a paper published in the international journal Journal of Intelligent & Robotic Systems in 2013.

8. "UPnP" is a chapter which has been published in 2015 in the book Domótica para ingenieros.

9. "Integración de robots mediante UPnP" (Integration of robots by means of UPnP) is a chapter which has been published in 2015 in the book Domótica para ingenieros.

10. Another paper entitled "Off the shelf cloud robotics for the smart home: empowering a wireless robot through cloud computing" has been sent to the international journal Sensors - Open Access Journal (published by MDPI).

Appendix A

Resumen en castellano

La interoperabilidad trata con la integracin de dispositivos heterogéneos en la misma red independientemente de su arquitectura, sistema operativo, lenguaje de programación o posición en la red.

El problema de la falta de interoperabilidad surge cuando hay diferentes dispositivos que cumplen con cierto sistema pero no son compatibles con otras tecnologías en el hogar digital, por lo que no es posible hacer que todos los dispositivos y tecnologías trabajen juntamente. Este problema no es nuevo, de hecho ha sido una constante durante la corta historia del hogar digital. En el mercado para el hogar digital han emergido muchas tecnologías, pero ninguna de ellas preocupadas por proporcionar mecanismos de interoperabilidad con tecnologías previas. Hay muchos factores que pueden explicar tal situación: normalmente las nuevas tecnologías llegan con nuevas posibilidades que no existían antes, compatibilidad implica mayor coste de desarrollo y un riesgo de clientes comprando dispositivos a otros fabricantes, fuerte falta de estandarización en el sector La lista de causas es casi interminable y ayuda a identificar errores que no deberían repetirse en el futuro. Sin embargo, no ayuda mucho a resolver el problema de la interoperabilidad al que tenemos que enfrentarnos hoy en día. En particular, es posible identificar diferentes áreas que trabajan más o menos de forma independiente y que tienen que ser integradas en el futuro hogar digital.

Como los robots de servicios están muy próximos a los humanos, la tecnología involucra más seguridad relativa a la interacción hombre-máquina. Por lo tanto, en la actualidad se mantiene el gran desafo de construir hogares digitales y robots inteligentes que puedan *pensar* como nosotros lo hacemos. Para alcanzar tal objetivo, científicos e ingenieros han trabajado duro para capturar la esencia de la inteligencia humana en nuestros hogares y robots para hacerlos inteligentes para funcionar en el mundo real. Este es una tarea ambiciosa, ya que robots y hogares inteligentes deben hacer frente a diversos problemas, incertidumbres y cambios dinámicos del mundo real. Como los humanos, los hogares digitales y los robots inteligentes deberían ser capaces de sentir el entorno, razonar y tomar decisiones, así como responder rápidamente a tareas y eventos inesperados. Esto nos lleva a la necesidad de impulsar en el hogar digital lo que conocemos como inteligencia artificial.

En general, los robots y hogares inteligentes son elementos que involucran a muchas tecnologías diferentes, como la integración de sensores, fusión de datos, redes de sensores inalámbricos, construcción de mapas, sistemas informáticos embebidos, navegación, planificación e inteligencia artificial.

En todos los mercados emergentes la compatibilidad entre sistemas juega un papel crucial para el éxito del propio mercado. En este sentido, el mercado del hogar digital es como el perro que se muerde la cola: como no hay un sistema idóneo entre los existentes, el mercado se guía por la oferta y no por la demanda, pero cómo va a haber una fuerte demanda de dispositivos y servicios cuando no hay una forma sencilla de combinarlos? Es necesario un sólido compromiso hacia la interoperabilidad por parte de todos los actores de la industria domótica si se desea acelerar la penetración de servicios avanzados a los usuarios finales.

A.1 Sistemas de Interoperabilidad

Hoy en día hay muchas iniciativas y diferentes estándares disponibles cuyo objetivo es interconectar todos los dispositivos de electrónica de consumo en el hogar digital. Sin embargo, la falta de interoperabilidad entre los sistemas propuestos, la incompatibilidad entre algunos de ellos y la falta de compromiso por parte de los fabricantes para hacer que los dispositivos cumplan con tales sistemas está evitando el despegue del hogar digital en nuevos edificios.

Durante las últimas décadas ha habido un crecimiento exponencial en robots de servicios y tecnologías domóticas, lo que ha permitido el desarrollo de nuevos productos en nuestra vida diaria. Los robots de servicios pueden ser usados para proporcionar asistencia para la tercera edad y discapacitados, proporcionando servicios que van desde la limpieza hasta el entretenimiento.

Los robots de servicios están divididos por funciones, tales como robots personales, robots de campo, robots de seguridad, robots sanitarios, robots médicos, robots para rehabilitación y robots de entretenimiento. La falta de interoperabilidad se hace más evidente con la inclusión de tales robots de servicio en hogares de todo el mundo. En este sentido, sería interesante controlar robots de forma remota y coordinar todos los robots para realizar tareas complejas y reducir el tiempo en completarlas. Por esta razón, el objetivo de alcanzar la interoperabilidad total entre los dispositivos de electrónica de consumo y robots de servicios se hace cada vez más importante.

Ante este contexto todos los esfuerzos se centran en alcanzar un estándar común y aceptado que permita interconectar todos los dispositivos en el hogar domótico. Otro enfoque interesante es usar un sistema de interoperabilidad existente (estándar intermedio) para interoperar con el resto de los sistemas, de tal forma que todo quede interconectado a través de este sistema intermedio.

Es importante tener en cuenta la necesidad de cooperación y expansión de las redes que ya están en uso. El principio es el concepto de middleware. Middleware es la conectividad software que ofrece un grupo de servicios y que hace posible ejecutar aplicaciones distribuidas sobre plataformas heterogéneas.

La idea de middleware, como una capa abstracta de software, es encapsular todos los recursos disponibles en una red que pueda comprender todo tipo de dispositivos e interconectarlos de forma transparente. En otras palabras, dar un API (Application Programming Interface) a los programadores para el uso de aplicaciones distribuidas.

La próxima revolución en el hogar domótico se espera que proceda del mundo de la robótica. Actualmente, el uso de la robótica está limitado a áreas industriales, aunque los robots de servicio que nos ayudan en tareas rutinarias como limpiar la casa, cortar el césped o incluso preparar comidas son cada vez más comunes. Sin embargo, se deben resolver diversos problemas antes de que los robots de servicios lleguen a ser tan populares como los ordenadores. En concreto, la interoperabilidad entre los diferentes sistemas que puedan existir en los hogares es un tema que debe resolverse.

Durante este trabajo se han analizado algunas de las iniciativas middleware más usadas para proporcionar interoperabilidad. Se describen los protocolos y sistemas existentes para comunicar plataformas heterogéneas. Las tecnologías estudiadas y detalladas en este trabajo son las siguientes: CORBA, Jini, RMI, OSGi, UPnP, DLNA, Web Services, Semantic Web Services, Military Standards, Salutation, Service Location Protocol, Ad hoc Developments, URBI, DH Compliant, ROS, OROCOS y OpenJAUS.

La introducción de los robots de servicios como una nueva área que debe interoperar con el resto de dispositivos en el hogar digital potencia todavía más el problema de la falta de interoperabilidad, y se hace necesario disponer de tecnologías que abarquen distintas áreas y funcionalidades.

En esta tesis la tendencia ha sido emplear un protocolo en uso y potenciar sus capacidades, así como plantear pasarelas para integrar otros dispositivos totalmente independientes al protocolo. De esta forma, se han incluido robots de servicios, se ha desarrollado un nuevo protocolo basado en el original y totalmente compatible con éste, y además para los procesos que requieren alta potencia de computación se ha propuesto resolverlo en el cloud.

A.2 Objetivos de la Tesis

Como se verá a lo largo de la tesis, la inclusión de nuevos servicios avanzados en el hogar digital hace necesario más que nunca potenciar la interoperabilidad entre las distintas áreas y sistemas disponibles en el mercado. Este es el objetivo principal del trabajo realizado en esta tesis. La primera etapa que se fija en este trabajo consiste en conocer las tecnologías más destacadas en este sector, con el fin de conocer los puntos de interés y las carencias de cada sistema, para así potenciar los aspectos beneficiosos para nuestras propuestas.

La inclusión de robots de servicios en el hogar digital que sean capaces de interoperar con el resto de sistemas en el hogar digital es un reto que es necesario resolver. Esta tesis intenta proporcionar mecanismos para integrar robots de servicios en el hogar digital, reduciendo costes de tiempo, desarrollos y maximizando los beneficios que originalmente proponía UPnP. Hemos propuesto soluciones para integrar en el hogar digital sistemas y robots distintos y heterogéneos. Como se ha hecho notar a lo largo de la tesis, la interoperabilidad es un tema muy importante en el hogar digital ya que su falta es el mayor obstáculo que evita la penetración de las tecnologías del hogar digital en el mercado. Es cierto que el mercado multimedia ha progresado mucho en los últimos años, sin embargo, la continua aparición de nuevos protocolos para llegar a una solución para compartir y transmitir contenido de un dispositivo a otro sigue aún en auge. Una vez más, el problema de la interoperabilidad evita que los dispositivos multimedia interoperen entre ellos en cualquier entorno y situación.

Durante esta tesis se han integrado algunos robots en redes UPnP y las capacidades de dicho protocolo se han potenciado como resultado de las investigaciones realizadas durante el proyecto DH Compliant. Hemos desarrollado un protocolo para interoperar de forma transparente con cada dispositivo en el hogar. Dicho protocolo, con el objetivo de mantener compatibilidad hacia atrás, usa un existente y popular estándar, y propone el uso de entidades UPnP con doble comportamiento (punto de control y dispositivo), permitiendo el intercambio de información y la cooperación entre entidades. Por el contrario, es necesario el desarrollo de lógicas más complejas.

En general, las soluciones de interoperabilidad previas se han centrado en mejorar las capacidades disponibles. Esto implica que muchos sistemas no sean compatibles con el resto de sistemas disponibles en el mercado. Uno de nuestros objetivos más importantes durante este trabajo ha sido presentar un procedimiento para integrar distintos dispositivos, y robots en particular, en el hogar digital. En este trabajo ha quedado demostrado que cualquier dispositivo es susceptible de ser integrado en un hogar a través del estándar UPnP.

Además, como parte de la tesis, se ha investigado acerca de como proporcionar servicios de computación existentes desde entornos *cloud* a robots disponibles en los hogares. Este estudio tiene el objetivo de proporcionar de forma transparente servicios con altas necesidades de computación al entorno del hogar digital.

A.3 Estructura de la Tesis

La tesis está organizada como se detalla a continuación:

- **Capítulo 2: Sistemas de Interoperabilidad**. En este capítulo se ha abordado el problema de la falta de interoperabilidad en general, y en el hogar digital en particular. Además, se ha realizado una revisión de los sistemas existentes hasta al fecha desde un punto de vista de interoperabilidad, detallando una amplia lista de propuestas de interoperabilidad. Se ha prestado especial atención al sistema UPnP debido a que es la base de esta tesis.

- **Capítulo 3: Integración de Robots de Servicios en el Hogar Digital por medio del Protocolo UPnP**. La integración de robots en el hogar digital proporciona una vida más confortable para sus habitantes. En particular, se ha propuesto una metodología para crear dispositivos UPnP para controlar los robots. Los robots estudiados en este capítulo han sido Roomba y Rovio. Ambos robots han sido descritos y la implementación de los dispositivos UPnP desarrollados para controlarlos también ha sido ampliamente detallada.

- **Capítulo 4: Tareas Colaborativas entre Robots basado en el Protocolo DH Compliant sobre UPnP**. En este capítulo se presenta un nuevo protocolo de interoperabilidad: Digital Home Compliant (DHC), que se centra en resolver el problema de la interoperabilidad entre dispositivos domóticos y robóticos. En concreto, se ha presentado el módulo DHC-Groups, que permite a un grupo de robots realizar tareas colaborativas. Además se ha desarrollado y presentado un adaptador para proporcionar las propiedades de este módulo al robot Roomba.

- **Capítulo 5: Cloud Robotics**. En este capítulo se presentan los conceptos básicos de la computación en la nube o *cloud computing* y de la computación en la nube para robots o *cloud robotics*. Se integra al robot Rovio en un hogar digital real a través de un controlador comercial. Este controlador permite enviar, localmente y de forma directa, comandos básicos al robot Rovio, y además proporciona una interfaz UPnP para controlar este robot mediante un punto de control. Finalmente, y con el objetivo de proporcionar capacidades más avanzadas al robot, se ha desarrollado un servicio y una aplicación en la nube que reciben peticiones desde el controlador, envían comandos al robot y, como resultado de las respuestas de éste, envían órdenes de vuleta al controlador del hogar digital para que ejecute acciones en el hogar.

- **Capítulo 6: Conclusiones**. La tesis finaliza con un capítulo que analiza las contribuciones más relevantes y, adicionalmente, señala líneas futuras de investigación en el campo de la interoperabilidad en el hogar digital.

A.4 Contribuciones

Una vez ubicada la tesis en el marco de la interoperabilidad, se repasarán en este apartado las principales aportaciones al estado del arte realizadas y se señalarán también las publicaciones que se han originado a partir del presente trabajo.

Esta tesis promueve la integración de robots de servicios en el hogar digital y la mejora de las capacidades UPnP. Básicamente el propósito ha sido mejorar la interoperabilidad en el hogar.

El estado del arte y el estudio de los sistemas de interoperabilidad, disponibles en el mercado y presentados en la tesis, han sido publicados como parte de dos capítulos de libro [10, 11].

El procedimiento para crear dispositivos virtuales UPnP, resultado de este trabajo, queda recogido en [205] como capítulo de un libro. En este libro también se presentan distintos desarrollos de dispositivos UPnP integrados en una misma red UPnP. Otro dispositivo UPnP que también fue integrado en la red es el detector de humos publicado en [206].

El primer robot que se integró en el hogar digital fruto de este trabajo fue el robot aspirador Roomba. El resultado de esta investigación se publicó en en una revista internacional [44]. Otro robot que, fruto de este trabajo, quedó integrado en una red UPnP es el Rovio. Como consecuencia de esta integración, junto con la presentación de varios escenarios de Rovio en la red UPnP, se publicó un nuevo artículo en otra revista [122]. Durante este trabajo, la interoperabilidad entre dispositivos domóticos y robóticos estuvo siempre en mente. En un capítulo de libro [121] ambos robots (Roomba y Rovio) son profundamente detallados y se revisa y analiza la integración de ambos en una red digital.

A pesar de que no se obtuvieron los resultados esperados, también es interesante reseñar la contribución acerca de la estimación de la posición de robots en interiores mediante balizas ZigBee que se presentó en las jornadas de automática [207].

Otro aporte muy interesante, realizado dentro del proyecto del consorcio DH Compliant, es la presentación de una nueva tecnología basada en UPnP, que permite extender el concepto de tareas colaborativas y que se presentó en otra revista [188].

Finalmente, el planteamiento de realizar procesamientos en la nube de complejos algoritmos por parte de los robots está siendo evaluado en otro artículo de revista internacional [184].

Bibliography

[1] M. Aleksy, A. Korthaus, and M. Schader, *Implementing Distributed Systems with Java and CORBA*. Berlin: Springer, 2005.

[2] B. Miller, T. Nixon, C. Tai, and M. Wood, "Home networking with universal plug and play," *Communications Magazine, IEEE*, vol. 39, no. 12, pp. 104 –109, dec 2001.

[3] M. Saaranen and D. Kalafonos, *Technologies for Home Networking*. John Wiley & Sons, Inc., 2007, ch. Mobile Device Connectivity in Home Networks, pp. 73–92.

[4] D. Cook, M. Youngblood, E. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, "Mavhome: an agent-based smart home," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, 2003.

[5] J. M. Maestre and E. F. Camacho, "Smart home interoperability: the domoesi project approach," *International Journal of Smart Home*, vol. 3, pp. 31–44, 2009.

[6] J. Tu, W.-W. Lin, J.-C. Wang, and Y.-T. Lin, "The scenario implementation of home networking," in *The 9th International Conference on Advanced Communication Technology*, vol. 3, feb. 2007, pp. 1861 –1863.

[7] T. Perumal, A. R. Ramli, C. Y. Leong, S. Mansor, and K. Samsudin, "Interoperability for smart home environment using web services," *International Journal of Smart Home*, vol. 2, no. 4, pp. 1–16, 2008.

[8] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, ser. Middleware '98.

London, UK: Springer-Verlag, 1998, pp. 191–206. [Online]. Available: http://portal.acm.org/citation.cfm?id=1659232.1659249

[9] G. Coulson, G. S. Blair, M. Clarke, and N. Parlavantzas, “The design of a configurable and reconfigurable middleware platform,” *Distributed Computing*, vol. 15, pp. 109–126, 2002, 10.1007/s004460100064. [Online]. Available: http://dx.doi.org/10.1007/s004460100064

[10] M. del Pilar Almudena Garca Fuente, J. R. de la Pinta, and A. L. Garca, *Interoperabiliy Systems (Service Robotics within the Digital Home: Appications and Future Prospects).* Springer, 2011, vol. 53, ch. 1, pp. 1–47.

[11] M. R. F. Alcalá, J. M. Maestre, and J. R. de la Pinta, *Integration of Service Robots in the Smart Home (Service Robotics within the Digital Home: Appications and Future Prospects).* Springer, 2011, vol. 53, ch. 4, pp. 115–142.

[12] S. Vinoski, “Corba: integrating diverse applications within distributed heterogeneous environments,” *Communications Magazine, IEEE*, vol. 35, no. 2, pp. 46 –55, feb 1997.

[13] M. Henning and S. Vinoski, *Advanced CORBA programming with C++.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[14] OMG. (1998) Corba services: Common object services specification. OMG (Object Management Group). [Online]. Available: http://www.ing.iac.es/~docs/external/corba/CorbaServices.pdf

[15] S. Bottazzi, S. Caselli, M. Reggiani, and M. Amoretti, “A software framework based on real-time corba for telerobotic systems,” in *in IEEE Intl Conf. Intelligent Robots and Systems*, 2002, pp. 3011–3017.

[16] E. Woo, B. MacDonald, and F. Trepanier, “Distributed mobile robot application infrastructure,” in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2, oct. 2003, pp. 1475 – 1480 vol.2.

[17] S. Knoop, S. Vacek, R. Zollner, C. Au, and R. Dillmann, “A corba-based distributed software architecture for control of service robots,” in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 4, sept.-2 oct. 2004, pp. 3656 – 3661 vol.4.

[18] K. Takeda, Y. Nasu, G. Capi, M. Yamano, L. Barolli, and K. Mitobe, "A corba-based approach for humanoid robot control," *Industrial Robot: An International Journal*, vol. 28, no. 3, pp. 242–250, 2001.

[19] Z. Zhang, Q. Cao, L. Zhang, and C. Lo, "A corba-based cooperative mobile robot system," *Industrial Robot: An International Journal*, vol. 36, no. 1, pp. 36–44, 2009.

[20] K. Arnold, "The jini architecture: dynamic services in a flexible network," in *Design Automation Conference, 1999. Proceedings. 36th*, 1999.

[21] R. Gupta, S. Talwar, and D. Agrawal, "Jini home networking: a step toward pervasive computing," *Computer*, vol. 35, no. 8, pp. 34 – 40, aug 2002.

[22] S. Morgan, "Jini to the rescue [computer network interconnection technology]," *Spectrum, IEEE*, vol. 37, no. 4, pp. 44 –49, apr 2000.

[23] H. Chen, "Developing a dynamic distributed intelligent agent framework based on the jini architecture," Master's thesis, University of Maryland Baltimore County, 2000.

[24] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "Iceni: An open grid service architecture implemented with jini," in *In SuperComputing 2002*, 2002.

[25] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington, "Implementations of a service-oriented architecture on top of jini, jxta and ogsi." in *European Across Grids Conference'04*, 2004, pp. 90–99.

[26] H. He. (2003, September) What is service-oriented architecture. [Online]. Available: http://www.xml.com/pub/a/ws/2003/09/30/soa.html

[27] J. Allard, V. Chinta, S. Gundala, and I. Richard, G.G., "Jini meets upnp: an architecture for jini/upnp interoperability," in *Applications and the Internet, 2003. Proceedings. 2003 Symposium on*, jan. 2003, pp. 268 – 275.

[28] C.-W. Chen, C.-K. Chen, J.-C. Chen, C.-T. Ko, J.-K. Lee, H.-W. Lin, and W.-J. Wu, "Efficient support of java rmi over heterogeneous wireless networks," in *Communications, 2004 IEEE International Conference on*, vol. 3, june 2004, pp. 1391 – 1395 Vol.3.

[29] Oracle. (2004) When should i use rmi-iiop? [Online]. Available: http://java.sun.com/j2se/1.5.0/docs/guide/rmi-iiop/rmiiiopUsing.html

[30] T. Spexard, F. Siepmann, and G. Sagerer, "Memory-based software integration for development in autonomous robotics," in *International Conference on Intelligent Autonomous Systems*, 2008.

[31] D. Westhoff, T. Scherer, H. Stanek, J. Zhang, and K. A., "A flexible framework for task-oriented programming of service robots," *VDI BERICHTE*, 2004.

[32] M. Zhang, B. P. Zeigler, and P. Hammonds, "Devs/rmi-an auto-adaptive and reconfigurable distributed simulation environment for engineering studies," *ITEA Journal*, 2005.

[33] O. Alliance, *OSGi service platform: the OSGi alliance.* Ios Pr Inc, 2003.

[34] N. Bartlett, *OSGi In Practice.* Online: amazon.com, 2009.

[35] T. Gu, H. Pung, and D. Zhang, "Toward an osgi-based infrastructure for context-aware applications," *Pervasive Computing, IEEE*, vol. 3, no. 4, pp. 66 – 74, oct.-dec. 2004.

[36] D.-O. Kang, K. Kang, S.-G. Choi, and J. Lee, "Upnp av architectural multimedia system with a home gateway powered by the osgi platform," in *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, jan. 2005, pp. 405 – 406.

[37] M. Jeronimo and J. Weast, *UPnP Design by Example. A Software Developers Guide to Universal Plug and Play.* Intel Press, 2003.

[38] G. A. Olleros, "Domotica: protocolo upnp y hogar digital," 2007, proyecto Fin de Carrera, Universidad de Sevilla, Sevilla. [Online]. Available: http://bibing.us.es/proyectos/abreproy/11557/fichero/Volumen+I\%252F1_\%CDndice.pdf

[39] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *Internet Computing, IEEE*, vol. 6, no. 2, pp. 86–93, March 2002. [Online]. Available: http://dx.doi.org/10.1109/4236.991449

[40] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. (2000, October) Extensible markup language (xml) 1.0 (second edition),

recommendation rec-xml-20001006. World Wide Web Consortium. [Online]. Available: http://www.w3.org/TR/2000/REC-xml-20001006

[41] M. Jeronimo. (2004) It just works: Upnp in the digital home. The Journal of Spontaneous Networking. [Online]. Available: http://www.artima.com/spontaneous/upnp_digihome.html

[42] T. Fout. (2001, July) Universal plug and play in windows xp. Microsoft Corporation. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.4096&rep=rep1&type=pdf

[43] W3C. (2008) Guía breve de tecnologías xml. World Wide Web Consortium (W3C). [Online]. Available: http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML

[44] J. de la Pinta, J. Maestre, E. Camacho, and I. Alonso, "Robots in the smart home: a project towards interoperability," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 7, no. 3, pp. 192–201, 2011.

[45] U. UPnP Forum. (2001) Universal plug and play vendor's implementation guide. [Online]. Available: http://www.upnp.org/download/UPnP_Vendor_Implementation_Guide_Jan2001.htm

[46] U. Members of the UPnP Forum. (2008) Upnp device architecture 1.1. [Online]. Available: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[47] S. C. Ahn, J. H. Kim, K. Lim, H. Ko, Y.-M. Kwon, and H.-G. Kim, "Upnp approach for robot middleware." in *Proceedings of the 2005 IEEE International Conference on In Robotics and Automation (ICRA'05)*, 2005, pp. 1959–1963.

[48] D.-S. Kim, J.-M. Lee, W. H. Kwon, and I. K. Yuh, "Design and implementation of home network systems using upnp middleware for networked appliances," *IEEE Transactions on Consumer Electronics*, vol. 48, pp. 963–972, 2003.

[49] S. M. Mok and C. Wu, "Automation integration with upnp modules," in *Electronic Design, Test and Applications, 2006. DELTA 2006. Third IEEE International Workshop on*, jan. 2006, p. 5 pp.

[50] R. Dobrescu, M. Dobrescu, M. Nicolae, and D. Popescu, "Using upnp services with an intelligent sensor network node," in *Proceedings of the 7th Conference on 7th WSEAS International Conference*

on Applied Informatics and Communications, vol. 7. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2007, pp. 371–374. [Online]. Available: http://portal.acm.org/citation.cfm?id=1348011.1348077

[51] H. Song, D. Kim, K. Lee, and J. Sung, "Upnp-based sensor network management architecture," in *Second International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2005)*, 2005. [Online]. Available: http://www.ishilab.net/icmu2005/papers/117390-1-050228235605.pdf

[52] S. Konno, "Cyberlink for java - programming guide v1.3," Cyber-Garage, 2004.

[53] S. Fuentes. (2007, September) Qué es dlna: a fondo. [Online]. Available: http://www.xataka.com/musica/que-es-dlna-a-fondo

[54] D. Digital Living Network Alliance. (2007) Dlna overview and vision whitepaper. [Online]. Available: https://wikileaks.org/sony/docs/05/docs/DLNA/DLNA_white_paper.pdf

[55] A. S. D. Corporation. (2006) Networked digital media standards a upnp/dlna overview. [Online]. Available: http://www.allegrosoft.com/downloads/UPnP_DLNA_White_Paper.pdf

[56] K. Arruda. (2008, March) Dtcp-ip for dlna.

[57] J.-T. Kim, Y.-J. Oh, H.-K. Lee, E.-H. Paik, and K.-R. Park, "Implementation of the dlna proxy system for sharing home media contents," in *Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers. International Conference on*, jan. 2007, pp. 1 –2.

[58] P. Ferguson and G. Houston, "What is a vpn?" in *OPENSIG'98 Workshop on Open Signalling for ATM, Internet and Mobile Networks*, Toronto, October 1998.

[59] M. C. Daconta, L. J. Obrst, and K. T. Smith, *The Semantic Web: A guide to the future of XML, Web services, and knowledge management.* Indianapolis: Ind: Wiley Pub, 2003.

[60] J. Levine and L. Vickers, "Robots controlled through web services: A technogenesis summer research," 2001.

[61] DARPA. (2011) Darpa urban challenge. [Online]. Available: http://archive.darpa.mil/grandchallenge/

[62] R. Lara, D. Roman, A. Polleres, and D. Fensel, "A conceptual comparison of wsmo and owl-s," in *ECOWS 2004*, ser. LNCS, vol. 3250. Springer, 2004, pp. 254–269. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30209-4_19

[63] W. English, "Joint architecture for unmanned systems (jaugs). reference architecture specification, volume ii, version 3.3," 2007.

[64] R. Wade. (2006) Joint architecture for unmanned systems. aviation and missile research, development and engineering center (amrdec). [Online]. Available: http://www.dtic.mil/ndia/2006targets/Wade.pdf

[65] SAE. (2011) Society of automotive engineers. [Online]. Available: http://www.sae.org/

[66] ——. (2006) Jaus history and domain model. architecture framework committee. [Online]. Available: http://www.sae.org/

[67] ——. (2009) Jaus/sdp transport specification. network environmental committee. [Online]. Available: http://www.sae.org/

[68] ——. (2011) Jaus core service set. information modeling and definition committee. [Online]. Available: http://www.sae.org/

[69] ——. (2009) Jaus mobility service set. information modeling and definition committee. [Online]. Available: http://www.sae.org/

[70] H. Everett, R. Laird, D. Carroll, G. Gilbreath, T. Heath-Pastore, R. Inderieden, T. Tran, K. Grant, and D. Jaffee, "Multiple resource host architecture (mrha) for the mobile detection assessment response system (mdars). spawar systems technical document 3026, revision a," 2000.

[71] H. Nguyen, "Overview and highlights of robotics research and development at the space and naval warfare systems center." SPAWAR Systems Center, San Diego, Tech. Rep., 2005.

[72] D. T. Carroll DM, Mikell K, "Unmanned ground vehicles for integrated force protection," in *In SPIE Proc. 5422*, 2004.

[73] G. Gothing and J. Hurdus, "Implementation of jaus on a 2004 cadillac srx using a potential fields architecture." in *AUVSI's Unmanned Systems North America 2006*, Orlando, FL, 2006.

[74] B. J. Koren Y, "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1991, pp. 1398–1404.

[75] N. F. of the Blind. (2011) Blind driver challenge. National Federation of the Blind. [Online]. Available: https://nfb.org/nfb-blind-driver-challenge

[76] R. Faruque, "A jaus toolkit for labview, and a series of implementation case studies with recommendations to the sae as-4 standards committee," Master's thesis, Mechanical Engineering, Virginia Tech, 2006.

[77] TORC. (2011) Bywire xgv - hybrid escape drive-by-wire platform. [Online]. Available: http://www.torctech.com/

[78] M. Clark, "Jaus implementation: Robots gather for successful interoperability experiment," *IEEE Robotics and Automation Magazine*, 2005.

[79] ——, "Jaus compliant systems offers interoperability across multiple and diverse robot platforms," in *Unmanned Systems North America Conference*, Baltimore, Maryland, 2005.

[80] S. Baity, "Development of a next-generation experimentation robotic vehicle (nerv) that supports intelligent and autonomous systems research," Master's thesis, Mechanical Engineering, Virginia Tech, 2005.

[81] J. Albus, H. Huang, and E. Messina, "4d/rcs a reference model architecture for unmanned vehicle systems, version 2.0," National Institute of Standards and Technology (NISTIR 6910), Gaithersburg, MD, Tech. Rep., 2002.

[82] J. Albus, K. Murphy, A. Lacaze, S. Legowik, S. Balakirsky, T. Hong, M. Shneier, and E. Messina, "4d/rcs sensory processing and world modeling on the demo iii experimental unmanned ground vehicles," in *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, 2002, pp. 885 – 890.

[83] C. Schlenoff, J. Albus, E. Messina, and T. B. R. Madhavan, "Using 4d/rcs to address ai knowledge integration," *AI Magazine*, vol. 27, 2006.

[84] C. Shoemaker and J. Bornstein, "The demo iii ugv program: a testbed for autonomous navigation research," in *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, sep 1998, pp. 644 –651.

[85] C. Systems. (2011) Nato stanag 4586. CDL Systems.

[86] D. U. Magazine. (2007) Stanag 4586 - nato complient ground control system for uav. [Online]. Available: http://defense-update.com/products/s/stanag_4586.htm

[87] N. S. 4586, "Standard interfaces of uav control system (ucs) for nato uav interoperability," in *UAV Interoperability*. Brussels: NATO Standardization Agency (NSA), 2004.

[88] M. Cummings, A. Kirschbaum, A. Sulmistras, and J. Platts, "Stanag 4586 human supervisory control implications," in *Unmanned Vehicle Systems Canada Conference*, Montebello, Quebec, November 2006.

[89] N. Suri, J. Bradshaw, M. Carvalho, T. Cowin, M. Breedy, P. Groth, and R. Saavedra, "Agile computing: bridging the gap between grid computing and ad-hoc peer-to-peer resource sharing," in *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, may 2003, pp. 618 – 625.

[90] B. Miller and R. Pascoe, "Mapping salutation architecture apis to bluetooth service discovery layer," *Bluetooth Consortium 1.C.118/1.0*, vol. 1, 1999.

[91] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. (1997, June) Service location protocol. [Online]. Available: http://tools.ietf.org/html/rfc2165

[92] C. Bettstetter and C. Renner, "A comparison of service discovery protocols and implementation of the service location protocol," *Proceedings of the 6th EUNICE Open European Summer School Innovative Internet Applications*, pp. 13–15, 2000. [Online]. Available: http://data.bettstetter.com/publications/bettstetter-2000-eunice-slp.pdf

[93] F. Zhu, M. Mutka, and L. Ni, "Classification of service discovery in pervasive computing environments," Michigan State University, Tech. Rep., 2002.

[94] S. Avancha, A. Joshi, and T. Finin, "Enhancing the bluetooth service discovery protocol," 2001.

[95] Gostai, "Urbi quick start guide, version 1.5," URBI, Tech. Rep., November 2007.

[96] Urbi. (2011) Urbi official website.

[97] Gostai, “The urbi software development kit,” URBI, Tech. Rep., April 2010.

[98] Urbi. (2011) Urbi walkthrough. [Online]. Available: http://www.youtube.com/watch?v=uD65ARxyrnw

[99] Sony. (2011) Sony aibo europe official website. [Online]. Available: http://www.sony-aibo.com/

[100] D. Compliant. (2012) Dh compliant. Available: http://www.dhcompliant.com/. [Online]. Available: http://www.dhcompliant.com/

[101] Ingenium. (2011) Ingenium official website. [Online]. Available: www.ingeniumsl.com

[102] D. Davinci. (2015). [Online]. Available: http://www.domoticadavinci.com/

[103] MoviRobotics. (2011) Movirobotics official website. [Online]. Available: http://www.movirobotics.com/SPmsecurit.php

[104] A. R. A. ARA. (2011). [Online]. Available: www.ara.com

[105] C. Foundation. (2011) Cartif official website. [Online]. Available: http://www.cartif.com/en/

[106] U. of Oviedo Infobotica Research Group. (2011) Dhcompliant stack architecture v1.1. Available: http://156.35.46.38/data/files/Architecture/DHCOMPLIANT-Architecture1.1.pdf. [Online]. Available: http://156.35.46.38/data/files/Architecture/DHCOMPLIANT-Architecture1.1.pdf

[107] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open source robot operating system,” in *Proceedings of the 2009 International Conference on Robotics and Automation, Open-Source Software Workshop*, 2009.

[108] H. Bruyninckx, “Open robot control software: the orocos project,” in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2523 – 2528.

[109] H. Bruyninckx and P. Soetens. (2007) The orocos project: Open robot control software. Available: http://www.orocos.org/. [Online]. Available: http://people.mech.kuleuven.be/~orocos/pub/documentation/rtt/v1.8.x/doc-xml/orocos-overview.html

[110] OpenJAUS. (2015) Openjaus official website. Available: http://openjaus.com/. [Online]. Available: http://openjaus.com/

[111] IPDomo. (2011) Ipdomo official website. [Online]. Available: http://www.ipdomo.com

[112] V. Ricquebourg, D. Menga, D. Durand, B. Marhic, L. Delahoche, and C. Loge, "The smart home concept : our immediate future," in *1ST IEEE International Conference on E-Learning in Industrial Electronics*, December 2006, pp. 23 –28.

[113] L. N. Hoang, "Middlewares for home monitoring and control," *TKK T-110.5190 Seminar on Internetworking (Home Networking)*, 2007, helsinki University of Technology. Telecommunications Software and Multimedia Laboratory.

[114] U. UPnP Forum. (2011) Upnp forum official website. [Online]. Available: http://www.upnp.org/

[115] C. Ramos, J. C. Augusto, and D. Shapiro, "Ambient intelligence - the next step for artificial intelligence," *Intelligent Systems, IEEE*, vol. 23, no. 2, pp. 15 –18, 2008.

[116] S. Dixit and R. Prasad, *Technologies for Home Networking.* John Wiley & Sons, Inc., 2008.

[117] S. Zeadally and P. Kubher, "Internet acces to heterogeneous home area network devices with an osgi-based residential gateway," *International Journal of Ad Hoc and Ubiquitous Computing 2008 - Vol. 3, No.1 pp. 48 - 56*, vol. 3, pp. 48–56, 2008.

[118] V. Miori, L. Tarrini, M. Manca, and G. Tolomei, "An open standard solution for domotic interoperability," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 1, pp. 97–103, 2006.

[119] H. Y. Lee and J. W. Kim, "An approach for content sharing among upnp devices in different home networks," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 4, pp. 1419 –1426, November 2007.

[120] J. Sung, D. Kim, H. Song, J. Kim, S. Y. Lim, and J. S. Choi, "Upnp based intelligent multimedia service architecture for digital home network," in *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006.*, April 2006, p. 6 pp.

[121] M. J.M. and de la Pinta J.R., *Integración de robots mediante UPnP (Domótica para ingenieros).* Paraninfo, March 2015, ch. 15.

[122] R. Borja, J. de la Pinta, A. Álvarez, and J. M. Maestre, "Integration of service robots in the smart home by means of upnp: A surveillance robot case study," *Robotics and Autonomous Systems*, vol. 61, no. 2, pp. 153–160, February 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889012001881

[123] IFR. (2011) International federation of robotics official website. [Online]. Available: http://www.ifr.org/

[124] Hisparob. (2011) Hisparob official website. [Online]. Available: http://www.hisparob.es/

[125] A. MALL. (2011) Aeon mall official website. [Online]. Available: http://www.aeonmall.com/en/index.html

[126] Tmsuk. (2011) Tmsuk co. official website. [Online]. Available: http://www.tmsuk.co.jp/english/robots.html

[127] ALSOK. (2011) Alsok official website. [Online]. Available: http://www.alsok.co.jp/en/

[128] I. Surgical. (2011) Intuitive surgical official website. [Online]. Available: http://www.intuitivesurgical.com/

[129] H. Motor. (2011) Honda official website. [Online]. Available: http://world.honda.com/ASIMO/new/

[130] MetraLabs. (2011) Metralabs official website. [Online]. Available: http://www.metralabs.com/

[131] T. M. Corporation. (2011) Toyota motor corporation global website. [Online]. Available: http://www.toyota.co.jp/en/index.html

[132] iRobot. (2011) irobot official website. [Online]. Available: http://www.irobot.com/

[133] H. Wiechman. (2005, December) Interoperability in the multi-format home network. [Online]. Available: http://www.hometoys.com/htinews/dec05/articles/ti/networkhome.htm

[134] R. A. Quinnell. (2007, July) Networking moves to home automation. [Online]. Available: http://www.edn.com/design/consumer/4315885/Networking-moves-to-home-automation

[135] S. C. Ahn, J.-W. Lee, K.-W. Lim, H. Ko, Y.-M. Kwon, and H.-G. Kim, “Requirements to upnp for robot middleware,” in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[136] D. Alliance. (2011) Dlna official website. Available: http://www.dlna.org/home. [Online]. Available: http://www.dlna.org/home

[137] S. C. Ahn, J.-W. Lee, K.-W. Lim, H. Ko, Y.-M. Kwon, and H.-G. Kim, “Upnp sdk for robot development,” in *SICE-ICASE International Joint Conference*, 2006, pp. 363–368.

[138] J. Jones, “Robots at the tipping point: the road to irobot roomba,” *Robotics & Automation Magazine, IEEE*, vol. 13, no. 1, pp. 76–78, March 2006.

[139] J. Forlizzi and C. DiSalvo, “Service robots in the domestic environment: a study of the roomba vacuum in the home,” in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, ser. HRI ’06. New York, NY, USA: ACM, 2006, pp. 258–265. [Online]. Available: http://doi.acm.org/10.1145/1121241.1121286

[140] J.-Y. Sung, L. Guo, R. E. Grinter, and H. I. Christensen, “”my roomba is rambo”: intimate home appliances,” in *Proceedings of the 9th international conference on Ubiquitous computing*, ser. UbiComp ’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 145–162. [Online]. Available: http://portal.acm.org/citation.cfm?id=1771592.1771601

[141] J. Forlizzi, “How robotic products become social products: an ethnographic study of cleaning in the home,” in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, ser. HRI ’07. New York, NY, USA: ACM, 2007, pp. 129–136. [Online]. Available: http://doi.acm.org/10.1145/1228716.1228734

[142] B. Tribelhorn and Z. Dodds, “Evaluating the roomba: A low-cost, ubiquitous platform for robotics research and education,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[143] T. E. Kurt, *Hacking Roomba: Extreme Tech.* Wiley, 2006.

[144] J. Elston, C. Dixon, M. Stachura, C. Eheim, and R. Witoff. (2003) Roomba pacman. [Online]. Available: http://pacman.elstonj.com/

[145] Infinuvo. (2011) Infinuvo official website. [Online]. Available: http://www.infinuvo.com/

[146] P. International. (2011) P3 international official website. [Online]. Available: http://www.p3international.com/

[147] iRobot. (2006) Roomba serial command interface (sci) specification. [Online]. Available: http://irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf

[148] WowWee. (2011) Wowwee group limited official website. WowWee Group Limited. [Online]. Available: http://www.wowwee.com

[149] S. Fladung and J. Mwaura, "Cs4758: Rovio augmented vision mapping project," in *CS4758*, 2010.

[150] M. Kandefer, "Cassie can speak: A .net interface to a robotic fevahr," in *Seminar on Cognitive Robotics*, Department of Computer Science and Engineering, University at Buffalo, 2009.

[151] H. Dang, J. Hundal, and R. Nachiappan, "Robot visual mapper," in *Robot Visual Mapper*, 2010.

[152] J. Bona and M. Prentice, "Pyrovio: Python api for wowwee rovio," 2009.

[153] J. Melville and T. Sams, "Thirsty rovio - autonomous mini-keg locating robot," 2010.

[154] OpenCV. (2011) Emgu cv: Open cv in .net. [Online]. Available: http://www.emgu.com

[155] WowWee. (2009) Rovio api specification for rovio (version 1.3). WowWee Group Limited. [Online]. Available: http://breckon.eu/toby/teaching/dip/rovio/Rovio_API_Specifications_v1.3.pdf

[156] Allegro. (2011) Romplug toolkits. [Online]. Available: http://www.allegrosoft.com/romplug.html

[157] Cybergarage. (2011). [Online]. Available: https://github.com/cybergarage/cybergarage-upnp

[158] GUPnP. (2011) gupnp official website. Disponible: http://www.gupnp.org/. [Online]. Available: http://www.gupnp.org/

[159] Atinav. (2011) Atinav official website. [Online]. Available: www.atinav.com

[160] Intel. Developer tools for upnp technologies. [Online]. Available: https://software.intel.com/en-us/blogs/2011/02/04/developer-tools-for-upnp-update

[161] Microsoft. (2010) Microsoft visual c# 2010 express. Disponible: http://www.microsoft.com/express/Windows/. [Online]. Available: http://www.microsoft.com/express/Windows/

[162] R. B. Pozo, J. de la Pinta, A. Álvarez, and J. Maestre. (2011) Integration of rovio in an upnp network. [Online]. Available: http://www.youtube.com/watch?v=K_d5BjlMOoA

[163] A. Kirillov. (2011) Aforge.net framework. [Online]. Available: http://code.google.com/p/aforge/

[164] R. B. Pozo, J. de la Pinta, A. Álvarez, and J. Maestre. (2011) Rovio tracks a ball. [Online]. Available: http://www.youtube.com/watch?v=0KMYmeIREgc

[165] R. B. Pozo, J. de la Pinta, A. Alvarez, and J. Maestre. (2011) Rovio tracks roomba (test 1). [Online]. Available: http://www.youtube.com/watch?v=9o41AyeNf_4

[166] R. B. Pozo, J. de la Pinta, A. Álvarez, and J. Maestre. (2011) Rovio tracks roomba (test 2). [Online]. Available: http://www.youtube.com/watch?v=xFqjwwSO8_o

[167] ——. (2011) Service robots integration over upnp scenario. [Online]. Available: http://www.youtube.com/watch?v=qWOWrw8MjYQ

[168] M. Chico, J. Maestre, and E. Camacho, "Upnp x10 sofware bridge," in *Iadis Multi Conference on Computer Science and Information Systems.*, 2008.

[169] R. Lobillo, J. Maestre, and E. Camacho, "Zigbee positioning system: applications to home automation (in spanish)," in *XXIX Jornadas de Automática*, Tarragona, 2008.

[170] J. Snape, J. van den Berg, S. Guy, and D. Manocha, "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009*, oct 2009, pp. 5917 –5922.

[171] WowWee. (2011) Rovio mobile webcam. [Online]. Available: http://www.wowwee.com/en/products/tech/telepresence/rovio/rovio

[172] MOBILEROBOTS. (2011) Mobilerobots official website. [Online]. Available: http://www.mobilerobots.com

[173] I. G. Alonso, O. Á. Fres, A. A. Fernández, P. G. del Torno, J. M. Maestre, and M. A. G. Fuente, "Towards a new open communication standard between homes and service robots, the dhcompliant case," *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 889–900, 2012.

[174] D. Compliant. (2010) Dhc-groups draft specification for the teamwork between robots. version 1.0. Available: http://156.35.46.38/data/files/Collaborative/DHC-Groups%201.0.pdf. [Online]. Available: http://156.35.46.38/data/files/Collaborative/DHC-Groups%201.0.pdf

[175] ——. (2010) Dhc-localization draft specification for the robot localization. version 1.0. Available: http://156.35.46.38/data/files/Localization/DHC-Localization%201.0.pdf. [Online]. Available: http://156.35.46.38/data/files/Localization/DHC-Localization%201.0.pdf

[176] ——. (2010) Dhc-energy draft specification for energy management and smart grids. version 1.0. Available: http://156.35.46.38/data/files/Energy%20Management/DHC-Energy%201.0.pdf. [Online]. Available: http://156.35.46.38/data/files/Energy\%20Management/DHC-Energy%201.0.pdf

[177] ——. (2010) Dhc-rules draft specification for checking rules. version 1.0. Available: http://156.35.46.38/data/files/Intelligence/Rules/DHC-Rules%201.0.pdf. [Online]. Available: http://156.35.46.38/data/files/Intelligence/Rules/DHC-Rules%201.0.pdf

[178] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[179] D. Compliant. (2010) Dhc-security & privacy draft specification for data protection, user data privacy and access restriction. version 1.0. Available: http://156.35.46.38/data/files/SecurityPrivacy/DHC-Security&Privacy%201.0.pdf. [Online]. Available: http://156.35.46.38/data/files/SecurityPrivacy/DHC-Security&Privacy%201.0.pdf

[180] B. P. Gerkey and M. J. Matarić, “Sold!: Auction methods for multi-robot coordination,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, oct 2002.

[181] A. A. Fernández, O. A. Fres, I. G. Alonso, and H. Hu, “Visual localisation of mobile devices in an indoor environment under network delay conditions,” *International Journal of Distributed and Parallel Systems (IJDPS)*, vol. 2, no. 2, March 2011. [Online]. Available: http://arxiv.org/abs/1103.5554

[182] A. A. Vazquez, I. G. Alonso, and M. A. G. Fuente, “Performance analysis of a upnp/dhcompliant robotic adapter for collaborative tasks development,” *International Journal of Distributed and Parallel Systems (IJDPS)*, vol. 3, pp. 1–14, 2012.

[183] D. Compliant2. (2012) Dh compliant 2. Available: http://dhcompliant2.com/. [Online]. Available: http://dhcompliant2.com/

[184] J. R. de la Pinta, J. M. Maestre, I. Jurado, and S. Reyes, “Off the shelf cloud robotics for the smart home: empowering a wireless robot through cloud computing,” 2017.

[185] G. Xiao, J. Guo, L. D. Xu, and Z. Gong, “User interoperability with heterogeneous iot devices through transformation,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1486–1496, May 2014.

[186] J. M. Maestre, *Domótica para ingenieros.* Paraninfo, 2015.

[187] Q. G. Services. (2015) Upnp in digital home networking. [Online]. Available: https://www.quest-global.com/wp-content/uploads/2015/08/UPnP-in_Digital_Home_Networking.pdf

[188] J. de la Pinta, A. Álvarez, J. Maestre, and I. Alonso, “Collaborative tasks between robots based on the digital home compliant protocol over upnp,” *Journal of Intelligent & Robotic Systems*, vol. 72, no. 3-4, pp. 357–371, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10846-012-9801-7

[189] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, “Miro - middleware for mobile robot applications,” *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 493–497, 2002.

[190] H. Bruyninckx, "Orocos: Design and implementation of a robot control software framework," in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington D.C., April 2002, pp. 1–9.

[191] C. Schlegel, "A component approach for robotics software: Communication patterns in the orocos context," in *Autonome Mobile Systeme 2003*, ser. Informatik aktuell, R. Dillmann, H. Wörn, and T. Gockel, Eds. Springer Berlin Heidelberg, 2003, pp. 253–263. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18986-9_26

[192] M. Baeg, J.-H. Park, J. Koh, K.-W. Park, and M.-H. Baeg, "Robomaidhome: A sensor network-based smart home environment for service robots," in *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, 2007, pp. 182–187.

[193] S. Rastkar, D. Quintero, D. Bolivar, and S. Tosunoglu, "Empowering robots via cloud robotics: Image processing and decision making boebots," in *Conference on Recent Advances in Robotics*, Florida, May 2012.

[194] M. Matskin, "Services, clouds and robots," in *In The Sixth International Conference on Internet and Web Applications and Services (ICIW 2011)*, St. Maarten, The Netherlands Antilles, March 2011.

[195] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, April 2015.

[196] J. Quintas, P. Menezes, and J. Dias, "Cloud robotics: Towards context aware robotic networks." in *In Proc. of the 16th IASTED International Conference on Robotics*, Pittsburgh (USA), November 2011.

[197] G. Hu, W.-P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.

[198] E. Guizzo. (2011, January) Cloud robotics: Connected to the cloud robots get smarter. Available: http://spectrum.ieee.org/automaton/robotics/robotics-software/cloud-robotics.

[199] ——, "Robots with their heads in the clouds," *IEEE Spectrum*, 2011.

[200] L. Lopez. (2013, Mayo) Rovio control plugin from z-wave gateway vera lite. Available: http://domotica4all.com/2013/05/plugin-of-rovio-control-from-vera-lite/. Authorship of the plugin: Sergio Reyes Cozar.

[201] Google. (2016) Google cloud speech. Available: https://cloud.google.com/speech/.

[202] J. R. de la Pinta and J. M. Maestre. (2016) Teddy bear search experiment. Available: https://www.youtube.com/watch?v=Z-jwaY_pUjA&feature=youtu.be.

[203] ——. (2016) New object for search service experiment. Available: https://www.youtube.com/watch?v=ZEcCFR_bs0s&feature=youtu.be.

[204] ——. (2016) Z-wave + rovio integration experiment. Available: https://www.youtube.com/watch?v=rUDAxxSrGDc&feature=youtu.be.

[205] M. J.M. and de la Pinta J.R., *UPnP (Domótica para ingenieros)*. Paraninfo, March 2015, ch. 14.

[206] J. de la Pinta, C. Martín-Macareno, A. Álvarez, and J. Maestre, "Smoke detectors: Development of an alarm management system for upnp," in *Workshop International Technology Robotics Applications, INTERA 2011*, Oviedo, February 2011.

[207] S. P. Ruiz, J. de la Pinta, J. Maestre, and E. Camacho, "Localización de robot móvil mediante zigbee," in *XXXI Jornadas de Automática*, vol. 31, no. 31. Jaén: CEA-IFAC, 2010.