RUBICON
RUBICON Robotic UBiquitous COgnitive Network
Project No.: 269914

SEVENTH FRAMEWORK PROGRAMME

D1.2 – Integration of Communication Layer and Robotic Components

Editor: Claudio Gennaro CNR
Mauro Dragone UCD

Contributor(s): Alessandro Saffiotti ORU
Héctor Lozano Tecnalia
Maurizio Di Rocco ORU
Claudio Vairo CNR

Dissemination level	
X	**PU** = Public
	PP = Restricted to other programme participants (including the Commission Services)
	RE = Restricted to a group specified by the consortium (including the Commission Services)
	CO = Confidential, only for members of the consortium (including the Commission Services)

© Copyright RUBICON - All rights reserved

Issue Date	30/04/201
Deliverable Number	D1.2
WP	WP 1 - Communication Layer
Status	☐ Draft ☐ Working ☒ Released ☐ Delivered to EC ☐ Approved by EC

Document history			
V	Date	Author	Description
0.1	13/02/2012	Mathias Broxvall	Creation of LaTeX template
0.1	11/03/2013	Claudio Gennaro	Very first draft
0.2	02/04/2013	Claudio Gennaro	Added Chapter about islands routing
0.3	22/04/2013	Maurizio Di Rocco	Added Chapter about PEIS-ROS
0.8	24/04/2013	Claudio Gennaro	Added Chapter about SALAD
0.9	02/05/2013	Héctor Lozano	Internal review
1.0	07/05/2013	Kylie O'Brien	Final QA review

Disclamer The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

The document reflects only the author's views and the Community is not liable for any use that may be made of the information contained therein.

Executive Summary

This deliverable (D1.2) describes the work carried out in the Task 1.2 *Integration of Robotic Components and WSAN*. In particular, T1.2 concerns the integration of previously existing robotic devices and software components to satisfy the requirements posed by Task 1.1, which involves the generic integration of pre-existing robotic communication tools (Part of the PEIS-ROS bridge) and the integration of specific PEIS Ecology functionalities to non-PEIS side of Communication Layer. This latter task includes the routing between islands and the access to non-WSN sensors inside WSN network.

Contents

Figures

Abbreviations

AF	Agent Factory
BDI	Belief Desire Intention (agent model)
MAC	Media Access Control
PEIS	Physically Embedded Intelligent System
ROS	Robotic Operating System
RUBICON	Robotic UBIquitous COgnitive Network
SPP	Serial Port Profile
SW	Software
WSN	Wireless Sensor Network
WSAN	Wireless Sensor and Actuator Network

Chapter 1

Introduction

This report describes the integration of existing robotic, sensing, and actuation components in the communication infrastructure of RUBICON and communication between remote WSAN islands based on TinyOS. This integration is based on the newer version of the PEIS-kernel, generation G6, described in D1.3.1 and D1.1 and on the gateway and proxy functionalities of the final release described in D1.4.

The document is divided into the following three parts:

- The interface between the PEIS middle-ware and ROS. This interface is used in order to allow multiple robots equipped with ROS (Robot Operating System) to be part of the RUBICON system, i.e. to exchange data with other RUBICON nodes.
- Support for multi-island communications. The aim of this feature is twofold: first, to guarantee the scalability of TinyOS WSAN, and second, to permit robotic devices to be integrated with the TinyOS WSAN. In fact, robots (i.e., PCs) can be viewed as special islands consisting of one just one node, which can be a basestation or basestation + sink mote connected via USB.
- Access to non-WSAN Sensors. This is about the integration of the SALAD platform, which allows the interconnection and cooperation among several systems of sensors and actuators in one platform, providing an easy access to the information and a flexible environment for application development.

The latter two aspects rely on the development of a gateway that is described in the deliverable D1.4.

Chapter 2

PEIS-ROS Bridge

In this section a description of the interface between the PEIS middle-ware and ROS is provided. This interface is used in order to allow multiple robots equipped with ROS (Robot Operating System) to be part of the RUBICON system, i.e. to exchange data with other RUBICON nodes. Such choice is dictated by the fact that ROS entails the presence of a centralized master program (ROS master) managing the connections between cooperating modules; this structure doesn't fit the purpose of having a distributed and dynamic system where robots can join and leave the network at any time.

To fill this gap, an interface enhancing the capabilities of ROS components has been implemented. Each robot leveraging ROS can be seen like a monolithic structure internally constituted by a set of nodes. The data exchange between these nodes is managed by the related ROS master while the data flow within the RUBICON network is achieved through the PEIS-ROS interface (see Fig. 2.1).

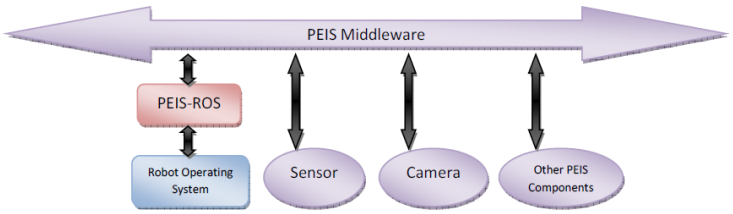

Figure 2.1: the communication between ROS nodes and PEIS middleware is managed by the PEIS-ROS interface

Capabilities are enhanced in both receiving and sending directions. The basic structure of the program implementing a PEIS-ROS node is depicted in 2.2:

In particular the program can be divided into the following parts:

- PREAMBLE: this part, generally present at the beginning of the program, defines tuples from which the node should gather data as well as tuples that have to be published by the node itself. Note that these routines can exploit the capabilities of the PEIS middle-ware to subscribe to abstract tuples. In this portion of code, the routines related to the callback section are launched.
- CALLBACK SECTION: this part is constituted by routines managing incoming data from other RUBICON

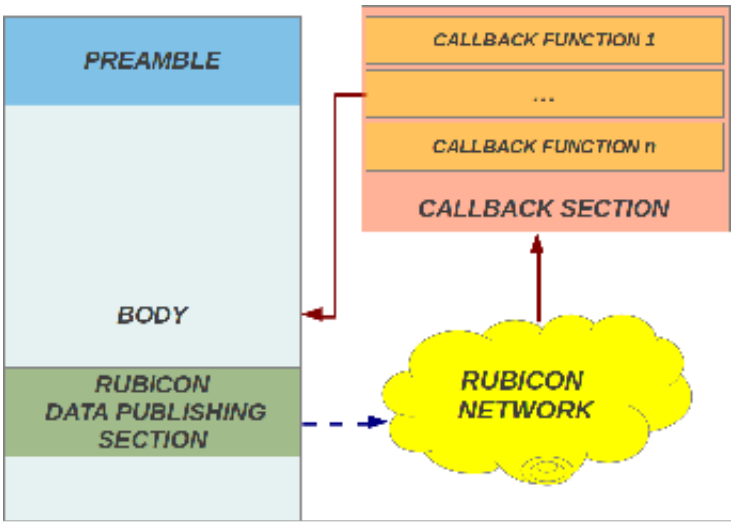

Figure 2.2: interface between a Ros node and the RUBICON network

nodes. In particular they asynchronously update data structures that are managed by the BODY of the program

- BODY: this part is the core of the program and usually interacts with other internal ROS nodes depending on the received data from the RUBICON network. During the execution of this portion of program, that is typically an infinite loop, data structures are modified upon the receiving of new data or information can be sent out over the network through standard PEIS routines
- RUBICON DATA PUBLISHING SECTION: this section writes into tuples meaningful data for other RUBICON nodes

Usually the most likely exchanged data refer to high level events (e.g. command received, task accomplishment notification) about which the nodes are able to reason about. Beside this configuration, the PEIS-ROS interface also offers the possibility to split standard ROS packets into tuples that can be read by other RUBICON nodes. Each ROS message is constituted by a set of static sub-packets; one of them, *data field*, contains the payload. While static sub-packets have a permanent representation, i.e. they are always expressed through the same data structures, the *data field* varies in accordance to the specific ROS packet: in this case the content of this field is written into a binary tuple that has to be properly managed by the related subscribers. The decomposition of a ROS packet into tuples is depicted in Fig. 2.3. This feature is useful when raw message exchanges between robots exploiting ROS are needed although this modality suffers the lack of a ROS-master and therefore introduce all the related problems concerning synchronization. These side effects can only be partially mitigated by the API provided by PEIS middle-ware.

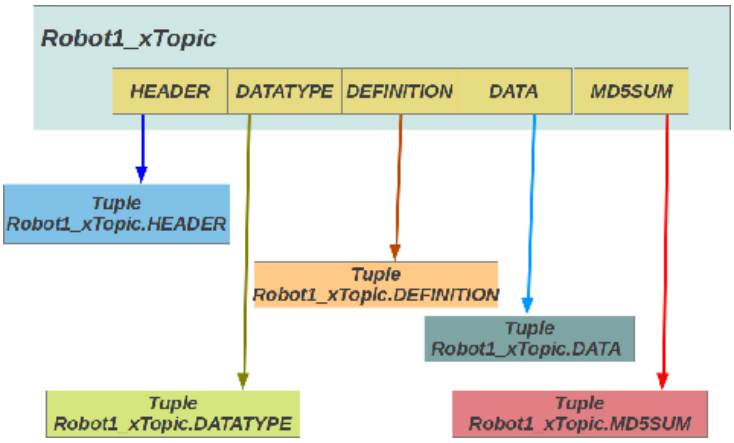

Figure 2.3: Decomposition of a ROS package in Peis-Tuples

Chapter 3

Routing between WSAN Islands and Robotic Devices

3.1 The Topology of the Communication Layer

One of the requirements for the Communication Layer is that the topology and the routing mechanisms must support a growing number of devices without degradation of performance in communications. Moreover, the Communication Layer must be distributed across the Motes, PCs, and Robots in order to minimize network bandwidth, latency, and energy consumption. In order to fulfill these requirements, the Communication Layer is based on a hybrid topology, in which a cluster of motes, called islands, are interconnected by mean of the PEIS "whiteboard" through a special mote acting as hub. This mote is called *sink* and it is in turn connected via USB with a basestation (a PC) that eventually is linked to the PEIS network. The nodes belonging to an island communicate with each other through the shared standard channel TinyOS (see Figure 3.1). This type of connection can be seen as a *bus topology*, in which messages are broadcast on the same mean to all nodes. Each node checks the destination address in the message header, and processes the messages addressed to it. The connectivity between distinct islands relies on a *second level* of connection provided by PEIS. The central hub is crucial to ensure the routing. All the nodes are connected to the central hub (i.e., the sink node together the basestation) and hence all the messages go through the central hub before going to the individual nodes. The main advantage of this two layer topology, which resemble a tree topology, is that it is easy to add new nodes/machines into the existing network. This topology is often refereed in literature [2, 1] as *Cluster Based Model*, the routing in these models is recognized as energy efficient compared with direct routing and multihop routing.

In addition to supporting network scalability, cluster based model has numerous advantages. It can localize the route set up within the cluster and thus reduce the size of the routing table stored at the individual node. Clustering can also conserve communication bandwidth since it limits the scope of inter-island interactions to sink node and avoids redundant exchange of messages among sensor nodes.

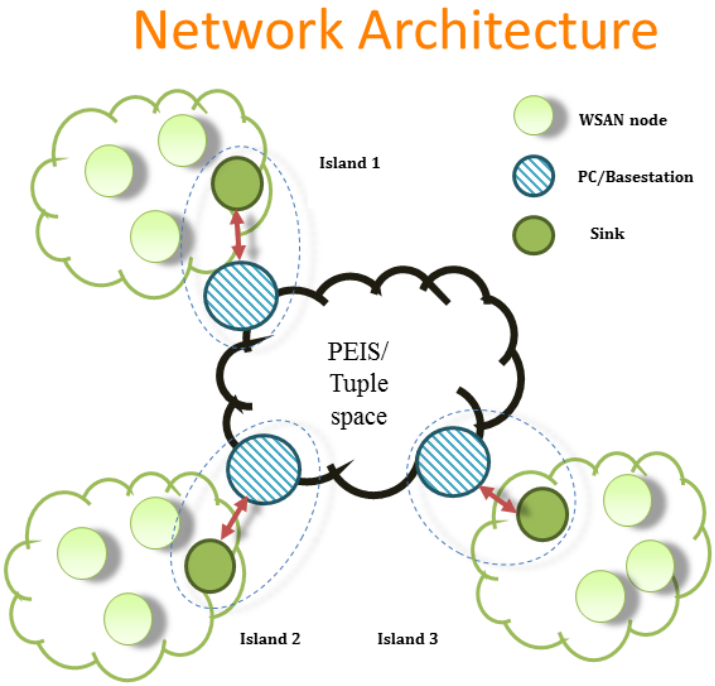

Figure 3.1: The general organization of the WSAN network

3.2 The Intra-Island Connectivity

The intra-island connectivity is ensured by the communication capabilities offered by TinyOS. Each node of the island is therefore able to communicate directly with other nodes without having to pass from the sink and without the use of algorithms for multi-hop. This has the advantage of simplicity, but requires that each node is within the communication range of the other nodes (see Figure 3.2).

3.3 The Inter-Island Connectivity

Connectivity between islands (inter-island) is guaranteed by the PEIS middleware. Whenever a node of an island must send a message to a remote node this must first be sent to the sink node, which in turn will forward it, taking advantage of the basestation, to the receiving basestation through PEIS. The latter is connected to the sink that then forward the message to the node of the island under its responsibility.

Figures 3.3 and 3.4 show, respectively, the message routing from a mote to PC/Robot connected to PEIS, and the

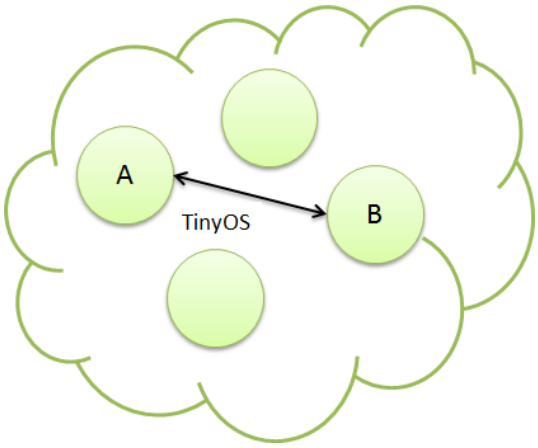

Figure 3.2: The intra-island communication.

message routing from two motes belonging to different island.

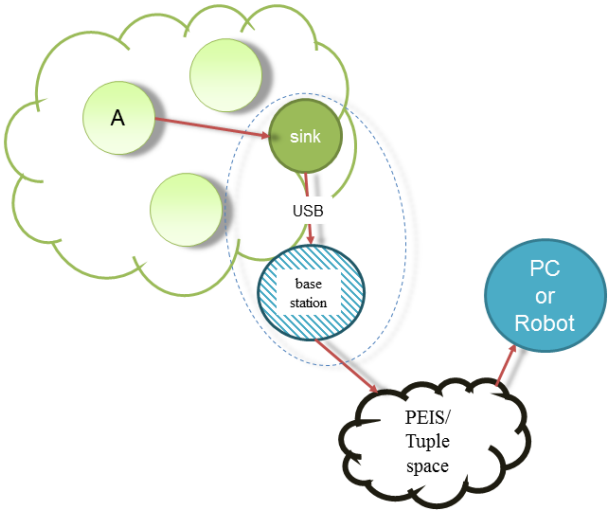

Figure 3.3: The inter-island communication between a mote and a robot/PC.

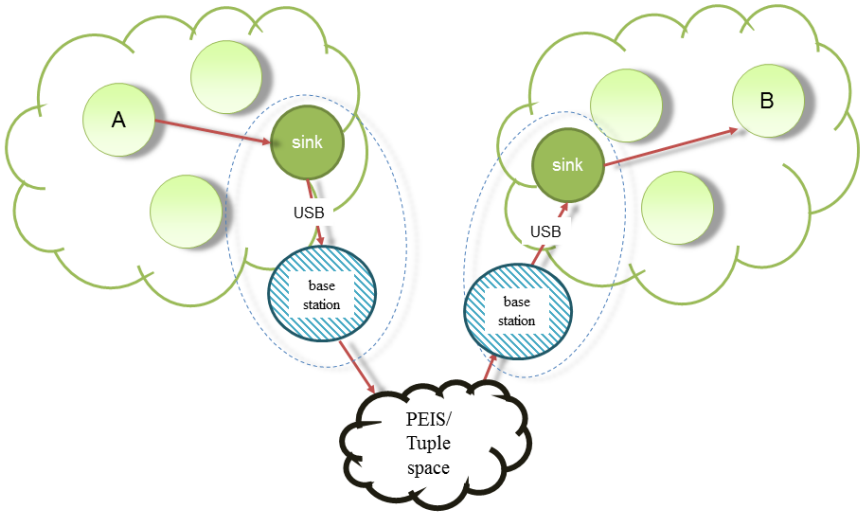

Figure 3.4: The inter-island communication between two foreign motes.

3.4 The RUBICON Addressing Scheme

Addressing algorithm in WSAN have to deal with storage consumption costs, processing costs and the energy, since addressing cost of storing, processing and energy costs consumption are important in evaluating the performance and improving scalability of an addressing scheme. In particular, the design of the RUBICON addressing scheme must satisfy two conflicting requirements, compactness and simplicity, and the ability to target and route messages to elements within the ecology that may be sensors inside the islands or devices connected to PEIS.

Compactness. WSAN nodes have strong memory constraint. Applying current address such as 32 bit IP addresses or the 128 bit IPv6 addresses is possible, but results in needless overhead. Significant savings can be realized by using only the minimum number of bits required to uniquely address all the nodes. However, an address-based WSAN has its side effects because constructing and maintaining an address structure usually requires additional cost. It is important to consider this cost in WSAN, where every bit transmitted can reduce the lifetime of the network. A key factor to consider is that both the packet size and data rate in WSAN are usually very small.

Simplicity. The low processing capabilities of motes prevents the use of multi-hop routing protocols. Moreover, multi-hop routing in WSN is affected by new nodes constantly entering or leaving the network. Our cluster-based routing protocols based on a hierarchical network organization provides an efficient and at the same time simple enough to be managed.

Dynamicity. The address of island of a mote is dynamic, i.e. it can be set at run time and not at compile time. In this way a mote can join any island. Moreover, each mote in the island can have a local address identical to motes of other islands. The address of the island is able to make independent islands. In this way, there is no need to know the addresses of the local mote in the other islands when you have to deploy a new island in the ecology since

there is no risk of address collision. The sub-addressing solves this problem by passing Island Address at Join time to the joining time.

The RUBICON address is defined as follows:

```
typedef r_addr {
    uint16_t pid;
    uint16_t devid;
} r_addr_t;
```

where `pid` is the `PEIS` identifier and it addresses software components in a PC or a Robot, and `devid` is the device identifier. The former identifies also the island where the PC is connected, while the latter is used only to address a mote displaced in the island identified by the `pid`.

```
// reserved island addresses
BROADCAST 0xFFFF
LOCAL_ISLAND 0xFFFE
PROXY_PEIS_ID 0xFFFD

// reserved devid addresses
AM_BROADCAST 0xFFFF
LOCA_MOTE 0xFFFE
NULL_MOTE 0xFFFD
```

3.4.1 Island Collision Resolution and Safe Join

One problem that has been addressed in the implementation of the islands topology, is the possible overlap of the islands. A mote may be in communication range of two sinks two different islands and create interferences. One possible solution is to take advantage of the multiple channels of the MAC protocol, which, however, are few (3 or 4).

The problem is solved by the network layer of the Communication Layer by looking up addresses of the recipient when a message is received by a mote. In the case of sink, if the recipient is another island then we check whether the message should be forwarded (remember that the sink has the duty to forward the messages to the basestation to the remote islands). In the latter case, if the island address of the sending mote is different from that of the island which the sink belongs to, then the message is ignored because it is a mote of another island. If instead, it is the same island, then the message must be forwarded to the basestation (which in turn will forward the message to the remote island via PEIS). In the case of a generic mote of an island (ie, different from the sink) if the message is addressed to another island then it can simply be ignored.

This behavior is represented by the flowcharts of Figures 3.5 and 3.6. The test of the message that avoids island collision is expressed by the right branches of the two flowcharts. The branches on the left side express the join management. The sink keep track of the list of device ids of the motes belonging to its island. When a join message is received from a mote that wants to take part of the island, the sink check if the mote is already in the list. If so, the join message is ignored (because the mote that sent the join message may be trying to join another

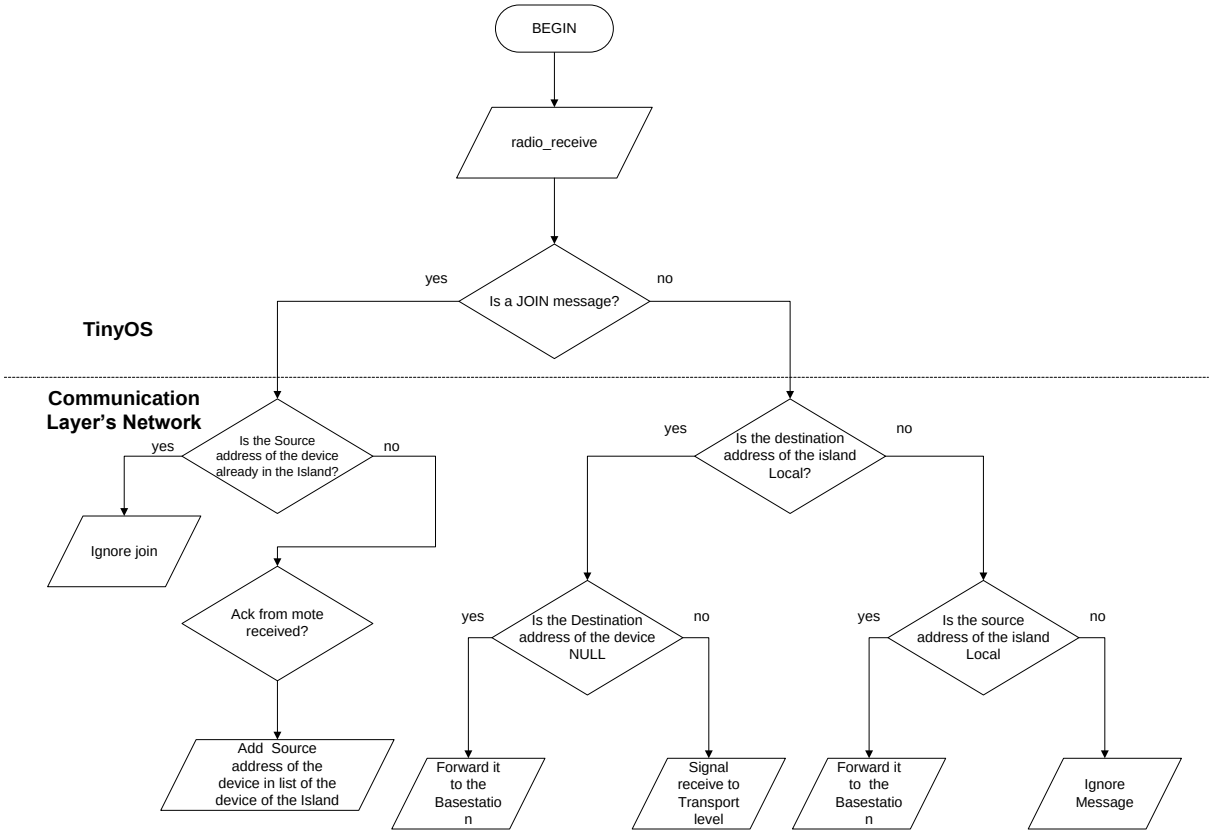

Figure 3.5: Flowchart of the reception of a message from the sink network level.

island and the message has arrived by chance to the sink). Otherwise, the sink send a join reply message to mote communicating that it is available to take the mote in the island. The mote that sent the join message replies with an Ack message. Only when the Ack message is received by the sink, the sink adds it in the list of the motes of the list. This additional check is necessary because the join message by the mote could reach two sinks, with the risk of joining two islands simultaneously. In this case, the mote decides which island to join and sends the Ack only to the sink of the island that he decided to join. The distinctions between ordinary messages and join messages is guaranteed by the mechanism of the Active Message Interfaces of TinyOS.

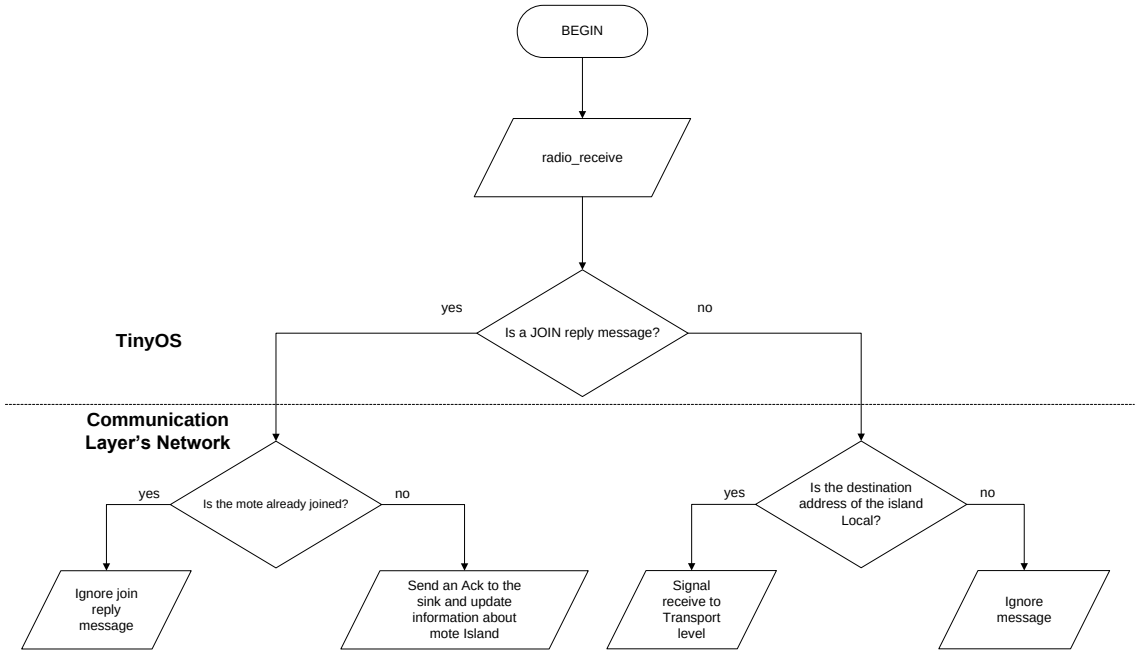

Figure 3.6: Flowchart of the reception of a message from the mote network level.

Chapter 4

Access to non-WSAN Sensors

With the aim of providing access to non WSAN sensors in Rubicon, the SALAD middleware developed by Tecnalia offers to third applications the possibility of communicating with a set of sensors and actuators installed in a homelab by mean of an easy access to the information and a flexible environment for application development.

4.1 The SALAD Middleware

SALAD is a framework which allows the interconnection and cooperation among several systems of sensors and actuators in one platform, providing an easy access to the information and a flexible environment for application development.

The main goal of SALAD is to keep the programmer away from the underlying technology, providing a common API to access the hardware, regardless the manufacturer and type of the devices which will be used. This feature allows the programmer to focus only on the development of the services which will make use of such devices.

The access to the information is made by several ways, depending on the needs of each application, or even depending on the needs of the modules of an application. The access can be carried out by using an enquiry, an event or by sampling.

1. *By enquiry*: The applications ask SALAD for the information they need, in the time they need it, in an explicit way. This is the preferred method to process the information in an off-line mode.

2. *By event*: Each time a change in a sensor status occurs, SALAD sends automatically the status updates to subscribed applications, by means of a message service. This is the preferred way to process information in real time.

3. *By sampling*: The applications which need to do so can subscribe to the Sample & Hold service, selecting the required sampling frequency. In this way they will receive, periodically at a fixed rate, an “image” of the status of the whole system at a certain time. This method may be useful for applications which require a periodic sampling of the information.

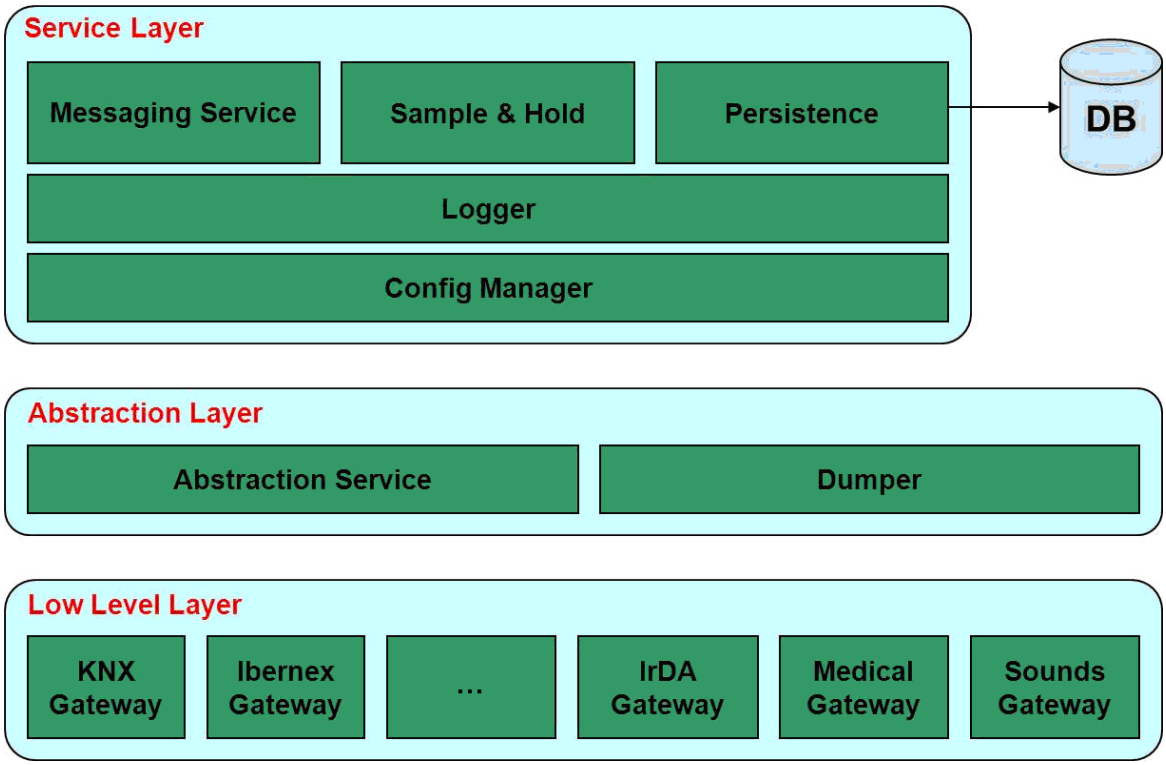

Figure 4.1: General module architecture of SALAD

4.1.1 Architecture

The following paragraphs show the modules of the SALAD architecture, see also Figure 4.1.

4.1.2 Low Level Layer

This is the lowest layer, in charge of performing the direct communication with the hardware. In this layer there is a module for each of the supported network. The main networks already integrated in the platform are the following ones:

Domotic networks based on KNX Connecting module with domotic networks KNX, both wired and wireless. The events generated by the sensors are sent by means of the serial interface of KNX to the SALAD host, where they are managed. Also an additional PC can be used as a gateway to perform tests in laboratory.

Ibernexs Network for Tele-assistance Connecting module with the home tele-assistance terminal of the manufacturer Ibernex and its network of wireless sensors. The connection is made by an USB port, and the terminal must be previously configured to be able to receive the events form the sensors.

Sensors are connected by RF with the tele-assistance module of Ibernex, which sends the events to SALAD.

Sound recognition system by Tecnalia This system collects the data regarding the probabilistic estimation of the recognition of home sounds. Data are sent via sockets to SALAD from a SW for sound detection, developed by Tecnalia, which detects the sounds generated inside the home, processes them (searching for previously learnt sounds, labeled as relevant), and sends them to SALAD with a probabilistic level.

There are 2 methods for processing the information within SALAD:

1. Sound event module: The sound recognition software sends the events generated by the detection system only if the sound is recognizable. That is to say, if the system recognizes the sound of bell or door, it sends an event indicating that this sound has been detected. There is a previous pre-processing step in the software, so SALAD simply stores that value.
2. Sound probabilistic module: SALAD continuously receives the probabilistic data of each frame. In this case, SALAD will perform the processing in order to store the samplings and to determine which sound has been detected within that frame.

Medical devices Bluetooth Taidoc/RGB The current available medical devices are connected via Bluetooth to SALAD. The connection profile is SPP, it is a serial virtual port by Bluetooth, allowing for 7 connection open simultaneously. The operation is based on automatic connections.

The SALAD system is continuously searching for Bluetooth devices. When it finds an active TAIDOC device, it asks for data and stores them. In this way, it allows the user to measure his/her physiological signals without the need to follow certain rules.

TV LG Gateway via Bluetooth This module for monitoring and actuation allows to obtain the current status of the TV set and to command it through the serial port of this device. In order for the communication to be performed with SALAD by means of a wireless interface, a RS-232/Bluetooth converter has been set up, allowing a BT connection for the TV set.

When SALAD starts its execution, it searches for devices with Bluetooth adapter which allows the RS-232 / BT conversion. Once the device is detected, the connection is established and it remains open as long as SALAD is running. The TV monitoring activity is performed for:

1. Switch On / Off
2. Video source used
3. Current volume level

In addition, commands simulating the TV remote control can be issued from any SALAD access point.

The manufacturer of the TV set is LG, with a proprietary communication protocol via RS-232.

4.1.3 Abstraction Layer

This layer collects the information produced by the low level layer, and processes it following SALAD rules. If the information gas been produced by a registered sensor, it is stored in the data base and a message is sent to all registered components within the message system. The abstraction modules are the following ones:

1. **Abstraction Service:** it checks the validity of sensors data. It makes requirements to the higher layer of the data base in order to know whether the sensor and its associated data have been registered correctly.
2. **Dumper Service:** once the events have been verified by the Abstraction Service, the Dumper Service sends them to the higher layer of the data base in order to be stored.

4.1.4 Service Layer

This is the core of the framework. It provides basic services to both the lower and higher layers. It includes the following services:

1. **Config Manager:** Service for supporting the configuration. With this service the components (from SALAD or from the application) have an easy access to files of external configuration. It also eases the parametrization of several components of the application.
2. **Logger:** Log service for printing messages on screen and in a file. It supports several levels depending on the type of message to be shown: informative, advertence, error, debugging, etc. It eases the tasks of testing and error fixing.
3. **Persistence:** Service of persistence in the data base. It allows to store information permanently and to access to it later. It also allows to make enquiries to SALAD about the networks, sensors and rooms in the installation.
4. **Messaging Service:** it is used by both SALAD and the applications to receive information about the stuatus of the system or the sensors. It allows the enquiries to sensor networks by means of events.
5. **Sample & Hold:** It allows to receive, periodically and with fixed intervals, an image of the status of the whole system in a certain time.

4.2 Integration with RUBICON

With the aim to provide to third applications (such as the RUBICON case) the information collected from the networks and to allow remote actuation, SALAD offers a communication protocol, UDP, for that purpose. The Tecnalia Middleware (SALAD) allows an external application (Rubicon) to:

1. Receive the status of all the sensors

2. Send actions to the actuators

The dedicated ports for the communication are the following ones:

Internal Service port (Tecnalia Middleware will be listening in this port)

Port: 9000

External Service port (Rubicon will be listening in this port)

Port: 9100

4.2.1 Receive the status of all the sensors:

Using the UDP protocol it is possible to communicate with SALAD to start the transmission of the data and then stop it. In order to do this there are three commands available:

//Request for the list of HarwareIds

#SALAD#0:2:2:0#END#

//Starting the sending data with frequency 2Hz (500ms)

#SALAD#0:1:2:0#END#

//Stopping the sending of data

#SALAD#0:0:2:0#END#

The first command "#SALAD#0:2:2:0#END#" asks for the name of the devices that are running in SALAD. When SALAD receives this command it will send a command such as:

#SALAD#:DeviceHardwareId1:DeviceHardwareId2:....:DeviceHardwareIdN:#END#

The order of the Devices names is the same one used by SALAD to send the data when it receives the command "#SALAD#0:1:2:0#END"

When SALAD receives the command "#SALAD#0:1:2:0#END" it will start to send the status of all the sensors each 500ms. The message which will be sent has the format:

#SALAD#:StatusDevice1:StatusDevice2:....:StatusDeviceN:#END#

The stopping of the transmission will be done when SALAD receives the command "#SALAD#0:0:2:0#END#".

In Figure 4.2 the graph of this sequence is shown.

4.2.2 Send actions to the actuators

Using the UDP protocol it is possible to communicate with SALAD, in order to send commands to the available actuators in the networks integrated in SALAD. The available actuators are:

1. KNX:
    (a) Open/Close the door

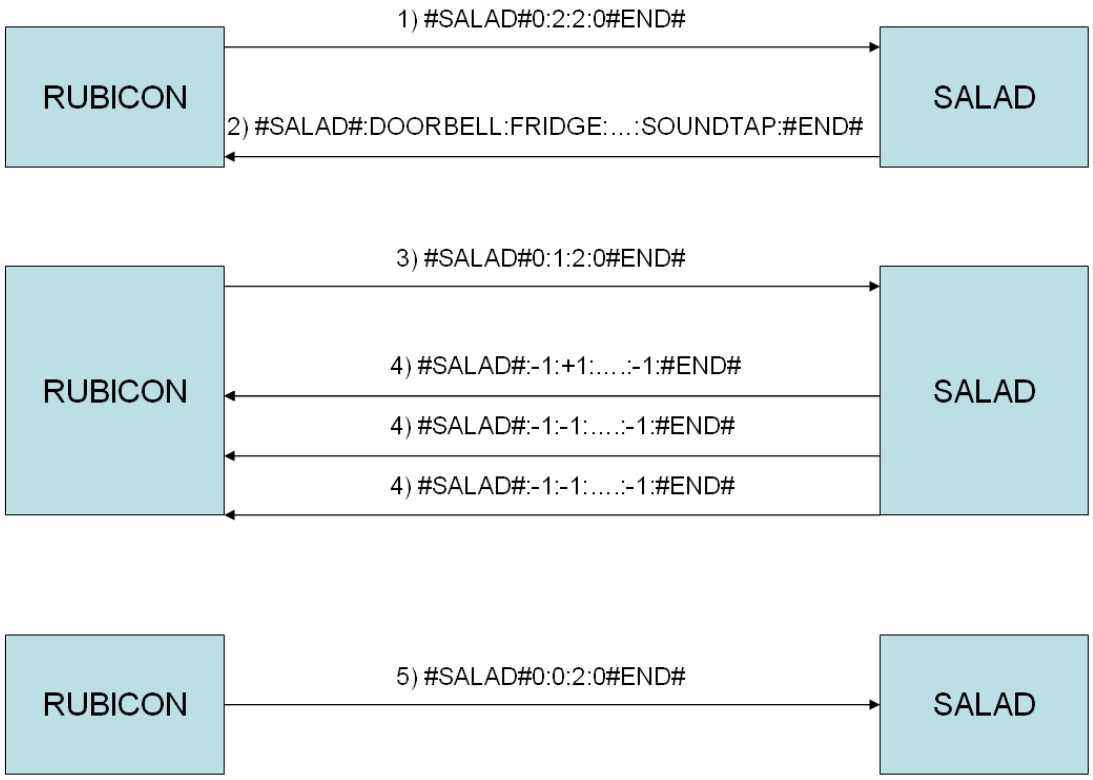

Figure 4.2: Graph of SALAD and Rubicon communication

(b) Switch On/Off of Outside and Bedroom lights

(c) Up/Down movement of the blind of the bedroom

2. TV:

(a) Switch On/Off the TV set

(b) Change the AV mode (for example between TDT and the AV3 for the Wii)

(c) Up/Down the volume

The format of the UDP message is:

#SALAD#2:DeviceHardwareId:Action:0#END#

where “DeviceHardwareId” is the name of the actuator, “Action” is what you want to do.

In the following table the HardwareIds are listed.

HardwareId	**Tag**
1/4/1	KNX_DOOR
1/1/1	KNX_LIGHT_ENTRANCE
2/1/1	KNX_LIGHT_BEDROOM
2/5/1	KNX_BLIND_UP
2/5/2	KNX_BLIND_DOWN
2/5/3	KNX_BLIND_STOP
1/4/2	KNX_DOOR_OPEN
LG	TV_STATE

In the following table the Actions are listed.

Description	**Action**
For all KNX actuators	0 or 1
TV channel 0	0
TV channel to 1	1
TV channel to 2	2
TV channel to 3	3
TV channel to 4	4
TV channel to 5	5
TV channel to 6	6
TV channel to 7	7
TV channel to 8	8
TV channel to 9	9
TV ON	10
TV OFF	11
TV volume Up	12
TV volume Down	13
TV mute	14
TV channel Up	15
TV channel Down	16

In the following tables you can find some example of actuation.

KNX Actuation:

Open the door	#SALAD#2**:1/4/2**:1:0#END#
Close the door	#SALAD#2**:1/4/2**:0:0#END#
Switch On the Bedroom light	#SALAD#2**:2/1/1**:1:0#END#
Switch Off the Bedroom light	#SALAD#2**:2/1/1**:0:0#END#
Switch On the Entrance/Outside light	#SALAD#2**:1/1/1**:1:0#END#
Switch Off the Entrance/Outside light	#SALAD#2**:1/1/1**:0:0#END#

TV Actuation:

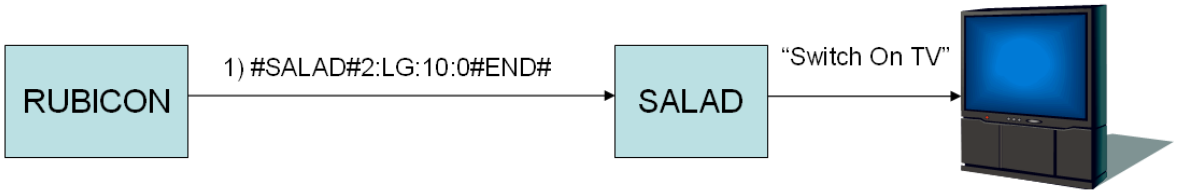

Figure 4.3: Graph TV Actuation

Switch On the TV	#SALAD#2**:LG**:10:0#END#
Switch Off the TV	#SALAD#2**:LG**:11:0#END#
Change to the Channel 4 (from 0 to 9)	#SALAD#2**:LG**:4:0#END#
Going Up one channel	#SALAD#2**:LG**:15:0#END#
Going Down one channel	#SALAD#2**:LG**:16:0#END#
Change the AV mode	It will be available,
Up the Volume (1 step)	#SALAD#2**:LG**:12:0#END#
Down the Volume (1 step)	#SALAD#2**:LG**:13:0#END#
Mute the Volume	#SALAD#2**:LG**:14:0#END#

In Figure 4.3 the graph of the sequence for the TV Actuation is shown.

Chapter 5

Conclusions

We summarize this report with an overview of the planned tasks for the second year of RUBICON within WP1, Task 1.2: Integration of robotic components and WSAN (M13-24), as documented in the Description of Work, and the performed work as documented in this deliverable and in the Interim Report.

Task 1.2 have been performed as planned and finished in March 2013. The result of this task has provided the RUBICON with the integration of specific previously existing robotic devices and software components in terms of sensors, actuators, and commands. These include the integration of autonomous robots developed at ORU by means of the interface between the PEIS middle-ware and ROS, support to the routing of communication between WSAN islands connected with PEIS, and access to non-WSAN Sensors within the RUBICON ecology.

The interface between the PEIS middleware and ROS is provided in order to allow multiple robots equipped with ROS (Robot Operating System) to be part of the RUBICON system, i.e. to exchange data with other RUBICON nodes. Such choice is dictated by the fact that ROS entails the presence of a centralized master program (ROS master) managing the connections between cooperating modules; this structure does not fit the purpose of having a distributed and dynamic system where robots can join and leave the network at any time.

Thanks to the effort made in the development of Communication Layer described in D1.3.2, the multi island routing allows the ecology to integrate “for free” Robots and PCs since any node that has a PEIS-id is seen in RUBICON as an autonomous islands consisting of a single node. The routing between the islands is obtained also by the gateway component described in D1.2.

With the aim of providing access to non WSAN sensors in RUBICON, the SALAD middleware developed by Tecnalia offers to third applications the possibility of communicating with a set of sensors and actuators installed in a homelab by mean of an easy access to the information and a flexible environment for application development.

The software is currently present at and in active use by all of the partners. Based on the feedback of the partners, further work is expected in order to validate the performance and tune all the mechanisms presented in this deliverable once they are operating in the final application testbeds.

Bibliography

[1] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14–15):2826–2841, 2007.

[2] Shio Kumar Singh, MP Singh, and DK Singh. A survey of energy-efficient hierarchical cluster-based routing in wireless sensor networks. *International Journal of Advanced Networking and Application (IJANA)*, 2(02):570–580, 2010.