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

D2.2 Second Prototype of sensors, Giraff platform and network system

WP related to the Deliverable:	2
Nature:	P
Dissemination Level :	PU
Version:	5
Author(s):	Filippo Palumbo (CNR-ISTI), Francesco Furfari (CNR-ISTI), Andrea Cardaci (CNR-ISTI), Maria Lindén (MdH), Gregory Koshmak (MdH), Stephen von Rump (Giraff), Silvia Coradeschi (ORU), Amy Loutfi (ORU), Ales Stimec (XLab)
Project Participant(s) Contributing:	UMA, CNR-ISTI, Giraff, MdH, ORU, XLab
Contractual Date of Delivery:	20130701
Actual Date of Delivery:	20130701

Document History

Version	Date	Type of editing	Editorial
0.1	02/05/13	Initial outline	ORU
0.2	22/06/13	Second draft	ORU
0.3	23/06/13	Addition CNR-ISTI	CNR-ISTI
0.4	24/06/13	Addition olfaction and intro	ORU
0.5	25/06/13	Addition context rec. and android sensor	ORU CNR-ISTI
0.6	26/06/13	Addition of section 6	XLAB
0.7	30/06/13	Final	ORU

Disclaimer:

No confidential material is included therein.

Deliverable Summary

This document reports on the second prototype of the system that is going to be deployed and installed at 6 test sites.
The second prototype at month 18 provides the Giraff robot and the sensors integrated in a flexible and robust communication infrastructure. A middleware solution helps to integrate the software components developed by the WP3 and WP4 and enhanced Giraff platform with safer and semi-autonomous mobility features.

Currently the prototype includes the Giraff robot, the Look4Myhealth kit, the monitoring sensors from Tunstall, additional environmental sensors, a physiological sensor for pulse oximetry measurements based on Android, the context recognition and configuration planning modules, and the remote storage and repository to collect user data. A new olfactory sensor is also considered that can be placed in strategic location like the fridge or near the garbage can and give alarms if needed. A new version of the middleware is presented and the semi-autonomy features for the giraffe robot are described.

Table of Contents

List of Tables

List of Figures

1 Introduction

Scope of the document

The document is a progress report with respect of the D2.1 and describes the second prototype of sensors, Giraff robot, GiraffPlus platform and network system. There have been improvements on all parts of the system and some novel additions. In particular the first version of the middleware is now completed based on OSGi platform and allowing plug-and-play use of sensors/actuator. A new version of the Giraff robot hardware platform has been delivered with improvements directed by user requirements. New ZigBee sensors have been selected and the use of an olfactory sensor has been investigated. Finally substantial improvements of the Giraff robot mobility have been implemented.
This second prototype of the GiraffPlus system is planned to be deployed at 6 test sites.

Deliverable structure

The document gives first an outline of what has been achieved and how it matches the DoW. In section 2 the sensor network design and implementation is described, while in section 3 the middleware is presented. Sections 4 and 5 present improvements of the Giraff robot and the addition and improvements of the sensors in particular the addition of the ZigBee sensors and the olfactory sensor. Section 6 presents an overview of the implemented security measures ensuring safe and secure communication in the GiraffPlus system. Section 7 is dedicated to the first implemented prototype of the Giraff robot with semi-autonomy.

Deviations with respect to the plan

The project is proceeding according to the plan outlined in the DoW with no deviations. In particular in the following table the requirements of the DoW are matched to what has been achieved by month 18 in the project.

Required from the DoW	Achieved at month 18
Sensor network design and implementation: A second version allowing "plug-and-play" and a richer set of sensors	Section 2 Open source solution has been considered to allow plug-and-play use of sensors/actuator and other equipment already available on the market. The OSGi platform has been chosen as the reference platform to allow smooth integration of components like drivers and gateways in order to facilitate the discovery, access and control of such sensors in the GiraffPlus ecosystem. State of the art technology ZigBee has been mainly considered for body and local area networks.
Middleware design and implementation: The first version of the middleware is delivered at month 18	Section 3 The middleware architecture described in the Deliverable 2.1 (Section 2.2) has been quite

	modified to reflect the requirements of hiding heterogeneity and distribution of both hardware and software resources. Instead of providing different components at different levels of the system to access storage and context recognition functionalities, the revised middleware architecture includes now dedicated modules at the same level. In addition to communication capabilities, the current middleware layer presents APIs to retrieve historical context data and to query for activities ongoing in the monitored home environment.
Development of Giraff robot platform: **A new version of the platform is delivered at month 18.**	Section 4 A new version of the Giraff hardware platform has been delivered. On the software side a new version has been released (2.0) that includes a completely re-designed UI, including sounds, and several improvements to the reliability. The Sentry system for managing Giraffs and user access has been updated to version 2.0.
Additional sensor selection and design: **Integration of more complex monitoring sensors by month18.**	Section 5 During Spring 2013 the work has progressed in three directions: investigation of the possibility to substitute the sensors provided by ORU with commercial sensors (ZigBee sensors); improvement of the android sensor wearable pulse oximeter with integrated fall detection and investigation of enhancement of the fall detection algorithm in the android sensor with the context recognition information; and integration of an additional environmental sensor that can detect smells.
Enabling safe and secure communication: **An intermediate version is available at month 18.**	Section 6 An intermediate version of the part of the GiraffPlus system handling safe and secure communication has been implemented following the indications provided in Section 5 of Deliverable D2.1.
Improvement of Giraff mobility: **• Automatic navigation to the "idle point" or to the recharging station completed by month 18** **• Reactive navigation to a user-specified place in the house completed by month 18.**	Section 7 Three major issues have been addressed: Reliable but inexpensive surrounding perception; Giraff self-localization; and reactive navigation to nearby places. In particular the two skills mentioned in the DoW have been developed.

2 Sensor network design and implementation

An open source solution has been considered to allow plug-and-play use of sensors/actuator and other equipment already available on the market, in addition to the already integrated sensors from Tunstall and Intellicare (see Section 2.3 of Deliverable D2.1).
The OSGi platform has been chosen as the reference platform to allow smooth integration of components like drivers and gateways and in order to facilitate the discovery, access and control of such sensors in the GiraffPlus ecosystem. State of the art technology ZigBee has been mainly considered for body and local area networks. The ZigBee standard defines a service-oriented framework for the implementation of Wireless Sensor Networks (WSNs). In the recent years ZigBee has received the attention of many research studies focused on the design and the implementation of network gateways able to access and to interact with ZigBee sensors from heterogeneous networks. To address the interoperability challenge, ZB4OSGi [1], an OSGi-based ZigBee gateway able to export the ZigBee network services in the OSGi execution environment without requiring any prior knowledge about the ZigBee protocol, is integrated in the system. ZB4OSGi exploits a 3-layered architecture enabling the access, abstraction and integration of ZigBee devices with different protocols like MQTT or SOAP/REST Web Services.

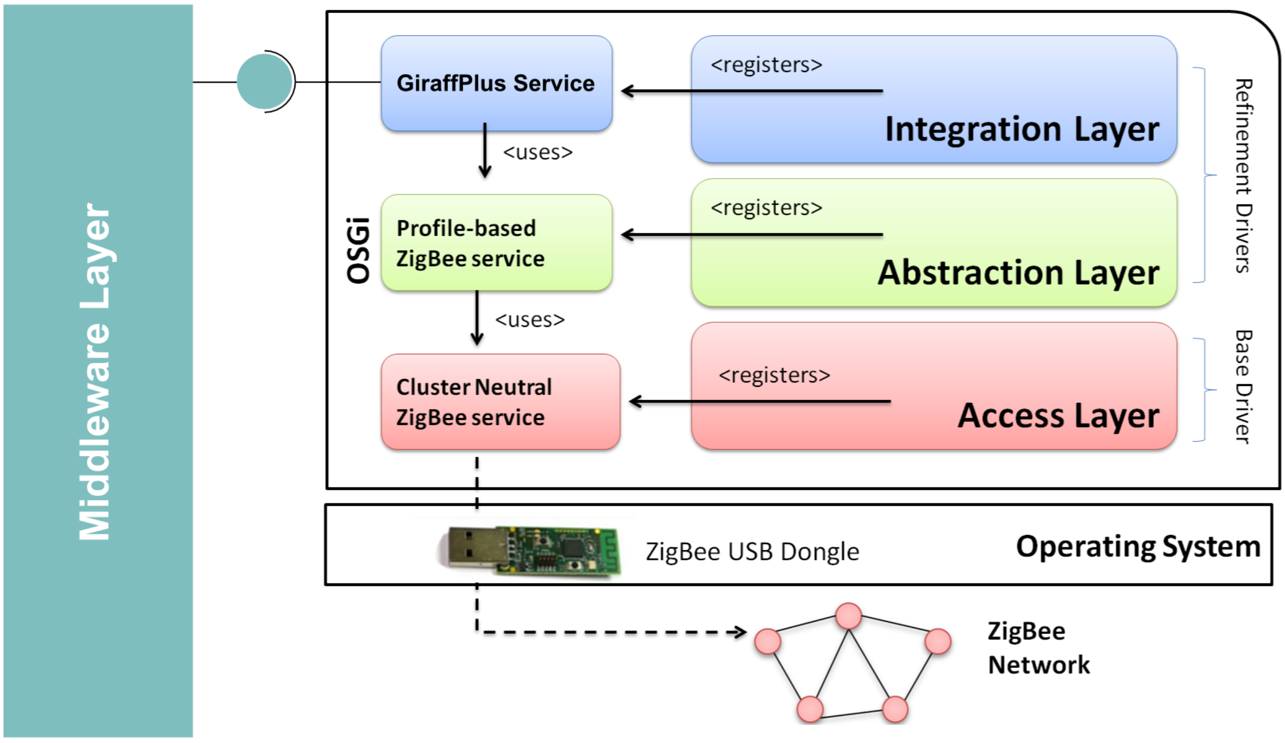

Figure 1 Integration design of ZB4OSGi in the GiraffPlus system

Figure 1 Integration design of ZB4OSGi in the GiraffPlus systemshows how the integration is done in the GiraffPlus system. A set of sensors available on the market has been tested. Sensors from Cleode [2] and Netvox [3] were purchased for testing and possible integration.

3 Middleware design and implementation

The middleware architecture described in the Deliverable 2.1 (Section 2.2) has been modified to reflect the requirements of hiding heterogeneity and distribution of both hardware and software resources. Instead of providing different components at different levels of the system to access storage and context recognition functionalities, now the revised middleware architecture includes dedicated modules at the same level. In addition to communication capabilities, the current middleware layer presents APIs to retrieve historical and real-time sensor data and to query for activities historical and ongoing in the monitored home environment.

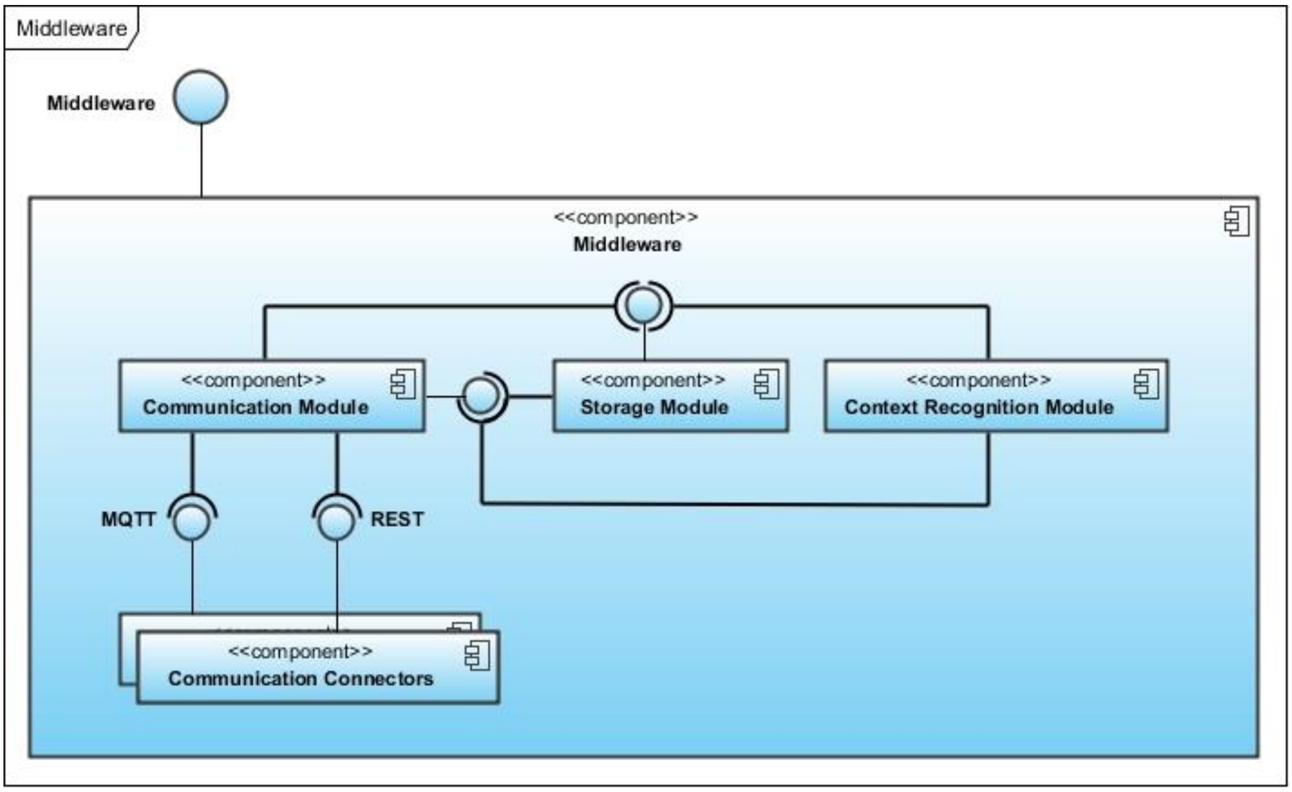

Figure 2 The Middleware Layer design

Figure 2 The Middleware Layer design shows the Middleware layer components. The Communication Connectors (formerly known as Communication Layer) have been already described in Section 2.2.2 of the previous deliverable D2.1. The modified Communication Module (formerly known as Middleware API Layer) and the new ones Storage Module and Context Recognition Module are going to be described in the following sections.

Communication Module

The previous version of the middleware API described in Section 2.2.1 of the Deliverable D2.1 has been slightly modified to reflect the architectural changes in the reference model. The new API that allows upper layer services to communicate and discover other services, exposes the following methods:

```
public void announce(ServiceDescriptor descriptor, ErrorHandler handler);
```

This method is used to announce a service on the service bus publishing its descriptor. A *ServiceDescriptor* contains all the required information to discover and access the service (*serviceBusTopic*, *id*, *type*, *category*, *room*, *contextBusTopic*, *uri*).

A service can register an *ErrorHandler* object to be notified if something goes wrong in the announcing phase. If this happens a callback is received with the relative error message:

```
public void handleError(String error);
```

```
public void remove(ServiceDescriptor descriptor, ErrorHandler handler);
```

This method is used to remove a service from the service bus when the service is stopped. A service can register an *ErrorHandler* object to be notified if something goes wrong in the removing phase.

```
public void publish(String topic, String message, Boolean retained,
                    ErrorHandler handler);
```

This method is used by a service to publish messages on the context bus. The message is a string serialized as a JSON file. A flag to set the message as persistent on the broker can be used. A service can register an *ErrorHandler* object to be notified if something goes wrong when publishing messages.

```
public void subscribe(String topic, MessageListener listener,
                    ErrorHandler handler);
```

This method is used by a service to subscribe to topics in which it is interested. Any messaged received will be handled by a *MessageListener*. A service can register an *ErrorHandler* object to be notified if something goes wrong when publishing messages. When a message is published on the subscribed topic the following callback method of *MessageListener* will be called:

```
public void messageReceived(String topic, String message);
```

```
public void unsubscribe(MessageListener listener, ErrorHandler handler);
```

This method allows a service to remove a previously registered *MessageListener*. A service can register an *ErrorHandler* object to be notified if something goes wrong when unsubscribing.

```
public void addServiceListener(ServiceDescriptor filter,
                    ServiceListener listener, ErrorHandler handler);
```

This method is the basis for service discovery in the GiraffPlus system. When a service is interested in finding a service with precise characteristics, defined by means of filter, it register a *ServiceListener* object to be notified when the required service becomes available. The filter is created defining the required field of the target *ServiceDescriptor*, if a field is left null any value for that field will be considered valid. The registered *ServiceListener* is used to receive notification about the state of the monitored target service by means of callbacks. The following are the available callbacks from *ServiceListener*:

```
public void serviceFound(ServiceDescriptor descriptor);
```

The callback used to receive asynchronously event about the monitored service, when it becomes available.

```
public void serviceRemoved(ServiceDescriptor descriptor);
```

The callback used to receive asynchronously event about the monitored service, when it is removed and becomes unavailable.

```
public void serviceChanged(ServiceDescriptor descriptor);
```

The callback used to receive asynchronously event about the monitored service, when it changes in its descriptor.

```
public void removeServiceListener(ServiceListener listener,
                   ErrorHandler handler);
```

This method is called when a service is no more interested on receiving notifications from the middleware about the status of the services related to the specified *ServiceListener*.

All the methods described are encapsulated in a dedicated OSGi bundle named *giraffplus.api*.

Storage Module

This module enables other middleware components to access the data stored in *the Giraff+ Long Term Storage* system. It exposes a number of methods that cater to the needs of various components for storage and retrieval of home configuration data – to store home configuration related data we created a set of data entities defined in the giraffplus.data.entities package – and most importantly to access the stored sensor data. As this module exposes a large number of methods beyond the scope of this document (description of all of them can be found on the wiki stated in section 0), we describe here only the most important method that enables access to the stored sensor data:

```
public String sendQuery(  String access_token,
                          ObjectId sensor_id,
                          Date start_time,
                          Date end_time) throws GiraffPlusStorageAPIException;
```

This method sends a request to the Giraff+ LTS system to retrieve a set of sensor readings for a sensor identified by a unique *sensor_id* for a time interval defined by a start and end time. To access the data one needs to provide a valid *access_token*, which is used on the server side to verify if the caller is authorized to access the requested data. If the *access_token* passes all tests, the requested data is returned in a string, which contains a JSON array of sensor readings. Deserialization is left the calling function leaving room for future developments, though a method for deserialization is implemented and exposed in the storage module. If the *access_token* does not pass the authorization tests, an exception is thrown and the attempt is logged on the server for the administrator.

Context Recognition Module

This module integrates the methods to call the Context Recognition component. The Context Recognition API is going to be implemented and it will expose the following methods:

```
public ActivitySet getActivityList (String home, String token);
```

This method retrieves the list of activities that can be monitored in a home. The home is specified by the *homeId* string and the *token* string is used for accessing the database in a secure manner.

```
public ActivityInstanceList queryActivity (String home, String token, String activity,
                                String start, String end);
```

This method queries the context recognition service for instances of an activity. The first two parameters serve the same purpose as in the method *getActivityList*. In addition, the string *activity* defines the name of the activity that is queried while *start* and *end* strings denote the limits of the temporal window in which activities are queried.

Android Version

The Android GiraffPlus project aims to extend the GiraffPlus system on the Android OS allowing mobile devices to interoperate with the existing infrastructure. The system is split into interchangeable components. Most of the business logic resides into a middleware so it's possible to have thin client applications and a set of transparent alternatives for the transport protocol.

Architecture

The system is composed of three components (Android applications) at its minimum: a client application, the middleware and a communication connector.
A typical client application uses data from the available sensors to implement a GiraffPlus service, and then the middleware takes care to serve the requests coming from the client application by using a communication connector, which represent a channel between the device and a GiraffPlus message broker (Figure 3).

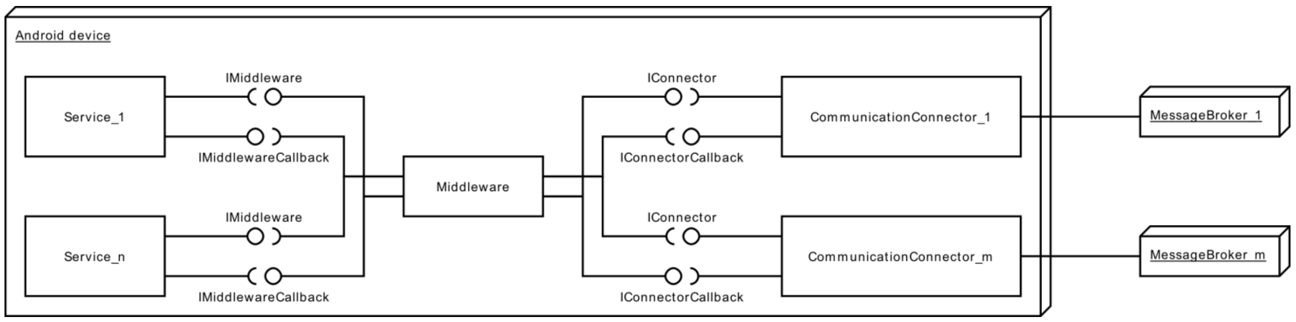

Figure 3 Architecture of the GiraffPlus system on Android

Components communicate with each other using an Inter-Process Communication (IPC) method provided by Android. Both middleware and communication connectors are implemented using

Android bound services that expose their own Android Interface Definition Language (AIDL) interface. AIDL requires that the interface must be available to the components that plan to use it. A connection with these services is established by using specific intent actions (prefixed with it.cnr.isti.giraff.android string):

- *BIND_TO_MIDDLEWARE*
- *BIND_TO_COMMUNICATION_CONNECTOR*

From the client applications point of view, the middleware side only is relevant, anyway AIDL is also used for the communication between the middleware and the connectors.
When more than one communication connector is available on the system, the middleware non-deterministically chooses one.

Interfaces

The provided interface of the middleware is a comparable subset of the usual GiraffPlus interface.

Middleware Component

This is the central component of the system, its purpose is not only to forward the client requests to the chosen communication connector, but it also needs to implement some business logic.
The middleware keeps track of the service descriptors announced or removed from the service bus so it can notify the clients that are interested to a particular service. This implies that the middleware must perform a preliminary subscribe on the *service bus*.
In the same way, for what concerns the *context bus*, it must maintain a similar correspondence to notify clients subscribed for a specific topic when a pertinent message arrives.
Keeping track of the subscription is also mandatory to avoid that one client may unsubscribe another, since the system is possibly used by many concurrent client applications. The desired (and implemented) behavior is shown in Figure 4 Subscribe / Unsubscribe example, where an unsubscribe operation is propagated to the communication connector only if there are no more clients interested to that topic.

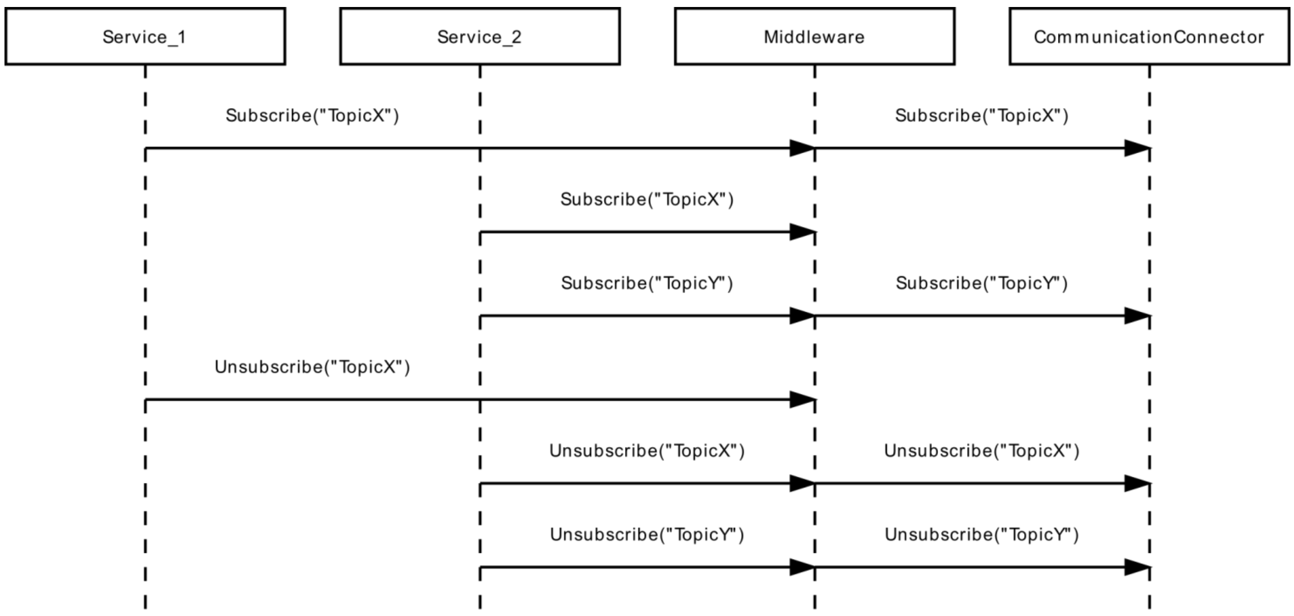

Figure 4 Subscribe / Unsubscribe example

Communication Connector Component

Communication connectors abstract the transport protocol used to reach the message broker in a way that is fully transparent to the client applications and to the middleware.

Error Handling

All the methods provided in the exposed interfaces are non-blocking (*oneway* in the AIDL jargon) this means that they cannot return any value, including exceptional status codes. Also, Android does not support inter-process exceptions by now, so each method takes an optional error listener as a parameter that is called whenever is impossible to complete a task.

Race Conditions

Due to the asynchronous nature of this system, race conditions may occur. For example, a client application that establishes a connection with the middleware may start to use the methods of the interface, but it may happen that the chosen communicator connector is not ready yet. To overcome this, a buffer of *Java Runnable* objects has been implemented on both middleware and communicator connector. This buffer has a fixed limited capacity and serves to fill the aforementioned time gap in the bootstrap phase of the process.

Best Practices

Client Applications

It's likely that common client applications do a lot of sensors gathering and background computation, so it's important to take some countermeasures to avoid that the OS kills the application because considered inactive. A strategy is to register all the sensors listeners in an

Android started service context that has the foreground flag set, so the system doesn't attempt to kill it to reclaim more memory.

Communication Connectors

Though it's not mandatory, it's highly advisable that each communicator connector implements some kind of buffering as described in 0.

Software Development

Code Management

As stated in the Section 3.1 of the Deliverable D2.1 the code is organized and maintained on an SVN server. To improve code organization a new structure of this repository has been set up. Since we have released a first stable version of the system, we created a *trunk* directory where stable code is stored. The old directories divided into WP folders have been were kept. Developers from different WPs can use those folders as *sandboxes*.
The following shows the new hierarchical view of folders:

```
\trunk
      \android
      \backend
            \giraffplus.storage.access_token.revoker
            \giraffplus.storage.ws
            \giraffplus.storage.access_control
      \commons
            \giraffplus.data.entities
      \middleware
            \giraffplus.api
            \giraffplus.drivers.tunstall
            \giraffplus.middleware
            \giraffplus.mqtt
            \giraffplus.pom
            \giraffplus.rest
            \giraffplus.storage.api.OSGi
            \giraffplus.tester
            \rundir
            \samples
```

Configuration and Installation

Tutorials and how-to on how to configure, install, and run the middleware and its modules are presented as wiki pages hosted on the CNR-ISTI server. The wiki can be accessed at the following address:

http://wnlab.isti.cnr.it/giraffplus/

Those pages will be updated regularly to support developers to easily integrate their components into the GiraffPlus system.

4 Development of the Giraff robot platform

Giraff AB has delivered a new version of the Giraff hardware platform with a new top section for higher reliability and the ability to flip the screen 180 degrees so the user easier can see who is calling. It also includes a touch screen so together with the DVPIS the user can interact more with the caller. The new hardware platform also includes an updated microphone.
On the software side a new version has been released (2.0) that includes a completely re-designed UI, including sounds, and several improvements to the reliability.
The Sentry system for managing Giraffs and user access has been updated to version 2.0. It includes an updated design but also several stability improvements and new features such as the possibility to get and E-mail or a SMS if a Giraff robot goes offline.

5 Additional sensor selection and design

The first deployed prototype of the system was completed in December 2012 and included the main sensors that are going to be used in the GiraffPlus system, that is the environmental sensors provided by Tunstall and the physiological sensors provided by Intellicare. In addition a few sensors provided by ORU were integrated, which provided additional environmental monitoring.
An android based sensor system was also integrated into the GiraffPlus system. The sensor system includes a remote monitoring system consisting of a wearable pulse oximeter and an integrated fall detection algorithm, utilizing the accelerometers of an android phone. The sensor is attached to the end-user and communicating with the android phone, which collects, saves and process the data (pulse rate, oxygen saturation, and fall detection). The system is only considered as a complementary platform for extended physiological monitoring and integrated into the project as a Middleware component.
During Spring 2013 the work has progressed in three directions: investigation of the possibility to substitute the sensors provided by ORU with commercial sensors (ZigBee sensors); investigation of enhancement of the fall detection algorithm in the android sensor with the context recognition information; and integration of an additional environmental sensor that can detect smells.

ZigBee sensors

As stated in 2 Sensor network design and implementation ZB4OSGi will be used as a gateway for ZigBee sensors. Since we want to use sensors with a high level of user acceptance, available sensors on market instead of development kit have been tested. After a deep investigation on available solutions, sensors from Cleode and Netvox have been chosen as candidate options. Following the list of purchased sensors from Cleode:

- ZDoor: detects intrusion with a magnet and reed-switch mechanism that reports an alarm during unauthorized entry by door or window.
- ZLum: illuminance sensor who contains a light sensor that detects the level of light in a room.
- ZMeter: measures the instantaneous power and cumulated consumption of the connected device.
- ZMove: a wireless PIR motion detector.

A preliminary test on these devices has been performed; unfortunately, the ZB4OSGi stack discovers the device (the hardware node) but not the endpoints on it (the actual implementation). More extensive tests will be carried out in the coming months.
More encouraging results were obtained from tests performed on Netvox devices. The developer community of ZB4OSGi is testing these devices too. An exhaustive list of sensors under testing is available on this address with the ongoing results from different testers. Following a list of the most promising devices:

- Z800: Mains Power Outlet
- Z711: Wireless Temperature & Humidity Sensor
- ZB01: IR Motion Detector sensor
- Z601: Warning Device
- Z307: Fall Sensor
- ZA02: Smoke Detector with Heat Sensor
- Z302G: Light Sensor
- Z302A: Door Contact Sensor
- Z302B: Light Sensor Switch
- Z302D: Emergency Button
- ZBWS3B: Three Gang Switch

We choose these subset of sensors because seems to be discovered and accessed by the ZB4OSGi stack. Only some commands are not handled correctly. More extensive tests will be carried out in the coming months.
A set of those sensors will be hopefully available and working for the end of the year.

Android based sensor 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 we installed android version of the middleware and established communication between the system and GiraffPlus database. 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 will be 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, we perform a simple check and 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.
Meanwhile, a new android version of the middleware has been successfully installed into the mobile-phone and now requires some minor parameters fixes.

Olfactory sensor

Among the user requirements specified in the deliverable D1.1 it is stated that the GiraffPlus should provide caregivers with direct notice if the house presents an unusual setting (first potential evidence of a personal disorder). The detection of the presence of smells in particular from food that has gone bad can be both used as an indication that the health of the person is deteriorating and as a reason to send an alarm to relatives and/or home care personal.
Olfactory sensors were not considered in the first year of the project where focus was placed in simple sensors. In the spring of 2013 we have started investigating the possible of using these more complex sensors in the kitchen.
An olfactory sensor consists of a semiconductor tin dioxide sensor chemical sensor. The sensor has a broad selectivity to a family of odours, and the first prototype has used one sensor selective to volatile organic compounds (VOC). The sensors electrical properties change when exposed to a target gas, thus providing the ability to quantify an odour. Although the precision of the sensor is poor compared to laboratory analysis (GC-MS), olfactory sensors can be coupled with several sensors of cross selectivity to provide a "fingerprint" of an odour. Used together with a pattern recognition component it is possible to train a system on specific odours which are to be recognized. Today, research in electronic olfaction has been used in a number of applications ranging from food detection, environmental monitoring, and specific robotic applications [8-10]. A subset of the GiraffPlus constellation has a particular expertise in developing such sensor arrays with specific focus on data interpretation from the sensor.

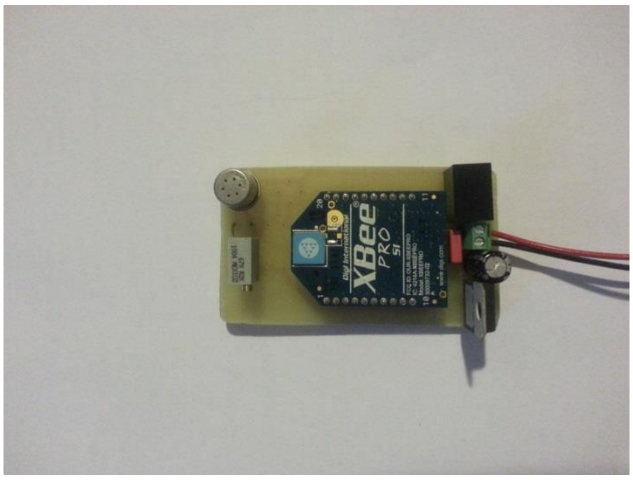
Figure 5 The gas sensor coupled with a Xbee device.

The olfactory sensor which has been developed currently is a singular gas sensor. The sensor is coupled with an Xbee device which transmits sensors readings at an interval of 1 Hz. Current issues to be addressed is power consumption (which is high and therefore difficult to run the sensor on battery alone), and sensor drift (which can occur after longer periods of monitoring e.g. months). So far the first prototype of the sensor has been developed and will be deployed at the Ängen apartment for further verification and testing. Of special interest is the rising of alarms in case of bad smell formation. In particular a sensor will be placed in the fridge and one in the cupboard of the trash bin that will detect food deterioration. The results from the olfactory sensor can also be used to improve context recognition (e.g. smell as an indication of cooking) and possibly the context recognition can be exploited to eventually improve the assessment of an odor (e.g. cupboard to trash bin has not been opened for long time could indicate that a VOC originates from the trash).

6 Enabling safe and secure communication

In the GiraffPlus project we are aware of various issues regarding the importance of privacy and security as described in Section 5 of Deliverable D2.1. Following sections describe how we address issues listed in the aforementioned deliverable in the current version of the GiraffPlus system.

Confidentiality

To ensure confidentiality of data we use a two layer approach. For the first layer of security we set up a **Virtual Private Network** (VPN), which uses certificates issued by the GiraffPlus VPN Certificate Authority to encrypt all data traffic between various computers and components in the GiraffPlus ecosystem. Thus we make sure that nobody outside the GiraffPlus network can listen in on the communication between various components.
As a second layer of security we set up a **GiraffPlus Certificate Authority**, which issues certificates for public-key cryptography to all components of the GiraffPlus software stack and all GiraffPlus system users and the **GiraffPlus Access Control** service, which controls access of users to the stored data using the *access control list*. The access control list can only be edited by the owners of data or by selected system administrators (both, owners of data and administrators, need to present valid certificates).
Each piece of information stored in the **GiraffPlus LTS** (Long Term Storage) system, has an owner and each primary user is the owner of any sensor data regarding him/her as well as all home configuration data. Before transmitting data the sensor component encrypts the sensor data using the home's private key (each home instance of the middleware has a separate certificate, which is also tied to the primary user in the home, who is the owner of all corresponding sensor data). In this way the GiraffPlus LTS system can verify the origin of each piece of information and can store the sensor data encrypted using owner's public key in the database.
On any data access the GiraffPlus LTS system contacts the GiraffPlus Access Control, which logs the access attempt and verifies if the user/component trying to access the data has proper access permissions. If the access is granted the GiraffPlus Access Control returns the private key, which enables the LTS system to decrypt the retrieved data, which is in turn encrypted using the public key of the component requesting the data. This component then decrypts the data using its

private key to reveal the requested data. Thus we ensure the confidentiality of all data stored in the GiraffPlus LTS system.

Integrity

Since all sensor and configuration data is encrypted using the procedure described in the previous section the integrity of the data is ensured. The data is double encrypted: fist encrypted by the VPN to ensure integrity en route and encrypted using the owner's public key to ensure it cannot be tampered with without a proper private key.

Availability

The *availability* of data is ensured using the replication and sharing features of the underlying MongoDB database. In addition we have set up an additional system of daily backups that prevents any data loss. To improve the *availability*, *reliability* and *integrity* for the upcoming installation in the first test sites, XLab began migrating servers to a completely new and secure server infrastructure. This migration is seamless and will at no point interrupt the normal functioning of the rest of the project.

Authenticity

Since all sensor data is encrypted using private keys ensuring authenticity is straightforward: if the data can be decrypted using proper public keys, then the sender is verified since only the sender possesses the corresponding private key.

Non-repudiation

Similar to authenticity the non-repudiation is straightforward. If the sender is authenticated as discussed above, and if the data integrity is guaranteed, then the non-repudiation problem is solved.

7 Improvement of Giraff mobility

The goal in the improvement of Giraff mobility is to develop and implement a variety of algorithms to both facilitate the Giraff guidance and mobility, and to provide it with safer motions, warning about or/and self-avoiding obstacles along its way. Three major issues have been addressed: Reliable but inexpensive surrounding perception; Giraff self-localization; and reactive navigation to nearby places. In particular the following skills have been developed by month 18 in the project for providing the Giraff platform a certain level of autonomy:

• Automatic navigation to the "idle point" or to the recharging station.
• Reactive navigation to a user-specified place in the house.

The specific improvements implemented in the Giraff robot are described in the following sections.

Improved Surrounding Perception

Previous versions of the Giraff Robot relied on 2D radial laser scanner for perceiving the environment. These sensors are very effective for self-localization but in terms of obstacles detection they exhibit a serious limitation: the robot can only detect obstacles in the plane scanned by the sensor, with the consequent risk of collision with objects out of this plane. This limitation has been addressed by enhancing the Giraff robot with a RGBD camera (Prime Sense Carmine 1.09). The RGB-D sensors bring significant benefits for the obstacles avoidance system by providing 3D information of the environment. Refer to sensors selection criteria and justification in the deliverable "D1.2-Technological Components Specification".
In summary, the more relevant features of the Prime Sense Carmine 1.09 Camera are:

RGB-D Camera CARMINE 1.09 Features	
Operation Range	0.35-1.4 [m]
Field of View (Horizontal, Vertical, Diagonal)	57.5,45,69(H,V,D) [deg]
Depth Image Size	640 x 480 (VGA)
Color Image CMOS @30FPS	640 x 480 (VGA)

Table 1. Main characteristics of the PrimeSense Carmine 1.09 RGB-D Camera

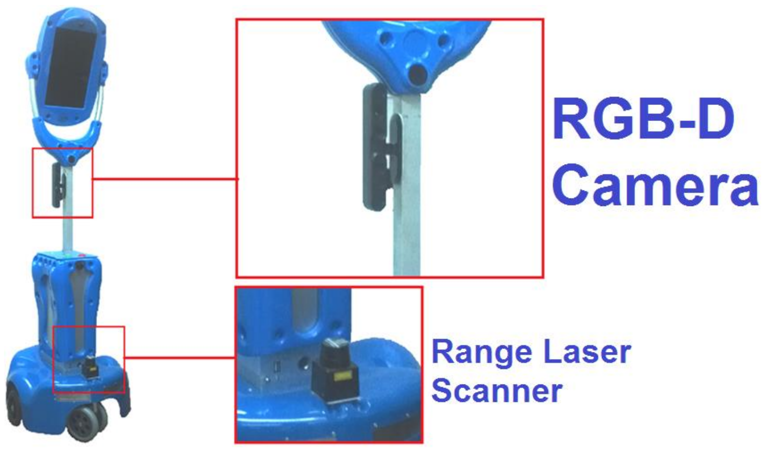

Figure 6. Sensorial system of the Giraff robot.

Reactive Navigation System

A reactive navigator automatically guides the robot to a nearby point negotiating the detected obstacles. It uses the robot pose and the sensor observations to derive the proper motors' commands to go from a point 'A' to a point 'B' negotiating any (possibly dynamic) obstacle found in the path [1].
A 3D reactive navigator has been developed by UMA to avoid obstacles at different heights by including 3D information of the environment [2]. For this purpose, the robot volume is modeled by a number of prisms or height sections (see **Figure 7**). In addition, the detected obstacles are sorted in height bands associated to the height sections used to model the robot. The 3D reactive navigation algorithm combines the information provided by the obstacles avoidance system for each of these horizontal bands and selects a safe motion command (linear and angular velocities

set to the motors controller), taking into account the whole 3D model of the Giraff robot, the obstacles around the robot and the final destination selected.

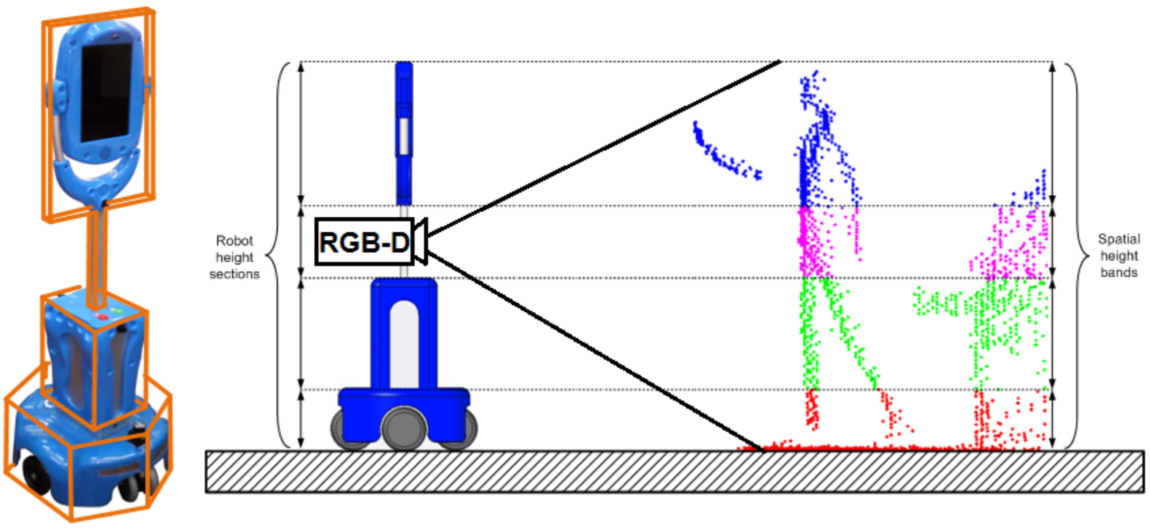

Figure 7. Geometric Model of the Giraff robot. The obstacles avoidance system takes into account four height bands for the 3D Reactive Navigator.

Hybrid Geometrical-Topological Map Building and Topological Navigator Module

A 2D geometrical map for semi-autonomous mobile robots is used to localize the robot in real-time by applying well known robotics techniques. Concretely, for building geometric maps of the environment, the system runs an implementation of the Iterative Closest Point algorithm (ICP). ICP aims to register point-based data coming from a number of scans by finding the geometrical transformations that minimizes the square error between the registered points. Moreover, the constructed geometric map is enriched manually in order to produce a suitable schematic-topological map (see Figure 8). For that, a GiraffPlus technician must perform the following two steps:

- Create a schematic map by adding graphical elements that represent pieces of furniture and environment structures, like doors, walls, etc., and
- Create a topological map by selecting distinctive places, connections, and friendly names, e.g. kitchen, corridor, bedroom, etc.

While the former only aims at enhancing the visualization of the environment, the latter, i.e., the creation of a topology, including human-friendly labels, permits the user to identify particular rooms of the elder home and to command the robot using this high-level information.
The Topological navigator exploits the topological map complementing the reactive navigation system which is not appropriated for achieving far destinations. It executes an A* algorithm for finding the best path, i.e. sequence of nodes with geometrical coordinates, to arrive to the destination given by the user. The coordinates of intermediate points in the sequence are sent to the reactive navigator. More information is provided at [3],[4].

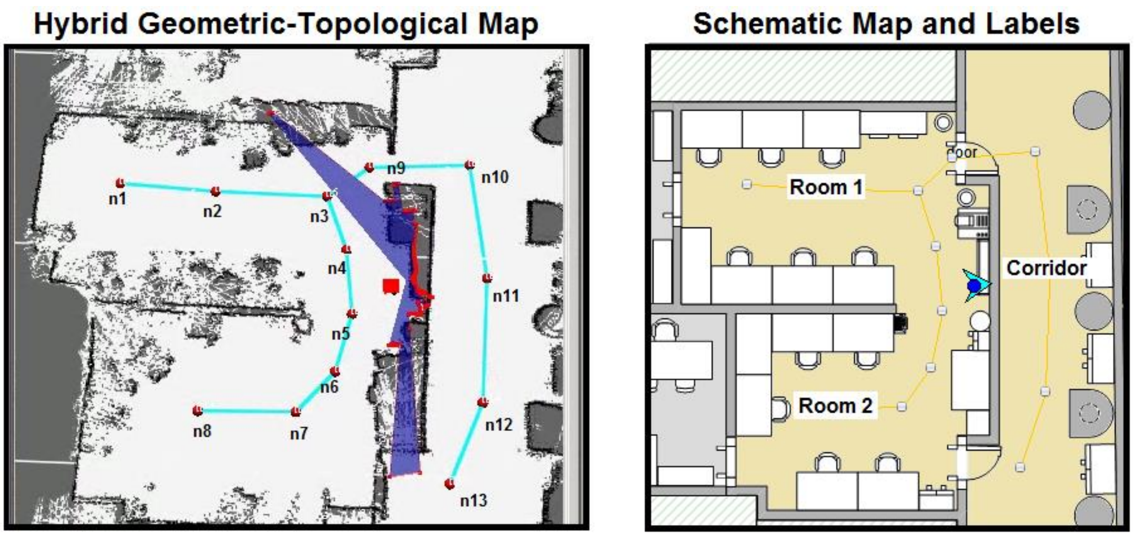

Figure 8. Left) Hybrid geometric-topological map. Topology has been manually created. Right) Schematic map where some nodes of the topology have been renamed with friendly labels that represents different parts of the environment.

Software Architecture

The Giraff platform software architecture consists of two parts: a) the client interface that runs on the secondary user´s computer, and b) a robotic architecture that runs in the Giraff robot to manage its motors and sensors. On both sides, the software provided by Giraff AB Technologies has been completed with the new functionalities described above.

Regarding the client interface, called Pilot, the UMA group has extended its functionality through a plug-in based on Java and the Giraff Plug-In Development Framework. This plug-in appears as an interactive panel on the Pilot application and allows the users to manage the new functionalities.

On the robot side, the improved robot architecture has been integrated with the Giraff AB software through a Java class. This class, called GiraffPlusAVR, fulfills two important functions:

- To communicate the GiraffAB software through a local TCP Socket with the software developed by the UMA group for self-localization and semiautonomous navigation.
- To manage the concurrent access of Giraff SW and UMA SW components to the robot sensor readings and motors controller.

Finally, a new module has been developed and integrated on both software applications, client and robot side, to allow communications with the GiraffPlus broker via the MQTT protocol.

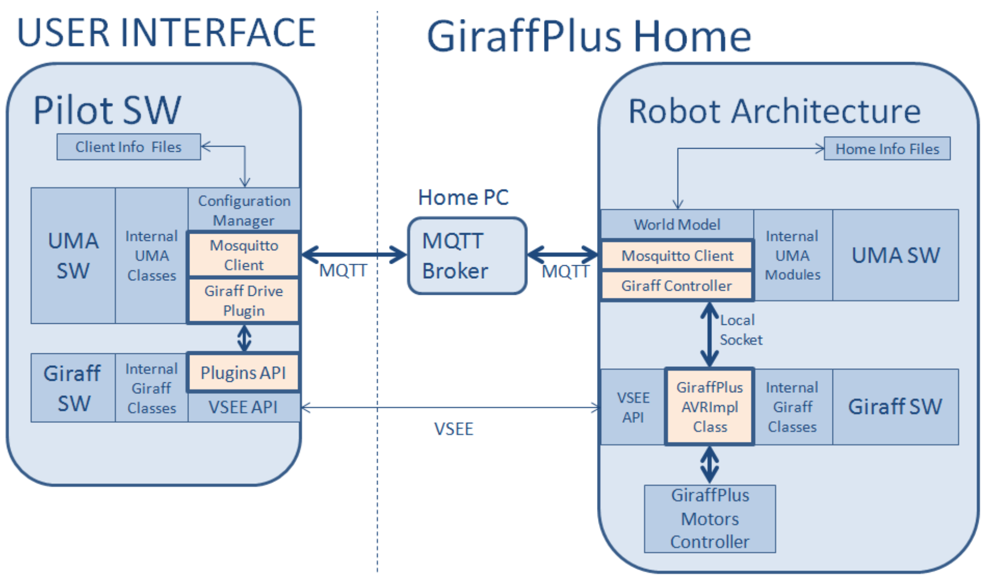

Figure 9. Giraff robotic platform software architecture.

Graphical User Interface

The secondary user´s interface has been extended by including two plug-ins, the first of them is developed by the UMA group for the improvement of the Giraff mobility, it is called OpenMORA plug-in. The second one is developed by Giraff AB and provides the Pilot application with an assisted procedure for docking the robot at the recharging station.
Figure 10 depicts the Pilot Interface. New contributions are displayed with numbered boxes in the figure. These contributions are:

1. Interactive viewport where the schematic map and the robot pose are represented. This viewport allows the user to select destinations for the Giraff Robot by clicking on intuitive labels.
2. Specific Viewport controls. For example, a button for finding the robot in the schematic map or creating a new label for semiautonomous navigation.
3. A plug-in selection toolbar. This toolbar is displayed by right clicking on the video image and allows the user to set the Pilot application to three different modes: Standard mode (Plug-ins disabled), OpenMORA (for assisted navigation), or Docking Assistant.

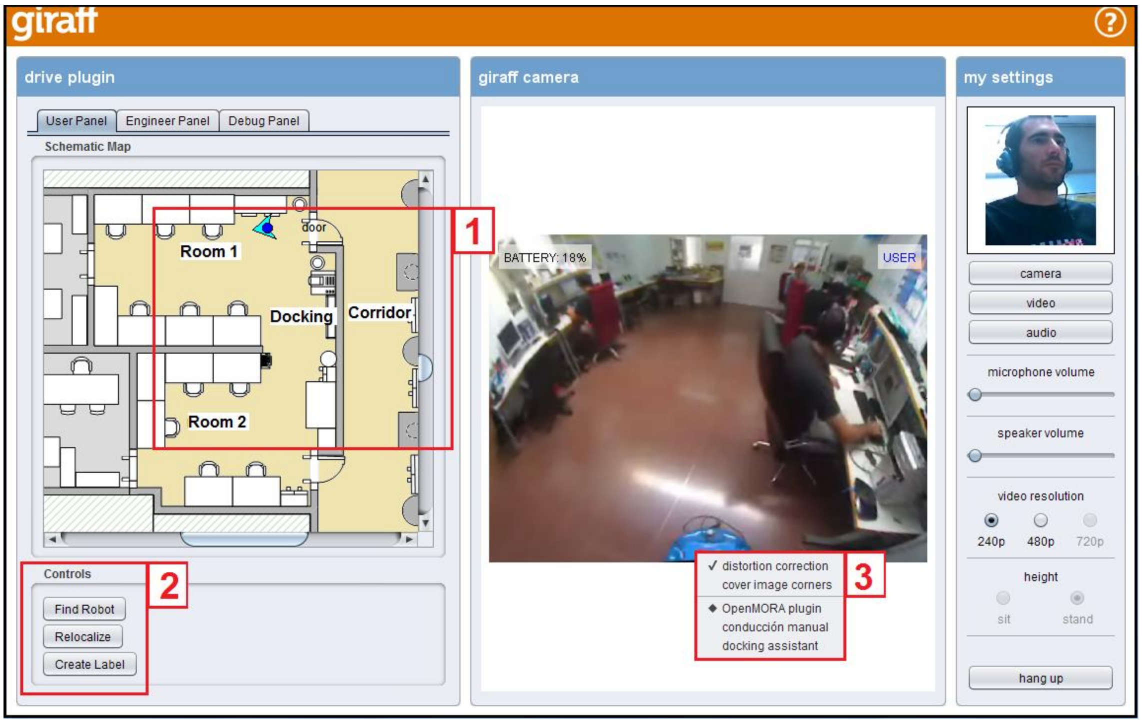

Figure 10. Pilot´s application extended with a plug-in developed for the improvement of Giraff Mobility.

8 Conclusion

This document presents an improved version of the GiraffPlus hardware and sensor network system with respect to the one presented in D2.1. In particular a revised version of the middleware has been implemented and integrated in the system and semi-autonomy functions have been integrated in the Giraff robot. In addition a new Giraff robot platform has been provided and additional sensors have been considered namely ZigBee and olfactory sensors. The development of the system has proceeded according to plan and the system is in the process to be deployed in the first six test sites.

References

[1] http://zb4osgi.aaloa.org/

[2] http://www.cleode.fr/en/index_beepack.php

[3] http://www.netvox.com.tw/

[4] J.L. Blanco, J. Gonzalez-Jimenez, J.A. Fernandez-Madrigal, **"Extending Obstacle Avoidance Methods through Multiple Parameter-Space Transformations"**, *Autonomous Robots,* vol. 24, no. 1, 2008.

[5] J. Gonzalez-Jimenez, J.R. Ruiz-Sarmiento, C. Galindo, **"Improving 2D Reactive Navigators with Kinect"**, *10th International Conference on Informatics in Control, Automation and Robotics (ICINCO),* Reykjavic, Iceland, 2013.

[6] J.L. Blanco, J. Gonzalez-Jimenez, J.A. Fernandez-Madrigal, **"Optimal Filtering for Non-Parametric Observation Models: Applications to Localization and SLAM"**, *The International Journal of Robotics Research (IJRR),* vol. 29, no. 14, 2010.

[7] J. Gonzalez-Jimenez, C. Galindo, F. Melendez-Fernandez, J.R. Ruiz-Sarmiento, **"Building and Exploiting Maps in a Telepresence Robotic Application"**, *10th International Conference on Informatics in Control, Automation and Robotics (ICINCO),* Reykjavic, Iceland, 2013.

[8] A.J. Lilienthal, A. Loutfi, T. Duckett, **Airborne Chemical Sensing with Mobile Robots. Sensors**, Volume 6, pages 1616- 1678, 2006.

[9] M. Trincavelli, S. Coradeschi, A. Loutfi**, Online Classification of Gases for Environmental Exploration**. In Proc. IEEE International Conference on Intelligent Robots and Systems, Volume 1, pages 3311- 3316, 2009

[10] Längkvist, M.; Coradeschi, S.; Loutfi, A.; Rayappan, J.B.B. **Fast Classification of Meat Spoilage Markers Using Nanostructured ZnO Thin Films and Unsupervised Feature Learning**. *Sensors* 2013, *13*, 1578-1592.