SEVENTH FRAMEWORK PROGRAMME	**Project Acronym: GiraffPlus** **Project Title: Combining social interaction and long term monitoring for promoting independent living** **Grant agreement no.: 288173** **Starting date: 1st January 2012** **Ending date: 31st December 2014**

GIRAFF PLUS

D2.4 Final version of sensors and GiraffPlus platform report

WP related to the Deliverable:	2
Nature:	P
Dissemination Level:	PU
Version:	0.5
Author(s):	Filippo Palumbo (CNR-ISTI), Davide La Rosa (CNR-ISTI), Erina Ferro (CNR-ISTI), Francesco Potortì (CNR-ISTI), Paolo Barsocchi (CNR-ISTI), Michele Girolami (CNR-ISTI), Cipriano Galindo (UMA), Mats Björkman (MDH), Maria Lindén (MDH), Stephen Von Rump (Giraff)
Project Participant(s) Contributing:	CNR-ISTI, UMA, XLAB, MDH, Giraff
Contractual Date of Delivery:	01/01/2015
Actual Date of Delivery:	11/01/2015

Document History

Version	Date	Type of editing	Editorial
0.0	28/11/14	Table of Content and Initial Draft	CNR-ISTI
0.1	10/12/14	Section 2 completed	CNR-ISTI
0.2	12/12/14	Table of Content modified (MDH input)	CNR-ISTI
0.3	19/12/14	MDH contribution (Section 3 and Appendix)	CNR-ISTI
0.4	24/12/14	UMA contribution to Section 4	CNR-ISTI
0.5	04/01/15	Complete version before internal review	CNR-ISTI
1.0	06/01/15	Stable version after internal review	CNR-ISTI
1.1	11/01/15	Additional contributions integrated	CNR-ISTI

Disclaimer:

No confidential material is included therein.

Deliverable Summary

This document reports on the final version of the sensor network and the GiraffPlus integrated system.

The final version delivered at month 36 incorporates the input from the last evaluation phase leading to a refined system.

The final version of the middleware infrastructure integrates a more complete set of sensors and functionalities. Indeed, the control bus functionality has been implemented and tested with new sensors and actuators, a new wearable technology capable of continuous heart rate monitoring has been analyzed, and a new web monitoring tool for checking the status of the test sites has been deployed. The final version of the Giraff platform is also described, addressing safety issues and general improvements requests coming from the last evaluation phase.

In addition, final considerations on the assessment of security and safety in the GiraffPlus system are also presented as an appendix.

Table of Content

1 INTRODUCTION 5

1.1 SCOPE OF THE DOCUMENT 5
1.2 DELIVERABLE STRUCTURE 5
1.3 DEVIATIONS WITH RESPECT TO THE PLAN 6

2 FINAL VERSION OF THE MIDDLEWARE 8

2.1 CONTROL BUS: IMPLEMENTATION AND TESTING OF ACTUATORS 8
2.2 DESKTOP AND MOBILE IMPLEMENTATION ENHANCEMENTS 11
2.3 REAL-TIME WEB MONITORING TOOL 12

3 FINAL SENSOR SELECTION AND INTEGRATION 14

3.1 ANDROID-BASED PHYSIOLOGICAL AND INERTIAL SENSOR 14
3.2 Z-WAVE ENVIRONMENTAL SENSOR NETWORK 16
3.3 WEARABLE SENSORS 19
3.3.1 Android wear platform 19
3.3.2 Heart rate monitor and accuracy testing 20

4 FINAL VERSION OF THE GIRAFF PLATFORM 24

4.1 IMPROVEMENT OF THE GIRAFF MOBILITY 24
4.1.1 Robotic platform safety issues 27
4.1.2 General improvements based on the reviewers' evaluation 29
4.1.3 Automatic docking based on laser sensor 30
4.2 THE NEW FEATURES OF THE GIRAFF HARDWARE AND SOFTWARE 34
4.3 THE DVPIS@HOME PLUGIN 35
4.3.1 Software architecture 36
4.3.2 Implementation details 37

5 CONCLUSION 39

APPENDIX 1 - ASSESSMENT OF SECURITY AND SAFETY IN THE FINAL VERSION OF THE GIRAFFPLUS SYSTEM 40

REFERENCES 42

List of Figures

Figure 1 Final middleware architecture 8
Figure 2 Class diagram with the new parts highlighted in red 9
Figure 3 Invocation of a command from a user application 10
Figure 4 Invocation of a command on the receiving side 11
Figure 5 The web page showing the homes status 12
Figure 6 The web page showing the tablets status 13
Figure 7 Additional Intellicare devices (the thermometer on the left and the spirometer on the right) 16
Figure 8 An example of Z-Wave sensors mesh network 17
Figure 9 The handheld sends a message to the wearable using the sendMessage method. On the receiving side, a WearableListenerService monitors the data layer and invokes the onMessageReceived callback when a message arrives 19
Figure 10 The PutDataMapRequest helper class simplifies the process of creating the DataMap, DataItem, and PutDataRequest objects that are needed for sending data. On the receiving side a WearableListener Service receives changed data and returns a buffer of DataEvents containing DataItem to be converted into DataMapItem containing the original handheld data 20
Figure 11 The Samsung Gear Live smartwatch 21
Figure 12 Heartbeat and accelerometer data while sleeping and walking 21
Figure 13 Heart rate data from the Samsung Gear Live and the Garmin Premium heart rate monitor compared (upper graph) while moving the wrist (bottom graph showing accelerometer data) when securely tightened 22
Figure 14 No heart rate obtained when moving the wrist while the smartwatch is not perfectly tightened 22
Figure 15 Erroneous heart rate (~20 heartbeats error) obtained when moving the wrist while the smartwatch is not perfectly tightened 23
Figure 16 Result of an autonomous navigation experiment carried out in a controlled environment: the Mapir Laboratory 25
Figure 17 Overview of the D2.4 prototype and its features 26
Figure 18 Taxonomy of failures. Engineering errors, human mistakes and environment conditions are the sources of system failures 28
Figure 19 Signals exchanged between different systems involved in telepresence platform 29
Figure 20 Simplified flow chart of the Auto-Docking module 31
Figure 21 Detailed views of the laser scans when detecting the pattern used for the auto-docking module. (a) Raw laser scan where the three sticks composing the pattern can be visually appreciated. (b-c) Colored pattern within the laser scans pointing the location and distance from the robot 32
Figure 22 Description of the robot movements when approaching the pattern. In phase 1 the robot will perpendicularly align to the pattern, while in phase 2 the robot will slowly advance straight 33
Figure 23 Schematic view of the pattern used in the auto-docking module 33
Figure 24 Auto-docking pattern implementation made of cardboard 34
Figure 25 DVPIS@Home architecture 37

1 Introduction

1.1 Scope of the document

The document is a progress report with respect of the D2.3 and describes the final version of middleware, sensors, Giraff robot, and the GiraffPlus integrated system. The refined version of the middleware infrastructure is provided together with final version of the Giraff platform.

There have been improvements on all the parts of the system and additional features have been developed in order to reflect the results of the evaluation of the system in the test sites. In particular, the middleware has been refined in its desktop and mobile adaptation, allowing the actuation of command by devices and services, a better management of the local backup of data, and the real-time monitoring of the status of all the middleware instances installed in the homes via a web monitoring tool. A new environmental sensory technology, based on a well-known standard, has also been integrated in order to guarantee flexibility in the GiraffPlus system when dealing with replacement or finding new suitable sensors for specific needs (e.g. monitor energy consumption [1]). This kind of sensors has been proved useful when exploited for indoor user localization purpose. Tests were made on this application scenario and results were published in an international conference [2] and a journal [3].

The use of a new wearable device embedding the new Android Wear operating system has also been investigated and its integration is presented in this deliverable. The device comes with a heart rate meter together with an inertial system like the previously analyzed devices (D2.3). Accuracy and robustness testing has been performed and the results are presented. This new technology, released on the market on July 2014, has been chosen in order to enable the GiraffPlus system of new possible future services, like fatigue detection or energy expenditure, by means of a continuously monitoring of the primary user's heart rate.

The Giraff platform in its improvements is presented addressing safety issues coming from the evaluation phase in the test sites, allowing a better mobility and featuring an automatic docking feature. These results are also described in an international journal [4].

Finally, an appendix describes the assessment of how the final GiraffPlus system fulfills the requirements of safe and secure communication as defined in D1.3. A considerable effort has been spent during this period on providing stability and robustness of the system. The document describes also the results of the tests performed.

1.2 Deliverable structure

The document starts with an outline of what has been achieved and how it matches the DoW. In section 2 the final version of the middleware is described, section 3 focuses on the final sensor selection with an insight on the possibilities offered by new wearable devices available on the market, while section 4 is dedicated to the final version of the Giraff platform and to improvements of its mobility capabilities. Finally, appendix 1 describes the assessment of how the final GiraffPlus system fulfills the requirements of safe and secure communication as defined in D1.3. Each section presents the effort made in terms of new features, tests, and bug fixing.

1.3 Deviations with respect to the plan

The project proceeded according to the plan outlined in the DoW with no deviations. In particular, Table 1 shows how the requirements of the DoW are matched to what has been achieved by month 36 in the project.

Table 1 Matching requirements from DoW with results achieved at M36

Required from the DoW	Achieved at month 36
Task 2.2 Middleware design and implementation: The final version of the middleware is refined	Section 2 The refined version of the middleware is described in its desktop and mobile adaptation. The mobile version of the middleware has been published on an international conference. The new features regarding the implementation of the control bus are presented in section 2.1, the enhancements made to the desktop and mobile version are described in section 2.2, while the new tool for real-time monitoring of the middleware instances status is detailed in section 2.3
Task 2.3 Development of Giraff robot platform: The final version of the platform incorporating the input from the last evaluation phase is ready	Section 4 The new Giraff avatar (version 4.1), the version 3.0 of the visitor “Pilot” and Giraff avatar software, and the sentry management system enhancements (version 2.0) are presented. The new features of the Giraff hardware and software are described in section 4.2.
Task 2.4 Additional sensor selection and design: The final version of the sensor system incorporating the input from the last evaluation phase is ready	Section 3 The Android-based physiological and inertial sensor already deployed in the test sites has been revised in order to address the results of the last evaluation phase and described in section 3.1. The Z-Wave technology has been also integrated in the system and devices able

	to perform actuation have been selected and presented in section 3.2 in order to show the potentialities of the control bus feature of the middleware. Also a new device embedding the Android wear operating system has been analyzed in its heart rate monitor capabilities in section 3.3 focusing on robustness and accuracy of the sensor.
Task 2.5 Enabling safe and secure communication: Final version is ready	Appendix 1 The assessment of how the final GiraffPlus system fulfills the requirements of safe and secure communication defined in D1.3 is presented in appendix 1.
Task 2.6 Improvement of Giraff mobility: The final version of the platform incorporating the input from the last evaluation phase is ready	Section 4 The final prototype of the Giraff robot featuring the semiautonomy abilities detailed in the DOW and developed along the project is described in section 4.1. Section 4.1 describes the improvement of Giraff mobility according to the last evaluation phase. In particular, safety issues have been addressed (section 4.1.1), general improvements based on the last evaluation period are presented in 4.1.2, and the new automatic docking feature based on laser data is described.

2 Final version of the middleware

The final version of the middleware includes both the additional implementation of features and the consolidation of the already developed parts in order to have a fully working software infrastructure. In the following sections the improvements with respect to the previous deliverable D2.3 will be presented in detail, highlighting the aspects and the functionalities of the additional components.

2.1 Control bus: implementation and testing of actuators

The main enhancement to the middleware is the implementation of the capability to send remote commands to devices that act as actuator, for instance to switch on/off lights and plugs or to set light levels, etc.

The commands, properly encoded in a JSON format, are sent on the control bus, which was previously used to convey the heartbeat signal from the active homes (D2.3 Section 3.2.2). A separate sub-topic is used to distinguish the heartbeat messages from the actuator operations.

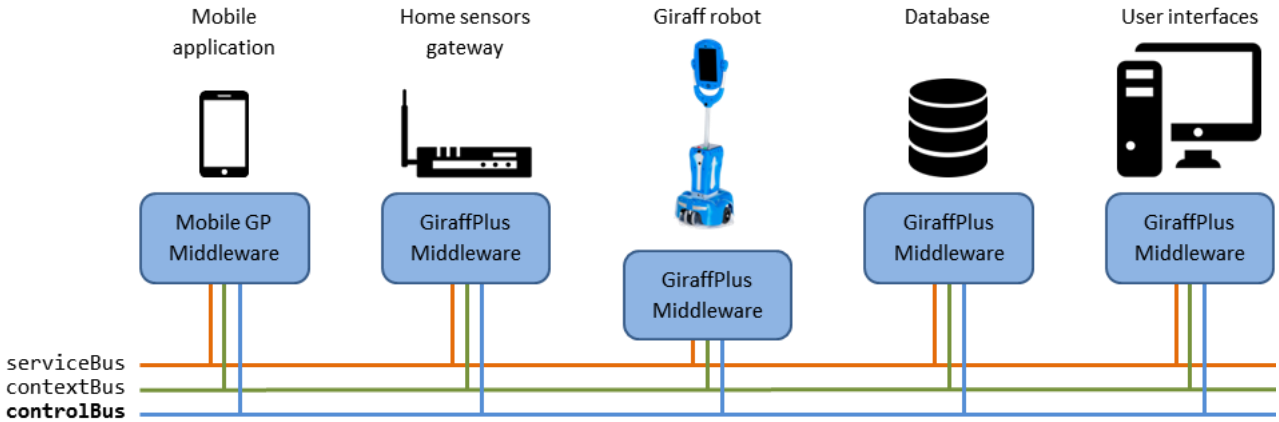


Figure 1 Final middleware architecture

A final view of the logical architecture is shown in Figure 1. The control bus is currently organized as follows:

- Heartbeat messages are sent on `<location>/controlBus/heartbeat/<instance>`
- Command messages are sent on `<location>/controlBus/command/<sensor>`

In order to allow a sensor to announce its actuators capabilities, the *ServiceDescriptor* class has been extended with the additional *controlBusTopic* and *commands* properties. Commands are described through a *CommandDescriptor* containing the name and the parameter definitions (name and type) of the offered operation.

A class diagram showing the main middleware components with the added parts is shown in Figure 2.

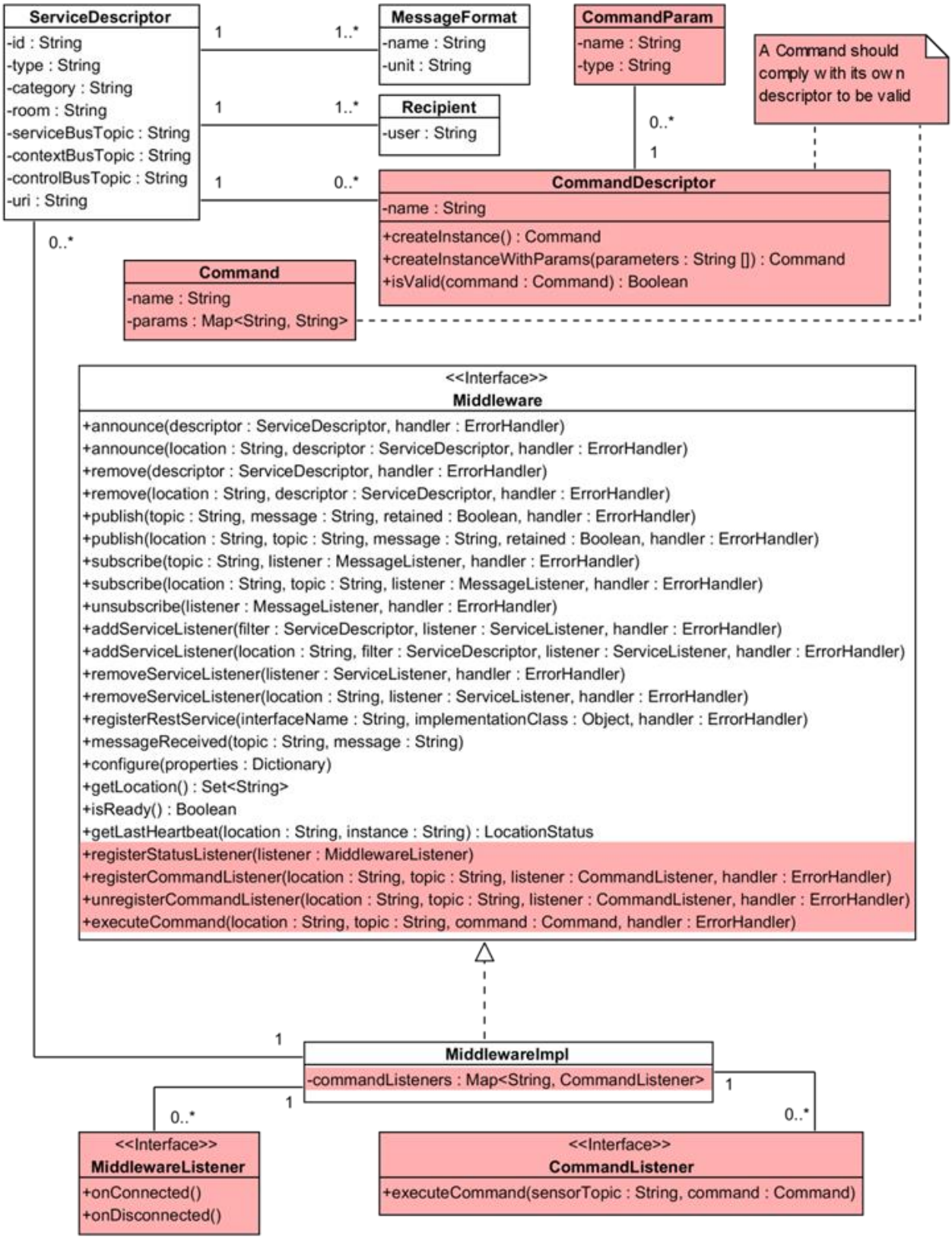


Figure 2 Class diagram with the new parts highlighted in red

The *Middleware* interface has been extended and some additional classes have been defined. The new *Middleware* methods are:

- `public void executeCommand (String location, String topic, Command command, ErrorHandler handler)`
  This method is used by a controller entity to invoke a command on a given device identified by its *topic* and placed at a remote *location*.
- `public void registerCommandListener (String location, String topic, CommandListener listener, ErrorHandler handler)`
  This method is used by an actuator entity to register a listener that will react to a command request directed to a sensor identified by its *topic* on a given *location*.
- `public void unregisterCommandListener (String location, String topic, CommandListener listener, ErrorHandler handler)`
  This method is complementary of the previous one and is used to remove a previously registered command listener.

When a client finds a remote sensor that can act as an actuator and wants to send a command to it, it calls the *executeCommand()* method passing an instance of the *Command* class loaded with the proper name and parameter values as defined in the relative descriptor (Figure 3). The middleware, after validating the command against the descriptor, translates the object into a JSON plain text message and sends it on the control bus.

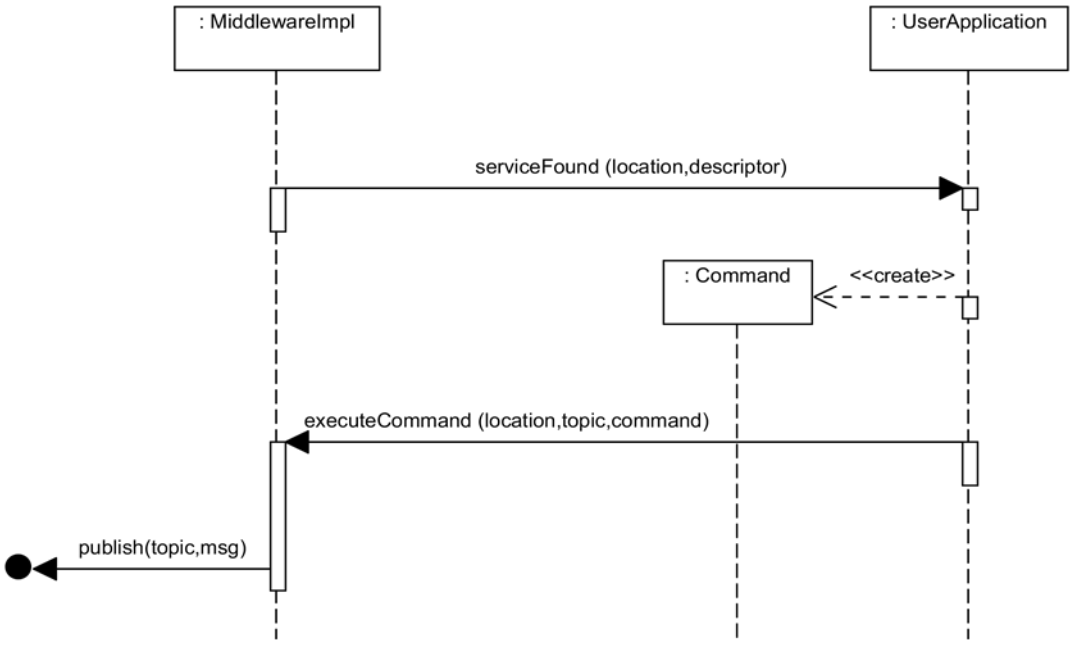


Figure 3 Invocation of a command from a user application

On the receiving side, the middleware assembles a new Command object from the received JSON file, and calls the listener registered for the target sensor through the interface *CommandListener* (Figure 4).

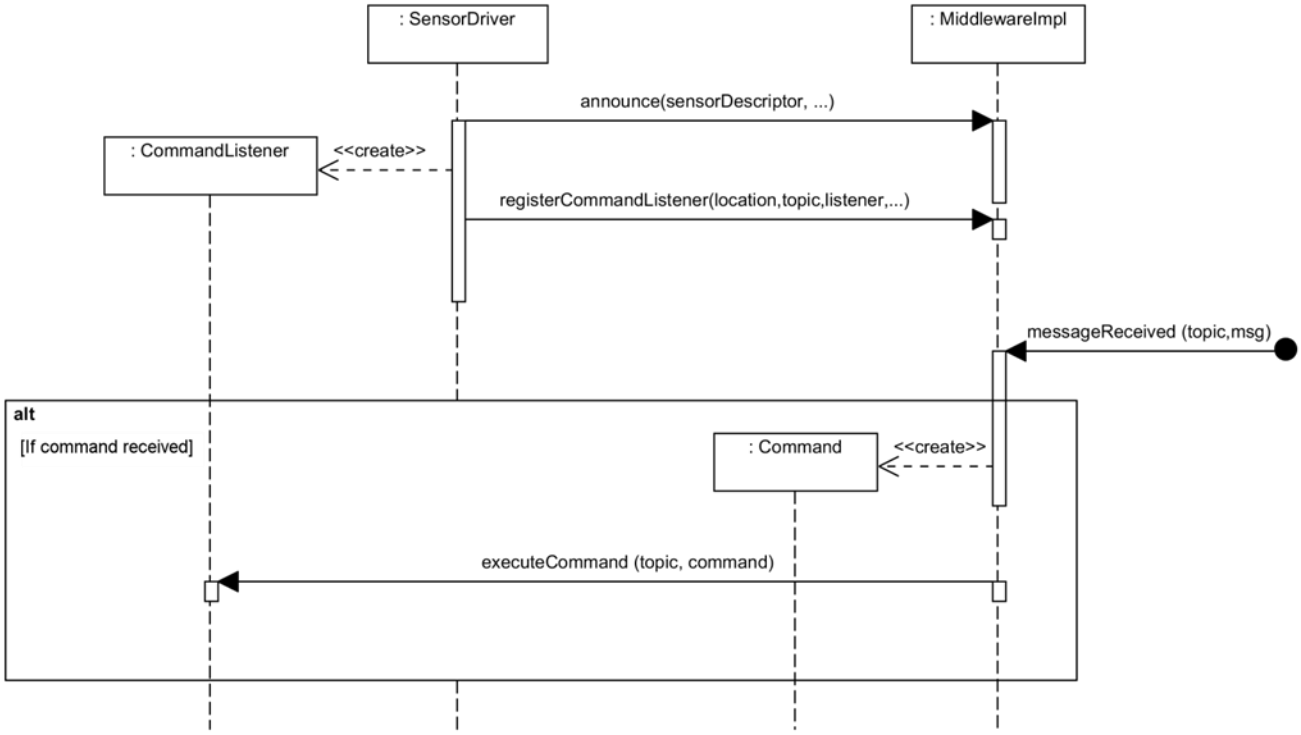


Figure 4 Invocation of a command on the receiving side

2.2 Desktop and mobile implementation enhancements

Since the last deliverable released at month 30, several enhancements have been implemented in the last six months in order to increase the robustness of the system, addressing the requests raised by other developers and internal continuous testing through the Mantis bug tracker tool (see D6.3 for details on the tool).

The desktop version is now able to notify user applications about connection problems to GiraffPlus external services. This is achieved by registering a listener *MiddlewareListener* that is notified whenever the connection drops or is re-established.

The callback executions are now run in separate threads with relative exception management. In this way, potential faulty user code cannot compromise the stability of the main modules.

A rare situation in which messages could be lost (if queued in the interval between a connection failure and the consequent notification to the middleware) has been discovered and fixed by strengthening the error handling procedures.

The local backup mechanism, described in D2.3, has also been improved: when the remote connection is re-established, messages published and buffered on the *serviceBus* are sent to the broker, while those regarding the *contextBus* are directly sent and stored on the remote database.

The mobile version of the middleware underwent a constant maintenance and, as an outcome, improvements were made:

- The binding/unbinding procedure between the middleware and the *MQTT communication connector* has been reinforced to increase the reliability of the apps update process.
- The *WiFi* module is automatically reset if the connection to the GiraffPlus server is not possible. This, to recover from issues related to the radio interface of the device.

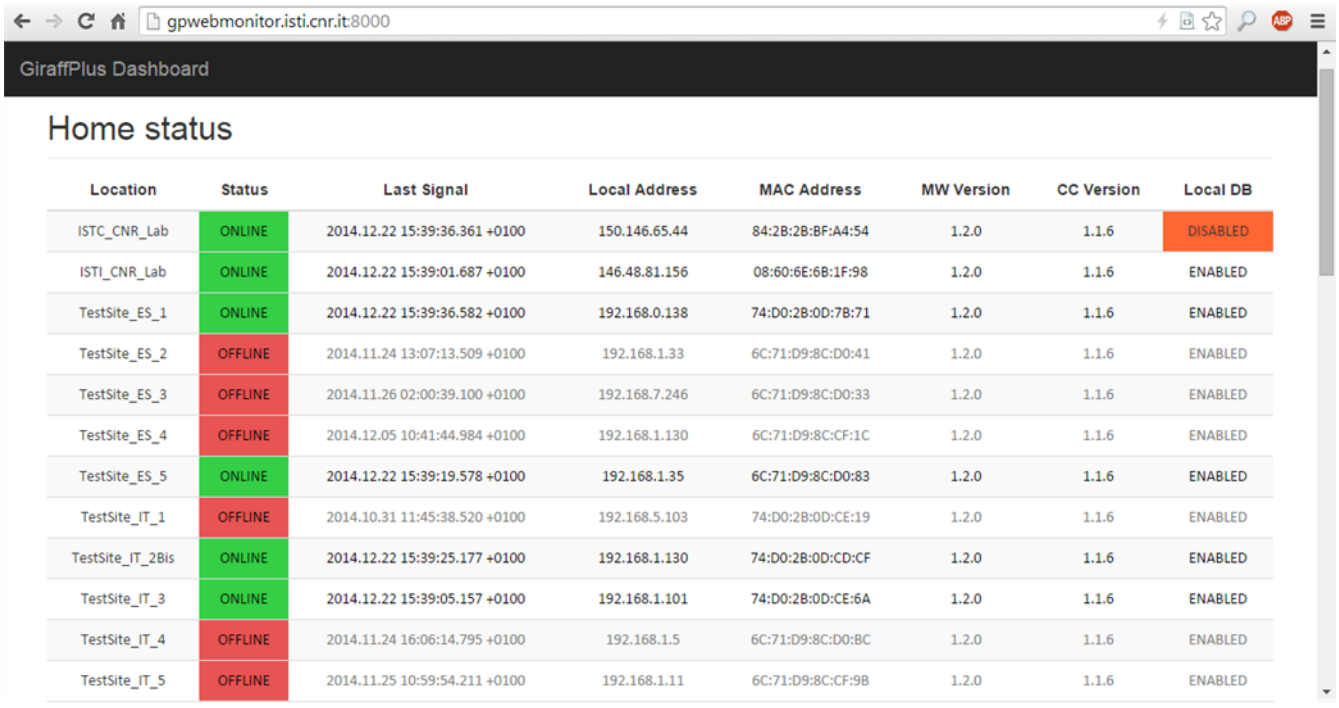
2.3 Real-time web monitoring tool

In order to increase the system reliability and recovery capabilities, a continuous 24/7 service for monitoring the running test sites status has been implemented.

The service collects the live on-site heartbeat sent every 60 seconds by both the tablets and the home gateways and, in case of more than 6 hours without any signal (an interval chosen to filter out temporary or short network outages, considering only the serious ones), it sends an email to all the registered engineers who are involved on the particular test site prompting for an urgent check.

The service also runs a compact open source HTTP webserver based on Jetty[1] that provides a web page currently hosted on the CNR servers in Pisa showing the real-time condition of the homes (Figure 5) and the tablets (Figure 6). The webpages are available at

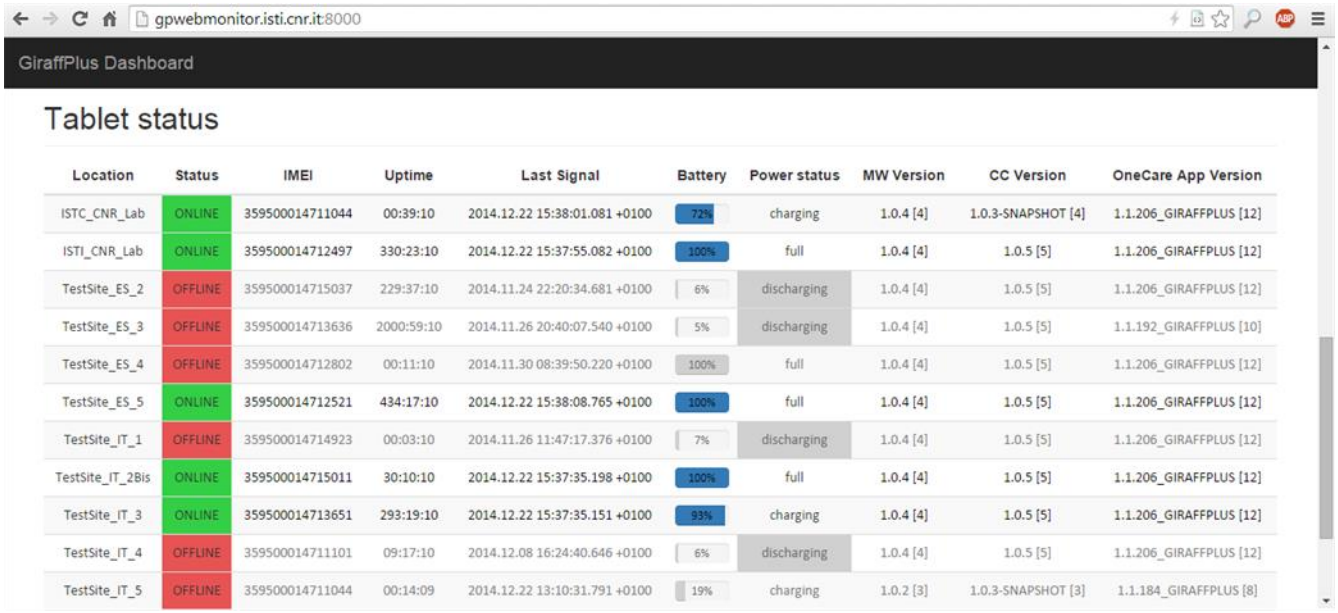
http://gpwebmonitor.isti.cnr.it:8000



Location	Status	Last Signal	Local Address	MAC Address	MW Version	CC Version	Local DB
ISTC_CNR_Lab	ONLINE	2014.12.22 15:39:36.361 +0100	150.146.65.44	84:2B:2B:BF:A4:54	1.2.0	1.1.6	DISABLED
ISTI_CNR_Lab	ONLINE	2014.12.22 15:39:01.687 +0100	146.48.81.156	08:60:6E:6B:1F:98	1.2.0	1.1.6	ENABLED
TestSite_ES_1	ONLINE	2014.12.22 15:39:36.582 +0100	192.168.0.138	74:D0:2B:0D:7B:71	1.2.0	1.1.6	ENABLED
TestSite_ES_2	OFFLINE	2014.11.24 13:07:13.509 +0100	192.168.1.33	6C:71:D9:8C:D0:41	1.2.0	1.1.6	ENABLED
TestSite_ES_3	OFFLINE	2014.11.26 02:00:39.100 +0100	192.168.7.246	6C:71:D9:8C:D0:33	1.2.0	1.1.6	ENABLED
TestSite_ES_4	OFFLINE	2014.12.05 10:41:44.984 +0100	192.168.1.130	6C:71:D9:8C:CF:1C	1.2.0	1.1.6	ENABLED
TestSite_ES_5	ONLINE	2014.12.22 15:39:19.578 +0100	192.168.1.35	6C:71:D9:8C:D0:83	1.2.0	1.1.6	ENABLED
TestSite_IT_1	OFFLINE	2014.10.31 11:45:38.520 +0100	192.168.5.103	74:D0:2B:0D:CE:19	1.2.0	1.1.6	ENABLED
TestSite_IT_2Bis	ONLINE	2014.12.22 15:39:25.177 +0100	192.168.1.130	74:D0:2B:0D:CD:CF	1.2.0	1.1.6	ENABLED
TestSite_IT_3	ONLINE	2014.12.22 15:39:05.157 +0100	192.168.1.101	74:D0:2B:0D:CE:6A	1.2.0	1.1.6	ENABLED
TestSite_IT_4	OFFLINE	2014.11.24 16:06:14.795 +0100	192.168.1.5	6C:71:D9:8C:D0:BC	1.2.0	1.1.6	ENABLED
TestSite_IT_5	OFFLINE	2014.11.25 10:59:54.211 +0100	192.168.1.11	6C:71:D9:8C:CF:9B	1.2.0	1.1.6	ENABLED

Figure 5 The web page showing the homes status

[1] http://eclipse.org/jetty/



Location	Status	IMEI	Uptime	Last Signal	Battery	Power status	MW Version	CC Version	OneCare App Version
ISTC_CNR_Lab	ONLINE	359500014711044	00:39:10	2014.12.22 15:38:01.081 +0100	72%	charging	1.0.4 [4]	1.0.3-SNAPSHOT [4]	1.1.206_GIRAFFPLUS [12]
ISTI_CNR_Lab	ONLINE	359500014712497	330:23:10	2014.12.22 15:37:55.082 +0100	100%	full	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_ES_2	OFFLINE	359500014715037	229:37:10	2014.11.24 22:20:34.681 +0100	6%	discharging	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_ES_3	OFFLINE	359500014713636	2000:59:10	2014.11.26 20:40:07.540 +0100	5%	discharging	1.0.4 [4]	1.0.5 [5]	1.1.192_GIRAFFPLUS [10]
TestSite_ES_4	OFFLINE	359500014712802	00:11:10	2014.11.30 08:39:50.220 +0100	100%	full	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_ES_5	ONLINE	359500014712521	434:17:10	2014.12.22 15:38:08.765 +0100	100%	full	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_IT_1	OFFLINE	359500014714923	00:03:10	2014.11.26 11:47:17.376 +0100	7%	discharging	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_IT_2Bis	ONLINE	359500014715011	30:10:10	2014.12.22 15:37:35.198 +0100	100%	full	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_IT_3	ONLINE	359500014713651	293:19:10	2014.12.22 15:37:35.151 +0100	93%	charging	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_IT_4	OFFLINE	359500014711101	09:17:10	2014.12.08 16:24:40.646 +0100	6%	discharging	1.0.4 [4]	1.0.5 [5]	1.1.206_GIRAFFPLUS [12]
TestSite_IT_5	OFFLINE	359500014711044	00:14:09	2014.12.22 13:10:31.791 +0100	19%	charging	1.0.2 [3]	1.0.3-SNAPSHOT [3]	1.1.184_GIRAFFPLUS [8]

Figure 6 The web page showing the tablets status

Together with the location names, other useful information is shown:

- current status
- last received heartbeat
- local IP and MAC address of the remote machine
- software versions
- local backup database status

For the tablet, the uptime is shown together with the battery level, while the IMEI code (an unique device identifier) replaces the MAC and IP addresses.

3 Final sensor selection and integration

In this section the final version of the sensor system incorporating the input from the evaluation phase in the test sites is described. In particular, the android-based physiological and inertial sensor already deployed in the test sites has been further tested and enhanced fulfilling the request of having additional control functionalities, including wireless network availability, average activity check and "switch on/off status" notifications. These requests have been reported by secondary users on the Mantis bug tracker and taken into account in the development.

The Z-Wave technology has also been integrated in the system and devices able to perform actuation have been selected and presented. The devices have been chosen in order to show the potentialities of the control bus feature of the middleware.

Also a new device embedding the Android wear operating system has been analyzed in its heart rate monitor capabilities. Tests focusing on robustness and accuracy of the sensor have been performed and presented in the next subsections.

3.1 Android-based physiological and inertial sensor

A pulse oximeter sensor and an acceleration sensor combined into a monitoring system based on a smart-phone android device are available in the apartment for the user evaluation. The system publishes data via an android interface and using the GiraffPlus middleware store the data in the GiraffPlus database where they are accessible to the rest of the system.

Several simultaneous tasks were performed in the direction of integrating android-based sensor system into the GiraffPlus environment:

- Evaluating fall detection algorithm
- Testing connection to the GiraffPlus database
- Updating the system with the latest version of the android middleware
- Developing reasoning architecture for context integration

Before integrating into the system, evaluation of the fall detection algorithm was performed, resulted in the conference publication entitled "Evaluation of the android-based fall detection system for elderly people". On the next stage the android version of the middleware was installed and the communication between the system and GiraffPlus database has been established.

After preliminary discussion concerning the on-going context recognition development, it was decided to provide the system with the following parameters:

- Current level of users' activity
- Users presence in the apartment
- Fall emergency alarm

Eventually, these parameters have been substituted with a high-level reasoning algorithm combining context information and acceleration-based fall detection into a more reliable joint

system. As a simple example we can describe one of the following scenarios based on this approach: context recognition will be calculating the fall risk probability during the monitoring process, based on the symbolic representation of the contextual data. Once the fall alarm is triggered on the phone, a simple check is performed to see if the current user's activity represents a high risk of fall. In this way, we confirm the android-based algorithm and increase reliability of the system. More scenarios and data fusion options are currently being investigated.

A new android version of the middleware was successfully installed into the mobile phone. As the next step, the Android-based sensor platform was integrated into the GiraffPlus platform. As a result, *MDHPulseOximeter* and *MDHFallDetector* were added to the system (via Engineering GUI) as complementary sensors. Additional control functionalities were introduced, including wireless network availability, average activity check and "switch on/off status" notifications.

A separate application has been developed to perform activity monitoring and fall detection with alarm notification functionally based on *Pushover*[2] and GiraffPlus middleware. The user is invited to choose the communication option depending on monitoring circumstances.

Subsequently, a number of real-life tests were initiated to perform continuous monitoring of physiological parameters and to assess both entities in terms of user acceptance, data collection and communication reliability. The application has been tested and integrated into the monitoring process.

With latest improvements, complementary sensors are able to communicate with caregivers directly in case fall alarm has been triggered. Subsequently, a number of real-life tests were initiated to perform continuous monitoring of physiological parameters and to assess both entities in terms of user acceptance, data collection and communication reliability. Both sensor functionalities were involved into the current research towards context aware fall risk assessment, resulting in a journal publication [5].

Various modifications of the android based monitoring system were developed during the latest phase of the GiraffPlus project. The standard modifications implied integration the mobile-based system into the global system monitoring through the special middleware application developed especially for android devices. All the measurements (pulse rate, oxygen saturation, body movement, fall event) were extracted on a continuous basis, automatically synced with the system and become available to all type of users on-line via DVPIS.

Special requirements were discussed while working with different test groups in Italy and Spain. Based on several requests a new modification of the system was developed in order to split pulse rate monitoring and fall event monitoring into two parallel processes. A special focus was made on simplification of the user interface to make the monitoring system available for elderly users.

Unlike the first version of the program, the latest application is not integrated via GiraffPlus middleware into the global database, but generates direct notifications to the secondary users in case of abnormal situation or fall event detected during the monitoring. The latest version of the system was successfully tested at one of the test sites in Sweden in collaboration with a local

[2] https://pushover.net/

homecare facility. Both modifications can be run simultaneously on the same device if required and be a part of the global monitoring system.

In addition to the above solution, Intellicare selected and integrated new sensors: thermometer and spirometer (shown in Figure 7). Both devices were chosen after a careful selection process where multiple technical and business parameters were evaluated. The key selection criteria were if the devices were fit for purpose, protocol availability and unit cost, both for prototypes and large scale deployment.

Figure 7 Additional Intellicare devices (the thermometer on the left and the spirometer on the right)

The thermometer selected, shown in, is supplied by HUAYI, Shenzhen Huayi Instrument Technology Co. Ltd. It's a general purpose infrared thermometer with Bluetooth connectivity with CE certification. The protocol was made available by the supplier.

The spirometer is used in monitoring of chronic obstructive pulmonary disease (COPD). The device used is supplied by Vitalograph, again after as careful selection process using the same parameters. The key difference to the thermometer is that Vitalograph supplied an SKD to facilitate data acquisition.

3.2 Z-Wave environmental sensor network

An essential characteristic of the GiraffPlus system is the ability to integrate different types of sensor like environmental, physiological and android-based ones. Next to the already deployed Tunstall environmental sensors, the Z-wave sensors have been investigated and integrated in the GiraffPlus system. These sensors have to comply with the Z-Wave protocol that defines the communication technology, device capabilities and network available operations. The integration of other sensor technologies relying on standards such as Z-wave is important for enabling flexibility in the GiraffPlus system and to allow for specific needs of users to be accounted for if not all sensors can be provided by the same manufacturer.

A consortium of more than 250 manufacturers called Z-Wave Alliance was founded, grouping all the companies who agreed to product home control devices complying with the Z-Wave standard. This ensures a high availability of the devices while guaranteeing their interoperability. The Z-

Wave wireless protocol is designed for home automation and remote control applications mainly targeted to residential environments. It uses low-power radiofrequency devices operating in the sub-gigahertz frequency spectrum (868 MHz in Europe) with a bandwidth of 9.6, 40 or 100 Kbit/s and a maximum range of 100 meters (between two devices).

The protocol uses a mesh network architecture (Figure 8), allowing the nodes to communicate even if they are not directly visible to each other, through some nodes acting as “repeaters”, usually being the mains powered devices like the wall plugs. This relay system greatly extends the network range, allowing covering large apartments as well. A Z-Wave network requires at least one controller, which can be a dedicated gateway, a USB pluggable key or even a hand-held controller. Every device, prior to be used, has to be included in the network by manually associating it with the controller. This operation has to be done just once, during the sensors installation and is not required again unless some changes in the network happen.

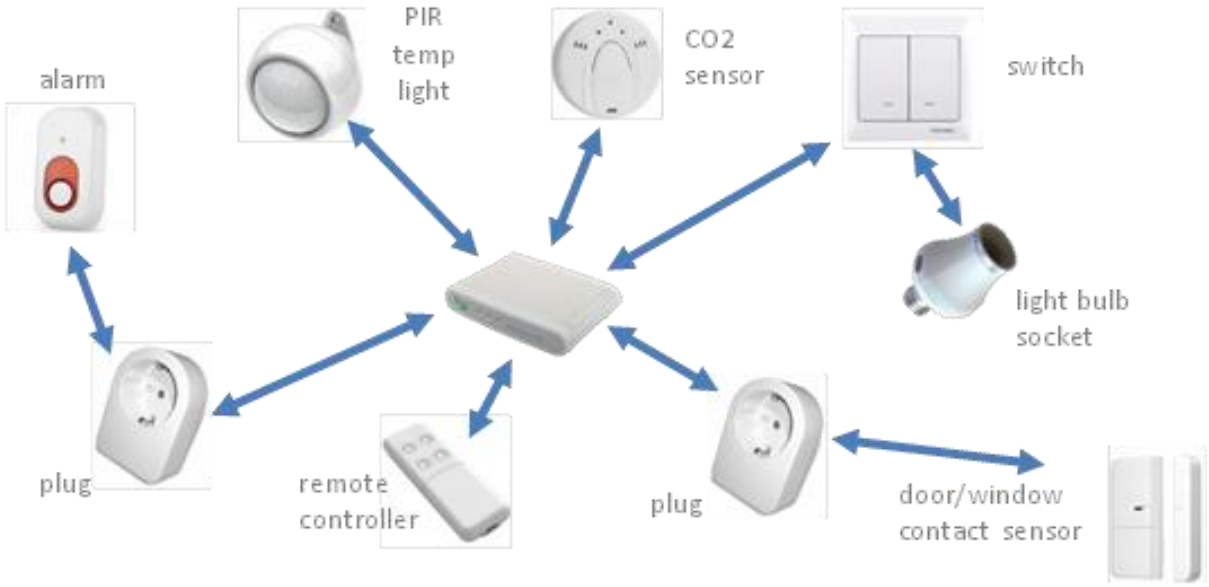


Figure 8 An example of Z-Wave sensors mesh network

During the association process, at every device is assigned a unique node ID by the controller and that ID is used by the GiraffPlus system to recognize events received from a device or commands sent to a device. This ID, together with the data fields provided by the device, has to be configured in the Engineering UI during the sensors configuration phase. The implemented GiraffPlus driver that interfaces with the Z-Wave controller takes care of loading the configuration from the database, connecting with the Z-Wave controller, announcing the sensors/actuators and dispatching the messages to/from the sensor network.

Some of the Z-Wave device selected to be used with the GiraffPlus system are the following:

This USB Z-Wave controller from Aeon Labs can be plugged into any PC and interfaced with the open source openzwave library (http://www.openzwave.com/), removing the need for a dedicated gateway

	This 4 in 1 multisensing device from Aeon Labs provides the temperature, humidity, light intensity and movement detection
	This wall plug from Aeon Labs provides real-time power consumption, voltage and current values. It can also act as a remote controllable switch to turn on/off the connected appliances
	This power meter from Aeon Labs uses clamps to detect the current absorbed on the electric line without the need to cut the wires
	This light bulb socket from Everspring can be remotely controlled to turn on/off a light bulb up to 100W
	This remote controller from BeNext has three buttons whose can be associated to custom functions, for instance triggering alarms whenever the user requests help
	This door sensor from Zipato detects whenever a door/window is opened or closed as well as the temperature and the light intensity

All the Z-Wave battery powered devices are able to go in a sleep state characterized by very low power consumption, waking up only if they have new data to send or at given intervals to check if there is any command directed to them. This allows reaching usually from one up to three years of operation before replacing the battery. When the battery is low, the devices also send a warning message to notify the system about the situation.

3.3 Wearable sensors

In addition to the wearable sensors already integrated in the system (see D2.3 Section 2.3), in the last period, an additional wearable sensors selection has been performed. The analysis has been conducted taking into account the new emerging devices available on the market and the new trends in health and activity monitoring [6]. For this reason, the Android wear platform has been integrated in the system focusing on the heart rate monitoring capabilities of smartwatches. In order to better understand the potentiality of such devices, a comparative analysis between the integrated device and a standard heart rate chest strap sensor has been conducted highlighting accuracy and possible limit of the proposed solution.

3.3.1 Android wear platform

Android Wear is a version of Google's Android operating system designed for smartwatches and other wearables[3]. By pairing with mobile phones running Android version 4.3+, Android Wear integrates Google Now[4] technology and mobile notifications into a smartwatch form factor.

An application that runs on a wearable device usually utilizes some of the capabilities of a paired handheld device. This means that two separate Android apps, one that runs on the wearable and another that runs on the handheld, are needed. These two apps communicate with one another over the Bluetooth link that connects the two devices. A Wearable Message API provides access to the data layer of a data communications link between the two devices and messages move down the protocol stack on the sending side, across the Bluetooth link, then up the stack on the receive side. The diagram in Figure 9 shows how a simple message flows through the wearable communications link.

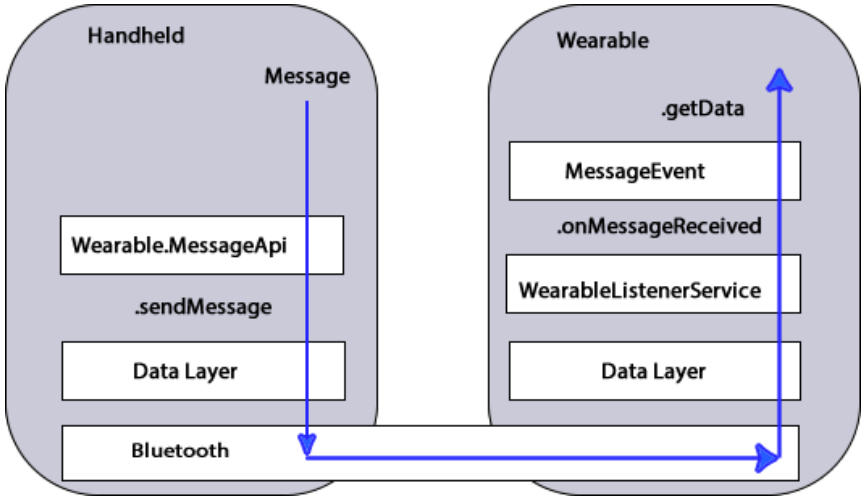


Figure 9 The handheld sends a message to the wearable using the sendMessage method. On the receiving side, a WearableListenerService monitors the data layer and invokes the onMessageReceived callback when a message arrives

[3] http://developer.android.com/wear/index.html
[4] http://www.google.com/landing/now/

The wearable data layer can sync either messages or data. A message contains a single text string while data is typically wrapped in a DataMap object. A DataMap contains a collection of one or more of data types, stored as key/value pairs. Figure 10 shows how DataMaps are managed by the wearable data layer. Using the Android version of the GiraffPlus middleware, different smartwatches running Android wear OS can be integrated in the system. As a result, the Samsung Gear Live smartwatch has been integrated in order to test its robustness and accuracy when used as a continuous heart rate monitor.

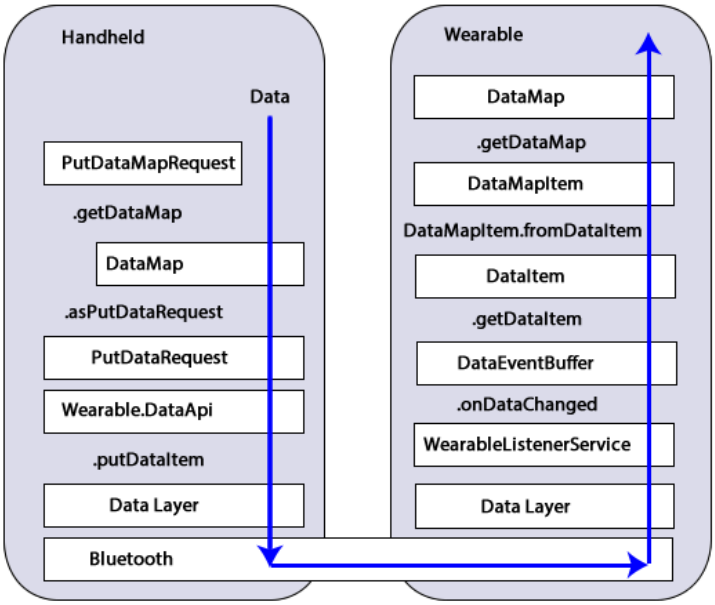


Figure 10 The PutDataMapRequest helper class simplifies the process of creating the DataMap, DataItem, and PutDataRequest objects that are needed for sending data. On the receiving side a WearableListener Service receives changed data and returns a buffer of DataEvents containing DataItem to be converted into DataMapItem containing the original handheld data

3.3.2 Heart rate monitor and accuracy testing

The presence of an Android OS on smartwatches eases the integration process in the GiraffPlus system of new sensors by means of the developed mobile middleware [7]. The Samsung Gear Live (Figure 11) has been chosen for a first experimental analysis in order to evaluate the possibility of using such a device in the GiraffPlus homes. The smartwatch comes with different sensors:

- Optical heart rate meter
- Accelerometer and gyroscope
- Compass
- Step counter

The most important sensing capability that has been identified in the device is the optical heart rate meter since it opens new health monitoring scenarios that do not involve an active interaction of the primary user with the device. Unfortunately, the heart rate monitor embedded in the smartwatch cannot be used continuously out-of-the-box since a still position of the user is required to be precise and it must be manually activated by the user via a tap on the screen. For these reasons, a wear application has been developed on the smartwatch that continuously

retrieves data from the heart rate sensor and sends them to the tablet[5] according to the GiraffPlus data format.

Figure 11 The Samsung Gear Live smartwatch

The developed application can collect data at different sampling frequencies in order to test how a continuous monitoring impacts on the battery life of the smartwatch. The results obtained show that it is possible to record up to 8 hours at 1Hz reaching a maximum of 4 hours at 10Hz. It must be noted that also accelerometer data were collected simultaneously to be able to replace the already integrated (see D2.3 for details) wearable inertial system with only one device.

Figure 12 shows the heartbeat and movements profiles while sleeping and during daily activities. Some "holes" in the data can be seen when walking due to a not perfect position of the device on the wrist. This is an important issue on this kind of devices since it can lead to incorrect readings or no readings at all. For this reason we tested the device under different positions and compared the heart rate data with the ones coming from a Garmin cardio chest strap (Garmin Premium heart rate monitor[6]) commonly used in fitness applications.

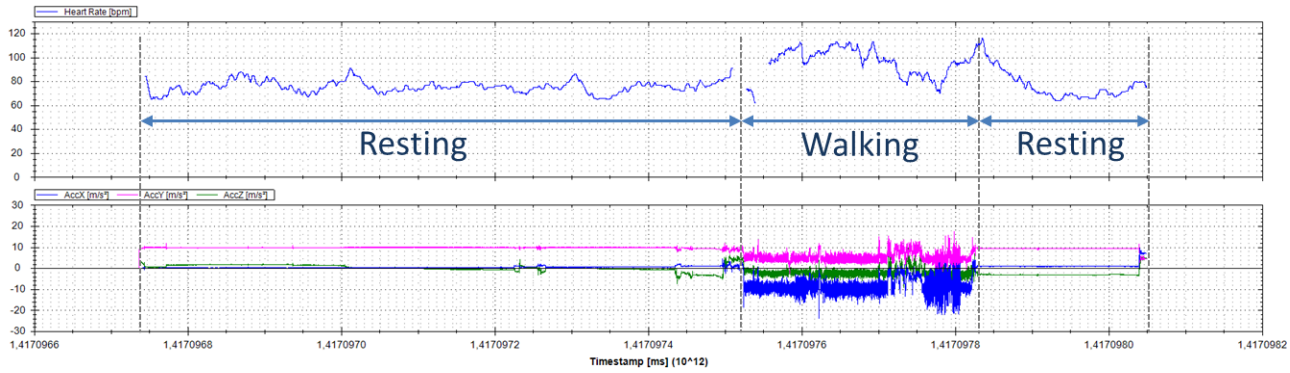


Figure 12 Heartbeat and accelerometer data while sleeping and walking

Figure 13 shows the tests made with the smartwatch well positioned and securely tightened to the wrist. In the upper graph the heart rate coming from both the devices is plot, while in the bottom

[5] http://marctan.com/blog/2014/07/08/reading-heart-rate-data-from-samsung-gear-live/
[6] https://buy.garmin.com/en-US/US/shop-by-accessories/fitness-sensors/soft-strap-premium-heart-rate-monitor/prod15490_010-10997-07.html

graph the three accelerometer's components are plotted. In these conditions, a good accuracy is obtained even in presence of sudden movements of the wrist.

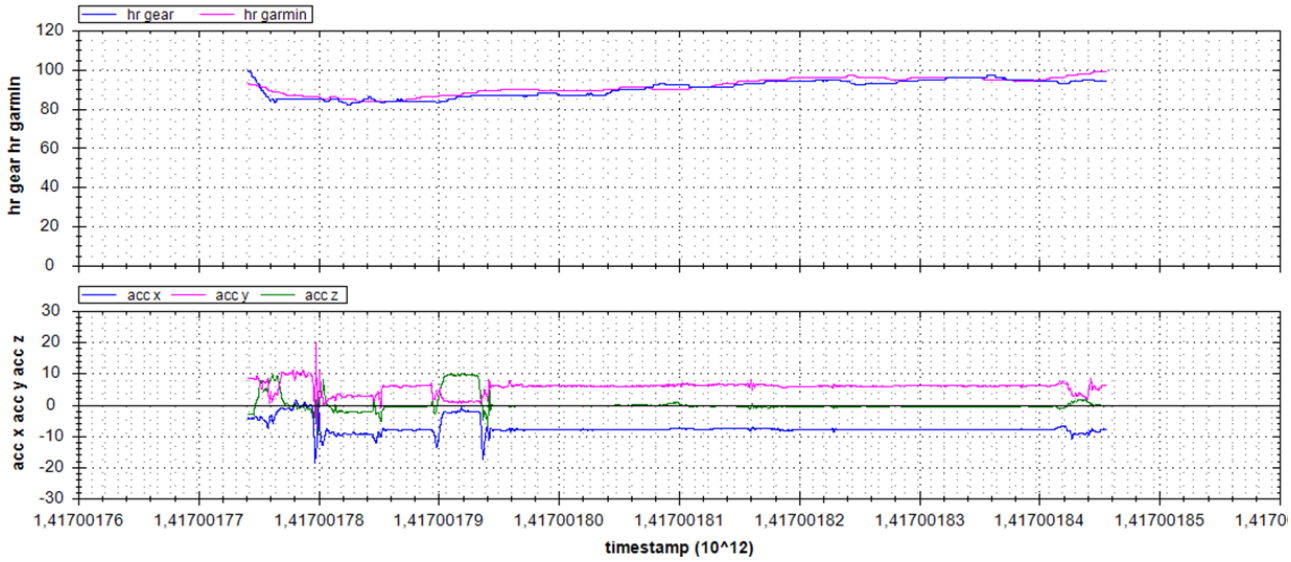


Figure 13 Heart rate data from the Samsung Gear Live and the Garmin Premium heart rate monitor compared (upper graph) while moving the wrist (bottom graph showing accelerometer data) when securely tightened

When the smartwatch is not securely tightened to the wrist it can easily give no output (Figure 14) or erroneous measurements (Figure 15). Overall, the tests show that, even in the case of a not perfectly tightened position, the measurement presents an error of ~20 heartbeats when a measure is obtained.

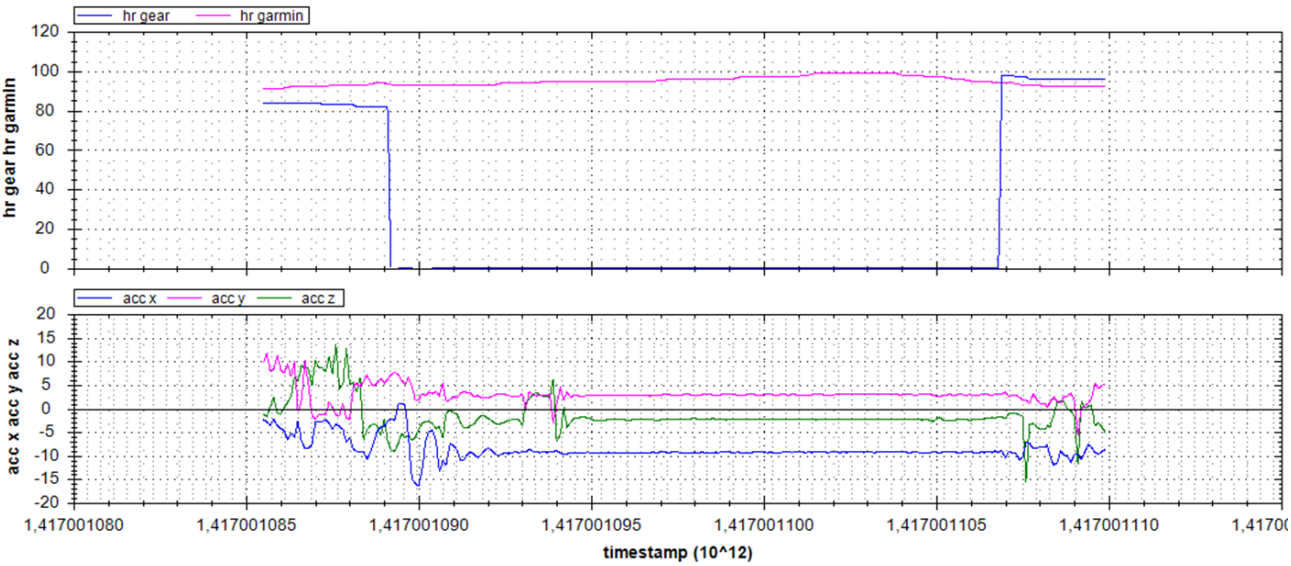


Figure 14 No heart rate obtained when moving the wrist while the smartwatch is not perfectly tightened

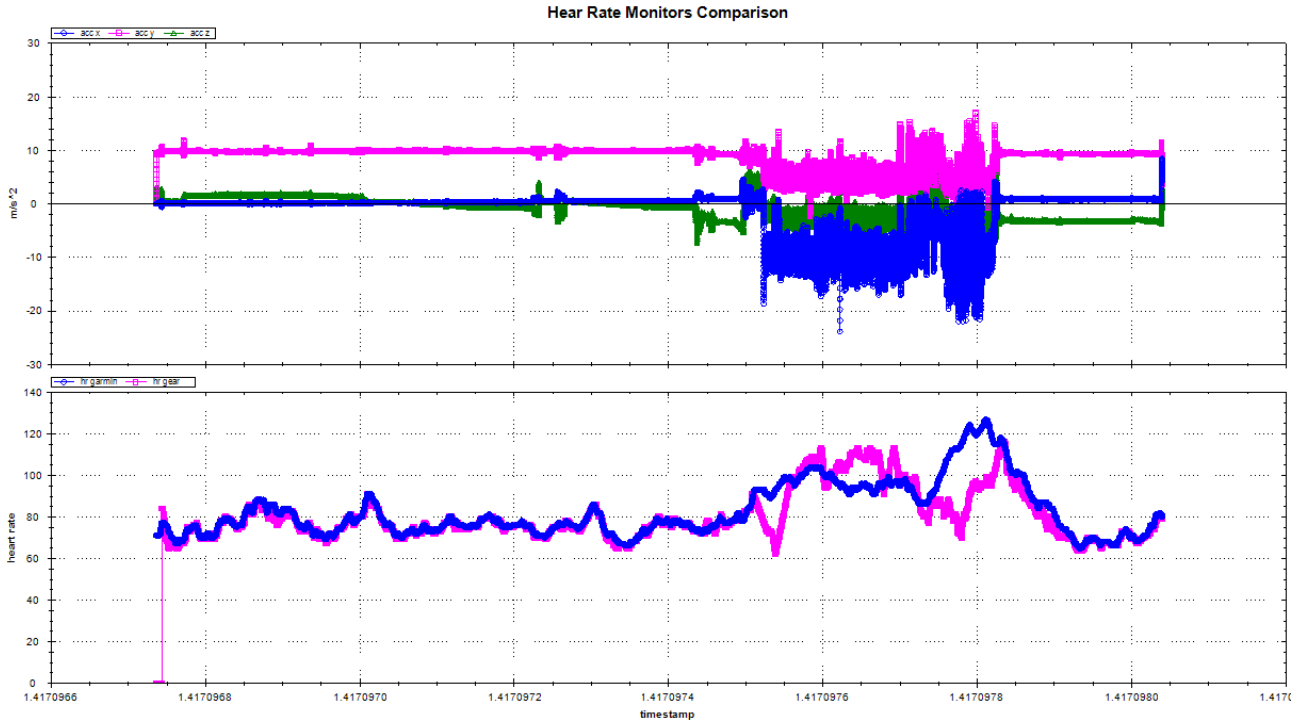


Figure 15 Erroneous heart rate (~20 heartbeats error) obtained when moving the wrist while the smartwatch is not perfectly tightened

When used in the GiraffPlus ecosystem, these results can be considered acceptable since the main focus of their usage is on long term monitoring for new possible services (e.g. providing fatigue detection or calories consumption over long periods). In this scenario, periods of missing data or with an error of 20 heartbeats over small periods can be tolerated or filtered [8].

4 Final version of the Giraff platform

In this section the final prototype of the Giraff robot featuring the semiautonomy abilities detailed in the DOW and developed along the project is described. In particular, the improvement of Giraff mobility according to the last evaluation phase addressing the safety issues emerged from the second intermediate evaluation is presented.

General improvements on the hardware robotic platform and software components are described together with the new automatic docking feature based on laser data.

4.1 Improvement of the Giraff mobility

The final prototype of the Giraff robot is featured with the semiautonomy abilities detailed in the DOW and developed along the project. Concretely, the final prototype of the robot includes:

- Safe, reactive navigation to predefined locations within the house, avoiding static and dynamic obstacles.
- Self-localization, providing the user with the estimated pose depicted in an illustrative map.
- Collaborative control that enables the user to drive the robot in a protected-mode, in which user commands that could end in a collision are override by the system.
- Automatic docking operation.

This version of the semiautonomous telepresence platform has been extensively tested in a controlled environment: the Mapir laboratory (Figure 16), at the University of Malaga, and in the test site ES1 where every experiment ran under the supervision of a GiraffPlus Engineer (physically present in the primary user environment). These tests showed up that some additional improvements were needed in order to increase the usability and ensure a higher level of security. More specifically, the final evaluation revealed the need to provide the platform with:

- Higher level of security when the secondary user delegates the robot control to the autonomy system.
- Assistance to the secondary user while driving the robot in environments with areas of poor lighting or low visibility.
- Response to the loss of communication between the different systems involved in the GiraffPlus telepresence service.

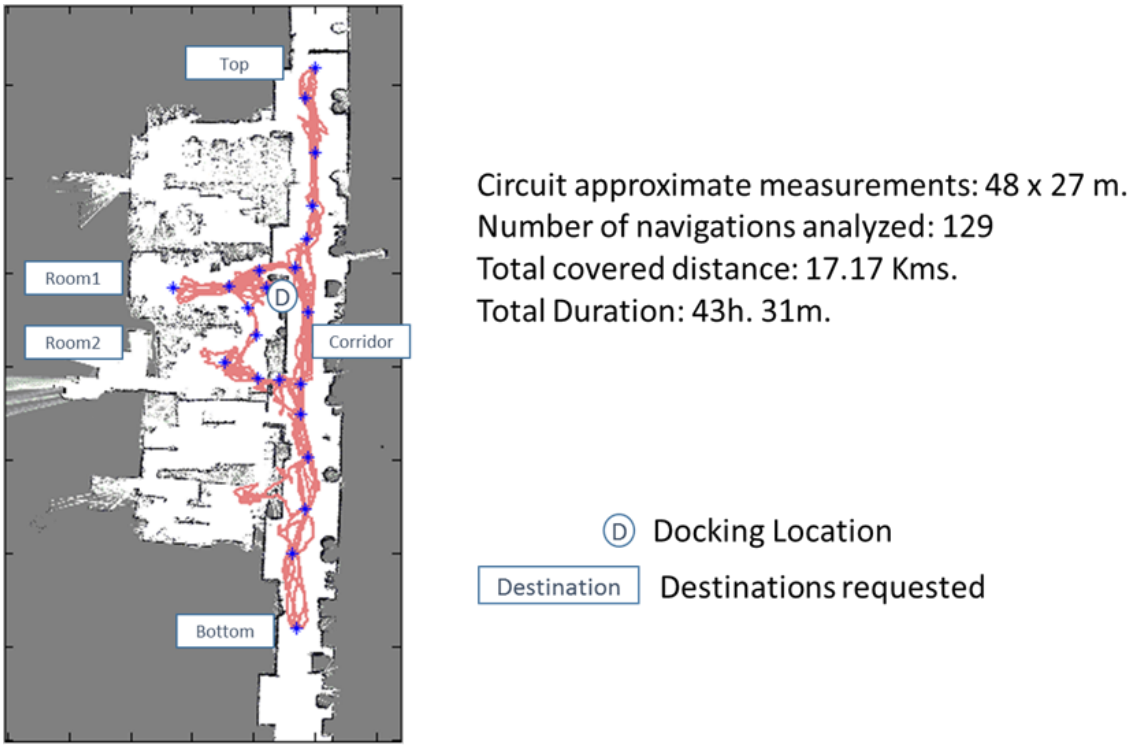


Figure 16 Result of an autonomous navigation experiment carried out in a controlled environment: the Mapir Laboratory

Therefore, in this period the final prototype has been enhanced with the following additional features and improvements:

- Communications status monitoring.
- Robotic platform internal status monitoring.
- Limitation of the Giraff motors' input range.
- Improvements in the collaborative control.
- Automatic docking based on laser data.

Figure 17 summarizes the systems involved in the telepresence service (remote interface, servers, and robot), the subsystems contained in them to fulfill the desired functioning of the service, and the final set of available features.

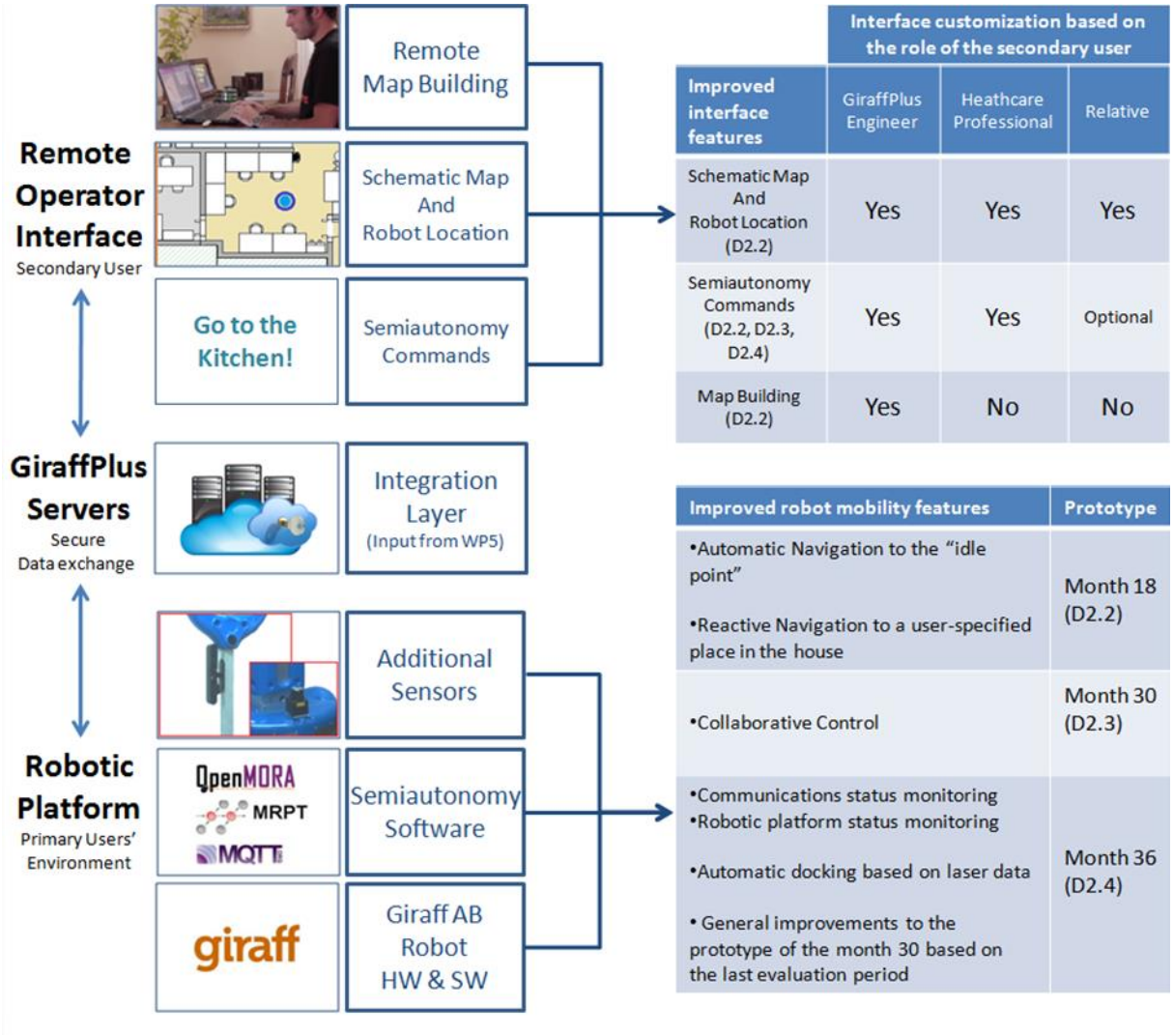


Figure 17 Overview of the D2.4 prototype and its features

Safe navigation and usability are essential to bring the robotic telepresence into households. In fact, security and usability are closely related to each other. In cases where the robotic platform has the ability to perform tasks autonomously, security becomes even more valuable.

The semiautonomous algorithms developed are a powerful tool to gaining in usability. Additionally, usability increases the safety of the system by reducing user errors, which prevents undesired states in the robot and avoids its consequences. As a result, a safe and usable robot navigation system makes driving easier given that remote users do not need to worry about possible collisions, and they can perform complicated maneuvers or even visit specific locations in the house without continuously guiding the robot.

The following subsections present the security mechanisms that are included in the final version of robotic platform and describe a key improvement for usability: an automatic parking based on laser data.

4.1.1 Robotic platform safety issues

Any failure occurred in the robotic platform may cause an unsafe situation. A failure in the platform means: “The inability of the robot or the equipment used with the robot to function normally” [9].

In order to analyze and improve the safety of a robotic platform designed for interaction with humans it is necessary to identify a) what is the greatest danger that can occur, b) who is the user or objects which are at risk c) which are the consequences of possible failure, and d) what are the most influential factors for safety.

Three different levels can be established when focusing on the dangerous situations that the system may reach, ranging from level 3, i.e., incidence of maximum danger that can lead to a collision, to level 1 in which incidents can cause the autonomous system to produce errors but still allowing remote user teleoperation (Level 1). Examples of the consequences of a failure are:

- Physical contact between the robot and the user or delicate objects that can be damaged. (Risk level 3).
- The robot becomes an obstacle, it clutters the environment, bothers the primary user and forces him/her to either move it manually, or to ask the help of a third person to carry the robot to its parking area (level 2).
- The autonomy system stops working and it all depends on the driving skills of the secondary user (level 1).

According to [10] the causes of incidents in a robotic platform can be classified into three groups: engineering errors, human mistakes and inadequate environmental conditions. Among the engineering errors, there are mechanical, electronic or controllers errors. Losses of connection between different parts of the system, programming errors or failures in the design of algorithms are examples of such problems. These failures are unpredictable for secondary users and, hence it is necessary to have mechanisms that anticipate or detect these faults to prevent an incident occurs. The other main source of failures are human errors, most come from inattention, little expertise in driving, or conducting inadequate procedures while using the robot. In these cases, the usability of the system plays a very important role. The environment is the other main factor affecting the safety of the robot. Among others, lighting conditions, obstacles that are not detectable by the perception system or cluttered spaces are the main causes of mobile robot failures.

Figure 18 shows the classification of the incidents described [11].

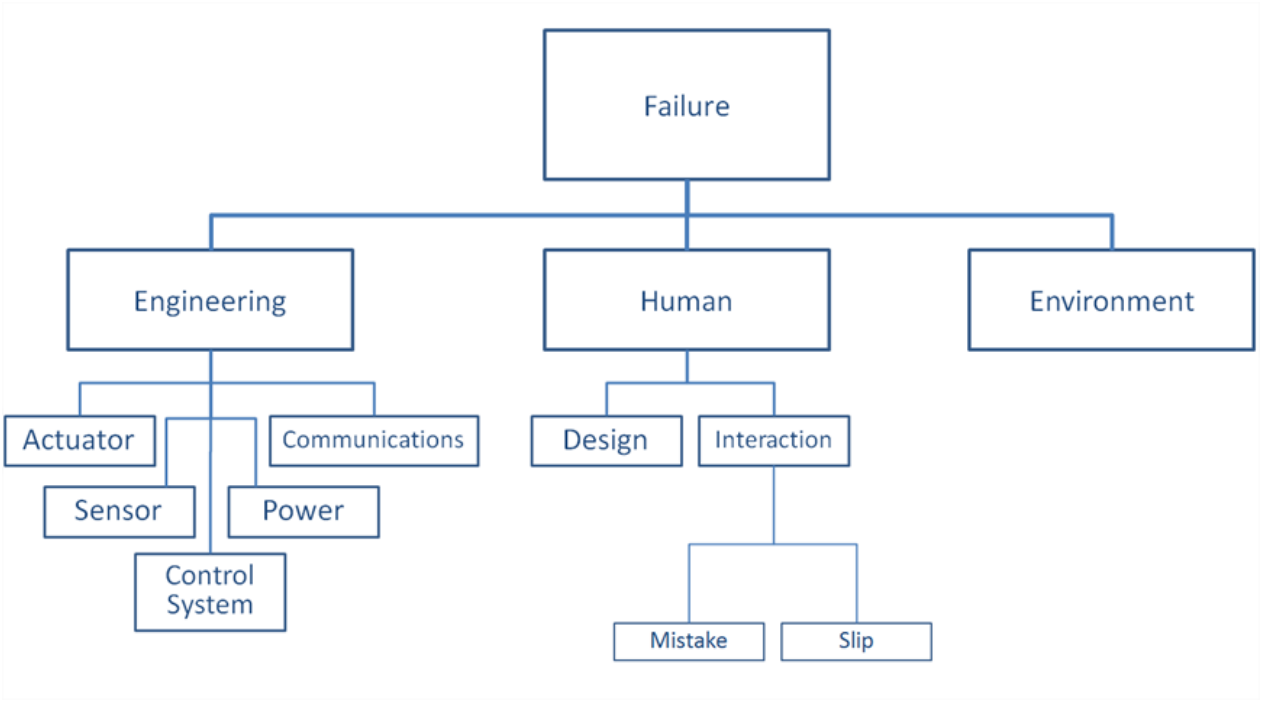


Figure 18 Taxonomy of failures. Engineering errors, human mistakes and environment conditions are the sources of system failures

Following this classification of failures and taking into account the specific characteristics of the GiraffPlus robotic system, in this period, security solutions have been designed and incorporated into the platform in order to address problems in each of the systems involved in telepresence (remote user, server and robot). Since the security risk occurs in the environment of the primary user and the agent that generates the danger is the robot, these solutions have been focused in the subsystems that compound the robot (perception, actuation, controllers and semiautonomy). The main security measures are:

1. Detection of an atypical behavior of the robot.
2. Detection of atypical performances in the integration layer of semiautonomy and controller software (Giraff AB teleoperation software).
3. Detection of an atypical operation of the actuators.
4. Improve usability of robotics platform to reduce secondary user errors.
5. Check the connections between remote user, server, and robot to respond to any unforeseen incidents.

Fulfilling the premise that the robot can only be in motion if a remote user supervises the navigation, the actions taken by the security mechanisms are only stopping the robot or limiting speeds if any anomaly is detected.

4.1.2 General improvements based on the reviewers' evaluation

This section covers the mechanisms implemented and integrated into the robotic platform to gain in security. Safe behavior of the robot is essential when performing in home environments, as it was highlighted in the second intermediate review meeting. Therefore, in the last ten months, an intense work has been done on improving the different software pieces and internal communications.

Communications status monitoring

To complement the security system and increase the usability of the system, the robot and the secondary user interface have been enhanced in this period to be able to detect connection loss in any of the systems involved. Specifically, each endpoint of the telepresence platform transmits an acknowledgment signal (Figure 19) when it is connected and monitors the status of other system's components (server and remote device).

In any case where the communication between secondary user and robot is interrupted, the robot is stopped and a wait state starts that is capable of restoring communications automatically once the device that had lost connection is recovered.

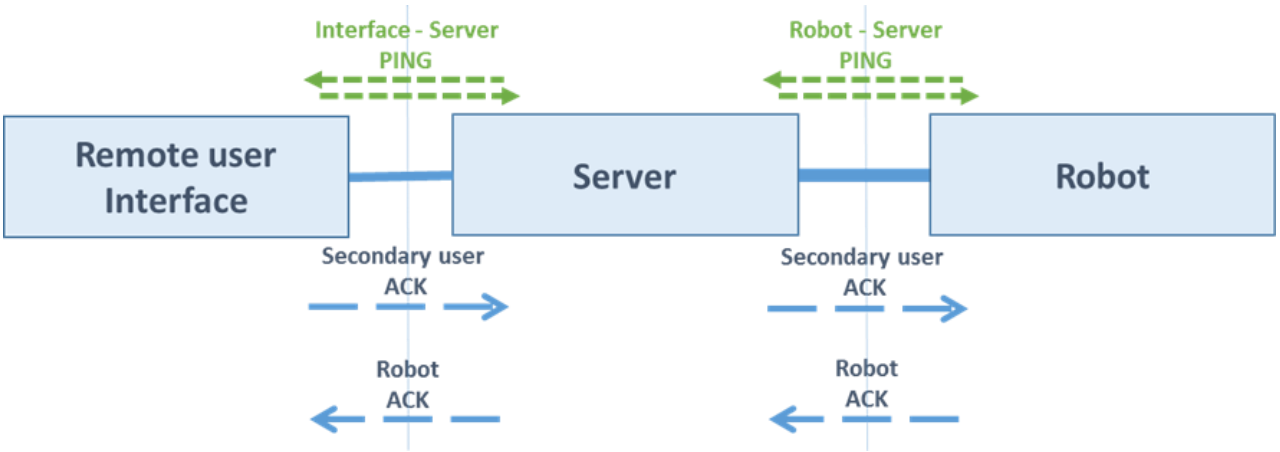


Figure 19 Signals exchanged between different systems involved in telepresence platform

Robotic platform internal status monitoring

Within the robot itself there are several subsystems working together to perform robot navigations: perception, actuation, control, and semi-autonomy. It is necessary to monitor the status of each of these subsystems and the communications between them to avoid an inappropriate behavior of the robot and be able to inform users that there is a problem in robotic platform.

The OpenMORA[7] robotic control architecture (RCA) has been improved to centralize and monitor the internal state of the robot. A central database inside the RCA collects information from modules that are active at all times and can be consulted by any software module in the architecture. The RCA receives feedback from the sensors and actuators through specifically

[7] http://sourceforge.net/projects/openmora/

developed modules for integrating these devices with openMORA. When an anomaly is detected, depending on the level of risk, the RCA communicates directly with the actuators control software to disable autonomy (teleoperation remains enabled), apply for reduced speeds, or stop the robot definitely.

Improvements in the collaborative control

During the last period of evaluation of the prototype, the speed range of collaborative control and thresholds that determine the system input autonomy over user-requested commands have been redesigned to make driving smoother and more intuitive. In previous versions collaborative speeds were too slow and had abrupt transitions when control passed from the user to the autonomous system or vice versa.

Limitation of the motors' input range

An additional component has been deployed in the telepresence software provided by GiraffAB Technologies in order to enhance the safety of the system. The maximum linear and angular speeds that can be commanded to the motors have been limited to safe ranges, avoiding high accelerations that may cause collisions or the robot to fall down. Although the reactive navigator and the teleoperation system provided by GiraffAB include their own safety procedures and they limit the range of their commands, this issue has been addressed as a monitoring mechanism for the final set of systems involved in the semiautonomous robotic platform.

4.1.3 Automatic docking based on laser sensor

The auto-docking (AD) assistant described in this section refers to a software module from the OpenMORA robotics architecture in charge of guiding the robot to the charging station using data from the laser scanner. Although other solutions could have been considered, e.g. using infrared sensors, we have opted for relying on the already mounted laser sensor. This solution does not result in additional hardware neither on the robot nor on the docking station, while provides precise information from the surroundings of the docking area. The inclusion of this module in the final version of the GiraffPlus system entails a number of enhancements, namely:

- Increases the usability of the system since it allows the user to command the robot to recharge its batteries from any room. A new high-level navigation command has been integrated in the interface that combines topological navigation to the node that represents the docking station and the subsequent execution of the implemented auto-docking module.
- Improves the robustness and safety of the system by providing an effective solution in scenarios where the manual navigation is difficult, as in the case of low light environments or when obstacles are close to the docking station.
- Improves the tolerance to connection errors (optional) by avoiding the robot to stay idle without recharging its batteries in situations where the user drops the call due to connectivity problems.

This module has been intensively tested in lab environments by commanding continuous, random topological navigations (see Figure 16 for the results of one of such tests). Every time the system

detects that the battery level is under a threshold, an automatic docking is issued from the current, unforeseeable location of the robot. Our tests entail, in total, more than 250 automatic docking operations considering 5 different locations for the docking station. The results yield a success rate of around 94%. The non-success cases are due to errors in detecting the docking station, requiring the user intervention to perform a manual docking.

This feature was integrated in the very last months of the projects, and thus it has been occasionally tested in some of the deployed test sites, being its limited use by secondary users insufficient for concluding the benefits in real scenarios.

Module description

As mentioned above, the objective of this module is to allow the robot to autonomously find and reach the docking station for recharging its batteries.

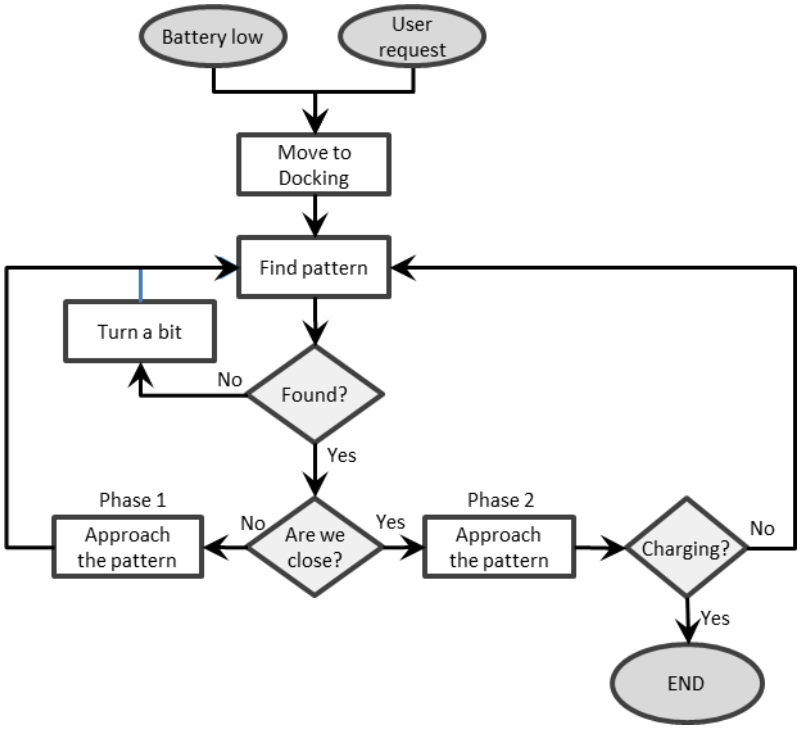


Figure 20 Simplified flow chart of the Auto-Docking module

Figure 20 shows a flow chart of the process where the main stages are described next:

- *Move to Docking.* The first step consists in getting the robot close to the docking station, that is, navigate to the room where the docking station is placed. It is to be noticed that the process itself may be started by the user directly, or if enabled, when the robot detects that the batteries are getting low. Thus, this stage commands other modules of the OpenMORA architecture to carry out a reactive navigation to the “Docking” node fro, the navigation topology.

- *Find Pattern*. This is the core stage of the auto-docking process. Here, the developed algorithm searches for the pattern within the laser scan. Figure 21 shows different laser scans where the pattern has been detected. Security and error tolerance have been considered as important requisites when implementing the pattern search algorithm since this module is expected to be used when the robot has low batteries, and so, failing the auto docking process will mean leaving the robot unusable. The pattern layout and dimensions are described next.

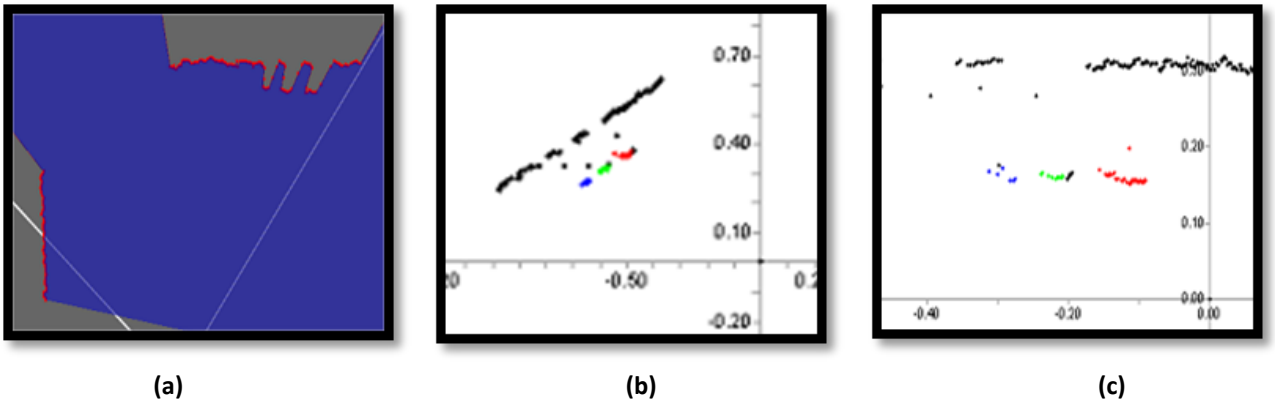


Figure 21 Detailed views of the laser scans when detecting the pattern used for the auto-docking module. (a) Raw laser scan where the three sticks composing the pattern can be visually appreciated. (b-c) Colored pattern within the laser scans pointing the location and distance from the robot

- *Turn.* This stage is performed when the pattern cannot be found in the current laser scan. Since the robot should be positioned close to the docking station (see "*Move to Docking*" stage), the most common cause is that the laser is heading an incorrect direction, thus, turning represents both a safe alternative and the best option to find the pattern.
- *Approach Pattern.* Once the pattern is found, the distance between the robot and the pattern is estimated to differentiate between two approaching modes. Initially, when the distance is bigger than a set threshold (Phase 1), the robot approaches the pattern heading to its perpendicular, trying to leave the pattern in front of it. Then, once the pattern is correctly aligned with the robot, the robot moves in a straight line to approach the pattern. When the distance to the pattern is below the threshold (phase 2), the robot reduces considerably the movement speeds to assure a soft and safe docking, keeping the straight movement fashion. This phase continues until the charging event is detected. Figure 22 depicts the movements performed by the robot when approaching the pattern depending on the working phase.

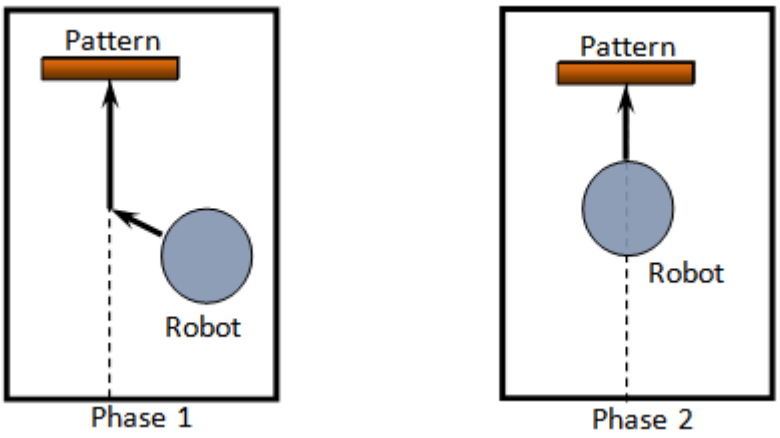


Figure 22 Description of the robot movements when approaching the pattern. In phase 1 the robot will perpendicularly align to the pattern, while in phase 2 the robot will slowly advance straight

The parking pattern

The pattern used for the auto-docking is based on three sticks of known width and separation, which can be configured in the mission file. Figure 23 shows a schematic view of the pattern and the controlled dimensions, while a real implementation is shown in Figure 24. Since the pattern is to be detected with the laser-range scan sensor onboard the robot, the pattern has to be placed at a height visible by it (the height of the Giraff docking station is perfect for that purpose). As described, the robot will approach the pattern until it detects the charging event, so it is important to always place the pattern on top of the charging station, facing the front.

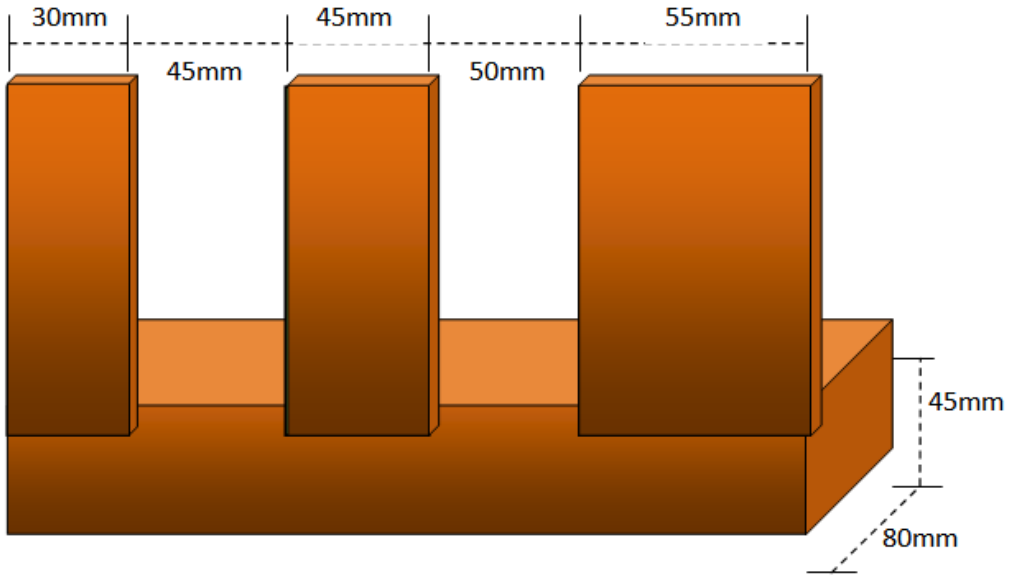


Figure 23 Schematic view of the pattern used in the auto-docking module

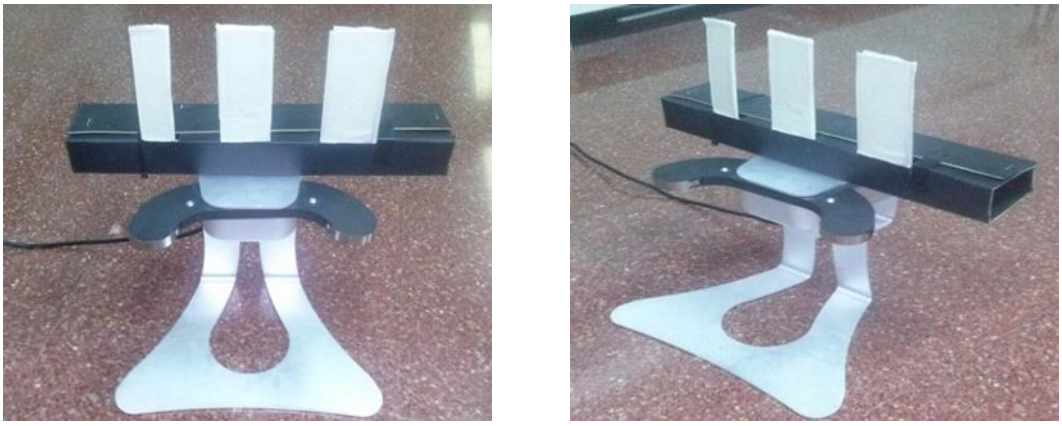


Figure 24 Auto-docking pattern implementation made of cardboard

4.2 The new features of the Giraff hardware and software

In addition of the work presented in Section 4.1, during the last 6 months of the project focus has been given to the development of fundamentally new features of the Giraff telepresence system based upon user input. These enhancements are now in Beta and are expected to be released as part of the initial commercial version of GiraffPlus in the first half of 2015.

A summary of these enhancements is as follows:

1. Giraff avatar (version 4.1)
   a. Numerous improvements to hardware and electronics reliability
   b. Improved WiFi connectivity including stronger signal strength (and therefore extended range)
   c. Improved WiFi management and the ability to automatically to numerous stored configurations
   d. Improved microcontroller software to reduce risk of damage to display tilt motor from user abuse (rapidly forcing the display head up and down)

2. Visitor “Pilot” and Giraff avatar software (version 3.0)
   a. Introduction of a new audio/video codec and videoconferencer, to support:
      i. Automatic video resolution selection based upon network performance
      ii. Ability to port Pilot visitor application to other operating systems including Mac OS and Android
   b. Enhanced security with option to provide 2-stage authentication (see system enhancements below)
   c. Enhanced Pilot UI to be more intuitive

3. Sentry management system enhancements (version 2.0)
   a. Numerous improvements to security and system data protection
   b. Enhanced security option that requires visitors to sign in, then receive a session password on their cell phone
   c. Completely redesigned UI that is organized by admin function instead of by database element as before

Overall, the Giraff system as a stand-alone commercial platform now has nearly all the features commonly requested by all user groups including residents, caregivers and care administrators/organizations. Furthermore, it also contains all of the integration support required to deliver a first commercial version of the GiraffPlus platform.

There are two areas of focus going forward, both associated with the Giraff avatar:

1. Cost reduction of the hardware – it is clear from the economic analysis of the various use cases for the GiraffPlus platform that it is necessary to achieve a major reduction (up to 50%) in the cost of the hardware/electronics platform. This will happen in two stages; first, cost reduction of current system via more efficient procurement process based upon volume commitments, and second, "value engineering" – redesign of the hardware for cost reduction and more efficient assembly.

2. A new form factor – the GiraffPlus project has produced a wide variety of feedback on the physical appearance of the avatar, often in direct conflict (e.g. some say it should look "high-tech" while others say it should look like a "piece of household furniture"). Feedback generally falls into 3 categories – "minimalistic," "furniture" and "playful" (e.g. it should look like an animal). Therefore, no substantial changes to the physical appearance have been made because there has been no clear direction. Nevertheless, it is expected that a clearer set of requirements to emerge with more commercial experience, and hope to combine this effort with the value engineering work described above.

4.3 The DVPIS@Home plugin

A separate contribution to the Giraff robot started after the work of WP1. In particular during the User Requirement Analysis there has been a frequent demand to offer additional services though the robot. A specific activity of WP4 has been dedicated to design an additional module to run of the robot responsible for managing interactions with the primary user. The goal is to create additional functionalities for the primary user with respect to the pure answering a call from a secondary user or making an emergency call. The result of such activity is an independent software plugin called *DVPIS@Home*. It takes advantage of the features of the hardware platform and in particular extensively uses the touch screen. Although detailed reporting of this activity is given in WP4 deliverables, we decided to insert a summary on the plugin in this last report of WP2 to offer in a single document a complete view of the improvements realized on the robot platform within GiraffPlus.

In detail, besides maintaining the ability to perform all activities of the standard Giraff software, the DVPIS@Home offers:

- **Messages**: the primary user can receive messages from DVPIS@Office software. They can be regular text messages, voice messages, reminders, or questions. The user can also reply with a voice message, or send voice messages to a secondary user of their choice. Text messages are read out by a Text-To-Speech system, to facilitate people with visual impairment.

- **Shared Space**: the software provides a “shared space” area that can be used to facilitate content exchange from a secondary user to the primary. At present it allows to send pictures or documents.
- **Physiological Data Display**: the DVPIS@Home provides the elder person with audio/visual feedback when physiological data is being acquired using the sensors of the GiraffPlus ecosystem: if, for instance, the primary user takes a blood pressure measurement, the collected data is displayed on the robot screen and read out, thus giving the primary user an acknowledgement of their action. This functionality is not a pure speech facilitator it has also the goal of communicating the old person that the GiraffPlus system “knows” her/his physiological data.
- **Multimedia capabilities**: the software integrates a remote-controlled media player based on YouTube: it allows the secondary user to define playlists for the elder person, and to control playback on the robot. The primary user can of course intervene by controlling playback on his own.
- **Robot head movement**: when the robot is charging, its screen faces the wall, making it very uncomfortable to use the abovementioned features. Hence, the primary user can rotate the screen by 180° by just touching whatever part of the screen (when the robot is charging). Also, the robot automatically rotates its head when it is charging and a notification arrives (such as a new message, or a new physiological measurement). The user can also choose to lower or raise the robot neck by tapping a button on the touch screen, thus making it easier to use when being seated.

For details on the different functionalities the interested reader can refer to the D4.3 report. The rest of the section presents some technical aspects on the plugin realization.

4.3.1 Software architecture

The DVPIS@Home is a plugin with respect to the standard Giraff software, and as such it follows the conventions defined for plugins by Giraff developers. Its code is packed in a series of java archive (jar) files, which are placed by its installer into the directory:

```
%PROGRAMFILES%\cygwin\home\giraff\telbot\thirdparty
```

Together with configuration files `classpath.conf` and `mainclass.conf`. During application start-up the Giraff software scripts load and execute the DVPIS@Home software, which in turn loads and initializes the standard Giraff software, embedding it into the DVPIS@Home user interface.

The GiraffPlus software is generally built upon the OSGI framework. To be able to use the middleware to communicate with the rest of the ecosystem, the DVPIS@Home has to rely upon OSGI (Figure 25). However, the standard Giraff software is a regular java application and the way the platform can be extended through plugins does not play well with complex frameworks like OSGI. To solve this problem, the @Home software is split between a Giraff plugin frontend and an OSGI backend, communicating through Java RMI (Remote Method Invocation). The backend is made up by the bare minimum needed to allow for the communication with DVPIS@Office through the Internet.

When the @Home frontend is started it tries to connect to a running instance of the backend, spawning if necessary the `rmiregistry` process and executing the backend in a separate instance of the Java Virtual Machine.

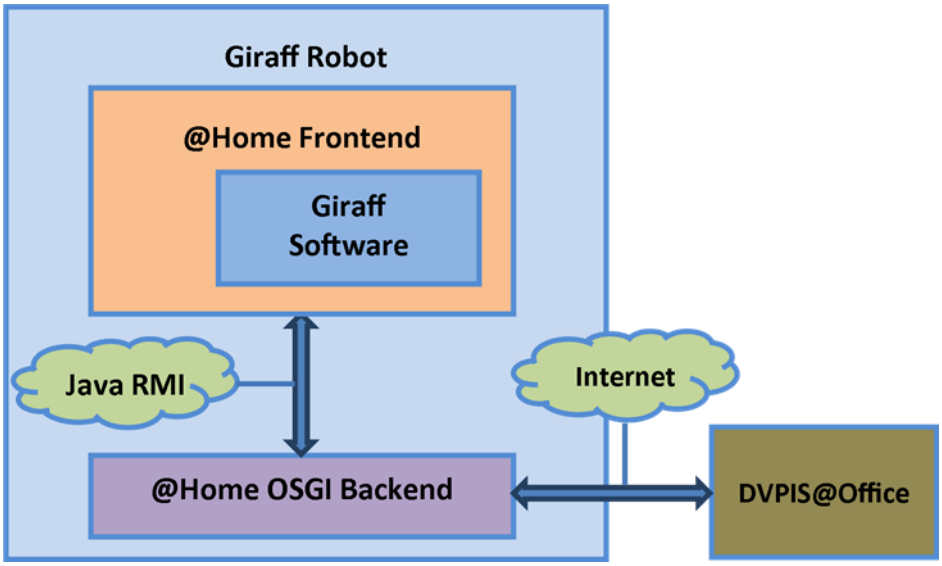


Figure 25 DVPIS@Home architecture

4.3.2 Implementation details

The Giraff software is embedded into the DVPIS@Home by the means of a modified version of the Giraff-provided *GiraffIntegration* class and *GiraffApplicationListener* interface. The modifications are meant to expose more Giraff features (e.g. robot head movement, screen saver interface, capabilities enquiry) and are mostly achieved via reflection at runtime.

The communication between the frontend and the OSGI backend happens through java RMI: the two sides export two façade objects, *GiraffApplicationInterface* and *ServerInterface* respectively to allow for communication in both directions. The backend communicates with DVPIS@Office using the APIs provided by the middleware, sending and receiving JSON-encoded messages to/from the appropriate MQTT topics.

Text-To-Speech services are provided by MaryTTS[8]. MaryTTS is open source (LGPL license), supports multiple languages and is written in Java. It is embedded in the @Home software by the means of the *LocalMaryInterface* API, which allow for the generation of java-compatible raw audio streams that can be easily played.

The system supports the playback of WAVE and FLAC-encoded audio files, the latter being possible through the jFLAC[9] library. The audio data is captured with a sampling rate of 16 KHz, mono, 16 bit resolution and is encoded in WAVE or FLAC formats thanks to javaFlacEncoder[10] library. Currently only WAVE data is transmitted for compatibility with DVPIS@Office.

[8] http://mary.dfki.de
[9] http://jflac.sourceforge.net
[10] http://javaflacencoder.sourceforge.net

YouTube integration is made possible through DJ Native Swing[11] library. The library provides a way to embed native components (such as a web browser) in java Swing code. Thus, the native browser's Flash Player plugin is used to load the YouTube player, which is then, controlled using the official YouTube API version 3.0 through some JavaScript boilerplate code.

Support to different languages is achieved with the standard tools the Java platform provides, that is, resource bundles based on property files.

[11] http://djproject.sourceforge.net/ns

5 Conclusion

This document represents the final report describing the prototype of the overall GiraffPlus system in terms of design and development work performed in the work package 2. As in previous deliverables, it focuses on changes and enhancements with respects to versions of the system delivered at previous milestones.

The last development cycle, completed at month 36, integrated the feedback and recommendations collected during the evaluation process in the test sites and taking into account the valuable comments coming from the reviewers. In this regard, the Mantis bug tracker is proven a useful tool to collect not only software bugs or malfunctions, but also for aggregating suggestions coming from secondary users involved in the project.

This document shows how issues regarding the different parts of the system have been addressed. It also describes improvements inspired by a continuous analysis of state-of-the-art and of new technologies as they come to the market. Indeed, it is described how the middleware exploited its communication infrastructure, based on a very popular protocol in the Internet-of-Things scenario like MQTT, to improve its functionalities in terms of controlling the devices in addition to the already described capabilities in terms of service discovery and context data transmission. Also new wearable devices have been investigated in their specifications and possible use in the GiraffPlus ecosystem. In this case, the described device has been release on the market only in July 2014 making difficult the deployment of such devices in the test sites. Anyhow, the limits and the opportunities given by such a device are described.

Improvement on the Giraff mobility has also been implemented. The development has been driven by users' and reviewers' comments and suggestions, leading to the realization of a system ready to be commercialized in the 2015. The improvements described consist of a safe and reactive navigation to predefined locations within the house, avoiding static and dynamic obstacles, a self-localization, providing the user with the estimated pose depicted in an illustrative map, a collaborative control that enables the user to drive the robot in a protected-mode, in which user commands that could end in a collision are override by the system, and of the automatic docking operation.

Appendix 1 - Assessment of security and safety in the final version of the GiraffPlus system

In the following, we assess how the final GiraffPlus system fulfills the requirements as defined in deliverable D1.3. The final version of the GiraffPlus system fulfills all security and safety requirements that were specified in delivery D1.3. The final version of the GiraffPlus system incorporates state of the practice solutions to confidentiality, data integrity, availability and authenticity. The final version will also be able to evolve to meet future requirements thanks to built-in configurability and adaptability.

Confidentiality

R1.1 *Confidentiality in the first prototype can be solved by shared symmetrical keys. For wireless links, encryption is sometimes already provided by the standard. Data stored should be encrypted to avoid unauthorized access.*

R1.2 *For later prototypes, a key management system like Public Key Infrastructure (PKI) should be considered.*

The final GiraffPlus system incorporates a public key management system, refined though the prototypes (D2.2, D5.3). It implements a certificate-based public key infrastructure with its own GiraffPlus Certificate Agency.

Certificate-based public key infrastructures are state of the practice for confidentiality and authenticity solutions, so the final GiraffPlus system fulfills the requirements R1.1 and R1.2 as specified in D1.3.

Integrity

R2.1 *Integrity for stored data is for hard disks provided by hardware. To avoid data loss in case of disk crashes, redundant disks or a RAID system should be used.*

The final version of the GiraffPlus system incorporates a cloud server solution from XLab, based on the MongoDB where redundant servers are employed (D2.2, D2.3, D5.2, D5.3). In addition, a local server is used for temporary storage in case of connection outage (D5.3). This guarantees storage of data also in the cases where connectivity to the cloud server is (temporarily) lost.

The redundant cloud servers represent state of the practice for robust and reliable data storage. In addition, the local temporary storage guarantees data integrity in case of connection outages. Hence, the final version of the GiraffPlus system fulfills the data integrity requirement R2.1 as specified in D1.3.

Availability

R3.1 *Redundant network paths should be considered for increased availability.*

The final version of the GiraffPlus system incorporates a cloud server solution from XLab, based on the MongoDB where redundant servers are employed (D2.2, D2.3, D5.2, D5.3).

For added safety, alarms from the additional android sensors can be sent directly to the caregiver (D5.5).

The redundant cloud servers represent state of the practice for robust and reliable data storage. With physical distribution and Internet accessibility, redundant paths are provided throughout the Internet, so availability is guaranteed as long as Internet connectivity is maintained. Alarms are sent directly to the caregiver also in the case of lost connection to the database. Together, the final version of the GiraffPlus system fulfills the data integrity requirement R3.1 as specified in D1.3.

Authenticity

R4.1 *For the first prototype, authenticity can be achieved by shared symmetric keys and nonce challenges to authenticate the communication counterpart.*

R4.2 *For later prototypes, the use of certificates should be considered together with the PKI infrastructure mentioned above.*

The final GiraffPlus system incorporates a public key management system, refined though the prototypes (D2.2, D2.3, D5.3). It implements a certificate-based public key infrastructure with its own GiraffPlus Certificate Agency.

Certificate-based public key infrastructures are state of the practice for confidentiality and authenticity solutions, so the final GiraffPlus system fulfills the requirements R4.1 and R4.2 as specified in D1.3.

References

[1] Barsocchi P, Ferro E, Palumbo F, Potortì F. Smart meter led probe for real-time appliance load monitoring. *SENSORS, IEEE*. 2014; 1451-1454.

[2] Palumbo F, Barsocchi P. SALT: Source-Agnostic Localization Technique Based on Context Data from Binary Sensor Networks. *AmI 14 European Conference on Ambient Intelligence*. 2014; 17-32.

[3] Potorti F, Palumbo F. CEO: a Context Event Only indoor localization technique for AAL. *To appear on Journal of Ambient Intelligence and Smart Environments*.

[4] Jaimez M, Blanco J.L, Gonzalez-Jimenez J. Efficient Reactive Navigation with Exact Collision Determination for 3D Robot Shapes. *To appear in International Journal of Advanced Robotic Systems*.

[5] Koshmak G, Linden M, Loutfi A. Dynamic Bayesian Networks for Context-Aware Fall Risk Assessment. *Sensors*. 2014; 14(5):9330-9348.

[6] Rawassizadeh R, Blaine A P, Marian P. Wearables: has the age of smartwatches finally arrived?. *Communications of the ACM*. 2014; 58.1:45-47.

[7] Palumbo F, La Rosa D, Chessa S. GP-m: Mobile middleware infrastructure for Ambient Assisted Living. *Computers and Communication (ISCC), 2014 IEEE Symposium on*. 2014; 1-6.

[8] Vyas N et al. Machine learning and sensor fusion for estimating continuous energy expenditure. *AI Magazine*. 2012; 33(2):55-66.

[9] Carlson J, Murphy R. R. How UGVs physically fail in the field. *Robotics, IEEE Transactions on*. 2015; 21(3):423-437.

[10] Ogorodnikova O. Methodology of safety for a human robot interaction designing stage. *Human System Interactions, 2008 Conference on*. 2008; 452-457.

[11] Vasic M, Billard A. Safety issues in human-robot interactions. *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. 2013; 197-204.